112404 15866 U.S. PTO

PTO/SB/05 (09-04)
Approved for use through 07/31/2006. OMB 0651-0032
U.S. Patent and Trademark Office. U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No.	2222.0300002
First Inventor	Danny Lange
Title	Network System Extensible By Users
Express Mail Label No.	

APPLICATION ELEMENTS
See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO: **Commissioner for Patents, P.O. Box 1450, Alexandria VA 22313-1450**

17541 U.S. PTO 10/995159 112404

1. [X] **Fee Transmittal Form** (e.g., PTO/SB/17) *(Submit an original and a duplicate for fee processing)*
2. [ ] **Applicant claims small entity status.** See 37 CFR 1.27.
3. [X] **Specification** [*Total Pages* 105] Both the claims and abstract must start on a new page *(For information on the preferred arrangement, see MPEP 608.01(a))*
4. [X] **Drawing(s)** (*35 U.S.C. 113*) [*Total Sheets* 17]
5. **Oath or Declaration** [*Total Sheets* ____]
   a. [ ] Newly executed (original or copy)
   b. [X] A copy from a prior application (37 CFR 1.63(d)) *(for continuation/divisional with Box 18 completed)*
      i. [ ] DELETION OF INVENTOR(S) Signed statement attached deleting inventor(s) name in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).
6. [X] **Application Data Sheet.** See 37 CFR 1.76
7. [ ] **CD-ROM or CD-R** in duplicate, large table or Computer Program *(Appendix)*
   [ ] Landscape Table on CD
8. **Nucleotide and/or Amino Acid Sequence Submission** *(if applicable, items a. – c. are required)*
   a. [ ] Computer Readable Form (CRF)
   b. Specification Sequence Listing on:
      i. [ ] CD-ROM or CD-R (2 copies); or
      ii. [ ] Paper
   c. [ ] Statements verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

9. [X] **Assignment Papers** (cover sheet & document(s))
   Name of Assignee ____
10. [ ] **37 CFR 3.73(b) Statement** *(when there is an assignee)* [ ] **Power of Attorney**
11. [ ] **English Translation Document** *(if applicable)*
12. [X] **Information Disclosure Statement** (PTO/SB/08 or PTO-1449)
    [ ] Copies of citations attached
13. [X] **Preliminary Amendment**
14. [X] **Return Receipt Postcard** (MPEP 503) *(Should be specifically itemized)*
15. [ ] **Certified Copy of Priority Document(s)** *(if foreign priority is claimed)*
16. [ ] **Nonpublication Request** under 35 U.S.C. 122(b)(2)(B)(i). Applicant must attach form PTO/SB/35 or equivalent.
17. [X] Other: Authorization under 37CFR 1.136(a)(3)
    please transfer microfice appendices A-D from Appl. ~~No. 09/178,366 to present application~~

18. If a CONTINUING APPLICATION, *check appropriate box, and supply the requisite information below and in the first sentence of the specification following the title, or in an Application Data Sheet under 37 CFR 1.76:*

[X] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No.: ....

Prior application information: *Examiner* Geckil, Mehmet B. *Art Unit:* 2142

19. CORRESPONDENCE ADDRESS

[X] The address associated with Customer Number: 26111 *OR* [ ] Correspondence address below

Name					
Address					
City		State		Zip Code	
Country		Telephone		Fax	

Signature	[illegible]	Date	11/24/04
Name (Print/Type)	Thomas C. Fiala	Registration No. (Attorney/Agent)	43,610

15866 U.S.PTO
112404

PTO/SB/17 (10-04v2)
Approved for use through 07/31/2006. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

FEE TRANSMITTAL for FY 2005

Effective 10/01/2004. Patent fees are subject to annual revision.

☐ Applicant claims small entity status. See 37 CFR 1.27

TOTAL AMOUNT OF PAYMENT ($) 1,016.00

Complete if Known	
Application Number	To be assigned
Filing Date	November 24, 2004
First Named Inventor	Danny Lange
Examiner Name	To be assigned
Art Unit	To be assigned
Attorney Docket No.	2222.0300002

METHOD OF PAYMENT *(check all that apply)*

☐ Check [X] Credit card ☐ Money Order [X] Other ☐ None

☐ Deposit Account: **Charge any deficiencies or credit any overpayments in the fees to Deposit Acct. No. 19-0036.

Deposit Account Number: 19-0036

Deposit Account Name: Sterne, Kessler, Goldstein & Fox P.L.L.C.

The Director is authorized to: *(check all that apply)*

☐ Charge fee(s) indicated below ☐ Credit any overpayments

☐ Charge any additional fee(s) or any underpayment of fee(s)

☐ Charge fee(s) indicated below, **except for the filing fee** to the above-identified deposit account.

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1001	790	2001	395	Utility filing fee	790.00
1002	350	2002	175	Design filing fee	
1003	550	2003	275	Plant filing fee	
1004	790	2004	395	Reissue filing fee	
1005	160	2005	80	Provisional filing fee	

SUBTOTAL (1) ($) 790.00

2. EXTRA CLAIM FEES FOR UTILITY AND REISSUE

			Extra Claims		Fee from below		Fee Paid
Total Claims	21	-20** =	1	X	18.00	=	18.00
Independent Claims	4	- 3** =	1	X	88.00	=	88.00
Multiple Dependent					0	=	0.00

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description
1202	18	2202	9	Claims in excess of 20
1201	88	2201	44	Independent claims in excess of 3
1203	300	2203	150	Multiple dependent claim, if not paid
1204	88	2204	44	** Reissue independent claims over original patent
1205	18	2205	9	** Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) ($) 106.00

**or number previously paid, if greater; For Reissues, see above

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1051	130	2051	65	Surcharge - late filing fee or oath	
1052	50	2052	25	Surcharge - late provisional filing fee or cover sheet	
1053	130	1053	130	Non-English specification	
1812	2,520	1812	2,520	For filing a request for *ex parte* reexamination	
1804	920*	1804	920*	Requesting publication of SIR prior to Examiner action	
1805	1,840*	1805	1,840*	Requesting publication of SIR after Examiner action	
1251	110	2251	55	Extension for reply within first month	
1252	430	2252	215	Extension for reply within second month	
1253	980	2253	490	Extension for reply within third month	
1254	1,530	2254	765	Extension for reply within fourth month	
1255	2,080	2255	1,040	Extension for reply within fifth month	
1401	340	2401	170	Notice of Appeal	
1402	340	2402	170	Filing a brief in support of an appeal	
1403	300	2403	150	Request for oral hearing	
1451	1,510	1451	1,510	Petition to institute a public use proceeding	
1452	110	2452	55	Petition to revive - unavoidable	
1453	1,370	2453	685	Petition to revive - unintentional	
1501	1,370	2501	685	Utility issue fee (or reissue)	
1502	490	2502	245	Design issue fee	
1503	660	2503	330	Plant issue fee	
1460	130	1460	130	Petitions to the Commissioner	
1807	50	1807	50	Processing fee under 37 CFR 1.17(q)	
1806	180	1806	180	Submission of Information Disclosure Stmt	
8021	40	8021	40	Recording each patent assignment per property (times number of properties)	120.00
1809	790	2809	395	Filing a submission after final rejection (37 CFR 1.129(a))	
1810	790	2810	395	For each additional invention to be examined (37 CFR 1.129(b))	
1801	790	2801	395	Request for Continued Examination (RCE)	
1802	900	1802	900	Request for expedited examination of a design application	

Other fee (specify) ____________

*Reduced by Basic Filing Fee Paid

SUBTOTAL (3) ($) 120.00

SUBMITTED BY				(Complete *(if applicable)*)	
Name *(Print/Type)*	Thomas C. Fiala	Registration No. *(Attorney/Agent)*	43,610	Telephone	(202) 371-2600
Signature	[illegible]			Date	11/24/04

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

This collection of information is required by 37 CFR 1.17 and 1.27. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Sterne Kessler Goldstein Fox
ATTORNEYS AT LAW

November 24, 2004

WRITER'S DIRECT NUMBER:
(202) 772-8835
INTERNET ADDRESS:
TFIALA@SKGF.COM

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Re: U.S. Non-Provisional Utility Patent Application under 37 C.F.R. § 1.53(b)
Appl. No. To be assigned; Filed: November 24, 2004
(Continuation of Appl. No. 09/712,712; Filed: November 14, 2000)
For: **Network System Extensible By Users**
Inventors: Lange *et al.*
Our Ref: 2222.0300002

Sir:

The following documents are forwarded herewith for appropriate action by the U.S. Patent and Trademark Office:

1. Credit Card Payment Form (PTO-2038) for $1,016.00 to cover:
   $790.00 Patent Application Fee;
   $106.00 Excess Claims Fee;
   $120.00 Recordation Fee;

2. Utility Patent Application Transmittal Form (PTO/SB/05);

3. Fee Transmittal Form (PTO/SB/17);

4. Authorization to Treat a Reply As Incorporating An Extension of Time Under 37 C.F.R. § 1.136(a)(3);

5. U.S. Utility Patent Application entitled:

   Network System Extensible By Users

   and naming as inventors:

   Danny Lange
   Barbara Nelson
   Jing Su
   James E. White

the application consisting of:

a. An Application Data Sheet (37 C.F.R. § 1.76);

b. A copy of the executed combined Declaration and Power of Attorney, as originally filed in U.S. Appl. No. 09/178,366;

c. A specification containing:

i. 86 pages of description prior to the claims;

ii. 18 pages of claims (76 claims);

iii. a one (1) page abstract;

d. 17 sheets of drawings (Figures 1-17);

6. A Preliminary Amendment;

7. A copy of the Revocation of Prior Power of Attorney and Appointment of New Attorneys of Record, as originally filed in U.S. Appl. No. 9/712,712;

8. Recordation Form Cover Sheet (Form PTO-1595);

9. A copy of the executed Assignment to General Magic, recordation of which is hereby respectfully requested;

10. Recordation Form Cover Sheet (Form PTO-1595);

11. A copy of the executed Assignment to Intellectual Ventures Patent Holding I, L.L.C., recordation of which is hereby respectfully requested;

12. Recordation Form Cover Sheet (Form PTO-1595);

13. A copy of the executed Change of Name to Ben Franklin Patent Holding L.L.C., recordation of which is hereby respectfully requested;

14. An Information Disclosure Statement;

15. Form PTO-1449 (2 sheets citing 22 documents); and

16. Two (2) return postcards.

It is respectfully requested that, of the two attached postcards, one be stamped with the filing date of these documents and returned to our courier, and the other, prepaid postcard, be stamped with the filing date and unofficial application number and returned as soon as possible. The U.S. Patent and Trademark Office is hereby authorized to charge any fee deficiency, or credit any overpayment, to our Deposit Account No. 19-0036.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Thomas C. Fiala
Attorney for Applicants
Registration No. 43,610

TCF/LAG/lam
Enclosures

337900.1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Confirmation No.: To be assigned
Lange *et al.*	Art Unit: To be assigned
Appl. No.: To be assigned *(Continuation of Appl. No. 09/712,712; Filed: November 14, 2000)*	Examiner: To be assigned
Filed: November 24, 2004	Atty. Docket: 2222.0300002
For: **Network System Extensible By Users**	

Authorization to Treat a Reply as Incorporating an Extension of Time Under 37 C.F.R. § 1.136(a)(3)

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

The U.S. Patent and Trademark Office is hereby authorized to treat any concurrent or future reply that requires a petition for an extension of time under this paragraph for its timely submission, as incorporating a petition for extension of time for the appropriate length of time. The U.S. Patent and Trademark Office is hereby authorized to charge all required extension of time fees to our Deposit Account No. 19-0036, if such fees are not otherwise provided for in such reply.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Thomas C. Fiala
Attorney for Applicants
Registration No. 43,610

Date: 11/24/04

1100 New York Avenue, N.W.
Washington, D.C. 20005-3934
(202) 371-2600

338176.1

15866 U.S. PTO
112404

17541 U.S. PTO
10/995159
112404

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No.	2222.0300002
First Inventor	Danny Lange
Title	Network System Extensible By Users
Express Mail Label No.	

ADDRESS TO: Commissioner for Patents
P.O. Box 1450
Alexandria VA 22313-1450

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

1. [X] **Fee Transmittal Form** (e.g., PTO/SB/17)
   (Submit an original and a duplicate for fee processing)
2. [ ] **Applicant claims small entity status.** See 37 CFR 1.27.
3. [X] **Specification** [*Total Pages* 105]
   Both the claims and abstract must start on a new page
   (For information on the preferred arrangement, see MPEP 608.01(a))
4. [X] **Drawing(s)** (*35 U.S.C. 113*) [*Total Sheets* 17]
5. **Oath or Declaration** [*Total Sheets* ______]
   a. [ ] Newly executed (original or copy)
   b. [X] A copy from a prior application (37 CFR 1.63(d)) *(for continuation/divisional with Box 18 completed)*
   i. [ ] DELETION OF INVENTOR(S)
   Signed statement attached deleting inventor(s) name in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).
6. [X] **Application Data Sheet.** See 37 CFR 1.76
7. [ ] **CD-ROM or CD-R** in duplicate, large table or Computer Program *(Appendix)*
   [ ] Landscape Table on CD
8. **Nucleotide and/or Amino Acid Sequence Submission** *(if applicable, items a. – c. are required)*
   a. [ ] Computer Readable Form (CRF)
   b. Specification Sequence Listing on:
   i. [ ] CD-ROM or CD-R (2 copies); or
   ii. [ ] Paper
   c. [ ] Statements verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

9. [X] **Assignment Papers** (cover sheet & document(s))
   Name of Assignee ______________________
10. [ ] **37 CFR 3.73(b) Statement** *(when there is an assignee)* [ ] **Power of Attorney**
11. [ ] **English Translation Document** *(if applicable)*
12. [X] **Information Disclosure Statement** (PTO/SB/08 or PTO-1449)
    [ ] Copies of citations attached
13. [X] **Preliminary Amendment**
14. [X] **Return Receipt Postcard** (MPEP 503) *(Should be specifically itemized)*
15. [ ] **Certified Copy of Priority Document(s)** *(if foreign priority is claimed)*
16. [ ] **Nonpublication Request** under 35 U.S.C. 122(b)(2)(B)(i). Applicant must attach form PTO/SB/35 or equivalent.
17. [X] Other: Authorization under 37CFR 1.136(a)(3)
    please transfer microfice appendices A-D from Appl. ~~No. 09/178,366 to present application~~

18. If a CONTINUING APPLICATION, *check appropriate box, and supply the requisite information below and in the first sentence of the specification following the title, or in an Application Data Sheet under 37 CFR 1.76:*

[X] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No.: ....

Prior application information: *Examiner* Geckil, Mehmet B. *Art Unit:* 2142

19. CORRESPONDENCE ADDRESS

[X] The address associated with Customer Number: 26111 **OR** [ ] Correspondence address below

Name					
Address					
City		State		Zip Code	
Country		Telephone		Fax	

Signature	[illegible]	Date	11/24/04
Name (Print/Type)	Thomas C. Fiala	Registration No. (Attorney/Agent)	43,610

15866 U.S. PTO
112404

FEE TRANSMITTAL for FY 2005

Effective 10/01/2004. Patent fees are subject to annual revision.

☐ Applicant claims small entity status. See 37 CFR 1.27

TOTAL AMOUNT OF PAYMENT ($) 1,016.00

Complete if Known	
Application Number	To be assigned
Filing Date	November 24, 2004
First Named Inventor	Danny Lange
Examiner Name	To be assigned
Art Unit	To be assigned
Attorney Docket No.	2222.0300002

METHOD OF PAYMENT *(check all that apply)*

☐ Check [X] Credit card ☐ Money Order [X] Other ☐ None

☐ Deposit Account: **Charge any deficiencies or credit any overpayments in the fees to Deposit Acct. No. 19-0036.

Deposit Account Number: 19-0036

Deposit Account Name: Sterne, Kessler, Goldstein & Fox P.L.L.C.

The Director is authorized to: *(check all that apply)*

☐ Charge fee(s) indicated below ☐ Credit any overpayments

☐ Charge any additional fee(s) or any underpayment of fee(s)

☐ Charge fee(s) indicated below, **except for the filing fee** to the above-identified deposit account.

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1001	790	2001	395	Utility filing fee	790.00
1002	350	2002	175	Design filing fee	
1003	550	2003	275	Plant filing fee	
1004	790	2004	395	Reissue filing fee	
1005	160	2005	80	Provisional filing fee	

SUBTOTAL (1) ($) 790.00

2. EXTRA CLAIM FEES FOR UTILITY AND REISSUE

			Extra Claims		Fee from below		Fee Paid
Total Claims	21	-20** =	1	x	18.00	=	18.00
Independent Claims	4	- 3** =	1	x	88.00	=	88.00
Multiple Dependent					0	=	0.00

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description
1202	18	2202	9	Claims in excess of 20
1201	88	2201	44	Independent claims in excess of 3
1203	300	2203	150	Multiple dependent claim, if not paid
1204	88	2204	44	** Reissue independent claims over original patent
1205	18	2205	9	** Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) ($) 106.00

**or number previously paid, if greater; For Reissues, see above

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1051	130	2051	65	Surcharge - late filing fee or oath	
1052	50	2052	25	Surcharge - late provisional filing fee or cover sheet	
1053	130	1053	130	Non-English specification	
1812	2,520	1812	2,520	For filing a request for *ex parte* reexamination	
1804	920*	1804	920*	Requesting publication of SIR prior to Examiner action	
1805	1,840*	1805	1,840*	Requesting publication of SIR after Examiner action	
1251	110	2251	55	Extension for reply within first month	
1252	430	2252	215	Extension for reply within second month	
1253	980	2253	490	Extension for reply within third month	
1254	1,530	2254	765	Extension for reply within fourth month	
1255	2,080	2255	1,040	Extension for reply within fifth month	
1401	340	2401	170	Notice of Appeal	
1402	340	2402	170	Filing a brief in support of an appeal	
1403	300	2403	150	Request for oral hearing	
1451	1,510	1451	1,510	Petition to institute a public use proceeding	
1452	110	2452	55	Petition to revive - unavoidable	
1453	1,370	2453	685	Petition to revive - unintentional	
1501	1,370	2501	685	Utility issue fee (or reissue)	
1502	490	2502	245	Design issue fee	
1503	660	2503	330	Plant issue fee	
1460	130	1460	130	Petitions to the Commissioner	
1807	50	1807	50	Processing fee under 37 CFR 1.17(q)	
1806	180	1806	180	Submission of Information Disclosure Stmt	
8021	40	8021	40	Recording each patent assignment per property (times number of properties)	120.00
1809	790	2809	395	Filing a submission after final rejection (37 CFR 1.129(a))	
1810	790	2810	395	For each additional invention to be examined (37 CFR 1.129(b))	
1801	790	2801	395	Request for Continued Examination (RCE)	
1802	900	1802	900	Request for expedited examination of a design application	

Other fee (specify) ____________

*Reduced by Basic Filing Fee Paid

SUBTOTAL (3) ($) 120.00

SUBMITTED BY				(Complete *(if applicable)*)	
Name *(Print/Type)*	Thomas C. Fiala	Registration No. (Attorney/Agent)	43,610	Telephone	(202) 371-2600
Signature	[illegible]			Date	11/24/04

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

This collection of information is required by 37 CFR 1.17 and 1.27. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Sterne Kessler Goldstein Fox
ATTORNEYS AT LAW

November 24, 2004

WRITER'S DIRECT NUMBER:
(202) 772-8835
INTERNET ADDRESS:
TFIALA@SKGF.COM

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Re: U.S. Non-Provisional Utility Patent Application under 37 C.F.R. § 1.53(b)
Appl. No. To be assigned; Filed: November 24, 2004
(Continuation of Appl. No. 09/712,712; Filed: November 14, 2000)
For: **Network System Extensible By Users**
Inventors: Lange *et al.*
Our Ref: 2222.0300002

Sir:

The following documents are forwarded herewith for appropriate action by the U.S. Patent and Trademark Office:

1. Credit Card Payment Form (PTO-2038) for $1,016.00 to cover:
   $790.00 Patent Application Fee;
   $106.00 Excess Claims Fee;
   $120.00 Recordation Fee;

2. Utility Patent Application Transmittal Form (PTO/SB/05);

3. Fee Transmittal Form (PTO/SB/17);

4. Authorization to Treat a Reply As Incorporating An Extension of Time Under 37 C.F.R. § 1.136(a)(3);

5. U.S. Utility Patent Application entitled:

   Network System Extensible By Users

   and naming as inventors:

   Danny Lange
   Barbara Nelson
   Jing Su
   James E. White

the application consisting of:

a. An Application Data Sheet (37 C.F.R. § 1.76);

b. A copy of the executed combined Declaration and Power of Attorney, as originally filed in U.S. Appl. No. 09/178,366;

c. A specification containing:

i. 86 pages of description prior to the claims;

ii. 18 pages of claims (76 claims);

iii. a one (1) page abstract;

d. 17 sheets of drawings (Figures 1-17);

6. A Preliminary Amendment;

7. A copy of the Revocation of Prior Power of Attorney and Appointment of New Attorneys of Record, as originally filed in U.S. Appl. No. 9/712,712;

8. Recordation Form Cover Sheet (Form PTO-1595);

9. A copy of the executed Assignment to General Magic, recordation of which is hereby respectfully requested;

10. Recordation Form Cover Sheet (Form PTO-1595);

11. A copy of the executed Assignment to Intellectual Ventures Patent Holding I, L.L.C., recordation of which is hereby respectfully requested;

12. Recordation Form Cover Sheet (Form PTO-1595);

13. A copy of the executed Change of Name to Ben Franklin Patent Holding L.L.C., recordation of which is hereby respectfully requested;

14. An Information Disclosure Statement;

15. Form PTO-1449 (2 sheets citing 22 documents); and

16. Two (2) return postcards.

It is respectfully requested that, of the two attached postcards, one be stamped with the filing date of these documents and returned to our courier, and the other, prepaid postcard, be stamped with the filing date and unofficial application number and returned as soon as possible. The U.S. Patent and Trademark Office is hereby authorized to charge any fee deficiency, or credit any overpayment, to our Deposit Account No. 19-0036.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Thomas C. Fiala
Attorney for Applicants
Registration No. 43,610

TCF/LAG/lam
Enclosures

337900.1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

Lange *et al.*

Appl. No.: To be assigned *(Continuation of Appl. No. 09/712,712; Filed: November 14, 2000)*

Filed: November 24, 2004

For: **Network System Extensible By Users**

Confirmation No.: To be assigned

Art Unit: To be assigned

Examiner: To be assigned

Atty. Docket: 2222.0300002

Authorization to Treat a Reply as Incorporating an Extension of Time Under 37 C.F.R. § 1.136(a)(3)

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

The U.S. Patent and Trademark Office is hereby authorized to treat any concurrent or future reply that requires a petition for an extension of time under this paragraph for its timely submission, as incorporating a petition for extension of time for the appropriate length of time. The U.S. Patent and Trademark Office is hereby authorized to charge all required extension of time fees to our Deposit Account No. 19-0036, if such fees are not otherwise provided for in such reply.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Thomas C. Fiala
Attorney for Applicants
Registration No. 43,610

Date: 11/24/04

1100 New York Avenue, N.W.
Washington, D.C. 20005-3934
(202) 371-2600

338176.1

EXPRESS MAIL LABEL NO:
EL 030 470 745 US

NETWORK SYSTEM EXTENSIBLE BY USERS

Danny Lange
Barbara Nelson
Jing Su
James E. White

TECHNICAL FIELD OF THE INVENTION

The present invention relates generally to the field of computer software systems and, more particularly, to a network system extensible by users.

CROSS-REFERENCE TO MICROFICHE APPENDICES

A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

CROSS-REFERENCE TO RELATED APPLICATIONS

This Application relates to the subject matter disclosed in the following United States Patent and co-pending United States Applications:

United States Patent No. 5,603,031 to White et al., entitled "System and Method For Distributed Computation Based Upon the Movement, Execution, and Interaction of Processes In a Network;"

United States Application Serial No. 08/609,699, filed March 1, 1996, entitled "Method and Apparatus For Telephonically Accessing and Navigating the Internet;"

United States Application Serial No. 08/798,675, filed February 10, 1997, entitled "System and Method For Distributed Computation Based Upon the Movement, Execution, and Interaction of Processes In a Network;" and

United States Application Serial No. 09/071,717, filed May 1, 1998, entitled "Voice User Interface With Personality."

The above patent and co-pending applications are assigned to the present Assignee and are incorporated herein by reference.

BACKGROUND OF THE INVENTION

Advances in computer and telephony systems have led to the development of numerous technology-driven services, such as electronic mail (e-mail), voice mail, electronic organizers (for appointments and addresses), on-line databases (e.g., for periodicals and stock quotes), and the like. An increasing popularity for these technological services in recent years has spawned an entire industry devoted to the provision and integration of the same. For example, numerous companies now offer e-mail service over the interconnection of computers widely known as the Internet. Other companies offer systems for voice mail services in both private branch exchange (PBX) and public telephone environments. Entities which offer, supply, or otherwise provide services are referred to as "service providers." Entities which purchase, consume, or otherwise use services are referred to as "subscribers."

Many technological services are supported by one or more software applications. These software applications are often developed with a broad spectrum of subscribers in mind. As such, the respective technological services may address the generalized needs of many subscribers, but not the specialized needs of any one particular subscriber or group of subscribers.

With previous techniques, when a subscriber desires to alter, change, modify, or otherwise customize a service to suit his or her own specialized needs, that subscriber must contact the appropriate service provider. If the service provider deems that there is sufficient demand for such customization, the provider will initiate a modification of the supporting software application for the relevant service. Software programmers or developers must then modify the existing software application to address the specialized needs of the requesting subscriber(s), and afterwards, test the modified software to ensure that it is functioning properly. Many iterations of modification and testing may be performed before the finished, customized service is available to the subscriber.

In light of the above, it is clear that previous techniques are problematic for numerous reasons. For example, a service provider is required to maintain or otherwise employ a staff of human software developers for making modifications to supporting software applications. This can be expensive. Furthermore, a substantial amount of time may be required to develop, modify, and test supporting software applications in response to the request of a particular subscriber or group of subscribers. This can lead to subscriber dissatisfaction, and ultimately, defection to another service provider.

SUMMARY OF THE INVENTION

The disadvantages and problems associated with previous techniques for providing technological services have been substantially reduced or eliminated using the present invention.

The present invention provides a network system extensible (e.g., programmable) by "end-users," and a method of operation for the same. In general, an end-user (or simply "user") is any individual, party, or entity which somehow interacts with the network system. A user can thus be an entity known to the network system (i.e., an entity having a log-in ID), such as, for example, a subscriber and or an individual affiliated with the service provider. A user can also be an arbitrary third-party which somehow interacts with the network system.

With the present invention, users may extend or customize the network system according to their own particular needs. To accomplish this, a network system is augmented with an agent system. Capabilities of the network system are programmatically exposed by means of one or more services, service resources, and service wrappers. Each service individually, or the network system as a whole, can be extended by adding agents (created by users). Furthermore, the consumption of computational and service resources are monitored within the network system, thus protecting the subscribers and the service provider from harm or misuse, whether intentional or inadvertent. Accordingly, the network system can admit agents programmed by users.

In addition, the present invention contemplates that a third party may modify existing agents and create new agents in the case where subscribers lack the desire or sophistication to do so themselves. The third party can then make such customized agents commercially available to subscribers.

According to an embodiment of the present invention, a network system includes a service. An agent uses the service on behalf of a principal. An agent server mediates the use of the service by the agent.

According to another embodiment of the present invention, a network system includes a user interface which allows a user to interact with the network system. An agent server is coupled to the user interface. The agent server manages agent use of the network system. Furthermore, the agent server in conjunction with the user interface is operable to create or modify an agent in response to interaction by the user.

According to yet another embodiment of the present invention, a method includes the following: admitting a user to a network system wherein at least one agent is operable to utilize a service to perform a task for the user; and allowing the user to create or modify the agent within the network system.

According to still another embodiment of the present invention, a network system includes an agent server which manages agent use of the network system. An agent is operable to utilize a service within the network system. A service wrapper, associated with the service, cooperates with the agent server to mediate interaction between the service and the agent.

According to yet another embodiment of the present invention, a method includes the following: allowing an agent to utilize a service; and mediating interaction between the service and the agent.

A technical advantage of the present invention includes providing a network system (and a method of operation therefor) which is programmable by users (including subscribers) according to their own particular needs. From the standpoint of subscribers, this facilitates the process of adding or deleting new services or extending existing services and, from the standpoint of a service provider, this is beneficial in that human software developers and testers can be reduced or eliminated altogether with the automated system of the present invention.

Other aspects and advantages of the present invention will become apparent from the following descriptions and accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention and for further features and advantages, reference is now made to the following description taken in conjunction with the accompanying drawings, in which:

Figure 1 illustrates a network system extensible by users, according to an embodiment of the present invention;

Figure 2 illustrates an exemplary computer-based system that can be used to implement the network system shown in Figure 1;

Figure 3 illustrates details for a graphical user interface, according to an embodiment of the present invention;

Figure 4 illustrates details for a voice user interface, according to an embodiment of the present invention;

Figure 5 illustrates details for an agent server, according to an embodiment of the present invention;

Figure 6 illustrates details for a service wrapper, according to an embodiment of the present invention;

Figure 7 illustrates details for specific exemplary service wrappers, services, and service resources;

Figure 8 illustrates details for an exemplary agent object, according to an embodiment of the present invention;

Figure 9 is a flow diagram of an exemplary method for a user session, according to an embodiment of the present invention;

Figure 10 is a flow diagram of an exemplary method for executing a selection command for selecting an agent or an agent template, according to an embodiment of the present invention;

Figure 11 is a flow diagram of an exemplary method for executing an agent template command, according to an embodiment of the present invention;

Figure 12 is a flow diagram of an exemplary method for executing an agent command, according to an embodiment of the present invention;

Figure 13 is a block diagram detailing the controlled consumption of service and computational resources, according to an embodiment of the present invention;

Figure 14 is a flow diagram of an exemplary method for executing time slices for an agent population, according to an embodiment of the present invention;

Figure 15 is a flow diagram of an exemplary method for executing a time slice for an agent, according to an embodiment of the present invention;

Figure 16 is a flow diagram of an exemplary method for executing an event handler, according to an embodiment of the present invention; and

Figure 17 is a flow diagram of an exemplary method for executing a service instruction, according to an embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The preferred embodiments of the present invention and their advantages are best understood by referring to Figures 1 through 17 of the drawings. Like numerals are used for like and corresponding parts of the various drawings.

Turning first to the nomenclature of the specification, the detailed description which follows is represented largely in terms of processes and symbolic representations of operations performed by conventional computer components, such as a central processing unit (CPU) or processor associated with a general purpose computer system, memory storage devices for the processor, and connected pixel-oriented display devices. These operations include the manipulation of data bits by the processor and the maintenance of these bits within data structures resident in one or more of the memory storage devices. Such data structures impose a physical organization upon the collection of data bits stored within computer memory and represent specific electrical or magnetic elements. These symbolic representations are the means used by those skilled in the art of computer programming and computer construction to most effectively convey teachings and discoveries to others skilled in the art.

For purposes of this discussion, a process, method, routine, or sub-routine is generally considered to be a sequence of computer-executed steps leading to a desired result. These steps generally require manipulations of physical quantities. Usually, although not necessarily, these quantities take the form of electrical, magnetic, or optical signals capable of being stored, transferred, combined, compared, or otherwise manipulated. It is conventional for those skilled in the art to refer to these signals as bits, values, elements, symbols, characters, text, terms, numbers, records, files, or the like. It should be kept in mind, however, that these and some other terms should be associated with appropriate physical quantities for computer operations, and that these terms are merely conventional labels applied to physical quantities that exist within and during operation of the computer.

It should also be understood that manipulations within the computer are often referred to in terms such as adding, comparing, moving, or the like, which are often associated with manual operations performed by a human operator. It must be understood that no involvement of the human operator may be necessary, or even desirable, in the present invention. The operations described herein are machine operations performed in conjunction with the human operator or user that interacts with the computer or computers.

In addition, it should be understood that the programs, processes, methods, and the like, described herein are but an exemplary implementation of the present invention and are not related, or limited, to any particular computer, apparatus, or computer language. Rather, various types of general purpose computing machines or devices may be used with programs constructed in accordance with the teachings described herein. Similarly, it may prove advantageous to construct a specialized apparatus to perform the method steps described herein by way of dedicated computer systems with hard-wired logic or programs stored in non-volatile memory, such as read-only memory (ROM).

Network System Overview

Referring now to the drawings, Figure 1 illustrates a network system 2 extensible by users, according to an embodiment of the present invention. To achieve this, network system 2 may incorporate an agent system comprising an agent server and one or more agents, as described below in more detail. An exemplary construction for an agent system is taught by United States Patent No. 5,603,031, issued to the Assignee of the present invention, the text of which is incorporated herein by reference.

It is contemplated that network system 2 may be maintained, managed, and/or operated by any provider of technological services, such as electronic mail (e-mail), voice mail, electronic organizer (for appointments and/or addresses), on-line data retrieval (for, e.g., periodicals and stock quotes), and the like. These services are offered and/or provided to one or more users who may be considered to be "subscribers." Each of the provider and the subscribers can be an individual, a business, a governmental department or agency, an academic institution, an organization, or the like, which provides or receives, respectively, any form of technological service.

The following primarily describes how network system 2 and its associated methods of operation can be used to provide information technology services over a telephony system. Furthermore, a model of a network system providing telephony services is discussed in detail in microfiche Appendix A, in accordance with one embodiment of the present invention. It should be understood, however, that the present invention is not so limited. That is, the teachings of the present invention generally encompass the provision of services by agents to one or more users in an environment which can be extended by the users, for example, by programming additional agents.

Network system 2 includes a programmable functionality component 4 and a hard-wired functionality component 6. In general, programmable functionality component 4 and hard-wired functionality component 6 each functions to provide and/or support the provision of technological services. Hard-wired functionality component 6 is implemented substantially with a number of "hard-wired" elements, and thus, its functionality is modified or changed primarily by connecting or re-connecting the same or additional elements. Programmable functionality component 4 is implemented with an agent system and, as such, its functionality can be programmed (for example, by subscribers or third parties), as described below in more detail. Computer code for an exemplary agent system implementing a programmable functionality component 4 is provided in microfiche Appendix B, in accordance with an embodiment of the present invention.

Graphical User Interface

Network system 2 includes a graphical user interface (GUI) 12. Graphical user interface 12 may be implemented and/or supported by a web browser--i.e., a client application that resides on (or is downloaded to) an electronic user device, such as a desktop computer. Any of a variety of browsers are available, such as NETSCAPE NAVIGATOR, MICROSOFT INTERNET EXPLORER, and others. The web browser can be a forms-capable browser which is able to interpret Hyper Text Markup Language (HTML) code which provides forms including fill-in text boxes, option buttons, drop-down list boxes, radio buttons, and the like.

Graphical user interface 12 allows a user to interact with network system 2 via a communication line, which may be any type of communication link capable of supporting data transfer. For example, the communication line may include any combination of an Integrated Services Digital Network ("ISDN") communication line, a hard-wired line or a telephone line. This enables communication via the interconnection of computers popularly known as the "Internet," using any suitable protocol, such as, Transmission Control Protocol/Internet Protocol

(TCP/IP), Internetwork Packet eXchange/Sequence Packet eXchange (IPX/SPX), or AppleTalk.

Graphical user interface 12--comprising a web browser and a web server--manages the connection between the user device and network system 2, supports the transfer of data therebetween, and interprets and displays the data. For example, graphical user interface 12 enables the downloading of one or more web pages (serving as graphical interfaces into network system 2) to the user device. The user device may include one or more suitable input devices, such as a keyboard, key pad, touch screen, input port, pointing device (e.g. mouse), and/or other device that can accept information, and one or more suitable output devices, such as a computer display, output port, speaker, or other device for conveying information including digital data, visual information, or audio information.

Graphical user interface 12 may comprise an agent area 14 which is dedicated to the activities of creating new agents and manipulating existing agents, as described herein. For example, in one embodiment, an "agent" icon is added to a screen menu; the interface screen which is accessed by "clicking" on the agent icon constitutes the agent area.

Voice User Interface

A voice user interface (VUI) 16 allows a user to interact with network system 2 via a telephone line, which can be an analog telephone line, a digital T1 line, a digital T3 line, or an OC3 telephony feed. In contrast to graphical user interface 12, voice user interface 16 does not require that a user have access to an electronic interface, such as a computer. Rather, voice user interface 16 interprets the vocalized expressions of a user so that the user may issue commands and other input into network system 2. Voice user interface 16 may also issue audible output in the form of speech that is understandable by a user. Such speech can be synthesized or previously recorded, as described below in more detail.

Voice user interface 16 may comprise speech recognition and speech synthesis software and/or hardware stored in or implemented as a suitable memory device and run on a suitable processor. Such speech recognition software allows network system 2 to recognize vocalized speech and may include grammar software that creates or selects a speech recognition grammar for determining which speech should be recognized. Commercially available speech recognition systems with recognition grammars are provided by ASR (Automatic Speech Recognition) technology vendors such as the following: Nuance Corporation of Menlo Park, CA; Dragon Systems of Newton, MA; IBM of Austin, TX; Kurzweil Applied Intelligence of Waltham, MA; Lernout Hauspie Speech Products of Burlington, MA; and PureSpeech, Inc. of Cambridge, MA. Recognition grammars are written specifying what sentences and phrases are to be recognized by the voice user interface 16. For example, a recognition grammar can be generated by a computer scientist or a computational linguist or à linguist. The speech synthesis software synthesizes human speech and may include speech markup software for determining the speech to be synthesized.

In addition to speech synthesis software and/or hardware, voice user interface 16 may include speech play-back capabilities for playing back previously recorded human speech. Exemplary play-back devices include a tape player, a laser disc player, etc. Here, an actual person (preferably an actor) recites various statements which may desirably be issued during an interactive session with a user of network system 2. The person's voice is recorded as the recitations are made. The recordings are separated into discrete messages, each message comprising one or more statements that would desirably be issued in a particular context (e.g., greeting, farewell, requesting instructions, receiving instructions, etc.). Afterwards, when a user interacts with network system 2, the recorded messages are played back to the user when the proper context arises. In one embodiment, such speech play-back capabilities can be used to implement a voice user interface with personality, as taught by United States Patent Application Serial No. 09/071,717, entitled "Voice User Interface With Personality," the text of which is incorporated herein by reference.

Voice user interface 16 may also comprise hardware and/or software supporting the interpretation and issuance of dual tone multiple frequency (DTMF) commands so that a user may alternatively interact with network system 2 using a telephone key pad.

Voice user interface 16 may comprise an agent area 18 which, like agent area 14 of graphical user interface 12, is dedicated to the activities of creating new agents and manipulating existing agents. In one embodiment, agent area 18 of voice user interface 16 can be implemented by translating the agent area 14 of graphical user interface 12 into voice.

Agent Server

Programmable functionality component 4 includes an agent server 20. Agent server 20 is in communication with graphical user interface 12 and voice user interface 16, and accordingly, may exchange (receive and transmit) information therewith. In general, agent server 20 controls, coordinates, and otherwise manages the overall operation of programmable functionality component 4. Among other things, agent server 20 may invoke, initiate, or execute various routines, processes, objects, and the like. For example, when a user wishes to interact with network system 2 via graphical user interface 12, agent server 20 may cause web pages to be downloaded to an electronic user device. As another example, agent server 20 may prompt voice user interface 16 to issue various statements at appropriate moments during an interactive session with a user via telephone. Additional functionality of agent server 20 includes, but is not limited to, executing agent objects, identifying computational and service permissions, and controlling the consumption of computational and service resources, as described below in more detail.

Agent server 20 is responsive to various commands which it may receive from a user, for example, via graphical user interface 12 or voice user interface 16. These commands can be of three types: agent commands, agent template commands, and selection commands for selecting agents and agent templates. Each of these types of commands is explained below in more detail. Agent server 20 executes these commands during its operation.

The functionality of agent server 20 can be performed by any suitable processor such as a main-frame, file server, workstation, or other suitable data processing facility running appropriate software. Agent server 20 may operate under the control of any suitable operating system such as MS-DOS, MacINTOSH OS, WINDOWS NT, WINDOWS 95, OS/2, UNIX, XENIX, GEOS, MAGIC CAP, and the like.

<u>Computational Resources</u>

A number of computational resources 21 are available to agent server 20. In general, computational resources 21 are resources provided or supported by a computer-based system (Figure 2) having one or more processors, data-storage devices, interfaces, suitable connections, etc. Computational resources 21 include processing time, memory storage space, and the like. As described herein, computational resources 21 may be "consumed" or "used up" during the operation of network system 2.

Agents

A number of agents 22 are in communication with agent server 20. Each agent 22 is associated with a particular user (e.g., a subscriber or an individual affiliated with the service provider), which may be deemed to be the "principal" for the respective agent. Generally speaking, agents 22 can be considered to be personal software assistants with authority delegated by the respective principals. That is, each agent 22 may be implemented as a software application, program, or process which autonomously, and possibly continuously, runs on behalf of its principal. As such, an agent 22 may be viewed by its principal as an electronic extension thereof. A particular principal may employ a plurality of agents 22, each of which serves only that principal.

The software applications for implementing agents 22 may each comprise a text file or document in, for example, a format prescribed by eXtensible Markup Language (XML). XML is a subset of Standard Generalized Markup Language (SGML) and, like SGML, is a meta-language--i.e., a language for specifying markup languages. One such markup language is Agent Definition Format (ADF) developed by General Magic, Inc. Various specifications for ADF are discussed in detail in microfiche Appendix C, in accordance with an embodiment of the present invention. A markup language such as ADF uses tags to provide programming language constructs to text. These tags, which may comprise instructions enclosed in angled brackets, are inserted before and after the text affected. Agent server 20 can interpret the tags and text of an ADF document to cause a respective agent 22 to act. Exemplary code for a number of agents 22 is provided in microfiche Appendix D, in accordance with an embodiment of the present invention.

Agents 22 are task-based. That is, each agent 22 is responsible for performing a particular task or set of tasks on behalf of the respective principal. These tasks may include, for example, answering telephone calls, taking voice mail messages, placing telephone calls, notifying the user of recently received messages (voice mail and/or e-mail), delivering messages, setting up meetings/appointments, gathering information, negotiating deals, transacting electronic commerce, etc.

Agents 22 can be "standard" or "customized." A standard agent is one which may be written and/or set-up by the service provider. In general, standard agents perform tasks that many users (e.g., subscribers) each would desirably have performed on his or her behalf. Such tasks may include organizing meetings and delivering messages. A separate copy of a standard agent may be engaged or selected by each user who wishes to have the respective tasks performed. Thus, many standard agents, each performing the same sort of tasks but for different users, may exist in network system 2.

In contrast, a customized agent is one which is written and/or set up by a particular subscriber or group of subscribers to perform certain tasks which are unique to that subscriber or group. For example, a particular subscriber or group may work in the real estate industry and thus desire to have certain tasks related to real estate transactions performed for him/her or them. In this case, such subscriber or group of subscribers may customize or create one or more agents that address the specialized needs of the subscriber or group. Furthermore, a third party may customize or create agents for subscribers or groups which are unable, unwilling, or lack the sophistication to do so for themselves. Such agents customized by a third party can be made commercially available to subscribers and other users. Accordingly, network system 2 is extensible in the sense that subscribers and/or third parties may program customized agents 22 according to particular needs.

The customization of agents 22 can be accomplished using an electronic user device (e.g., desktop computer) communicating with network system 2 via graphical user interface 12. In one embodiment, the service provider may maintain a website which users can access for customizing agents 22.

While performing the respective tasks, agents 22 may use or consume various computational resources 21 (e.g., memory storage space, processing time, and the like). Furthermore, agents 22 may also use or consume various service resources (described below in more detail) during the performance of their respective tasks. The consumption of computational and service resources by various agents 22 can be monitored, and the respective principals charged for the same.

In one embodiment, each agent 22 is given permission to consume up to a pre-authorized amount of each computational resource and each service resource that the agent may use when performing its respective task(s). For each computational resource, the relevant permission constitutes a "computational permission." For each service resource, the relevant permission constitutes a "service permission." With respect to a particular agent 22, the computational and service permissions ensure that other agents 22 (acting on behalf of the same or other principals/users) have adequate computational and service resources, even in the case of a maliciously or incorrectly programmed customized agent.

Alternatively, a computational or a service permission may be associated with a particular principal and specifies a predetermined amount of a respective resource which is allowed to be consumed on behalf of that principal. That is, multiple agents 22 having the same principal may each use the same resource. By authorizing a predetermined amount of resource for the principal, the associated agents 22 are given a "pool" of that resource from which to draw.

In one embodiment, each agent 22 may direct its own movement (i.e., transportation) through a computer-based system (Figure 2) by executing an instruction which identifies a destination computer within the computer-based system and directs that the particular agent 22 be transported there. While executing within the destination computer, the agent may have access to information which is not available elsewhere in the computer-based system. The particular agent can access and use that information to determine another destination computer to which the agent should travel.

Agents 22 are created from agent templates. An agent template can be a "blueprint" for one or more agents 22. In object-oriented terminology, each agent 22 is an object and each template is an agent class from which agents 22 can be created by instantiating the class/template. In one embodiment, an agent template is essentially an agent object in its initial state. After the agent has been created, it may be executed. During execution, the agent object begins to receive events, handle these events, and so change its state.

In accordance with an embodiment of the present invention, network system 2 allows users (e.g., subscribers) to create, copy, modify, edit, or delete agents 22 and the associated templates as desired, thereby affording extensibility.

Services

A number of services 24 may each comprise one or more software applications providing various capabilities that are available to a principal. Each service 24 may be utilized by one or more agents 22 in order to perform their respective tasks. For example, one service 24 may support call processing, including a voice mail system. With this service 24, an agent 22 may place an outgoing telephone call for its principal, answer an incoming telephone call for the principal, and record a voice message from a caller. Another exemplary service 24 may support an electronic mail (e-mail) system. With an e-mail service, an agent 22 may collect, forward, and store e-mail messages addressed to the respective principal, generate e-mail messages for the principal, and notify the principal when new e-mail messages have been received. Yet another exemplary service 24 may support an electronic appointment book. With this service, an agent 22 can plan a schedule, check conflicts therewith, and organize various meetings, interviews, etc. for its respective principal. Still another exemplary service 24 can support an electronic address book which maintains information about one or more contacts with whom a principal may interact. The information maintained for each of these contacts may include the contact's name, title, employer, business address, home address, e-mail address, work phone number, home phone number, and the like. Still yet another service 24 can support on-line data retrieval for various information. With this service, an agent 22 can retrieve electronic copies of periodicals (e.g., newspapers, magazines, etc.) and the latest quotes for stocks, bonds, mutual funds, etc. In order to use a particular service 24, an agent 22 must first be authorized to do so. This authority may be given by the service provider and/or the respective principal.

In one embodiment, a plurality of services 24 may be used by a particular agent 22 which acts as a "virtual assistant" for its respective principal. This virtual assistant may, for example, answer an incoming telephone call, identify the caller, schedule a meeting between the caller and the principal, and generate an e-mail message notifying the principal of such meeting. Additional services 24 can be used (by this agent 22) by extending or customizing the agent, as described herein.

<u>Service Resources</u>

One or more service resources 25 can support each service 24. In one embodiment, at least a portion of service resources 25 can be integral to the respective services 24. In general, a service resource 25 is a resource which enables a service to be performed. For example, for a call processing service, service resources 25 may include a telephone, an answering machine, a telephone line, a local telephone provider service, a long-distance telephone provider service, etc. As another example, for an on-line data retrieval service, service resources 25 may include a telephony connection, a modem for on-line communication, an on-line database provider service, etc. As yet another example, for an e-mail service, service resources 25 may include an Internet connection and disk space for storing e-mail messages.

At least some of service resources 25 may comprise discrete units which are "consumed" during utilization of the respective resource by an agent 22. For example, a service resource 25 related to telephony services (e.g., voice mail and call placement) may comprise units of long-distance calling time which are consumed as an agent 22 places one or more calls utilizing such services 24. Likewise, a service resource 25 related to on-line data retrieval may comprise units of data-access time or inquiry which are consumed as an agent retrieves data (e.g., stock quotes, newspaper articles, etc.) utilizing the respective service 24.

Service Wrappers

A number of service wrappers 26 link services 24 and respective service resources 25 to agent server 20. In the depicted embodiment, a separate service wrapper 26 is provided for each service 24. Each service wrapper 26 can mediate the interaction between a service 24 (and its respective resources 25) and the remainder of programmable functionality component 4. In one embodiment, at least a portion of service wrappers 26 can be integral to the respective services 24 and/or service resources 25. Service wrappers 26, in conjunction with agent server 20, may act as "gatekeepers" to the respective services 24. That is, a service wrapper 26 grants access, to a respective service 24, only to agents 22 which have been authorized to utilize such service. For example, a voice message held by a service 24 supporting voice mail is accessible only by an agent 22 which has authority from the user for whom the message was left.

Similarly, for a service 24 supporting call answering, the respective service wrapper 26 only allows agents 22 having authority from a particular user to answer telephone calls placed to a number associated with such a user.

Service wrappers 26 enable communication between services 24 and agent server 20, for example, by converting between a computer language (or instruction set) used within agent server 20 and the computer language (or instruction set) of the respective service 24. For this purpose, in one embodiment, each service wrapper 26 may be implemented, at least in part, with an application programming interface (API). An API allows access to the respective service 24, thereby pragmatically exposing the capabilities of the service 24 to one or more agents 22. That is, each service wrapper 26 lets any agent (having the proper authorization) use the respective service 24. A service wrapper 26 may also allow agent server 20 to replace one service 24 with another service 24 of the same type, but located on a different computer, without any interruption to principals. This operation of a service wrapper 26 is hidden or invisible to users, and thus, service wrappers 26 provide a point of abstraction between the respective service 24 and agent server 20.

Furthermore, at least some of service wrappers 26 may monitor the amount of service resources 25 expended or otherwise consumed by one or more authorized agents 22 when utilizing the respective service 24. For example, a service wrapper 26 may identify the respective service resource 25 to agent server 20, allow agent server 20 to impose a predetermined limit on each authorized agent's use of such service resource 25 (i.e., allocate a predetermined amount of service resource 25 for each agent), and debit/credit the amount consumed from the predetermined amount previously allocated for the agent 22. Thus, each service wrapper 26, in conjunction with agent server 20, controls an agent's consumption of service resources 25, thereby ensuring that an agent 22 does not use more than a predetermined amount of resource when performing its respective task or tasks. In this way, service wrappers 26 protect against the over-consumption, whether intentional or inadvertent, of the respective service resources 25. A service wrapper 26 thus serves to ensure that sufficient units of a service resource 25 are available for each agent 22 authorized to utilize the respective service 24. In addition, a service wrapper 26 may maintain a record of the amount of service resource 25 consumed by various agents 22 so that the appropriate users/principals can be billed accordingly.

<u>Operational Overview</u>

In operation, one or more agents 22 may be set up for each user who is a subscriber to the services offered by the operator/provider of network system 2. These agents 22 can be standard (i.e., set up by the service provider) or customized (i.e., set up by the respective user). Each agent for a particular user performs one or more tasks on behalf of that user. These tasks may include answering calls, taking voice mail messages, placing calls, notifying the user of recently received e-mail, setting up appointments, negotiating deals, transacting electronic commerce, etc. To perform these tasks, each agent 22 utilizes one or more services 24, during which it may consume various respective service resources 25. Access to each service 24 is granted by the respective service wrapper 26, which may also monitor the amount of each respective service resource 25 consumed to ensure that no particular agent 22 uses more than an amount authorized for that agent when performing its specific task(s). This protects both the authorizing user/subscriber and the service provider against undue consumption of service resources 25. Via a suitable interface (e.g., graphical user interface 12 or voice user interface 16), a user may direct, command, instruct, or otherwise communicate with each of his or her respective agents 22. At any time, a user can extend the services provided thereto by customizing existing agents 22, or alternatively, employing (and possibly customizing additional) agents 22.

In this manner, the present invention enables subscribers of a network service provided by network system 2 to create and employ personal agents 22 utilizing services 24 in a manner that is safe and secure for the subscribers and the service provider.

Computer-Based System Implementation

Figure 2 is a simplified diagram of an exemplary computer-based system 30 that can be used to implement network system 2 shown in Figure 1. Referring to the embodiment shown in Figure 2, computer-based system 30 can include a communications hub 1000 and a firewall 1002 which support an interface between computer-based system 30 and the Internet. Collectively, communications hub 1000 and firewall 1002 provide communication, protection, and security between the remainder of computer-based system 30 and the Internet. Hub 1000 and firewall 1002 allow users to interact with computer-based system 30 using an electronic user device, which may support a graphical user interface 12 (Figure 1).

A switch 1004 enables communication with computer-based system 30 via a telephone instrument. Switch 1004 allows users to interact with computer-based system 30, for example, via a voice user interface 16

(Figure 1). A call detail records data base 1005 is coupled to switch 1004. This data base 1005 stores information relating to calls received by or initiated out of computer-based system 30. For each call, such call-related information may specify calling party, called party, telephone number of outside party, date of call, time of call, duration of call, cost of call, and the like.

A plurality of fast Ethernet hubs 1006 are coupled to firewall 1002 and switch 1004. Fast Ethernet hubs 1006 support a local area network (LAN) for computer-based system 30 and enable the routing of information signals therein. These Ethernet hubs 1006 may implement the Fast Ethernet technique in which information is transferred at a rate of 100 Mbps.

One or more application server clusters 1008 are coupled to fast Ethernet hubs 1006. Each application server cluster 1008 may comprise a plurality of servers which provide processing capability to support various functions, such as, for example, speech recognition, speech synthesis, pronunciation generation, speech playback, etc. As such, application server clusters 1008 may support a voice user interface 16 (Figure 1).

A number of sub-systems 1010, 1012, 1014, 1016, 1018, 1020, 1022 are also coupled to fast Ethernet hubs 1006. Each of these sub-systems may comprise one or more servers, data storage devices, and other hardware components. The servers provide processing capability, and the data storage devices provide data storage capability.

At least a portion of the sub-systems in computer-based system 30 may support one or more services 24, and the respective service wrappers 26 and service resources 25 (Figure 1). For example, as shown in Figure 2, sub-systems 1010, 1012, 1014, 1016, and 1018 support an e-mail service, a voice mail service, a paging/facsimile service, an address book and calendar service, and a business news and stocks information service, respectively.

The remaining sub-systems support other operational aspects of computer-based system 30. In particular, as shown, sub-system 1020 maintains profiles for one or more subscribers to a network system 2. For each subscriber, a profile may include the name of the subscriber, the home address of the subscriber, the billing address of the subscriber, an e-mail address, a telephone number, a listing of all agents operating for the subscriber, the computational and service permissions granted to the subscriber's agents, and the like. Sub-system 1022 maintains various rules for the operation of network system 2. For example, such rules may govern how network system 2 processes incoming e-mail messages for particular subscribers (e.g., a subscriber may be paged upon the delivery of messages satisfying certain criteria).

Each of the servers within application server clusters 1008 and sub-systems 1010-1022 can be implemented using any of a number of server hardware configurations running a suitable server operating system, such as SUN SOLARIS, WINDOWS NT, OS/2 WARP, and NETWARE. Each data storage device within application server clusters 1008 and sub-systems 1010-1022 can be a mass storage subsystem of tapes or disk drives, which is electronically coupled to the respective servers. Information or data supporting each of graphical user interface 12, voice user interface 16, agent server 20, agents 22, services 24, service resources 25, and service wrappers 26 (shown in Figure 1) can be contained in the data storage devices. The servers may retrieve, process, and store this information into the data storage devices.

In one embodiment, as shown, application server clusters 1008 and sub-systems 1010-1022 are linked by the same LAN. In another embodiment, at least a portion of application server clusters 1008 and/or sub-systems 1010-1022 can be remote from the remainder and linked as a wide area network (WAN); consequently, computer-based system 30 may provide a distributed network.

Any one or a combination of interested parties (e.g., subscribers, third parties, or service providers) can use computer-based system 30 to collect, maintain, generate, or process the information supporting graphical user interface 12, voice user interface 16, agent server 20, computational resources 21, agents 22, services 24, service resources 25, and service wrappers 26. Any of the servers or other computers in application server clusters 1008 and sub-systems 1010-1022, either individually or in combination, can perform the functionality of agent server 20 shown in Figure 1. Furthermore, because these servers and other computers are linked together, each can directly access (e.g., store and retrieve) any of the information described above, if necessary.

In one embodiment, agents 22 may travel throughout the environment of computer-based system 30. That is, each agent 22 may move to the servers and other computers in application server clusters 1008 and sub-systems 1010-1022, and afterward, execute thereon. As an agent 22 moves and executes, it may perform its respective tasks on behalf of its principal.

The elements of computer-based system 30, and the functions provided thereby, constitute computational resources 21 which may be expended, consumed, or used during the operation of network system 2 (Figure 1). In one embodiment, the consumption of these computational resources 21 by different agents 22 is monitored so that no particular agent 22 utilizes more than a proportionate, predetermined, or allotted share thereof, as such agent executes.

Graphical User Interface (Details)

Figure 3 illustrates details for a graphical user interface 12, according to an embodiment of the present invention. As depicted, graphical user interface 12 comprises a web browser 64 and a web server 66 which are connected via a line 68 supporting Internet communication.

Web browser 64 is a client application that may reside on (or is downloaded to) a client device such as a desktop computer. Such desktop computer preferably has at least a "486" processor or an operational equivalent and runs a suitable desktop operating system, such as, for example, MS-DOS, MacINTOSH OS, WINDOWS NT, WINDOWS 95, OS/2, UNIX, XENIX, GEOS, or MAGIC CAP.

Web server 66 is a server application that resides on a service provider device, such as a server. Such server can be any of the servers in application server clusters 1008 or sub-systems 1010-1022 of computer-based system 30 shown in Figure 2. Web server 66 may support a number of web pages 70 which can be downloaded from web server 66 to web browser 64 as appropriate during operation. At least one of these web pages 70 may constitute an agent area 14 which is dedicated to the activities of creating new agents and manipulating existing agents.

Communication line 68 can be any link capable of supporting data transfer between a client device and a service provider device. For example, communication line 68 may include any combination of an Integrated Services Digital Network ("ISDN") communication line, a hard-wired line, or a telephone line. This enables communication via the Internet, using any suitable protocol, such as, Transmission Control Protocol/Internet Protocol (TCP/IP), Internetwork Packet eXchange/Sequence Packet eXchange (IPX/SPX), or AppleTalk.

<u>Voice User Interface (Details)</u>

Figure 4 illustrates details for a voice user interface 16, according to an embodiment of the present invention. As shown, voice user interface 16 comprises a telephone instrument 72 coupled to a telephone server 74 via a telephone line 76.

Telephone instrument 72 is a user (e.g., subscriber) device which may comprise a conventional telephone. Telephone instrument 72 may include a key pad for entering dual tone multiple frequency (DTMF) commands.

Telephone server 74 is a service provider device. Such device may comprise any of the servers in application server clusters 1008 or sub-systems 1010-1022 of computer-based system 30 shown in Figure 2. Telephone server 74 may support a number of grammars, prompts, and other speech functionalities 78 for enabling communication with a user. At least one of these speech functionalities 78 may constitute an agent area 18 which is dedicated to the activities of creating new agents and manipulating existing agents 22.

Telephone line 76 can be an analog telephone line, a digital T1 line, a digital T3 line, or an OC3 telephony feed.

Agent Server (Details)

Figure 5 illustrates details for agent server 20, in accordance with a preferred embodiment. Agent server 20 generally functions to control, coordinate, and otherwise manage the overall operation of programmable functionality component 4. This includes the use of network system 2 by agents 22. As depicted, agent server 20 includes an engine 42, a scheduler 44, and one or more agent objects 46.

Scheduler 44 maintains an internal clock, which can keep track of real time or elapsed time. In one embodiment, scheduler 44 may be supported with a piezo-electric crystal or oscillating circuit implemented with transistors. Scheduler 44 generally functions to trigger the further execution of particular agents 22 by engine 42 upon the occurrence of certain events. Such events may include the lapse of a predetermined amount of time (e.g., 24 hours) or the occurrence of a specified time (e.g., 6:00 a.m.). Scheduler 44 may maintain a record or schedule with a separate entry for each triggering event. Scheduler 44 may also receive information from engine 42. This received information may be used to create new entries within scheduler 44.

Agent objects 46 each correspond to a particular agent 22 of network system 2 (Figure 1). Each agent object 46 can be an internal representation within agent server 20 for the corresponding agent 22. Agent objects 46 comprise software objects, each of which, in general, has (i) an internal state defined by a number of properties, (ii) an internal behavior defined by a number of methods, and (iii) an external behavior defined by a number of features.

More simply, each agent object 46 is an organization of data and instructions which are executable within agent server 20. In one embodiment, the data represents a "state" of the agent object and the instructions are grouped into tasks that are to be performed. The data may specify the corresponding agent 22, the services 24 which may be utilized by the agent, and the computational and service permissions which have been granted to the agent 22. The instructions may comprise one or more event handlers which direct the corresponding agent 22 upon the occurrence of predefined events. Each agent object 46 may also comprise a pending event queue which queues events to which the agent 22 should, but has not yet, responded.

Engine 42 is in communication with scheduler 44 and agent objects 46. In addition, engine 42 is in bi-directional communication with graphical user interface 12, voice user interface 16, and service wrappers 26. Engine 42 generally controls and/or manages the operation of agent server 20. Engine 42 may be implemented as a computer process, executing within a computer system, which invokes or manages other processes or routines, and executes instructions. For example, engine 42 can execute each agent object 46 which, in turn, may provide instructions to engine 42. Engine 42 may function to identify each of the computational permissions and service permissions specified within an agent object 46 and to control the consumption, by the corresponding agent 22, of the relevant computational resources 21 and service resources 25. Also, engine 42 may invoke routines in each of interfaces 12 and 16 and service wrappers 26.

Furthermore, engine 42 may receive and be responsive to information from the same.

In one embodiment, scheduler 44 may be considered to "create" pending events to which engine 42 responds by executing an agent's handler for such event. This is described below in more detail.

In an exemplary operation for agent server 20, one agent object 46 may correspond to an agent 22 which is responsible for waking up its principal at a certain time each weekday morning. This time constitutes an event for which scheduler 44 may contain an entry. Each weekday, at the appointed time or shortly before, scheduler 44 invokes engine 42. In response engine 42 executes the particular agent object 46. When this agent object 46 is executed, the corresponding agent 22 (using the appropriate services 24) performs the task of waking up the principal, for example, by calling the principal over a telephone or, alternatively, generating an audible alarm on an electronic user device (e.g., a desktop computer).

<u>Service Wrapper (Details)</u>

Figure 6 illustrates details for a service wrapper 26, in accordance with a preferred embodiment. A service wrapper 26 generally functions to mediate the interaction between a respective service 24 and the remainder of programmable functionality component 4. In one embodiment, service wrapper 26 is associated with, and supports, only a single service 24. As shown, service wrapper 26 comprises a converter 48 and a monitor 50.

Converter 48 generally functions to convert between a computer language (or instruction set) used within agent server 20 and a computer language (or instruction set) used within the respective service 24. In one embodiment, the computer language of agent server 20 can be a high-level language, whereas the computer language of service 24 may be a low-level language. An agent 22 may comprise instructions of the set used in agent server 20; a service 24 may comprise or issue instructions of the set used therein.

As depicted, converter 48 comprises an agent-server-to-service converter 52 and a service-to-agent-server converter 54. Agent-server-to-service converter 52 operates unidirectionally to convert from the language (e.g., a high-level language) used by agent server 20 to the language (e.g., a low-level language) used by the corresponding service 24. Service-to-agent-server converter 54 also operates unidirectionally, but in contrast to converter 52, converts from the language (e.g., low-level language) used by service 24 into the language (e.g., high-level language) used by agent server 20.

Monitor 50 is coupled by bi-directional lines to each of agent server 20 and the respective service 24. Monitor 50 generally functions to monitor the amount of respective service resources 25 expended, used, or otherwise consumed by one or more agents 22 which have been authorized to access the service 24. Monitor 50 identifies respective service resources 25 to agent server 20, allows agent server 20 to impose a predetermined limit on each authorized agent's use of the respective service resources 25, and informs agent server 20 of the amount consumed against the predetermined limit. The predetermined limit may be derived from a service permission for the respective agent 22.

Exemplary Service Wrappers, Services, and Service Resources (Details)

Figure 7 illustrates details for specific exemplary services, and their respective service wrappers and service resources. In particular, these services include a web server service 81, a web browser service 86, and a call processing service 91. Each of exemplary services 81, 86, and 91 constitutes a user interface service; that is, these services 81, 86, and 91 each provides a user interface by which an agent 22 can interact with a subscriber or third party. In Figure 7, the services and respective service resources have been combined because, with these examples, the services and resources are not logically distinct.

Web server service 81 provides a web server from network system 2 (Figure 1) and can be used, for example, to notify a user about recently received e-mail messages. Web server service 81 (and its respective service resources) resides in network system

2 and includes one or more web pages 84. A web server service wrapper 80 controls access from the remainder of programmable functionality component 4 into web server service 81 and its associated service resources. A web browser 82 is connected to web server service 81 via an Internet line 83. Web browser 82 may reside in a client device. Web pages 84 can be downloaded from web server service 81 to web browser 82. Collectively, web server service 81, web pages 84, web browser 82, and Internet line 83 can implement a graphical user interface with which an agent 22 can interact with its principal or another party.

Web browser service 86 provides a web browser from network system 2 and can be used, for example, for on-line data access to stock quotes, news, etc. Web browser service 86 and its respective service resources are resident on network system 2. A web browser service wrapper 85 provides access from the remainder of programmable functionality component 4 into web browser service 86 and its service resources. A web server 87 is connected to web browser service 86 via an Internet line 88. Web server 87 resides in a website server device and may comprise one or more web pages 89 which can be downloaded to web browser service 86. Collectively, web browser service 86, web server 87, web pages 89, and Internet line 88 can implement a graphical user interface with which an agent 22 can interact with an arbitrary website on behalf of its principal.

Call processing service 91 provides call processing from network system 2 and can be used, for example, to take voice mail messages for one or more subscribers. Call processing service 91 and its respective service resources reside in network system 2. Call processing service 91 may include grammars, prompts, and other speech functionality 94. A call processing service wrapper 90 controls access from the remainder of programmable functionality component 4 to call processing service 91 and its service resources. A telephone instrument 92, which resides outside of network system 2, is connected to call processing service 91 via a telephone line 93. Collectively, call processing service 91, speech functionality 94, telephone instrument 92, and telephone line 93 can implement a voice user interface with which an agent 22 can interact with its principal or another party, for example, by placing a call to, or answering a call from, the principal or other party.

Agent Object (Details)

Figure 8 illustrates details for an agent object 46, in accordance with an exemplary embodiment. An agent object 46 generally represents (within agent server 20) a corresponding agent 22, which can be considered to be a personal software assistant to a particular principal. Agent object 46 can be a software object and may reside within agent server 20. As depicted, agent object 46 comprises a permissions component 56, an event handlers component 58, a datastore component 60 and a pending event queue component 62. In other embodiments, the event handlers component and the pending event queue component may be external to an agent object 46.

Permissions component 56 generally comprises data and instructions related to permissions that have been granted to the agent 22 represented by agent object 46.

These permissions include computational permissions and service permissions. A computational permission can specify that a respective agent 22 is authorized to consume a particular computational resource 21 (e.g., memory storage space, processing time, elapsed time, and the like which may be provided by the elements and functions of computer-based system 30) when performing its particular task(s). Furthermore, in some instances, a computational permission can specify a pre-authorized amount of computational resource 21 which may be allowably consumed by the respective agent 22. A service permission can specify that the respective agent is authorized to consume a particular service resource 25 (e.g., long-distance time, on-line data access time) when the agent utilizes a respective service 24 (e.g., e-mail, phone mail, electronic appointment book, electronic contact book, etc.) in performing its task(s). In some instances, a service permission can specify a pre-authorized amount of service resource 25 which is allowably consumed by the respective agent 22.

Event handlers component 58 includes data and instructions for directing agent server 20 and/or engine 42 upon the occurrence of various events which may arise during the operation of network system 2. In particular, an event handler comprises a routine for handling an event of a specified type. For example, these events can be the lapse of a previously specified amount of time or the delivery of an e-mail message.

In one embodiment, an event is identified by a uniform resource locator (URL) which expresses or provides an address for a web page. The URL specifies both the event's type and the agent 22 which is event's intended recipient. The URL may be chosen by agent server 20 and the particular agent 22 in combination. This allows a web server providing a graphical user interface to network system 10 to receive HyperText Transfer Protocol (HTTP) requests for the web page at the URL so that agent server 20 can relay the event to the particular agent 22. A standard web browser can send an event to an agent 22 by fetching the web page identified by the URL for the event. The web page returned to the browser is supplied by the agent's event handler and relayed by agent server 20. One of the instructions in an agent instruction set allows an agent 22 to send an event (for example, to another agent 22 as a means of inter-agent communication).

Datastore component 60 may contain various information related to agent object 46. This may include information specifying the corresponding agent 22, the principal or user for whom that agent 22 performs tasks, the tasks which the agent 22 may perform, etc. Datastore component 60 may also include information used by agent object 46 for guiding its interaction with various service wrappers 26. For example, datastore component 60 may contain the parameters of a request the agent intends to make of a particular service wrapper 26 or the wrapper's response to such a request.

Pending event queue component 62 comprises information for events to which the corresponding agent 22 should, but has not yet, responded. Pending event queue component 62 queues these events so that the agent 22 may respond, for example, using the event handlers specified in event handlers component 58.

User Session

Figure 9 is a flow diagram of an exemplary method 100 for a user session, according to an embodiment of the present invention. During method 100, a user is interacting with network system 2.

Method 100 begins at step 102 where network system 2 admits a user at a user interface (UI), which can be either graphical user interface 12 or voice user interface 16. Essentially, at this step, a user logs on to network system 2.

At step 104, network system 2 admits the user to the agent area of the relevant user interface. For voice user interface 16, this is agent area 18. For graphical user interface 12, this is agent area 14. With graphical user interface 12, a user is admitted to the agent area when the user "clicks" on an agent icon which is displayed on a menu screen. In the agent area, the user can create new agents and manipulate existing agents. Specifically, the user can enter various commands, depending on his or her intentions.

Each command can be one of three types: an agent command, a template command, or a selection command. Agent commands are directed to the running of agents. These agents include ones which are currently executing and which the user now would like to manipulate. Template commands are directed to the manipulation of agent templates. An agent template can be a "blueprint" for agents. In object-oriented terminology, each agent is an object and each template is an agent class from which agents can be created by instantiating the class/template. Selection commands select agents or templates. A selection command allows network system 2 to focus on a particular template or a particular agent in order to provide a context for any subsequent template command or agent command.

At step 106, network system 2 accepts a command which has been entered by the user via the user interface. The command is forwarded from the user interface to agent server 20.

If the command is a selection command, agent server 20 executes such command at step 108. If the command is a template command, agent server 20 executes the command at step 110. Otherwise, if the command is an agent command, agent server 20 executes the command at step 112. The executions of a selection command, a template command, and an agent command are described below in more detail.

At step 114, network system 2 prompts a user for more commands. If the user enters additional commands, method 100 returns to step 106 where the next command is accepted. Method 100 repeats steps 106-114 until no additional commands are entered.

At step 116, network system 2 excuses the user from the agent area of the user interface. At step 118, the user is excused from the user interface itself. That is, the user logs off network system 2. Afterwards, method 100 ends.

<u>Selection Command Execution</u>

Figure 10 is a flow diagram of an exemplary method 200 for executing a selection command, according to an embodiment of the present invention. Method 200 is initiated within and performed by network system 2 when a user (e.g., subscriber) has entered a selection command. A selection command can be one of two types: a select agent command or a select template command.

A select agent command is one which selects a particular agent. Specifically, when a user enters the agent area, a list of executing agents may be visible. The user enters a select agent command to select a particular agent from the list in order to manipulate the same; afterwards, the user can enter an agent command to perform the manipulation.

Similarly, a select template command is one which selects a particular template. In particular, a list of agent templates may be provided from which a user may select. The user enters a select template command to select a particular template from the list; afterwards, the user can enter a template command to manipulate the template.

Method 200 begins at step 202 where agent server 20 determines whether the selection command is a select template command. If the selection command is a select template command, method 200 moves to step 204 where agent server 20 notes the particular template which has been selected. Afterwards, method 200 ends.

Otherwise, if at step 202 it is determined that the selection command is not a select template command, then at step 206 agent server 20 determines whether the selection command is a select agent command. If the command is a select agent command, method 200 moves to step 208 where agent server 20 notes the agent which has been selected, after which method 200 ends.

Otherwise, if the selection command is not a select agent command, method 200 ends.

Template Command Execution

Figure 11 is a flow diagram of an exemplary method 300 for executing a template command, according to an embodiment of the present invention. Method 300 is initiated within and performed by network system 2 when a user has entered a template command. In general, a template command is one which affects a particular template. A template command can be one of four types: a create template command, a copy template command, an edit template command, and a delete template command.

Method 300 begins at step 302 where agent server 20 determines whether the template command which has been entered is a create template command. A create template command is one which creates a new template. Each new template can be based on a generic blank template which can be modified as desired by a user.

Accordingly, if the template command is a create template command, then at step 304 agent server 20 stores a blank template into memory (a computational resource 21) where the template can be modified. In the context of a graphical user interface, an entry for the new template is added to the list of templates appearing on a screen. At step 306, agent server 20 selects the blank template as the current template, after which method 300 ends.

Referring again to step 302, if it is determined that the template command is not a create template command, than method 300 moves to step 308 where agent server 20 determines whether the command is a copy template command. A copy template command is one which copies an existing template. A user may enter this command when it is desirable to create a new template from a previously created template rather than the generic blank template.

If it is determined that the command is a copy template command, then at step 310 agent server 20 loads the previously created template. "Loading" a template means that the template is stored into memory, such as random access memory (RAM). At step 312, agent server 20 creates and stores a copy of the template into persistent memory, such as a hard drive. At step 314, agent server 20 selects the copy of the template as the current template, after which method 300 ends.

Referring again to step 308, if it is determined that the template command is not a copy template command, then method 300 moves to step 316 where agent server 20 determines whether the command is an edit template command. An edit template command is one which edits an existing template. A user may enter this command when it is desirable to edit either a newly created or previously created template.

If it is determined that the command is an edit template command, agent server 20 loads the template at step 318 and allows changes to be made to the template at step 320. At step 322, the edited template is stored back into persistent memory and then is made the current (selected) template, after which method 300 ends.

Otherwise, if it is determined at step 316 that the template command is not an edit template command, then method 300 moves to step 324 where agent server 20 determines whether the command is a delete template command. A delete template command is one which deletes an existing template. A user may enter this command to remove from permanent memory any template which is obsolete, incorrect, or otherwise undesirable.

If it is determined that the command is a delete template command, agent server 20 deselects the current template at step 326. Deselection is performed because a deleted template cannot be subsequently selected. Next, at step 328, agent server 20 deletes the template from persistent memory. In the context of a graphical user interface, this step would also remove an icon for such template from the screen. Method 300 then ends.

Referring again to step 324, if it is determined that the command is not a delete template command, method 300 ends because the command was not any of the types of commands for a template.

For each of steps 310, 318, and 326 above, if no template is currently selected, agent server 20 will generate an error message.

Method 300, as described, thus provides user extensibility for network system 2 by allowing a user to create, copy, edit, and delete agent templates as desired.

Agent Command Execution

Figure 12 is a flow diagram of an exemplary method 400 for executing an agent command, according to an embodiment of the present invention. Method 400 is initiated within and performed by network system 2 when a user has entered an agent command. Generally speaking, an agent command is one which affects a particular agent 22. Agent commands can be of three types: a create agent command, an edit agent command, and a delete agent command.

Method 400 begins at step 402 where agent server 20 determines whether the command is a create agent command. A create agent command is one which creates a new agent from an agent template. After an agent 22 has been created, it is generally allowed to execute, thereby performing tasks for the user who created the agent.

If it is determined that the command is a create agent command, then at step 404 agent server 20 loads the respective agent template. If no template is currently selected, agent server 20 will generate an error message. Otherwise, at step 406, agent server 20 stores the newly created agent 22 and at step 408 selects that agent 22 as the current agent. Afterwards, method 400 ends.

With reference again to step 402, if it is determined that the command is not a create agent command, then at step 410 agent server 20 determines whether the command is an edit agent command. An edit agent command is one which allows the user to edit an executing or running agent. If no agent 22 is currently selected, however, agent server 20 generates an error message.

If it is determined that the command is an edit agent command, then agent server 20 loads the executing agent 22 at step 412. Agent server 20 suspends the execution of such agent 22 at step 414 and allows changes to be made to the agent by a user at step 416. At step 418, agent server 20 resumes the execution of the agent 22 and, at step 420, stores the edited agent. Afterwards, method 400 ends.

Referring again to step 410, if it is determined that the agent command is not an edit agent command, then at step 422 agent server 20 determines whether the command is a delete agent command. In general, a delete agent command is one which ends the execution of an agent 22 and removes it from persistent memory. If no agent 22 is currently selected, however, agent server 20 generates an error message.

If it is determined at step 422 that the command is a delete agent command, then at step 424 agent server 20 deselects the agent 22, because an agent cannot be selected once it has been deleted. At step 426, agent server 20 deletes the agent 22. Method 400 then ends.

Referring again to step 422, if it is determined that the command is not a delete agent command, method

400 ends because the command was not any of the types of commands for an agent.

Consumption of Service and Computational Resources

Figure 13 is a block diagram detailing the controlled consumption of service resources 25 and computational resources 21 by an agent 22, according to an embodiment of the present invention.

As previously described, an agent 22 may consume various service resources 25 (e.g., long-distance calling time, on-line data access time, etc.) and computational resources 21 (e.g., memory space, processing time, etc.) in the performance of its tasks.

The amount of the resources consumed by a particular agent 22 is limited by permissions which are specified in permissions component 56 of an agent object 46 corresponding to the agent 22. More specifically, for a particular agent 22, permissions component 56 includes one set of service permissions 64 for each service 24 utilized by that agent. The service permissions 64 bound the service resources 25 expended on behalf of the agent 22 by that service 24.

That is, each service permission 64 specifies whether the agent 22 is authorized to consume a particular service resource 25 and, in some instances, the amount of such service resource 25 that is allowably consumed by that agent 22.

Agent server 20 and service wrappers 26 cooperate in order to ensure that an agent 22 does not consume more than its allotted amount of any particular service resource 25 as specified by a respective service permission 64.

Permissions component 56 also includes a set of computational permissions 66 for the agent 22. These computational permissions 66 bound the computational resources 21 expended on the agent's behalf by agent server 20. In other words, each computational permission 66 specifies whether the agent 22 is authorized to consume a particular computational resource 21 and, in some instances, the amount of the computational resource 21 which is allowably consumed by that agent.

Agent server 20, together with service wrappers 26, monitors and ensures that the agent 22 does not consume more than its allotted amount of any computational resource 21 as specified by a respective computational permission 66.

Agent Population Execution

Figure 14 is a flow diagram of an exemplary method 500 for executing time slices for an agent population, according to an embodiment of the present invention. Method 500 is performed by agent server 20 in order to enable each agent 22 to make progress in performing its respective tasks.

In general, the processing capability of agent server 20 cannot be dedicated solely to any particular agent 22, but rather must be allocated among all executing agents. Processing time by agent server 20 is thus divided into time slices. The time slices can be of predetermined duration. During each time slice, the processing capability of agent server 20 is directed to a particular agent 22.

With reference to Figure 14, method 500 begins at step 502 where agent server 20 selects an agent 22, which is responsible for performing one or more tasks for a respective user. At step 504, agent server 20 executes a time slice for the selected agent 22. During this time slice, the processing capability of server 20 is directed to or used for the selected agent 22.

At step 506, agent server 20 determines whether there are any other agents 22 for which a time slice should be executed. If there are more agents, method 500 returns to step 502 where the next agent is selected. Agent server 20 repeats steps 502-506 until a time slice has been executed for each agent 22. Afterwards, method 500 ends.

Agent Execution

Figure 15 is a flow diagram of an exemplary method 600 for executing a time slice for an agent 22, according to an embodiment of the present invention. Method 600 is performed by agent server 20 for a particular agent 22.

Method 600 begins at step 602 where an agent object 46 corresponding to the particular agent 22 is loaded into agent server 20. This agent object 46 comprises a permissions component 56, an event handlers component 58, a datastore component 60, and a pending event queue 62.

At step 604, agent server 20 determines whether execution of the agent 22 has been suspended (e.g., because the user is editing the agent). If execution has been suspended, it is unnecessary to provide any processing for the agent, and accordingly, method 600 ends.

Otherwise, if execution of the agent 22 has not been suspended, method 600 proceeds to step 605 where agent server 20 determines whether an event is currently being handled by agent 22. An event is something that triggers or causes an agent 22 to take action, or something to which agent 22 must respond. If it is determined that there is an event currently being handled, method 600 moves to step 612.

Alternatively, if it is determined at step 605 that there is not an event currently being handled, method 600 proceeds to step 606 where agent server 20 determines whether there is an event pending for the agent--i.e., an event for which handling has not yet begun. This can be accomplished by examining pending event queue component 62 of the corresponding agent object 46. If there is no event pending for agent 22, method 600 ends.

Otherwise, if it is determined that there is an event pending for agent 22, then at step 608 agent server 20 removes or dequeues the event from the pending event queue of the corresponding agent object 46.

At step 610, agent server 20 determines whether agent 22 has a handler for responding to the event. Generally, an event handler comprises a routine for responding to an event of a particular type. An event handler can be divided into a plurality of portions, each of which is separately executable during different time slices. Event handlers are specified in the event handlers component 58 of agent object 46. If it is determined that agent 22 does not have a handler for the relevant event, method 600 moves to step 614. Otherwise, if it is determined that agent 22 does have a handler for responding to the event, method 600 moves to 612.

At step 612, agent server 20 executes a portion of the event handler for agent 22. Method 600 then proceeds to step 614.

At step 614, agent server 20 stores the agent 22, after which method 600 ends.

Event Handler Execution

Figure 16 is a flow diagram of an exemplary method 700 for executing an event handler, according to an embodiment of the present invention. Method 700 is performed by agent server 20 when an event handler is invoked in response to the occurrence of a particular event. The event handler routine may include a number of instructions which are directed to agent server 20 and/or one or more service wrappers 26.

Method 700 begins its step 702 where agent server 20 receives an instruction to execute in a patroller time slice. At step 704, agent server 20 determines whether the instruction is a service instruction. A service instruction is an instruction relating to and utilizing a particular service 24. Accordingly, a service instruction can only be executed via a respective service wrapper 26. All other instructions do not require the cooperation of a respective service wrapper 26, but rather, can be executed by agent server 20 itself.

If it is determined at step 704 that the instruction is a service instruction, method 700 proceeds to step 708 where agent server 20 asks the respective service wrapper 26 to execute the instruction for agent 22. Method 700 then moves to step 710.

Referring again to step 704, if it is determined that the instruction is not a service instruction, then agent server 20 itself executes the instruction at step 706. Method 700 then proceeds to step 710.

At step 710, agent server 20 determines whether there are any more instructions in the event handler. If so, method 700 returns to step 702 where agent server 20 fetches the next instruction. Agent server 20 repeats steps 702-710 until all instructions of the event handler to be executed as part of the current time slice have been executed, either by agent server 20 or respective service wrappers 26. Afterwards, method 700 ends.

Service Instruction Execution

Figure 17 is a flow diagram of an exemplary method 800 for executing a service instruction, according to an embodiment of the present invention. Method 800 is performed by a service wrapper 26 when agent server 20 asks the service wrapper to execute a service instruction relating to the respective service 24.

Method 800 begins at step 802 where service wrapper 26 identifies the service permissions required by the service instruction. In other words, the execution of the service instruction may cause particular service resources 25 (e.g., long-distance calling time, on-line data access time, or memory storage space) to be consumed. Thus, an assessment is made as to the service permissions needed in order to carry out the service instruction.

At step 804, service wrapper 26 asks agent server 20 for the permissions held by the agent 22 for which the instruction is being executed. The permissions held by an agent 22 are not necessarily the same as the permissions which are required in order to execute an instruction. For example, agent 22 may have permissions only for long-distance calling time and memory storage space, but execution of the instruction requires permission for on-line data access time.

Thus, at step 806, service wrapper 26 determines whether the permissions held by agent 22 include the permissions required in order to execute the instruction. If it is determined that the permissions held do not include the permissions required, service wrapper 26 executes an error routine at step 808, after which method 800 ends.

Otherwise, if it is determined at step 806 that the permissions held do include the permissions required, method 800 precedes to step 810 where, for each service permission, service wrapper 26 identifies the amount of a service resource 25 which is allotted to agent 22 and the amount of that service resource 25 which must be consumed in order to execute the instruction. As with the permissions, the amount of a service resource 25 allotted to an agent 22 is not necessarily the same as the amount of that service resource 25 which is required to execute the instruction. For example, an agent 22 may have permission to utilize up to a predefined amount of memory storage space to store a voice mail message, but the actual amount required for a particular message is greater than the predefined amount.

At step 812, service wrapper 26 determines whether the amount allotted to agent 22 is at least as great as the amount required to execute the instruction. If it is determined that the amount allotted is not at least as great as the amount required, service wrapper 26 executes an error routine at step 814, after which method 800 ends.

Referring again to step 812, if it is determined that the amount of the service resource 25 allotted to agent 22 is at least as great as the amount which is required to execute the instruction, then method 800 moves to step 816 where service wrapper 26 asks the respective service 24 to execute the instruction, thereby enabling agent 22 to perform one or more tasks for its user/principal.

At step 818, service wrapper 26 identifies the amount of each service resource 25 actually consumed or used to execute the instruction. At step 820, for each service resource 25, service wrapper 26 asks agent server 20 to decrement the amount allotted to agent 22 by the amount actually used. Method 800 then ends.

In method 800 described above, it is assumed that service wrapper 26 can determine in advance what amount of service resource 25 will be consumed in the execution of an instruction. This, however, is not always the case. For example, while a predefined maximum amount of memory space may be set aside to store any given voice mail message, the amount of storage space actually consumed depends on how long the caller speaks. Other service resources 25 can be consumed on an on-going basis even after an instruction has executed. For example, if a long-distance call is made on behalf of a subscriber, connect time for a long-distance service continues until the call has been completed. In both cases, service wrapper 26 actively monitors service resource consumption and halts further consumption whenever the amount held by an agent 22 is exhausted.

Accordingly, the present invention provides a system and method which allow subscribers or third parties to create and customize personalized agents 22 which utilize services 24 (e.g., e-mail, voice mail, electronic address book, electronic contact book, etc.) in order to perform respective tasks for the subscribers. Thus, the present invention affords user extensibility of the services 24.

Furthermore, the present invention monitors and controls the consumption of service resources 25 and computational resources 21 by agents 22 during execution. Thus, the present invention protects the subscribers and a service provider from misuse or overuse, whether intentional or inadvertent, of such resources.

While particular embodiments of the present invention and their advantages have been shown and described, it should be understood that various changes, substitutions, and alterations can be made therein without departing from the spirit and scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. A network system comprising:

a service;

an agent operable to use the service on behalf of a principal; and

an agent server operable to mediate the use of the service by the agent.

2. The network system of Claim 1 further comprising a computational resource which can be consumed by the agent.

3. The network system of Claim 2 wherein the agent server is operable to monitor the consumption of the computational resource by the agent.

4. The network system of Claim 3 wherein the agent server is operable to charge the principal for the consumption of the computational resource.

5. The network system of Claim 3 wherein the agent server is operable to terminate the agent in response to the monitoring.

6. The network system of Claim 2 further comprising a computational permission associated with the computational resource and the agent, the computational permission specifying a predetermined amount of the computational resource which the agent is allowed to consume.

7. The network system of Claim 2 further comprising a computational permission associated with the computational resource and the principal, the computational permission specifying a predetermined amount of the computational resource which is allowed to be consumed on behalf of the principal.

8. The network system of Claim 2 wherein the computational resource can be processing time for the agent or elapsed time for the agent.

9. The network system of Claim 1 further comprising a service resource associated with the service and which can be consumed by the agent when the agent uses the service.

10. The network system of Claim 9 further comprising a service wrapper associated with the service, the service wrapper operable to monitor the consumption of the service resource by the agent.

11. The network system of Claim 10 wherein the service wrapper is operable to terminate the agent in response to the monitoring.

12. The network system of Claim 9 wherein the agent server is operable to charge the principal for the consumption of the service resource.

13. The network system of Claim 9 further comprising a service permission associated with the service resource and the agent, the service permission specifying a predetermined amount of the service resource which the agent is allowed to consume.

14. The network system of Claim 9 further comprising a service permission associated with the service resource and the principal, the service permission specifying a predetermined amount of the service resource which is allowed to be consumed on behalf of the principal.

15. The network system of Claim 9 wherein the service resource comprises elapsed time for a telephone call.

16. The network system of Claim 9 wherein the service resource comprises cost for a telephone call.

17. The network system of Claim 1 further comprising a service wrapper associated with the service, the service wrapper operable to translate between a first instruction set and a second instruction set.

18. The network system of Claim 1 further comprising a user interface coupled to the agent server, the user interface operable to allow a user to create, modify, or delete an agent template from which the agent is created.

19. The network system of Claim 1 further comprising a user interface coupled to the agent server, the user interface operable to allow a user to create, modify, or delete the agent.

20. The network system of Claim 19 wherein the user interface comprises a graphical user interface.

21. The network system of Claim 20 wherein the graphical user interface comprises:

a web server resident in the network system; and

a web browser resident in a user device.

22. The network system of Claim 19 wherein the user interface comprises a voice user interface.

23. The network system of Claim 22 wherein the voice user interface comprises a speech recognition application.

24. The network system of Claim 22 wherein the voice user interface comprises a speech synthesis application.

25. The network system of Claim 22 wherein the voice user interface comprises a speech playback device.

26. The network system of Claim 1 wherein the service comprises a user interface service operable to provide a user interface by which the agent can interact with a user.

27. The network system of Claim 26 wherein the user interface comprises a graphical user interface.

28. The network system of Claim 27 wherein the graphical user interface comprises:

a web server resident in the network system; and

a web browser resident in a user device.

29. The network system of Claim 27 wherein the user interface comprises a voice user interface.

30. The network system of Claim 29 wherein the voice user interface comprises a speech recognition application.

31. The network system of Claim 29 wherein the voice user interface comprises a speech synthesis application.

32. The network system of Claim 29 wherein the voice user interface comprises a speech playback device.

33. A network system comprising:

a user interface operable to allow a user to interact with the network system; and

an agent server coupled to the user interface, the agent server operable to manage the operation of a programmable functionality component of the network system, the agent server in conjunction with the user interface operable to create, modify, or delete an agent in response to interaction by the user.

34. The network system of Claim 33 wherein the user interface comprises a graphical user interface.

35. The network system of Claim 34 wherein the graphical user interface comprises a web browser application.

36. The network system of Claim 33 wherein the user interface comprises a voice user interface.

37. The network system of Claim 36 wherein the voice user interface comprises a speech recognition application.

38. The network system of Claim 36 wherein the voice user interface comprises a speech synthesis application.

39. The network system of Claim 36 wherein the voice user interface comprises a speech playback device.

40. The network system of Claim 33 wherein the user interface comprises an agent area for creating, modifying, or deleting the agent.

41. The network system of Claim 33 further comprising:

a service which can be utilized by the agent; and

a service wrapper associated with the service, the service wrapper operable to cooperate with the agent server to mediate interaction between the service and the agent.

42. The network system of Claim 41 wherein the service wrapper is operable to convert between a first computer language used by the agent and a second computer language used by the service.

43. The network system of Claim 41 further comprising a service resource associated with the service, the service resource being consumed when the agent utilizes the service.

44. The network system of Claim 43 wherein the service wrapper is operable to monitor the amount of service resource consumed by the agent when the agent utilizes the service.

45. A method comprising:

admitting a user to a network system wherein at least one agent is operable to utilize a service to perform a task for the user; and

allowing the user to create, modify, or delete the agent within the network system.

46. The method of Claim 45 further comprising accepting an agent command from the user.

47. The method of Claim 46 wherein the agent command is one of a create agent command, an edit agent command, and a delete agent command.

48. The method of Claim 45 further comprising suspending execution of the agent.

49. The method of Claim 45 further comprising accepting a template command from the user.

50. The method of Claim 49 further comprising allowing the user to create, modify, or delete an agent template within the network system.

51. The method of Claim 45 further comprising admitting the user to an agent area of a user interface.

52. A network system comprising:

an agent server operable to manage the operation of a programmable functionality component of the network system;

a service;

an agent operable to utilize the service; and

a service wrapper associated with the service, the service wrapper operable to cooperate with the agent server to mediate interaction between the service and the agent.

53. The network system of Claim 52 further comprising a service resource associated with the service, the service resource being consumed when the agent utilizes the service.

54. The network system of Claim 53 wherein the service wrapper comprises a monitor, the monitor operable to monitor the amount of the service resource consumed by the agent when the agent utilizes the service.

55. The network system of Claim 52 wherein the service wrapper comprises a converter, the converter operable to convert between a first computer language

used by the agent and a second computer language used by the service.

56. The network system of Claim 55 wherein the converter comprises an agent server to service converter for converting from the first computer language to the second computer language.

57. The network system of Claim 55 wherein the converter comprises a service to agent server converter for converting from the second computer language to the first computer language.

58. The network system of Claim 52 further comprising a user interface operable to allow a user to interact with the network system in order to create the agent.

59. The network system of Claim 58 wherein the user interface comprises a graphical user interface.

60. The network system of Claim 59 wherein the graphical user interface comprises a web browser application.

61. The network system of Claim 58 wherein the user interface comprises a voice user interface.

62. The network system of Claim 61 wherein the voice user interface comprises a speech recognition application.

63. The network system of Claim 61 wherein the voice user interface comprises a speech synthesis application.

64. The network system of Claim 61 wherein the voice user interface comprises a speech playback device.

65. The network system of Claim 52 further comprising a computational resource which can be consumed by the agent.

66. The network system of Claim 65 wherein the agent server is operable to monitor the amount of the computational resource consumed by the agent.

67. The network system of Claim 52 further comprising an agent object associated with the agent and resident in the agent server, the agent object comprising data and instructions executable within the agent server.

68. The network system of Claim 52 further comprising an agent object associated with the agent and resident in the agent server, the agent object comprising a permission for the agent.

69. A method comprising:

allowing an agent to utilize a service; and

mediating interaction between the service and the agent.

70. The method of Claim 69 wherein allowing comprises allowing the agent to consume a service resource associated with the service.

71. The method of Claim 69 wherein mediating comprises monitoring the amount of a service resource consumed by the agent.

72. The method of Claim 69 wherein mediating comprises:

identifying a service permission required in order for the agent to consume a service resource associated with the service; and

determining whether the agent holds the required service permission.

73. The method of Claim 69 wherein mediating comprises:

receiving an instruction which requires that the agent consume a service resource associated with the service when the agent utilizes the service;

identifying an amount of the service resource which would be consumed if the instruction is executed;

identifying an amount of the service resource allotted to the agent; and

determining whether the amount which is allotted is greater than or equal to the amount which would be consumed.

74. The method of Claim 73 wherein mediating comprises:

executing the instruction if the amount which is allotted is greater than or equal to the amount which would be consumed; and

subtracting the amount which would be consumed from the amount which is allotted.

75. The method of Claim 69 further comprising converting between a first computer language used by an agent and a second computer language used by the service.

76. The method of Claim 69 further comprising:

allowing an agent to consume a computational resource; and

monitoring the amount of the computational resource consumed by the agent.

NETWORK SYSTEM EXTENSIBLE BY USERS

Danny Lange
Barbara Nelson
Jing Su
James E. White

ABSTRACT OF THE DISCLOSURE

In one aspect, a network system includes a user interface which allows a user to interact with the network system. An agent server is coupled to the user interface. The agent server manages the operation of the network system. Furthermore, the agent server in conjunction with the user interface is operable to create or modify an agent in response to interaction by the user. In another aspect, a network system includes an agent server which manages the operation of the network system. An agent is operable to utilize a service within the network system. A service wrapper, associated with the service, cooperates with the agent server to mediate interaction between the service and the agent.

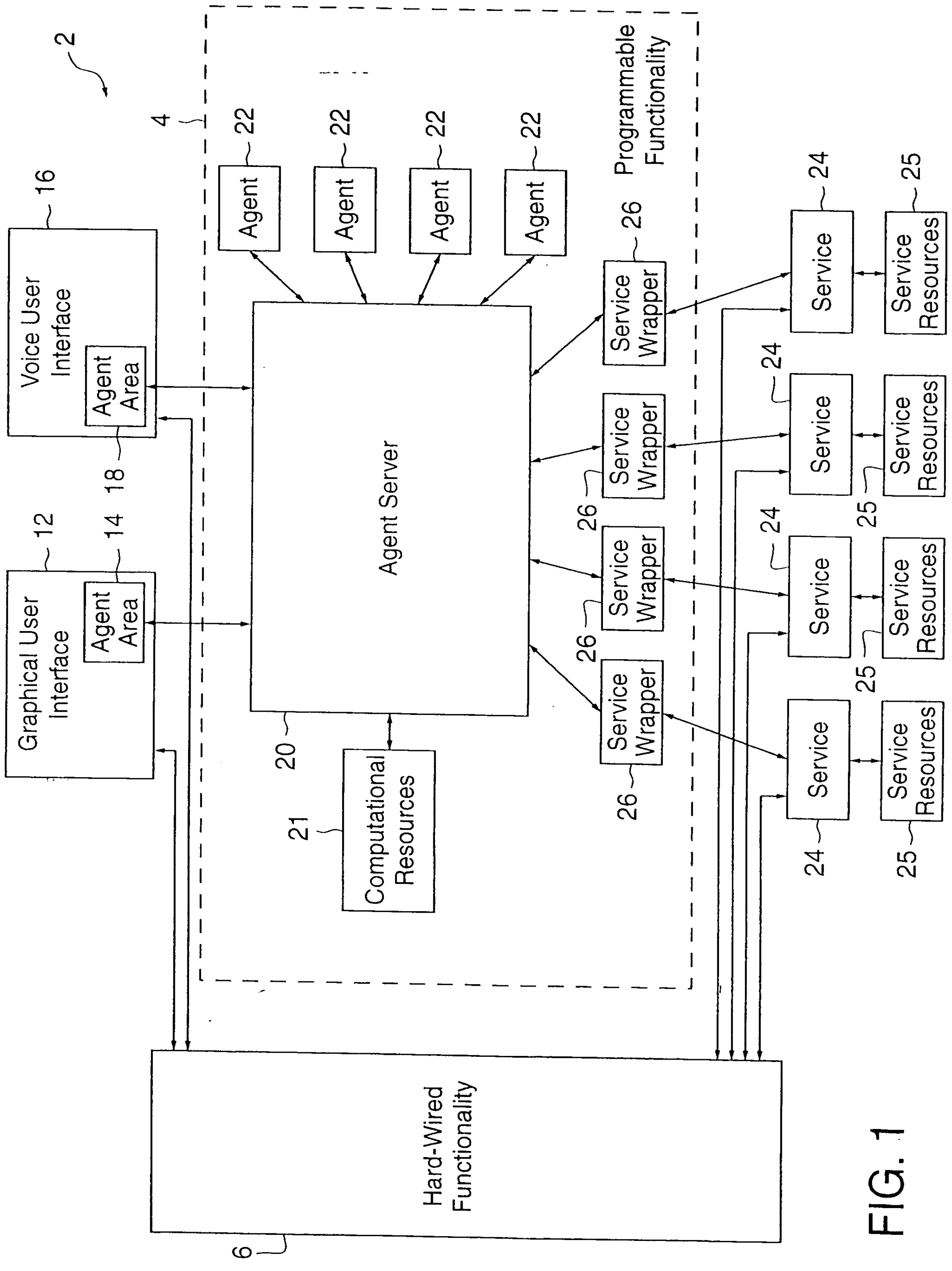

2
4
Programmable Functionality
Agent
22
Agent Server
20
Computational Resources
21
Service Wrapper
26
Service
24
Service Resources
25
Graphical User Interface
12
Agent Area
14
Voice User Interface
16
Agent Area
18
Hard-Wired Functionality
6

FIG. 1

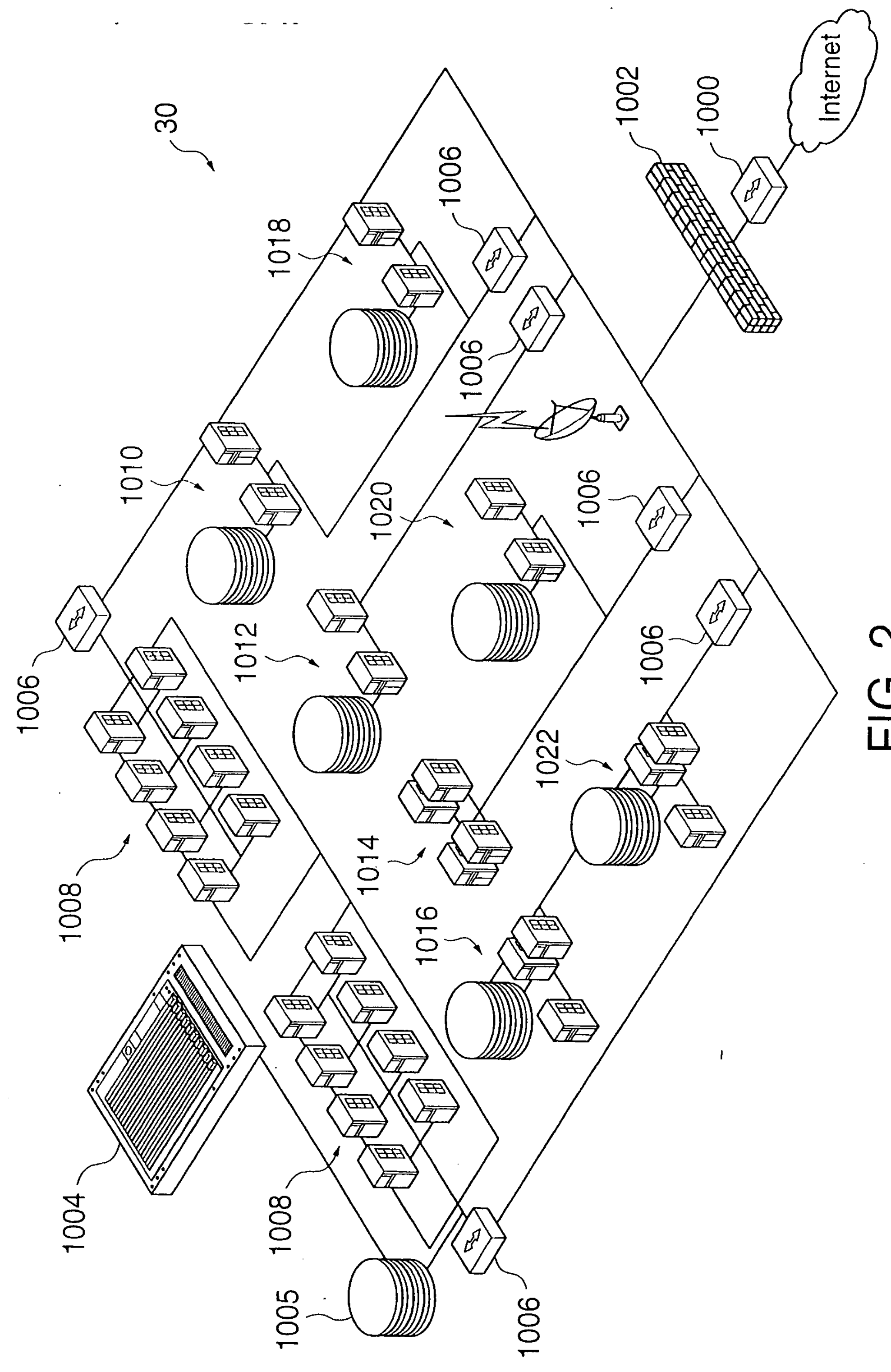

30
Internet
1000
1002
1004
1005
1006
1006
1006
1006
1006
1006
1008
1008
1010
1012
1014
1016
1018
1020
1022

FIG. 2

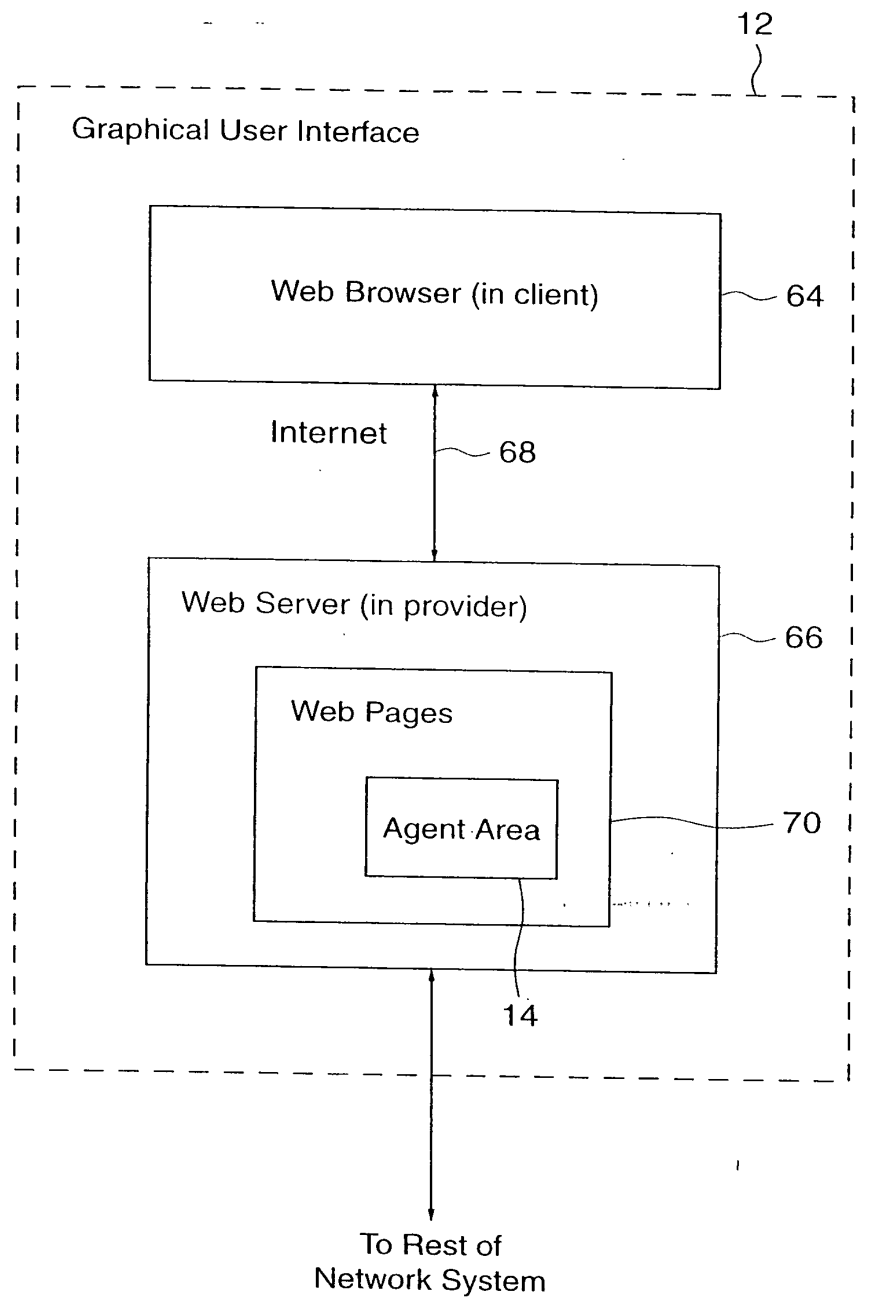

12
Graphical User Interface
Web Browser (in client)
64
Internet
68
Web Server (in provider)
66
Web Pages
Agent Area
70
14
To Rest of Network System

FIG. 3

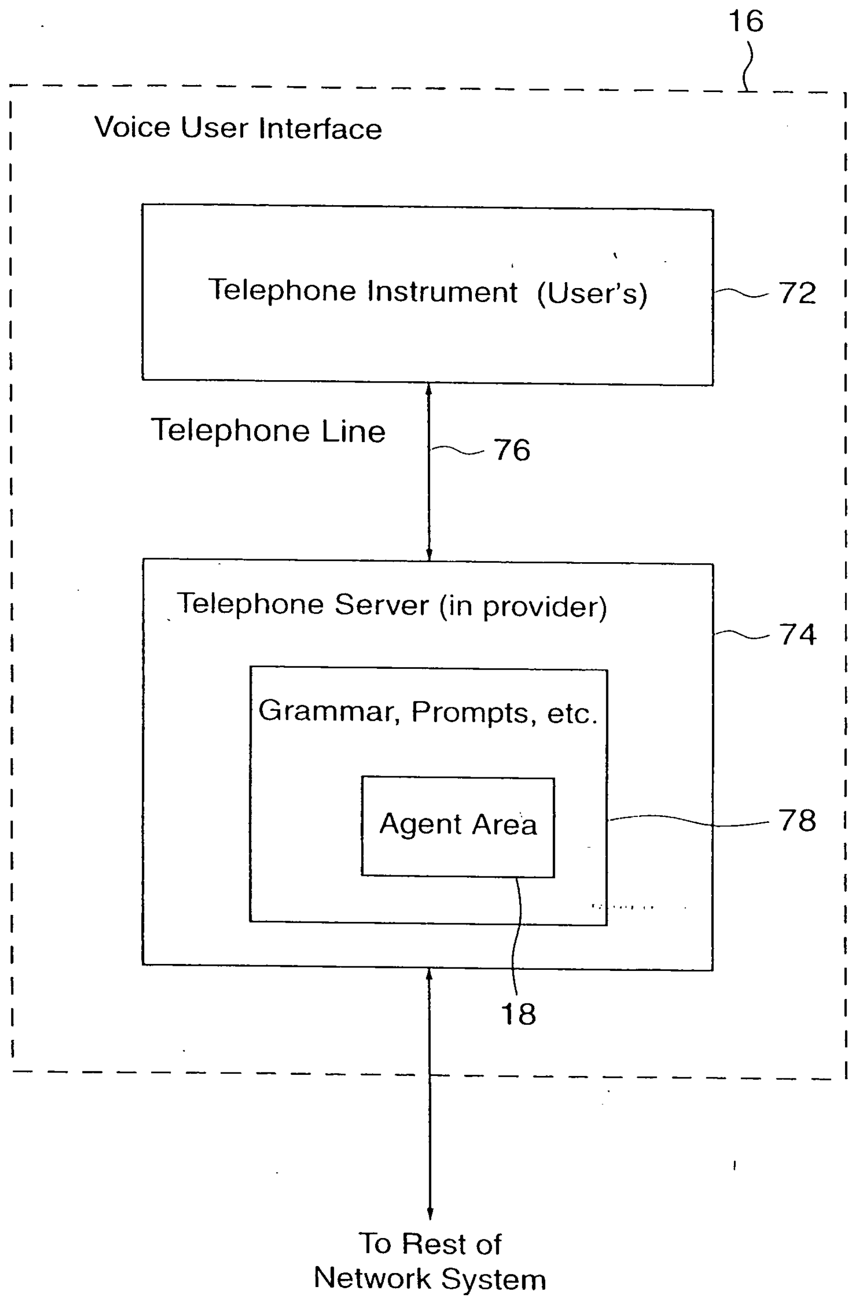

16
Voice User Interface
Telephone Instrument (User's)
72
Telephone Line
76
Telephone Server (in provider)
74
Grammar, Prompts, etc.
Agent Area
78
18
To Rest of Network System

FIG. 4

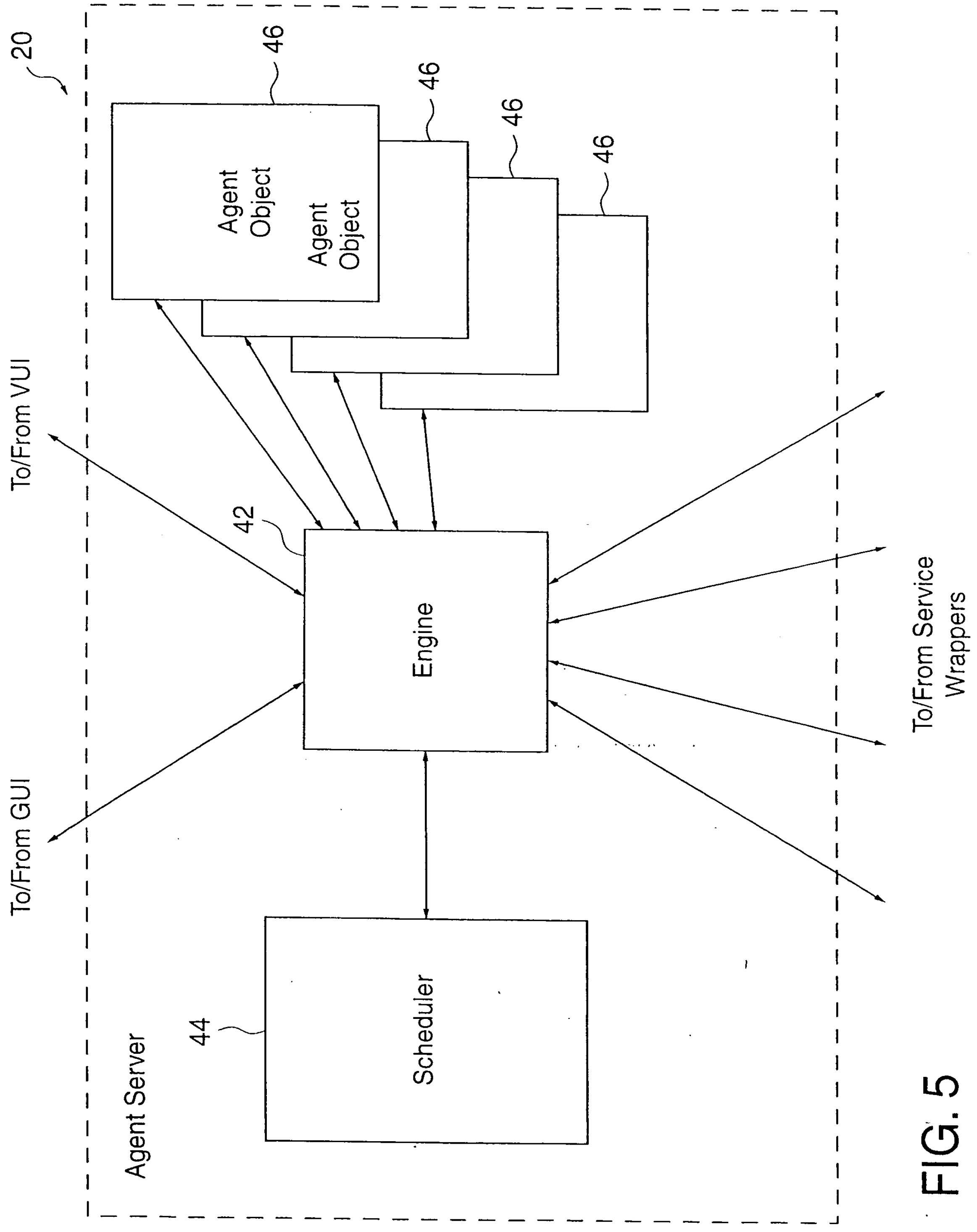

20
Agent Server
Agent Object
Agent Object
46
46
46
46
To/From VUI
To/From GUI
42
Engine
44
Scheduler
To/From Service Wrappers

FIG. 5

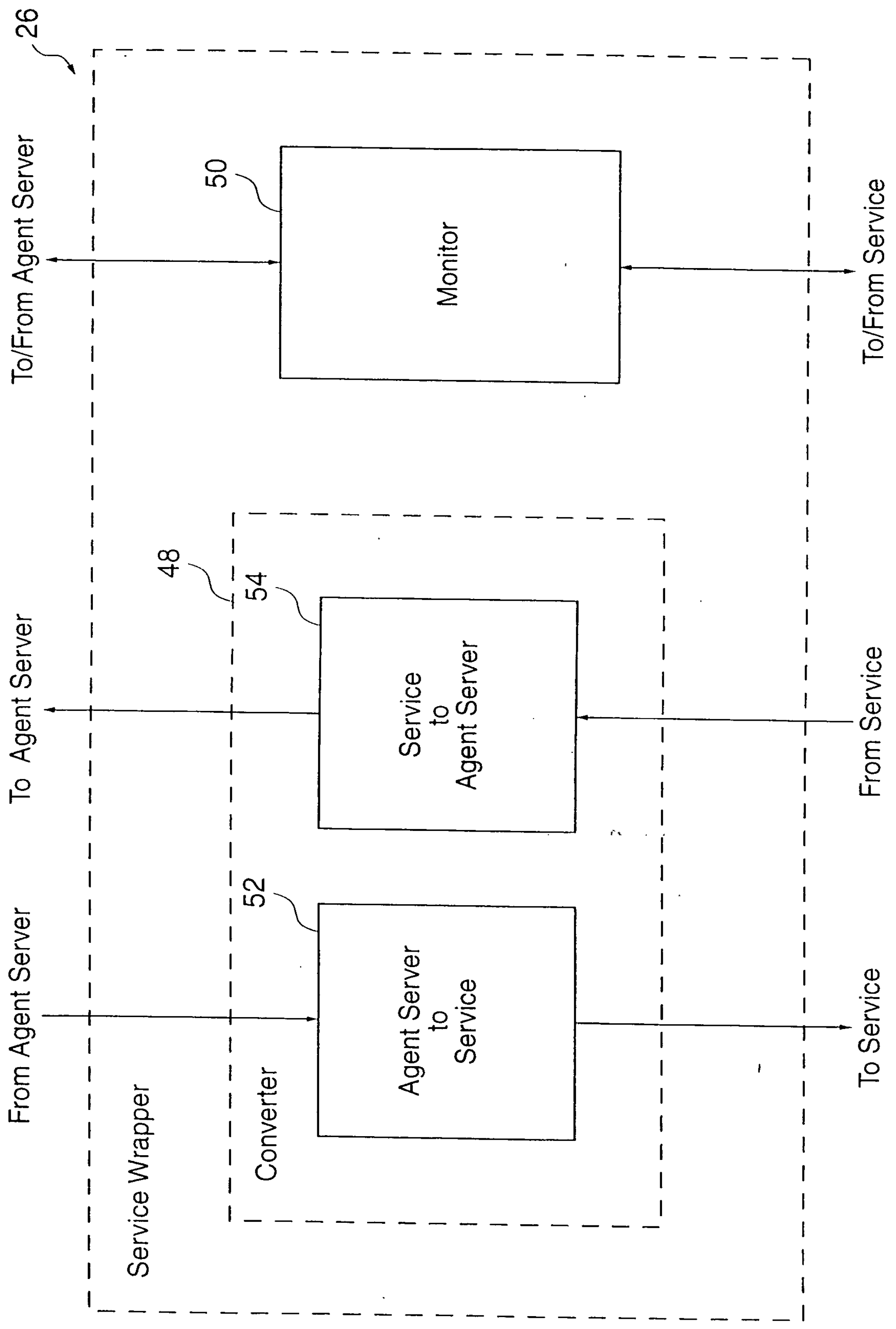
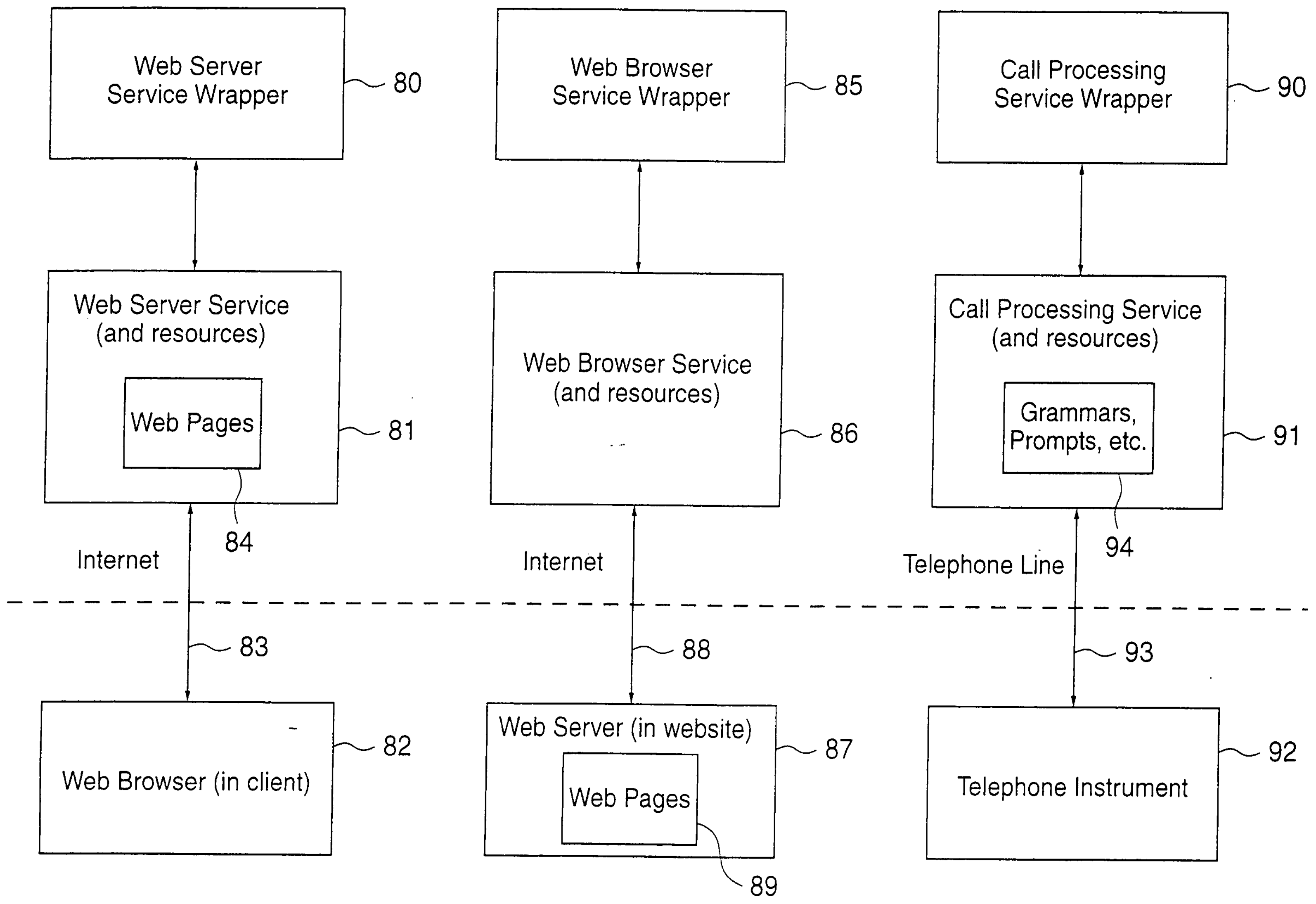

26
To/From Agent Server
50
Monitor
To/From Service
To Agent Server
48
54
Service to Agent Server
From Service
From Agent Server
52
Agent Server to Service
To Service
Service Wrapper
Converter

FIG. 6



Web Server Service Wrapper
80
Web Browser Service Wrapper
85
Call Processing Service Wrapper
90
Web Server Service (and resources)
Web Pages
81
84
Web Browser Service (and resources)
86
Call Processing Service (and resources)
Grammars, Prompts, etc.
91
94
Internet
Internet
Telephone Line
83
88
93
Web Browser (in client)
82
Web Server (in website)
Web Pages
87
89
Telephone Instrument
92

FIG. 7

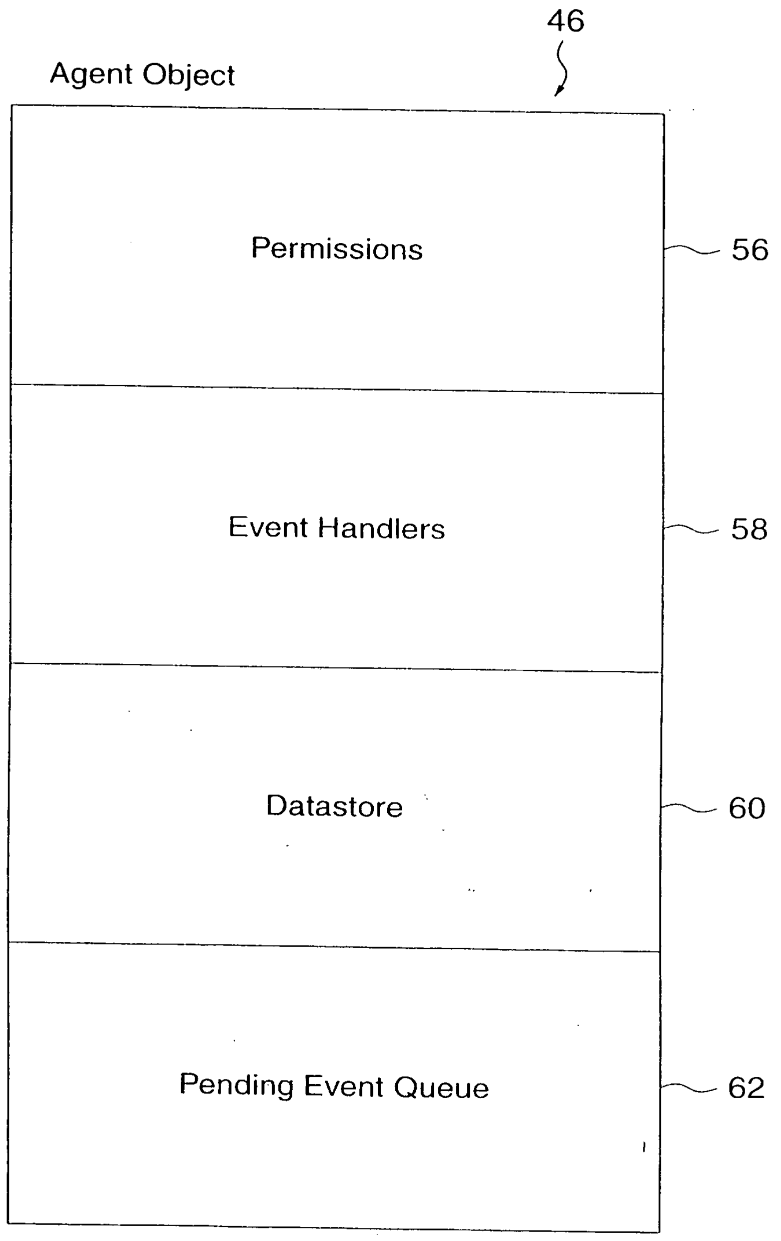

46
Agent Object
Permissions
56
Event Handlers
58
Datastore
60
Pending Event Queue
62

FIG. 8

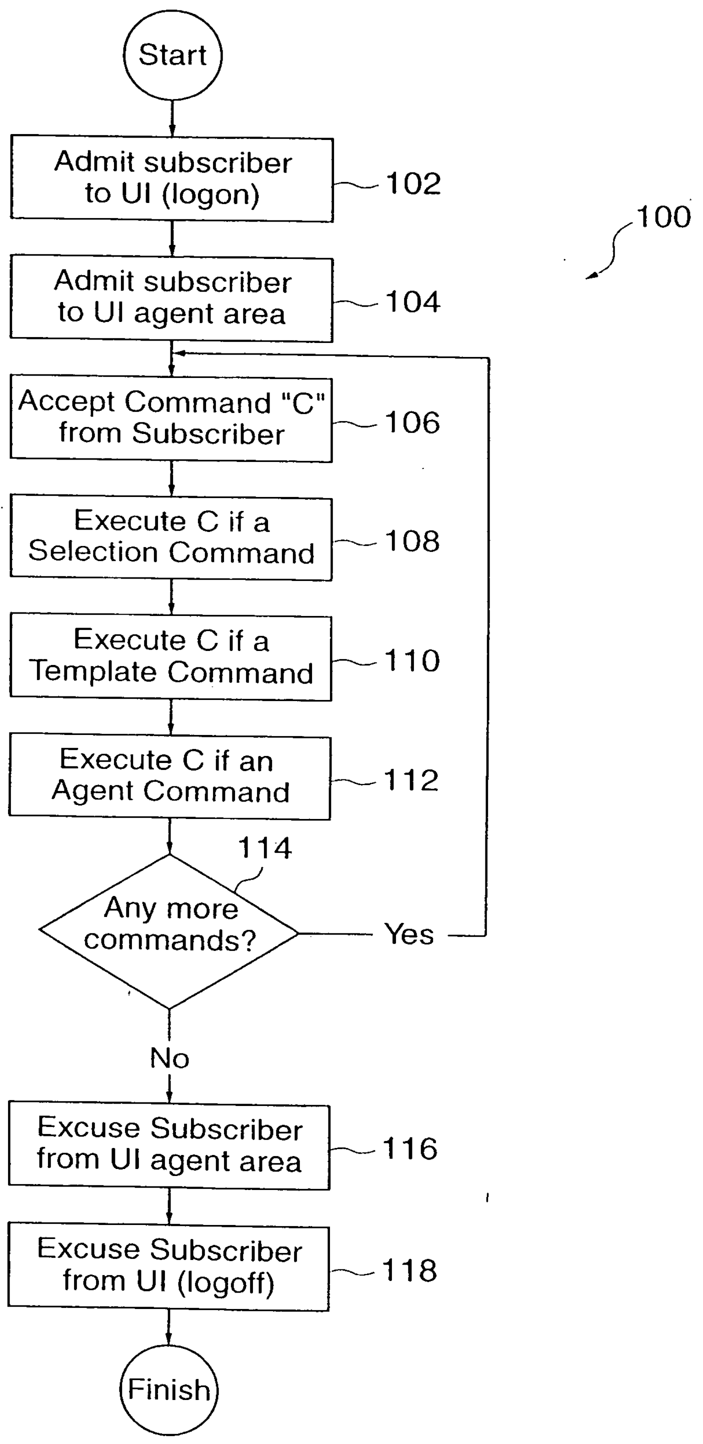

Start
Admit subscriber to UI (logon)
102
100
Admit subscriber to UI agent area
104
Accept Command "C" from Subscriber
106
Execute C if a Selection Command
108
Execute C if a Template Command
110
Execute C if an Agent Command
112
114
Any more commands?
Yes
No
Excuse Subscriber from UI agent area
116
Excuse Subscriber from UI (logoff)
118
Finish

FIG. 9

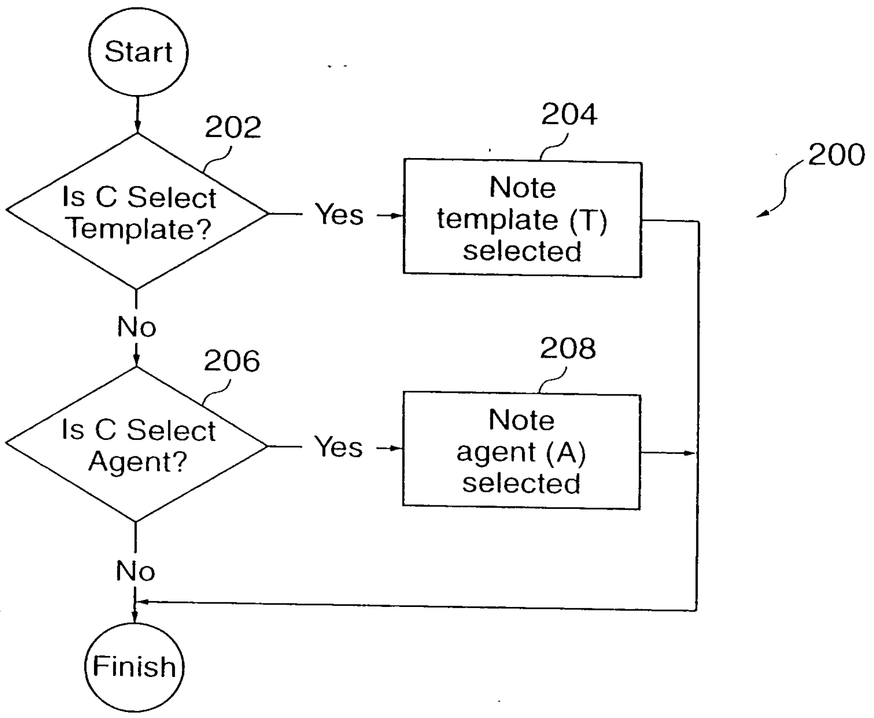

Start
202
Is C Select Template?
Yes
204
Note template (T) selected
200
No
206
Is C Select Agent?
Yes
208
Note agent (A) selected
No
Finish

FIG. 10

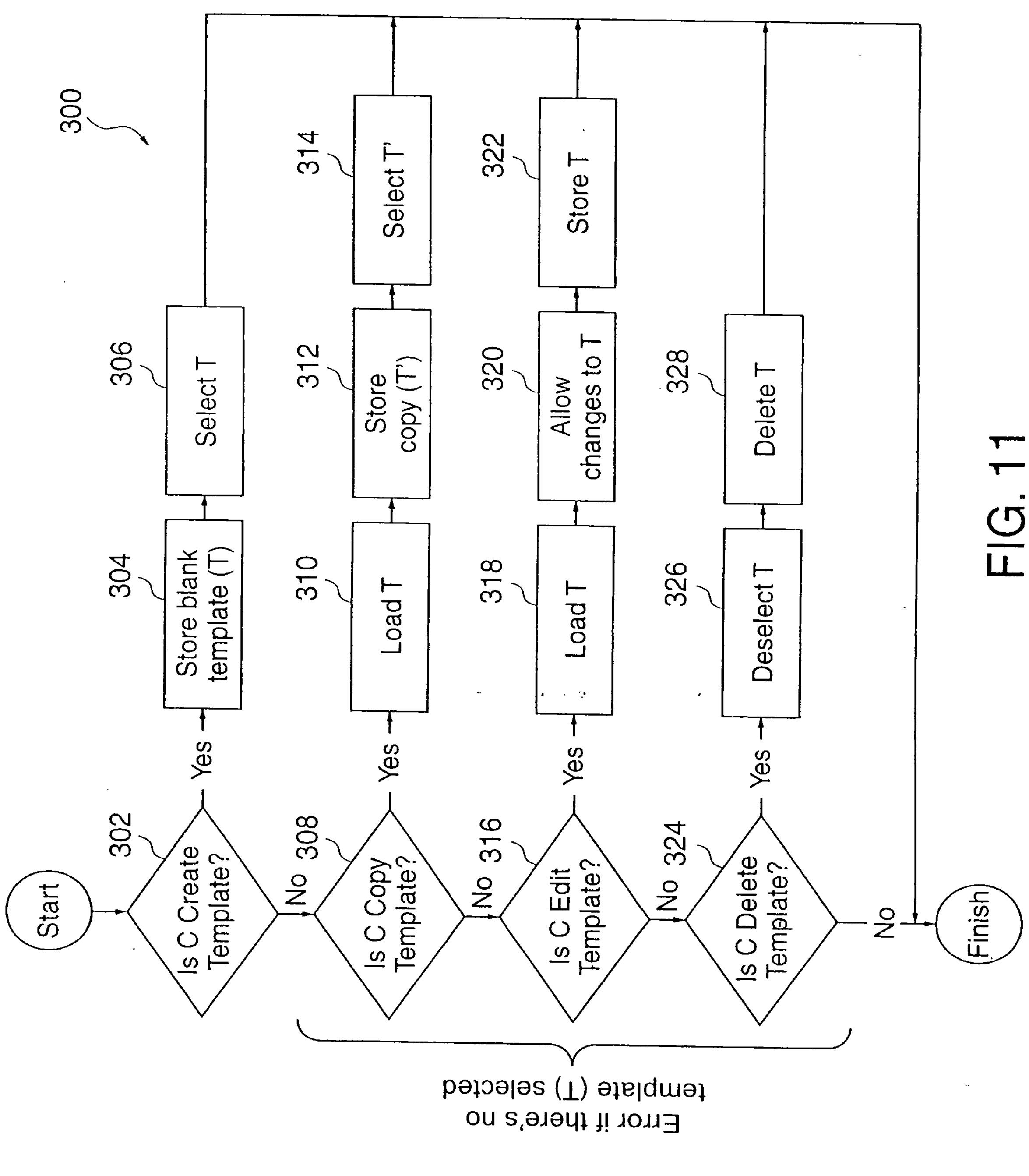

300
Start
302
Is C Create Template?
Yes
304
Store blank template (T)
306
Select T
No
308
Is C Copy Template?
Yes
310
Load T
312
Store copy (T')
314
Select T'
No
316
Is C Edit Template?
Yes
318
Load T
320
Allow changes to T
322
Store T
No
324
Is C Delete Template?
Yes
326
Deselect T
328
Delete T
No
Finish
Error if there's no template (T) selected

FIG. 11

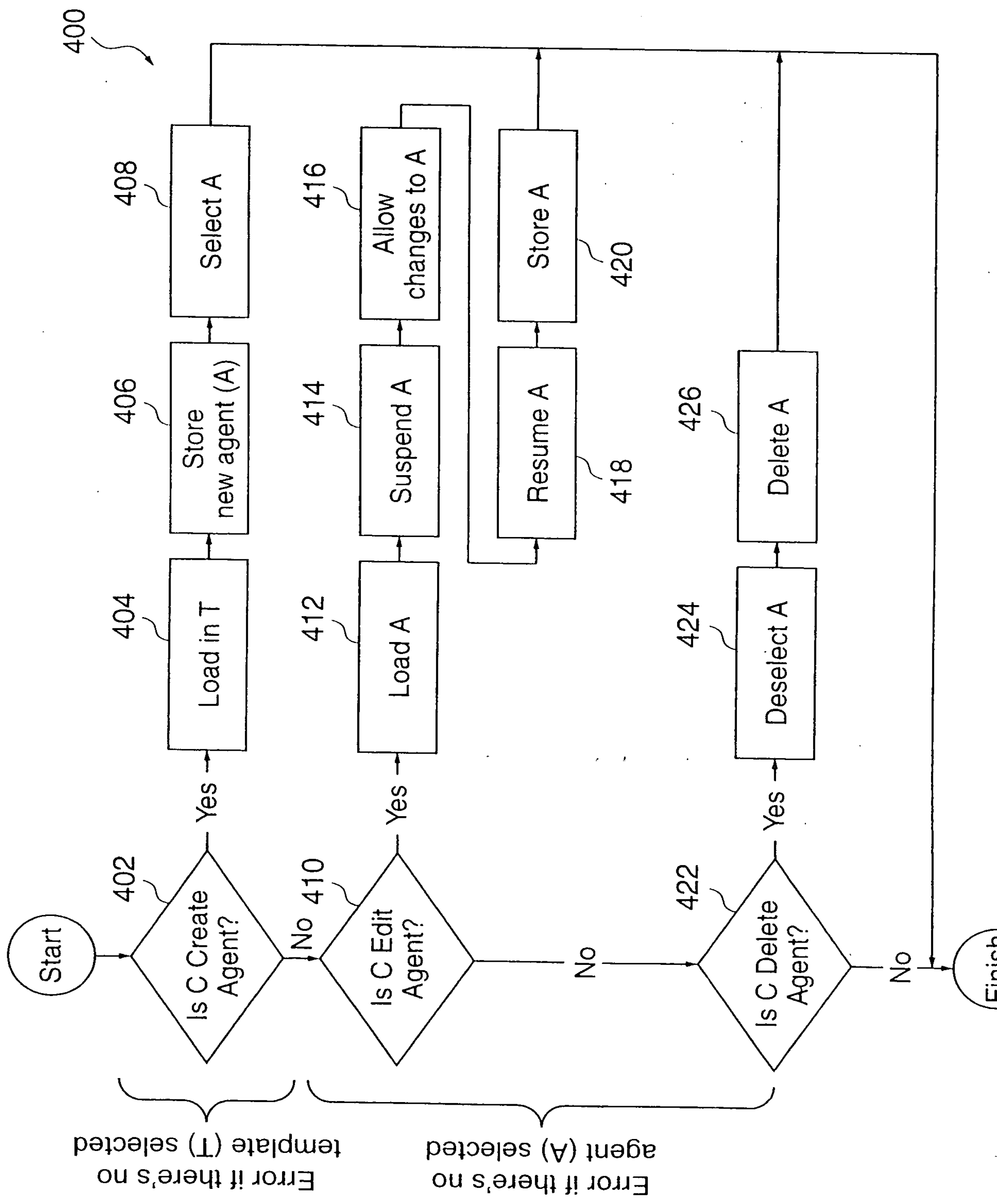

400
Start
402
Is C Create Agent?
Yes
404
Load in T
406
Store new agent (A)
408
Select A
No
410
Is C Edit Agent?
Yes
412
Load A
414
Suspend A
416
Allow changes to A
418
Resume A
420
Store A
No
422
Is C Delete Agent?
Yes
424
Deselect A
426
Delete A
No
Finish
Error if there's no template (T) selected
Error if there's no agent (A) selected

FIG. 12

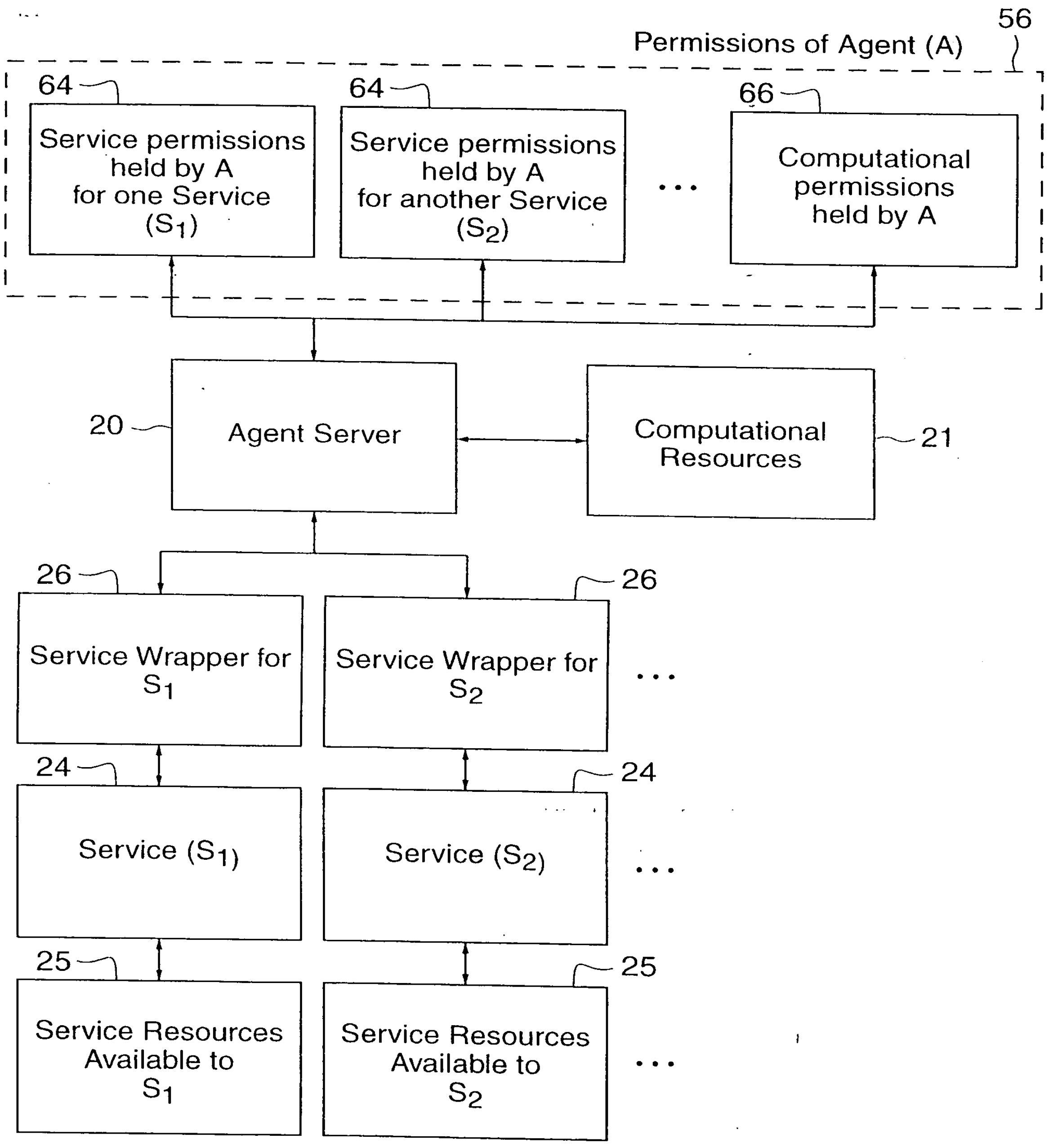

56
Permissions of Agent (A)
64
Service permissions held by A for one Service (S_1)
64
Service permissions held by A for another Service (S_2)
...
66
Computational permissions held by A
20
Agent Server
Computational Resources
21
26
Service Wrapper for S_1
26
Service Wrapper for S_2
...
24
Service (S_1)
24
Service (S_2)
...
25
Service Resources Available to S_1
25
Service Resources Available to S_2
...

FIG. 13

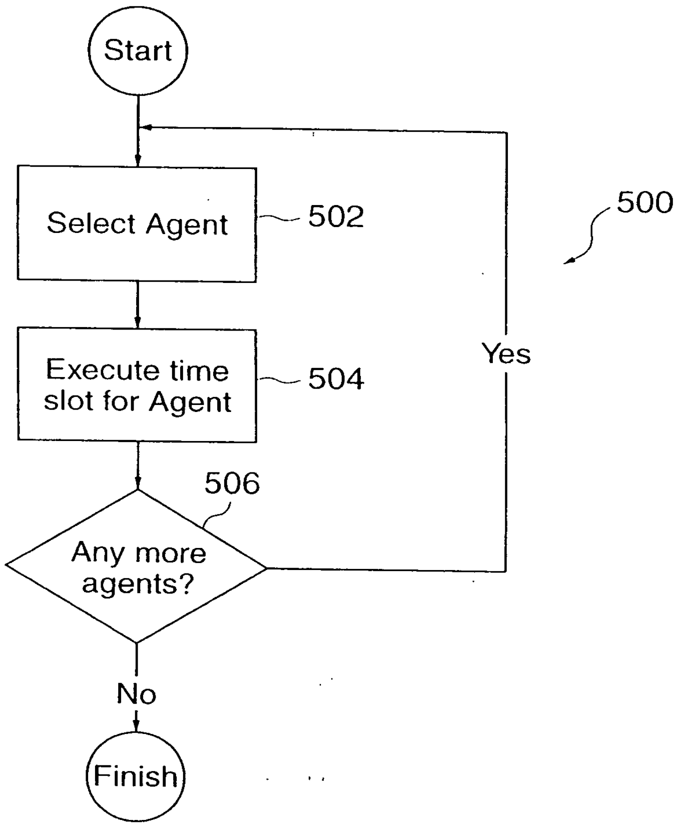

Start
Select Agent
502
Execute time slot for Agent
504
506
Any more agents?
Yes
No
Finish
500

FIG. 14

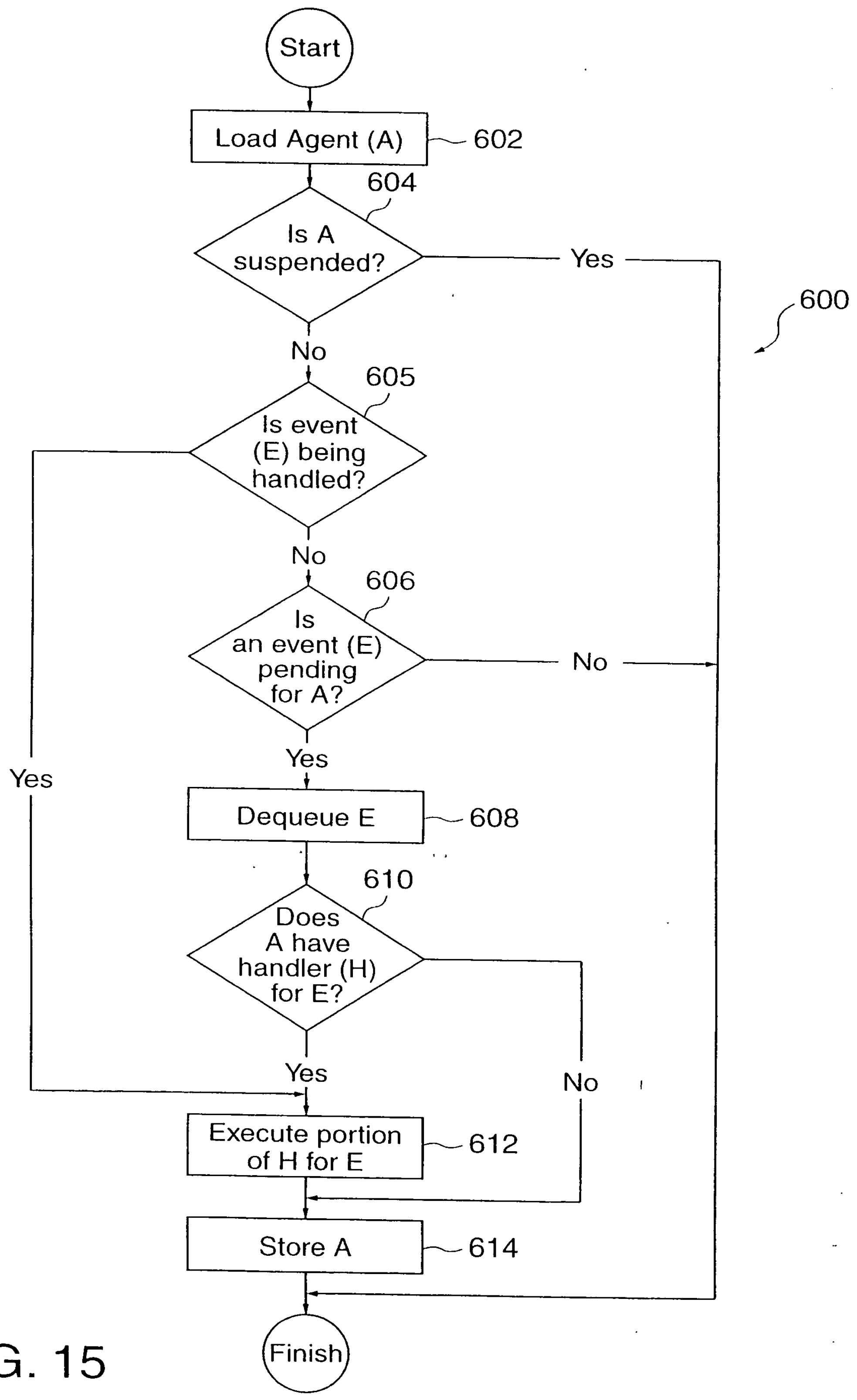

Start
Load Agent (A)
602
604
Is A suspended?
Yes
600
No
605
Is event (E) being handled?
No
606
Is an event (E) pending for A?
No
Yes
Yes
Dequeue E
608
610
Does A have handler (H) for E?
Yes
No
Execute portion of H for E
612
Store A
614
Finish

FIG. 15

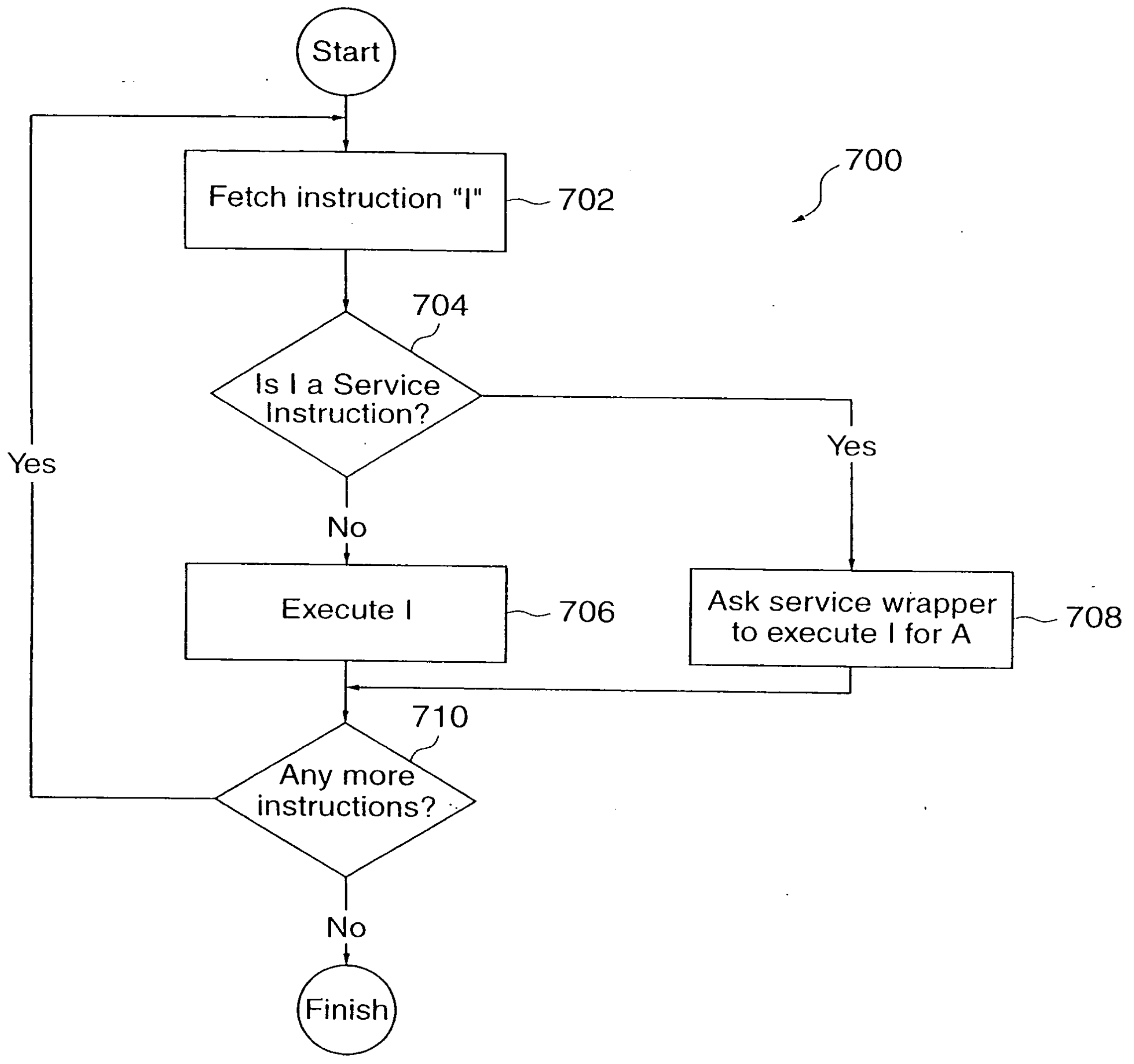
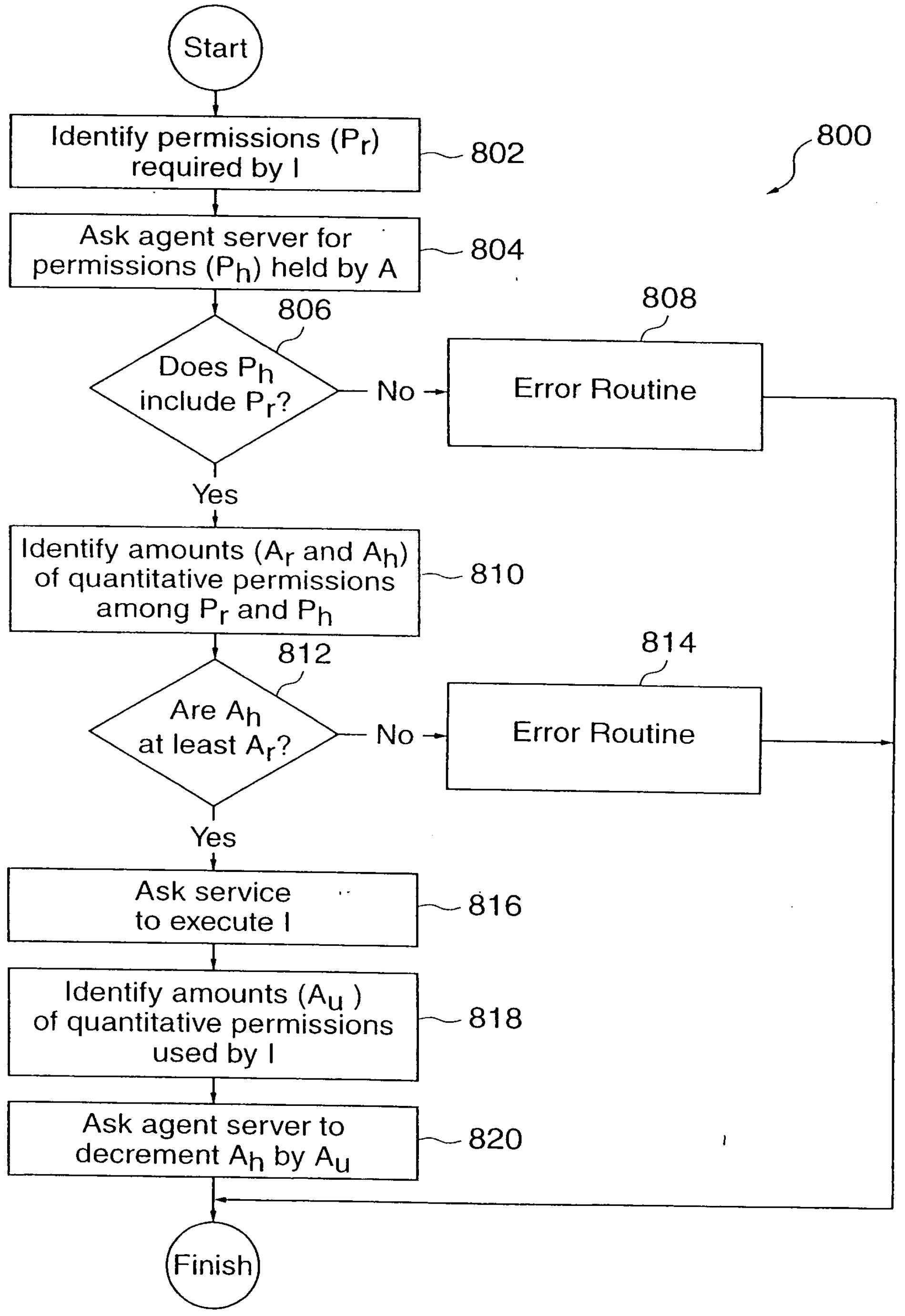

Start
Fetch instruction "I"
702
700
704
Is I a Service Instruction?
Yes
No
Yes
Execute I
706
Ask service wrapper to execute I for A
708
710
Any more instructions?
No
Finish

FIG. 16



800
Start
Identify permissions (P_r) required by I
802
Ask agent server for permissions (P_h) held by A
804
806
Does P_h include P_r?
No
808
Error Routine
Yes
Identify amounts (A_r and A_h) of quantitative permissions among P_r and P_h
810
812
Are A_h at least A_r?
No
814
Error Routine
Yes
Ask service to execute I
816
Identify amounts (A_u) of quantitative permissions used by I
818
Ask agent server to decrement A_h by A_u
820
Finish

FIG. 17

COPY

.orney Docket No.: M-6011 US

DECLARATION FOR PATENT APPLICATION AND POWER OF ATTORNEY

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below adjacent to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of subject matter (process, machine, manufacture, or composition of matter, or an improvement thereof) which is claimed and for which a patent is sought by way of the application entitled **"Network System Extensible By Users"**

which (check) ☒ is attached hereto.
☐ and is amended by the Preliminary Amendment attached hereto.
☐ was filed on ____________ as Application Serial No. ____________
☐ and was amended on ___ (if applicable).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information, which is material to patentability as defined in Title 37, Code of Federal Regulations, § 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, § 119(a)-(d) of any foreign application(s) for patent or inventor's certificate or any PCT international application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application(s) for patent or inventor's certificate or any PCT international application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) of which priority is claimed:

Prior Foreign Application(s)			Priority Claimed	
Number	Country	Day/Month/Year Filed	Yes	No
N/A			☐	☐

I hereby claim the benefit under Title 35, United States Code, § 119(e) of any United States provisional application(s) listed below:

Provisional Application Number	Filing Date
N/A	

I hereby claim the benefit under Title 35, United States Code, § 120 of any United States application(s) or PCT international application(s) designating the United States of America listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior application(s) in the manner provided by the first paragraph of Title 35, United States Code, § 112, I acknowledge the duty to disclose information, which is material to patentability as defined in Title 37, Code of Federal Regulations, § 1.56, which became available between the filing date of the prior application(s) and the national or PCT international filing date of this application:

464058 v1

Application Serial No.	Filing Date	Status (patented, pending, abandoned)
N/A		

I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and to transact all business in the United States Patent and Trademark Office connected therewith:

Alan H. MacPherson (24,423); Thomas S. MacDonald (17,774); Kenneth E. Leeds (30,566); Brian D. Ogonowsky (31,988); David W. Heid (25,875); Norman R. Klivans (33,003); Edward C. Kwok (33,938); David E. Steuber (25,557); Michael Shenker (34,250); Stephen A. Terrile (32,946); Peter H. Kang (40,350); Ronald J. Meetin (29,089); Ken John Koestner (33,004); Omkar K. Suryadevara (36,320); David T. Millers (37,396); Kent B. Chambers (38,839); Serge J. Hodgson (40,017); Michael P. Adams (34,763); Bernard Berman (37,279); Michael J. Halbert (40,633); Gary J. Edwards (41,008); William B. Tiffany (41,347); James E. Parsons (34,691); Daniel P. Stewart (41,332); Philip W. Woo (39,880); John T. Winburn (26,822); Tom Chen (42,406); Fabio E. Marino (43,339); William W. Holloway (26,182); Elaine H. Lo (41,158); Don C. Lawrence (31,975); Marc R. Ascolese (42,268); Carmen C. Cook (42,433); David G. Dolezal (41,711); Adrian J. Lee (42,785); Michael P. Noonan (42,038); Roberta P. Saxon (43,087); Bernice Chen (42,403); Mary Jo Bertani (42,321); and Dale R. Cook (42,434).

Please address all correspondence and telephone calls to:

Edward Kwok
Attorney for Applicant(s)
SKJERVEN, MORRILL, MacPHERSON, FRANKLIN & FRIEL LLP
25 Metro Drive, Suite 700
San Jose, California 95110-1349

Telephone: 408-453-9200
Facsimile: 408-453-7979

I declare that all statements made herein of my own knowledge are true, all statements made herein on information and belief are believed to be true, and all statements made herein are made with the knowledge that whoever, in any matter within the jurisdiction of the Patent and Trademark Office, knowingly and willfully falsifies, conceals, or covers up by any trick, scheme, or device a material fact, or makes any false, fictitious or fraudulent statements or representations, or makes or uses any false writing or document knowing the same to contain any false, fictitious or fraudulent statement or entry, shall be subject to the penalties including fine or imprisonment or both as set forth under 18 U.S.C. 1001, and that violations of this paragraph may jeopardize the validity of the application or this document, or the validity or enforceability of any patent, trademark registration, or certificate resulting therefrom.

Atty. Docket No.: M-6011 US

Full name of first inventor:	Danny Lange		
Inventor's Signature:	[illegible]	Date:	10-22-98
Residence:	Cupertino, California		
Post Office Address:	11626 Cedar Spring Court Cupertino, California 95014	Citizenship:	Denmark
Full name of second inventor:	Barbara Nelson		
Inventor's Signature:	Barbara Nelson	Date:	10/22/98
Residence:	San Mateo, California		
Post Office Address:	2825 Juniper Street San Mateo, California 94403	Citizenship:	Ireland
Full name of third inventor:	Jing Su		
Inventor's Signature:	[illegible]	Date:	10/22/98
Residence:	Cupertino, California		
Post Office Address:	20094 Wheaton Drive Cupertino, California 95014	Citizenship:	China
Full name of fourth inventor:	James E. White		
Inventor's Signature:	[illegible]	Date:	10/22/98
Residence:	San Carlos, California		
Post Office Address:	112 Wycombe Avenue San Carlos, California 94070	Citizenship:	U.S.A.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Confirmation No.: 4612
Lange *et al.*	Art Unit: 2142
Appl. No.: 09/712,712	Examiner: Geckil, Mehmet B.
Filed: November 14, 2000	Atty. Docket: 2222.0300001
For: **Network System Extensible by Users**	

Revocation of Prior Power of Attorney and Appointment of New Attorneys of Record

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Sir:

The undersigned, having express authority to represent Ben Franklin Patent Holding L.L.C., the assignee of the entire right, title, and interest in the above-captioned application, by assignment filed at the U.S. Patent and Trademark Office on April 21, 2004 (copy enclosed), hereby revokes all powers of attorney heretofore given in the above-captioned application and appoints as his attorneys the attorneys associated with Customer Number 28393, those attorneys currently being: Robert Greene Sterne, Esq., Registration No. 28,912; Edward J. Kessler, Esq., Registration No. 25,688; Jorge A. Goldstein, Esq., Registration No. 29,021; David K.S. Cornwell, Esq., Registration No. 31,944; Robert W. Esmond, Esq., Registration No. 32,893; Tracy-Gene G. Durkin, Esq., Registration No. 32,831; Michele A. Cimbala, Esq., Registration No. 33,851; Michael B. Ray, Esq., Registration No. 33,997; Robert E. Sokohl, Esq., Registration No. 36,013; Eric K. Steffe, Esq., Registration No. 36,688; Michael Q. Lee, Esq., Registration No. 35,239; Steven R. Ludwig, Esq., Registration No. 36,203; John M. Covert, Esq., Registration No. 38,759; Linda E. Alcorn, Esq., Registration No. 39,588; Lawrence B. Bugaisky, Esq., Registration No. 35,086; Donald J. Featherstone, Esq., Registration No.

33,876; Robert C. Millonig, Esq., Registration No. 34,395; Michael V. Messinger, Esq., Registration No. 37,575; Judith U. Kim, Esq., Registration No. 40,679; Timothy J. Shea, Jr., Esq., Registration No. 41,306; Patrick E. Garrett, Esq., Registration No. 39,987; with full power of substitution, association, and revocation, to prosecute said application and to transact all business in the United States Patent and Trademark Office connected therewith.

For the purpose of PAIR, the Customer Number is **26111**.

The undersigned hereby grants said attorneys the power to insert on this Power of Attorney any further identification that may be necessary or desirable in order to comply with the rules of the U.S. Patent and Trademark Office.

Send all correspondence to:

Customer Number 26111
STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.
1100 New York Avenue, N.W.
Washington, D.C. 20005-3934.

Direct telephone calls to (202) 371-2600.

FOR: Ben Franklin Patent Holding L.L.C.

SIGNATURE: [illegible]

BY: PETER DETKIN

TITLE: MANAGING DIRECTOR

DATE: 26 APRIL 2004

252194.1

BEST AVAILABLE COPY

State of Delaware
Secretary of State
Division of Corporations
Delivered 11:23 AM 11/24/2003
FILED 10:41 AM 11/24/2003
SRV 030753952 - 3649814 FILE

CERTIFICATE OF AMENDMENT
OF
INTELLECTUAL VENTURES PATENT HOLDING I, L.L.C.

The undersigned, being duly authorized to execute and file this Certificate of Amendment, does hereby certify as follows:

1. The name of the limited liability company is Intellectual Ventures Patent Holding I, L.L.C.

2. Paragraph 1 of the Certificate of Formation is amended in its entirety to read as follow:

"1. *Name*. The name of the limited liability company is Ben Franklin Patent Holding LLC."

IN WITNESS WHEREOF, the undersigned has duly executed this Certificate of Amendment on the 18th day of November, 2003.

Acquisition Management LLC, Manager

By: [illegible]
Gregory Gorder, Managing Director

{98033100245.DOC} 11/19/2003

BEST AVAILABLE COPY

PATENT APPLICATION SERIAL NO. ____________________

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

```
11/26/2004 SMINASS1 00000056 10995159

01 FC:1001                 790.00 OP
02 FC:1202                  18.00 OP
03 FC:1201                  88.00 OP
```

PTO-1556
(5/87)

*U.S. Government Printing Office: 2002 — 489-267/69033

PATENT APPLICATION FEE DETERMINATION RECORD
Effective October 1, 2004

Application or Docket Number

10995159

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS	21	
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	21 minus 20=	* 1
INDEPENDENT CLAIMS	4 minus 3 =	* 1
MULTIPLE DEPENDENT CLAIM PRESENT		☐

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE ☐ OR OTHER THAN SMALL ENTITY

RATE	FEE		RATE	FEE
BASIC FEE	395.00	OR	BASIC FEE	790.00
X$ 9=		OR	X$18=	18
X44=		OR	X88=	88
+150=		OR	+300=	
TOTAL		OR	TOTAL	996

CLAIMS AS AMENDED - PART II

AMENDMENT A

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

SMALL ENTITY OR OTHER THAN SMALL ENTITY

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X44=		OR	X88=	
+150=		OR	+300=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X44=		OR	X88=	
+150=		OR	+300=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X44=		OR	X88=	
+150=		OR	+300=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
***If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

FORM PTO-875 (Rev. 10/04) Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Confirmation No.: To be assigned
Lange, et al	Art Unit: To be assigned
Appl. No.: To be assigned *(Continuation of Appl. No. 09/712,712; Filed: November 14, 2000)*	Examiner: To be assigned
Filed: November 24, 2004	Atty. Docket: 2222.0300002
For: **Network System Extensible By Users**	

Preliminary Amendment Under 37 C.F.R. § 1.115

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

In advance of prosecution, Applicant submits the following amendments and remarks. This Preliminary Amendment is provided in the following format:

(A) Each section begins on a separate sheet;

(B) Starting on a separate sheet, amendments to the specification by presenting replacement paragraphs marked up to show changes made;

(C) Starting on a separate sheet, a complete listing of all of the claims:

- in ascending order;
- with status identifiers; and
- with markings in the currently amended claims;

(D) Starting on a separate sheet, the Remarks.

It is not believed that extensions of time or fees for net addition of claims are required beyond those that may otherwise be provided for in documents accompanying this paper. However, if additional extensions of time are necessary to prevent abandonment of this application, then such extensions of time are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required therefor (including fees for net addition of claims) are hereby authorized to be charged to our Deposit Account No. 19-0036.

Amendment to the Specification

On Page 1, before line 10, "<u>TECHNICAL FIELD OF THE INVENTION</u>", insert the following:

<u>CROSS REFERENCE TO RELATED APPLICATIONS</u>

The present application is a continuation of U.S. Patent Appl. No. 09/712,712, filed November 14, 2000, allowed, which is a continuation of Appl. No. 09/178,366, filed October 23, 1998, now U.S. Patent No. 6,163,794, each of which is incorporated herein by reference in its entirety.

Amendments to the Claims

1-76. (canceled)

77. (new) A system for performing user customized network-based operations, comprising:

an agent associated with a user, wherein the agent is configured to perform an operation on behalf of the user; and

an agent server coupled to the agent and coupled to the user via a network communications link, wherein the agent server manages the execution of the operation by the agent,

wherein the agent is operable to use a service and a service resource associated with the service when performing the operation on behalf of the user.

78. (new) The system of claim 77, further comprising a service wrapper associated with the service, wherein the service wrapper is configured to mediate the interaction between the service and the agent.

79. (new) The system of claim 77 wherein the network communications link is a communications link in a public-switched communications network.

80. (new) The system of claim 77, wherein the agent server comprises:

an engine, wherein the engine is configured to control the operation of the agent server;

a scheduler coupled to the engine, wherein the scheduler is configured to trigger the execution of the agent upon occurrence of one or more events; and

an agent object coupled to the agent, wherein the agent object includes data and executable instructions associated with the agent.

81. (new) The system of claim 80, wherein the agent object comprises:

a permission associated with the agent; and an event handler, wherein the event handler includes data and executable instructions for directing the operation of the engine upon the occurrence of the one or more events.

82. (new) The system of claim 81, wherein the permission is a computational permission defining one or more computational resources that the agent is permitted to use.

83. (new) The system of claim 82, wherein the computational permission further defines the extent to which the agent is permitted to use the one or more computational resources.

84. (new) The system of claim 81, wherein the permission is a service permission defining one or more services that the agent is permitted to use.

85. (new) The system of claim 84, wherein the service permission further defines the extent to which the agent is permitted to use the one or more services.

86. (new) A system for performing user customized network-based operations, comprising:

means for allowing a user to create a network-based agent associated with the user, wherein the network-based agent is configured to perform an operation on behalf of the user;

means for invoking the execution of the network-based agent on the occurrence of an event;

means for using a service and a service resource when performing the operation on behalf of the user; and

means for communicating the result of the operation to the user over a network communications link.

87. (new) The system of claim 86, wherein the network communications link is a communications link in a public-switched communications network.

88. (new) The system of claim 87, further comprising:

means for mediating the interaction between the means for using the service and the service.

89. (new) The system of claim 88, wherein the means for mediating comprises:

means for monitoring the amount of the service resource used by the network-based agent.

90. (new) The system of claim 89, wherein the means for mediating further comprises:

means for converting between a first messaging protocol used by the network-based agent and a second messaging protocol used by the service.

91. (new) The system of claim 86, further comprising:

means for allowing the user to modify the network-based agent associated with the user.

92. (new) A computer program product comprising a computer useable medium having computer program logic recorded thereon for enabling a processor in a computer system to perform user customized network-based operations, comprising:

means for enabling the processor to allow a user to create a network-based agent associated with the user, wherein the network-based agent is configured to perform an operation on behalf of the user;

means for enabling the processor to invoke the execution of the network-based agent on the occurrence of an event;

means for enabling the processor to use a service and a service resource when performing the operation on behalf of the user; and means for enabling the processor to communicate the result of the operation to the user over a network communication link.

93. (new) The system of claim 92, further comprising:

means for enabling the processor to allow the user to modify the network-based agent associated with the user.

94. (new) A method for performing user customized network-based operations, comprising:

(a) receiving data for creating an agent customized to perform a task for a user upon the occurrence of an event;

(b) creating the agent, wherein the agent has a plurality of executable instructions for performing the task;

(c) executing the agent instructions upon the occurrence of the event, wherein step (c) includes:

(i) providing instructions to a service to define the operations supported by the service required to perform the task,

(ii) receiving a response from the service including parameters required by the agent to complete task, and

(iii) providing an output associated with the task to the user over a network communications link.

95. (new) The method of claim 94, wherein the response received from the service includes data.

96. (new) The method of claim 94, wherein the instructions include a request to access a service resource.

97. (new) The method of claim 94, wherein the network communications link is a communications link in a public-switched communications network.

Remarks

Upon entry of the foregoing amendment, claims 77-97 are pending in the application, with 77, 86, 92, and 94 being the independent claims. Claims 1-76 are sought to be cancelled without prejudice to or disclaimer of the subject matter therein. New claims 77-97 are sought to be added. These changes are believed to introduce no new matter, and their entry is respectfully requested.

Conclusion

Prompt and favorable consideration of this Preliminary Amendment is respectfully requested. Applicant believes the present application is in condition for allowance. If the Examiner believes, for any reason, that personal communication will expedite prosecution of this application, the Examiner is invited to telephone the undersigned at the number provided.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Thomas C. Fiala
Attorney for Applicant
Registration No. 43,610

Date: November 24, 2004

1100 New York Avenue, N.W.
Washington, D.C. 20005-3934
(202) 371-2600

334885.1

APPLICATION DATA SHEET

Electronic Version v14
Stylesheet Version v14.0

Title of Invention	Network System Extensible By Users

Application Type: regular, utility
Attorney Docket Number: 2222.0300002

Correspondence address:
Customer Number: 26111 *26111*

Continuing Data:

This is a Continuation of US application number 09/712,712, filed 2000-11-14.

US application number 09/712,712, filed 2000-11-14 is a Continuation of US application number 09/178,366, filed 1998-10-23.

Inventors Information:

Inventor 1:

Applicant Authority Type:	Inventor
Citizenship:	DK
Given Name:	Danny
Family Name:	LANGE
City of Residence:	Cupertino
State of Residence:	CA
Country of Residence:	US
Address-1 of Mailing Address:	11626 Cedar Spring Court
Address-2 of Mailing Address:	
City of Mailing Address:	Cupertino
State of Mailing Address:	CA
Postal Code of Mailing Address:	95014
Country of Mailing Address:	US

Phone:	
Fax:	
E-mail:	

Inventor 2:

Applicant Authority Type:	Inventor
Citizenship:	IE
Given Name:	Barbara
Family Name:	NELSON
City of Residence:	San Mateo
State of Residence:	CA
Country of Residence:	US
Address-1 of Mailing Address:	2825 Juniper Street
Address-2 of Mailing Address:	
City of Mailing Address:	San Mateo
State of Mailing Address:	CA
Postal Code of Mailing Address:	94403
Country of Mailing Address:	US
Phone:	
Fax:	
E-mail:	

Inventor 3:

Applicant Authority Type:	Inventor
Citizenship:	CN
Given Name:	Jing
Family Name:	SU
City of Residence:	Cupertino
State of Residence:	CA
Country of Residence:	US
Address-1 of Mailing Address:	20094 Wheaton Drive
Address-2 of Mailing Address:	
City of Mailing Address:	Cupertino
State of Mailing Address:	CA
Postal Code of Mailing Address:	95014

Country of Mailing Address:	US
Phone:	
Fax:	
E-mail:	

Inventor 4:

Applicant Authority Type:	Inventor
Citizenship:	US
Given Name:	James
Middle Name:	E.
Family Name:	WHITE
City of Residence:	San Carlos
State of Residence:	CA
Country of Residence:	US
Address-1 of Mailing Address:	112 Wycombe Avenue
Address-2 of Mailing Address:	
City of Mailing Address:	San Carlos
State of Mailing Address:	CA
Postal Code of Mailing Address:	94070
Country of Mailing Address:	US
Phone:	
Fax:	
E-mail:	

Attorney Information:

practitioner(s) at Customer Number:

28393 *28393*

as our attorney(s) or agent(s) to prosecute the application identified above, and to transact all business in the United States Patent and Trademark Office connected therewith.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Confirmation No.: To be assigned
Lange *et al.*	Art Unit: To be assigned
Appl. No.: To be assigned *(Continuation of Appl. No. 09/712,712; Filed: November 14, 2000)*	Examiner: To be assigned
Filed: November 24, 2004	Atty. Docket: 2222.0300002
For: **Network System Extensible By Users**	

Information Disclosure Statement

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Listed on accompanying Form PTO-1449 are documents that may be considered material to the examination of this application, in compliance with the duty of disclosure requirements of 37 C.F.R. §§ 1.56, 1.97 and 1.98.

Where the publication date of a listed document does not provide a month of publication, the year of publication of the listed document is sufficiently earlier than the effective U.S. filing date and any foreign priority date so that the month of publication is not in issue. Applicants have listed publication dates on the attached PTO-1449 based on information presently available to the undersigned. However, the listed publication dates should not be construed as an admission that the information was actually published on the date indicated.

Applicants reserve the right to establish the patentability of the claimed invention over any of the information provided herewith, and/or to prove that this information may not be prior art, and/or to prove that this information may not be enabling for the teachings purportedly offered.

This statement should not be construed as a representation that a search has been made, or that information more material to the examination of the present patent application does not exist. The Examiner is specifically requested not to rely solely on the material submitted herewith.

Applicants have checked the appropriate boxes below.

☐ 1. Statement under 37 C.F.R. 1.704(d). Each item of information contained in this Information Disclosure Statement was cited in a communication from a foreign patent office in a counterpart application and this communication was not received by any individual designated in 37 C.F.R. § 1.56(c) more than thirty days prior to the filing of this information disclosure statement.

☒ 2. Filing under 37 C.F.R. § 1.97(b). This Information Disclosure Statement is being filed within three months of the date of filing of a national application other than a continued prosecution application (CPA), OR within three months of the date of entry of the national stage as set forth in 37 C.F.R. § 1.491 in an international application, OR before the mailing date of a first Office Action on the merits OR before the mailing of a first Office Action after the filing of a request for continued examination under 37 C.F.R. § 1.114. No statement or fee is required.

☐ 3. Filing under 37 C.F.R. § 1.97(c). This Information Disclosure Statement is being filed more than three months after the U.S. filing date AND after the mailing date of the first Office Action on the merits, but before the mailing date of a Final Rejection, or Notice of Allowance, or an action that otherwise closes prosecution in the application.

  ☐ a. Statement under 37 C.F.R. § 1.97(e)(1). I hereby state that each item of information contained in this Information Disclosure Statement was first cited in any communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this Information Disclosure Statement. 37 C.F.R. § 1.97(e)(1).

  ☐ b. Statement under 37 C.F.R. § 1.97(e)(2). I hereby state that no item of information in this Information Disclosure Statement was cited in a communication from a foreign patent office in a counterpart foreign application and, to my knowledge after making reasonable inquiry, was known to any individual designated in 37 C.F.R. § 1.56(c) more than three months prior to the filing of this Information Disclosure Statement. 37 C.F.R. § 1.97(e)(2).

☐ c. Attached is our PTO-2038 Credit Card Payment Form in the amount of $________ in payment of the fee under 37 C.F.R. § 1.17(p).

☐ 4. Filing under 37 C.F.R. § 1.97(d) This Information Disclosure Statement is being filed more than three months after the U.S. filing date and after the mailing date of a Final Rejection or Notice of Allowance, but before payment of the Issue Fee. Enclosed find our PTO-2038 Credit Card Payment Form in the amount of $_________ in payment of the fee under 37 C.F.R. § 1.17(p); in addition:

☐ a. Statement under 37 C.F.R. § 1.97(e)(1). I hereby state that each item of information contained in this Information Disclosure Statement was cited in a communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this Information Disclosure Statement. 37 C.F.R. § 1.97(e)(1).

☐ b. Statement under 37 C.F.R. § 1.97(e)(2). I hereby state that no item of information in this Information Disclosure Statement was cited in a communication from a foreign patent office in a counterpart foreign application and, to my knowledge after making reasonable inquiry, was known to any individual designated in 37 C.F.R. § 1.56(c) more than three months prior to the filing of this Information Disclosure Statement. 37 C.F.R. § 1.97(e)(2).

☐ 5. The document(s) was/were cited in a search report by a foreign patent office in a counterpart foreign application. Submission of an English language version of the search report that indicates the degree of relevance found by the foreign office is provided in satisfaction of the requirement for a concise explanation of relevance. 1138 OG 37, 38.

☐ 6. A concise explanation of the relevance of the non-English language document(s) appears below in accordance with 37 C.F.R. § 1.98(a)(3).

☐ 7. Copies of documents are submitted. However, in accordance with 37 C.F.R. § 1.98(a)(2), no copies of U.S. patents and patent application publications cited on the attached Form PTO-1449 are submitted.

☒ 8. Copies of the documents were cited by or submitted to the Office in an IDS that complies with 37 C.F.R. § 1.98(a)-(c) in Application No. 09/712,712, filed November 14, 2000, which is relied upon for an earlier filing date under 35 U.S.C. § 120. Thus, copies of these documents are not attached. 37 C.F.R. § 1.98(d).

☒ 9. It is expected that the examiner will review the prosecution and cited art in the parent application no(s). 09/712,712 in accordance with MPEP 2001.06(b), and indicate in the next communication from the office that the art cited in the earlier prosecution history has been reviewed in connection with the present application.

It is respectfully requested that the Examiner initial and return a copy of the enclosed Form PTO-1449, and indicate in the official file wrapper of this patent application that the documents have been considered.

The U.S. Patent and Trademark Office is hereby authorized to charge any fee deficiency, or credit any overpayment, to our Deposit Account No. 19-0036.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Thomas C. Fiala
Attorney for Applicants
Registration No. 43,610

Date: November 24, 2004

1100 New York Avenue, N.W.
Washington, D.C. 20005-3934
(202) 371-2600

337874.1

FORM PTO-1449 INFORMATION DISCLOSURE STATEMENT	ATTY. DOCKET NO. 2222.0300002	APPLICATION NO. To be assigned
	FIRST NAMED INVENTOR Danny Lange	
	FILING DATE November 24, 2004	ART UNIT To be assigned

U.S. PATENT DOCUMENTS

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE
	AA1	6,657,990	12/2003	Dilip et al.	370	352	
	AB1	6,457,063	09/2002	Chintalapati et al.	709	317	
	AC1	6,363,411	03/2002	Dugan et al.	709	202	
	AD1	6,285,977	09/2001	Miyazaki, Kazuya	703	26	
	AE1	6,163,794	12/2000	Lange et al.	709	202	
	AF1	5,825,759	10/1998	Liu, George	370	331	
	AG1	6,067,568	05/23/2000	Li et al.	709	223	
	AH1	6,016,520	01/18/2000	Facq et al.	710	33	
	AI1	5,983,267	11/09/1999	Shklar et al.	709	217	
	AJ1	5,983,190	11/09/1999	Trower II, et al.	704	276	
	AK1	5,974,441	10/26/1999	Rogers et al.	709	200	

FOREIGN PATENT DOCUMENTS

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUB-CLASS	TRANSLATION
	AL1						Yes No
	AM1						Yes No
	AN1						Yes No
	AO1						Yes No
	AP1						Yes No

OTHER (Including Author, Title, Date, Pertinent Pages, etc.)

	AR	1	Jonathan Dale, "A Mobile Agent Architecture to Support Distributed Resource Information Management", University of Southampton, Department of Electronics and Computer Science, 79 pages, 23 June 1998.
	AS	1	JP Morgenthal, "XML Agents," NC.Focus website (www.ncfocus.com), 1998, pages 1-4.
	AT	1	D. Tsichritzis, et al., "KNOs: Knowledge Acquisition, Dissemination, and Manipulation Objects," ACM Transactions on Office Information Systems, Vol. 5, No. 1, January 1987, pages 96-112.

EXAMINER	DATE CONSIDERED

EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to Applicant.

FORM PTO-1449	ATTY. DOCKET NO.	APPLICATION NO. To be assigned
INFORMATION DISCLOSURE STATEMENT	FIRST NAMED INVENTOR Danny Lange	
	FILING DATE November 24, 2004	ART UNIT To be assigned

U.S. PATENT DOCUMENTS

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE
	AA2	5,963,949	10/05/1999	Gupta et al.	707	100	
	AB2	5,913,214	06/15/1999	Madnick et al.	707	10	
	AC2	5,826,258	10/20/1998	Gupta et al.	707	4	
	AD2	5,665,081	08/05/1997	Bonnell et al.	709	202	
	AE2	6,016,393	01/18/2000	White et al.	395	6983	
	AF2	5,953,392	09/14/1999	Rhie et al.	379	8813	
	AG2	5,603,031	02/11/1997	White et al.	395	683	
	AH2						
	AI2						
	AJ2						
	AK2						

FOREIGN PATENT DOCUMENTS

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUB-CLASS	TRANSLATION
	AL2						Yes No
	AM2						Yes No
	AN2						Yes No
	AO2						Yes No
	AP2						Yes No

OTHER (Including Author, Title, Date, Pertinent Pages, etc.)

	AR	2	C. Daniel Wolfson, et al., "Intelligent Routers," The 9th International Conference on Distributed Computing Systems, IEEE Computer Society Press, 1989, pages 371-376.
	AS	2	
	AT	2	

EXAMINER	DATE CONSIDERED

EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to Applicant.

IFW

Sterne Kessler Goldstein Fox
ATTORNEYS AT LAW

February 9, 2005

WRITER'S DIRECT NUMBER:
(202) 772-8862
INTERNET ADDRESS:
LGORDON@SKGF.COM

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Art Unit 2642

Re: U.S. Utility Patent Application
Application No. 10/995,159; Filed: November 24, 2004
For: **Network System Extensible By Users**
Inventors: LANGE *et al.*
Our Ref: 2222.0300002

Sir:

Transmitted herewith for appropriate action are the following documents:

1. Request For Corrected Official Filing Receipt;

2. Copy of Filing Receipt with changes marked in red;

3. Copy of PTO date stamped receipt card; and

4. ONE (1) return postcard.

It is respectfully requested that the attached postcard be stamped with the date of filing of these documents, and that it be returned to our courier. In the event that extensions of time are necessary to prevent abandonment of this patent application, then such extensions of time are hereby petitioned.

The U.S. Patent and Trademark Office is hereby authorized to charge any fee deficiency, or credit any overpayment, to our Deposit Account No. 19-0036.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Lori A. Gordon
Attorney for Applicants
Registration No. 50,633

TCF/LAG/mjg
Enclosure

O I P E
FEB 0 9 2005
JC25
PATENT & TRADEMARK OFFICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Confirmation No.: 5640
LANGE *et al.*	Art Unit: 2642
Appl. No.: 10/995,159	Examiner: To be assigned
Filed: November 24, 2004	Atty. Docket: 2222.0300002
For: **Network System Extensible By Users**	

Request for Corrected Official Filing Receipt

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Attn: Office of Initial Patent Examination's Filing Receipt Corrections

Sir:

Applicants hereby request that a corrected Official Filing Receipt be issued and sent to the undersigned representative. Specifically, the following corrections to the Official Filing Receipt are requested:

In the section Assignment For Published Patent Application, please add:

Ben Franklin Patent Holding L.L.C..

In support of the above request, a photocopy of the instant Official Filing Receipt is enclosed with the corrections noted in red. Additionally, a copy of the PTO date stamped post card reflecting that a change of name document was filed is enclosed.

It is requested that a corrected Official Filing Receipt be issued, and sent to the undersigned at the earliest possible time.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

Lori A. Gordon

Lori A. Gordon
Attorney for Applicants
Registration No. 50,633

Date: 2/9/05

1100 New York Avenue, N.W.
Washington, D.C. 20005-3934
(202) 371-2600

OIPE JC25 FEB 09 2005 PATENT & TRADEMARK OFFICE

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPL NO.	FILING OR 371 (c) DATE	ART UNIT	FIL FEE REC'D	ATTY.DOCKET NO	DRAWINGS	TOT CLMS	IND CLMS
10/995,159	11/24/2004	2642	896	2222.0300002	17	21	4

CONFIRMATION NO. 5640

FILING RECEIPT

OC000000014993349

26111
STERNE, KESSLER, GOLDSTEIN & FOX PLLC
1100 NEW YORK AVENUE, N.W.
WASHINGTON, DC 20005

Date Mailed: 01/21/2005

Receipt is acknowledged of this regular Patent Application. It will be considered in its order and you will be notified as to the results of the examination. Be sure to provide the U.S. APPLICATION NUMBER, FILING DATE, NAME OF APPLICANT, and TITLE OF INVENTION when inquiring about this application. Fees transmitted by check or draft are subject to collection. Please verify the accuracy of the data presented on this receipt. **If an error is noted on this Filing Receipt, please write to the Office of Initial Patent Examination's Filing Receipt Corrections, facsimile number 703-746-9195. Please provide a copy of this Filing Receipt with the changes noted thereon. If you received a "Notice to File Missing Parts" for this application, please submit any corrections to this Filing Receipt with your reply to the Notice. When the USPTO processes the reply to the Notice, the USPTO will generate another Filing Receipt incorporating the requested corrections (if appropriate).**

Applicant(s)

Danny Lange, Cupertino, CA;
Barbara Nelson, San Mateo, CA;
Jing Su, Cupertino, CA;
James E. White, San Carlos, CA;

JAN 24 2005

Assignment For Published Patent Application

General Magic
Intellectual Ventures Patent Holding I, L.L.C.
Ben Franklin Patent Holding L.L.C. LAG

[illegible] TCF
[illegible] 1/25
m 1/26

Power of Attorney: The patent practitioners associated with Customer Number 28393.

Domestic Priority data as claimed by applicant

This application is a CON of 09/712,712 11/14/2000 PAT 6,839,733
which is a CON of 09/178,366 10/23/1998 PAT 6,163,794

Foreign Applications

If Required, Foreign Filing License Granted: 01/18/2005

The country code and number of your priority application, to be used for filing abroad under the Paris Convention, is US10/995,159

Projected Publication Date: 04/28/2005

Non-Publication Request: No

Early Publication Request: No

Title

Network system extensible by users

Preliminary Class

379

LICENSE FOR FOREIGN FILING UNDER
Title 35, United States Code, Section 184
Title 37, Code of Federal Regulations, 5.11 & 5.15

GRANTED

The applicant has been granted a license under 35 U.S.C. 184, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" followed by a date appears on this form. Such licenses are issued in all applications where the conditions for issuance of a license have been met, regardless of whether or not a license may be required as set forth in 37 CFR 5.15. The scope and limitations of this license are set forth in 37 CFR 5.15(a) unless an earlier license has been issued under 37 CFR 5.15(b). The license is subject to revocation upon written notification. The date indicated is the effective date of the license, unless an earlier license of similar scope has been granted under 37 CFR 5.13 or 5.14.

This license is to be retained by the licensee and may be used at any time on or after the effective date thereof unless it is revoked. This license is automatically transferred to any related applications(s) filed under 37 CFR 1.53(d). This license is not retroactive.

The grant of a license does not in any way lessen the responsibility of a licensee for the security of the subject matter as imposed by any Government contract or the provisions of existing laws relating to espionage and the national security or the export of technical data. Licensees should apprise themselves of current regulations especially with respect to certain countries, of other agencies, particularly the Office of Defense Trade Controls, Department of State (with respect to Arms, Munitions and Implements of War (22 CFR 121-128)); the Office of Export Administration, Department of Commerce (15 CFR 370.10 (j)); the Office of Foreign Assets Control, Department of Treasury (31 CFR Parts 500+) and the Department of Energy.

NOT GRANTED

No license under 35 U.S.C. 184 has been granted at this time, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" DOES NOT appear on this form. Applicant may still petition for a license under 37 CFR 5.12, if a license is desired before the expiration of 6 months from the filing date of the application. If 6 months has lapsed from the filing date of this application and the licensee has not received any indication of a secrecy order under 35 U.S.C. 181, the licensee may foreign file the application pursuant to 37 CFR 5.15(b).

OIPE JC25
FEB 0 9 2005
PATENT & TRADEMARK OFFICE

		Due Date:	NONE
Applicants:	Lange *et al.*	**Art Unit:**	To be assigned
		Confirmation No.:	To be assigned
Application No.:	To be assigned *(Continuation of Appl. No. 09/712,712; Filed: November 14, 2000)*	**Examiner:**	To be assigned
Filed:	November 24, 2004	**Docket:**	2222.0300002
For:	**Network System Extensible By Users**	**Atty:**	TCF/LAG

OIPE JC25
NOV 2 4 2004
PATENT & TRADEMARK OFFICE

When receipt stamp is placed hereon, the USPTO acknowledges receipt of the following documents:

1. SKGF Cover Letter;
2. Credit Card Payment Form (PTO-2038) for $1,016.00 to cover: $790.00 Patent Application Fee; $106.00 Excess Claims Fee and $120.00 Recordation Fee;
3. Utility Patent Application Transmittal Form (PTO/SB/05);
4. Fee Transmittal Form (PTO/SB/17);
5. Autnorization to Treat a Reply As Incorporating An Extension of Time Under 37 C.F.R. § 1.136(a)(3);
6. U.S. Utility Patent Application entitled: **Network System Extensible By Users**, and naming as inventors: Danny Lange, Barbara Nelson, Jing Su and James E. White, the application consisting of: **a.** An Application Data Sheet (37 C.F.R. § 1.76);
   b. A copy of the executed combined Declaration and Power of Attorney, as originally filed in U.S. Appl. No. 09/178,366;
   c. A specification containing: **i.** 86 pages of description prior to the claims; **ii.** 18 pages of claims (76 claims); **iii.** a one (1) page abstract; **d.** 17 sheets of drawings (Figures 1-17);
7. A Preliminary Amendment;
8. A copy of the Revocation of Prior Power of Attorney and Appointment of New Attorneys of Record, as originally filed in U.S. Appl. No. 9/712,712;
9. Recordation Form Cover Sheet (Form PTO-1595); **10.** A copy of the executed Assignment to General Magic, recordation of which is hereby respectfully requested; **11.** Recordation Form Cover Sheet (Form PTO-1595); **12.** A copy of the executed Assignment to Intellectual Ventures Patent Holding I, L.L.C., recordation of which is hereby respectfully requested; **13.** Recordation Form Cover Sheet (Form PTO-1595); **14.** A copy of the executed Change of Name to Ben Franklin Patent Holding L.L.C., recordation of which is hereby respectfully requested; **15.** An Information Disclosure Statement; **16.** Form PTO-1449 (2 sheets citing 22 documents); and **17.** Two (2) return postcards.

Please Date Stamp And Return To Our Courier

PATENT APPLICATION FEE DETERMINATION RECORD
Effective October 1, 2004

Application or Docket Number: 10995159

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS	21	
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	21 minus 20=	* 1
INDEPENDENT CLAIMS	4 minus 3 =	* 1
MULTIPLE DEPENDENT CLAIM PRESENT		☐

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE ☐ OR OTHER THAN SMALL ENTITY

RATE	FEE		RATE	FEE
BASIC FEE	395.00	OR	BASIC FEE	790.00
X$ 9=		OR	X$18=	18
X44=		OR	X88=	88
+150=		OR	+300=	
TOTAL		OR	TOTAL	996

CLAIMS AS AMENDED - PART II

AMENDMENT A

11/24/04	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	* 21	Minus	** 21	= —
Independent	* 4	Minus	*** 4	= —
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

SMALL ENTITY OR OTHER THAN SMALL ENTITY

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=	\	OR	X$18=	
X44=	\	OR	X88=	\
+150=	\	OR	+300=	\
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X44=		OR	X88=	
+150=		OR	+300=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X44=		OR	X88=	
+150=		OR	+300=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
***If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

FORM PTO-875 (Rev. 10/04) Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE

January 10, 2007

WRITER'S DIRECT NUMBER:
(202) 772-8862
INTERNET ADDRESS:
LGORDON@SKGF.COM

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Art Unit 2642

Re: U.S. Utility Patent Application
Application No. 10/995,159; Filed: November 24, 2004
For: **Network System Extensible By Users**
Inventors: LANGE *et al.*
Our Ref: 2222.0300002

Sir:

Transmitted herewith for appropriate action are the following documents:

1. Power of Attorney to Prosecute Applications Before the USPTO; and

2. Statement Under 37 CFR 3.73(b). The Assignment document was, or concurrently is, submitted for recordation pursuant to § 3.11;

The above-listed documents are filed electronically through EFS-Web.

In the event that extensions of time are necessary to prevent abandonment of this patent application, then such extensions of time are hereby petitioned.

The U.S. Patent and Trademark Office is hereby authorized to charge any fee deficiency, or credit any overpayment, to our Deposit Account No. 19-0036.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Lori A. Gordon
Attorney for Applicants
Registration No. 50,633

MVM/LAG:smn
Enclosure(s)

613539_1.DOC

POWER OF ATTORNEY TO PROSECUTE APPLICATIONS BEFORE THE USPTO

I hereby appoint:

[✓ X] Practitioners associated with the Customer Number: 26111

OR

[ ] Practitioner(s) named below (if more than ten patent practitioners are to be named, then a customer number must be used):

Name	Registration Number

as attorney(s) or agent(s) to represent the undersigned before the United States Patent and Trademark Office (USPTO) in connection with any and all patent applications assigned only to the undersigned according to the USPTO assignment records or assignment documents attached to this form in accordance with 37 CFR 3.73(b).

Assignee Name and Address:

Ben Franklin Patent Holding LLC
171 Main Street, #271
Los Altos, CA 94022

A copy of this form, together with a statement under 37 CFR 3.73(b) (Form PTO/SB/96 or equivalent) is required to be filed in each application in which this form is used. The statement under 37 CFR 3.73(b) may be completed by one of the practitioners appointed in this form if the appointed practitioner is authorized to act on behalf of the assignee, and must identify the application in which this Power of Attorney is to be filed.

SIGNATURE of Assignee of Record
The individual whose signature and title is supplied below is authorized to act on behalf of the assignee

Name	JULIA UFFALO		
Signature	[illegible]	Date	29 JULY 2004
Title	AUTHORIZED PERSON	Telephone	

STATEMENT UNDER 37 CFR 3.73(b)

2222.0300002

Applicant/Patent Owner: LANGE et al.

Application No./Patent No.: 10/995,159 Filed/Issue Date: November 24, 2004

Entitled: Network System Extensible By Users

Ben Franklin Patent Holding LLC, a Corporation
(Name of Assignee) (Type of Assignee, e.g., corporation, partnership, university, government agency, etc.)

states that it is:

1. [X] the assignee of the entire right, title, and interest; or
2. [ ] an assignee of less than the entire right, title and interest
(The extent (by percentage) of its ownership interest is _________%)

in the patent application/patent identified above by virtue of either:

A. [ ] An assignment from the inventor(s) of the patent application/patent identified above. The assignment was recorded in the United States Patent and Trademark Office at Reel ___________, Frame __________, or for which a copy thereof is attached.

OR

B. [X] A chain of title from the inventor(s), of the patent application/patent identified above, to the current assignee as follows:

1. From: LANGE et al. To: General Magic
The document was recorded in the United States Patent and Trademark Office at Reel 016028, Frame 0732, or for which a copy thereof is attached.

2. From: General Magic, Inc. To: Intellectual Ventures Patent Holding I, LLC
The document was recorded in the United States Patent and Trademark Office at Reel 016028, Frame 0725, or for which a copy thereof is attached.

3. From: Intellectual Ventures Patent Holding I, L.L.C. To: Ben Franklin Patent Holding L.L.C.
The document was recorded in the United States Patent and Trademark Office at Reel 016028, Frame 0730, or for which a copy thereof is attached.

[ ] Additional documents in the chain of title are listed on a supplemental sheet.

[X] As required by 37 CFR 3.73(b)(1)(i), the documentary evidence of the chain of title from the original owner to the assignee was, or concurrently is being, submitted for recordation pursuant to 37 CFR 3.11.
[NOTE: A separate copy (*i.e.*, a true copy of the original assignment document(s)) must be submitted to Assignment Division in accordance with 37 CFR Part 3, to record the assignment in the records of the USPTO. See MPEP 302.08]

The undersigned (whose title is supplied below) is authorized to act on behalf of the assignee.

[illegible]	12/29/06
Signature	Date
Thomas C. Fiala	202-371-2600
Printed or Typed Name	Telephone Number
Attorney	
Title	

613,533

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Electronic Acknowledgement Receipt	
EFS ID:	1428664
Application Number:	10995159
International Application Number:	
Confirmation Number:	5640
Title of Invention:	Network system extensible by users
First Named Inventor/Applicant Name:	Danny Lange
Customer Number:	26111
Filer:	Lori Ann Gordon
Filer Authorized By:	
Attorney Docket Number:	2222.0300002
Receipt Date:	10-JAN-2007
Filing Date:	24-NOV-2004
Time Stamp:	16:26:43
Application Type:	Utility

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)	Multi Part /.zip	Pages (if appl.)
1		22220300002.pdf	158762	yes	4

	Multipart Description/PDF files in .zip description		
	Document Description	**Start**	**End**
	Miscellaneous Incoming Letter	1	2
	Power of Attorney	3	3
	Assignee showing of ownership per 37 CFR 3.73(b).	4	4

Warnings:

Information:

Total Files Size (in bytes):	158762

Electronic Acknowledgement Receipt

EFS ID:	2814394
Application Number:	10995159
International Application Number:	
Confirmation Number:	5640
Title of Invention:	Network system extensible by users
First Named Inventor/Applicant Name:	Danny Lange
Customer Number:	26111
Filer:	Lori Ann Gordon
Filer Authorized By:	
Attorney Docket Number:	2222.0300002
Receipt Date:	05-FEB-2008
Filing Date:	24-NOV-2004
Time Stamp:	17:27:22
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes) /Message Digest	Multi Part /.zip	Pages (if appl.)
1	NPL Documents	NPL_33.pdf	842321 c1bba409d8ba32266e701d1c0727c5152df6f009	no	11

Warnings:

Information:

2	NPL Documents	NPL_34.pdf	200053 bedeff39e79dc249e4f0020b234ca77daad5f253	no	13
Warnings:					
Information:					
3	NPL Documents	NPL_35.pdf	1047668 1526d919653c60cfbaf7d9b17a5257df12121720	no	11
Warnings:					
Information:					
4	NPL Documents	NPL_37.pdf	530159 0b833bcde267fdebd6dab241588f851094e4cef1	no	4
Warnings:					
Information:					
5	NPL Documents	NPL_38.pdf	551500 dc83704070831c7320d89d563480e066a7b55028	no	8
Warnings:					
Information:					
6	NPL Documents	NPL_39.pdf	850516 6aee93b864337bbdc3c34ce34c53784597c3676c	no	8
Warnings:					
Information:					
7	NPL Documents	NPL_40.pdf	167292 485e2710407c58345f2595718f02ba4da9ed62ec	no	2
Warnings:					
Information:					
8	NPL Documents	NPL_41.pdf	67723 766e03876579f17b87ae98806be7704265ee726f	no	3
Warnings:					
Information:					
9	NPL Documents	NPL_42.pdf	800642 d0bff02fe94bad8df28834e89137e30293a5dbf1	no	8
Warnings:					
Information:					
10	NPL Documents	NPL_43.pdf	875540 7460017720c066b655c0493653b3a45e0d17fcc1	no	20
Warnings:					
Information:					

11	NPL Documents	NPL_44.pdf	78284 5f1c04c25c768503c36cdfae2b3037cba0c0c874	no	2
Warnings:					
Information:					
12	NPL Documents	NPL_45.pdf	40373 c8fa1cfea0c48496b69cae532d447504eff98cfe	no	1
Warnings:					
Information:					
13	NPL Documents	NPL_46.pdf	69681 19169a9898017d6015bfb1d33104bf90c28ec6b8	no	1
Warnings:					
Information:					
14	NPL Documents	NPL_47.pdf	203588 c15a084309073a7a2645cbd69e7f42877b2e6bc9	no	3
Warnings:					
Information:					
15	NPL Documents	NPL_36.pdf	2109657 8b1360fc8e8e93e49fdd0d749fcd3ed96111f844	no	59
Warnings:					
Information:					
			Total Files Size (in bytes):	8434997	

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

<u>New Applications Under 35 U.S.C. 111</u>
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

<u>National Stage of an International Application under 35 U.S.C. 371</u>
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

<u>New International Application Filed with the USPTO as a Receiving Office</u>
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

Electronic Acknowledgement Receipt

EFS ID:	2814238
Application Number:	10995159
International Application Number:	
Confirmation Number:	5640
Title of Invention:	Network system extensible by users
First Named Inventor/Applicant Name:	Danny Lange
Customer Number:	26111
Filer:	Lori Ann Gordon
Filer Authorized By:	
Attorney Docket Number:	2222.0300002
Receipt Date:	05-FEB-2008
Filing Date:	24-NOV-2004
Time Stamp:	17:28:05
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes) /Message Digest	Multi Part /.zip	Pages (if appl.)
1	NPL Documents	NPL_13.pdf	1011028 7ad6ba56c835924aa28e213046f98576 72bb1b7d	no	13

Warnings:

Information:

2	NPL Documents	NPL_14.pdf	301356 2a112470acecc122ac155a12a81f03fb7fa59df2	no	7
Warnings:					
Information:					
3	NPL Documents	NPL_15.PDF	9904208 9f85ff23e17535ecefcf605d104e94d6ac0dc4c8	no	92
Warnings:					
Information:					
4	NPL Documents	NPL_16.PDF	4816967 217d5dd8c088f6f1911ab4b6b9ae168b019850a3	no	32
Warnings:					
Information:					
5	NPL Documents	NPL_17.PDF	6248348 300f1eb272e9cd46d1d62ebd2828523b33789a87	no	45
Warnings:					
Information:					
6	NPL Documents	NPL_18.PDF	5290570 1596527c72d10d809ef5044c9439d8e44602499b	no	48
Warnings:					
Information:					
7	NPL Documents	NPL_19.pdf	1436989 01ef81d12ceffadd2429ce435821fb804bfa7700	no	25
Warnings:					
Information:					
8	NPL Documents	NPL_20.PDF	2224931 0aa4969c35a68e7a4d55a408b186e03062e8622b	no	12
Warnings:					
Information:					
9	NPL Documents	NPL_21.PDF	1444434 ad1ca8a41b7618321875bde5bc0442d5aeeae84a	no	10
Warnings:					
Information:					
10	NPL Documents	NPL_22.PDF	2363380 23c973fdafc5686f7b7a7cecad8e0d563c6a6e77	no	21
Warnings:					
Information:					

11	NPL Documents	NPL_23.PDF	1808858 c520a727a5a8ad6a79613a3784b2e73 3f774394c	no	10
Warnings:					
Information:					
12	NPL Documents	NPL_24.PDF	1814764 e1821809d787b246bf90311e7707a5c3 e0caac0c	no	11
Warnings:					
Information:					
13	NPL Documents	NPL_25.PDF	1706533 4099558072a7e42a2f7dc0042e2a6e14 fcaf8a15	no	8
Warnings:					
Information:					
14	NPL Documents	NPL_26.pdf	1301000 cde5941de5593da11b59e42a44bbddcf 1e720590	no	23
Warnings:					
Information:					
15	NPL Documents	NPL_27.pdf	166561 fcf3475eaff9c7764d20d6a5e2efe05180 f18f2b	no	3
Warnings:					
Information:					
16	NPL Documents	NPL_28.pdf	621010 1da5ae5e72e260f4cf92368806d982dd 1e132558	no	6
Warnings:					
Information:					
17	NPL Documents	NPL_29.pdf	256593 17bdf5378a801c69685414756cc4c4e4 d2fe971c	no	3
Warnings:					
Information:					
18	NPL Documents	NPL_30.pdf	1313658 0ff829e80198c3d14232fd9ac292e4435 909f5fe	no	16
Warnings:					
Information:					
19	NPL Documents	NPL_31.pdf	447868 efb1ce09a97ba08a9799cc89d58b47fc8 17100a1	no	4
Warnings:					
Information:					

20	NPL Documents	NPL_32.pdf	171553 a023527ff7bf054f60ea84661731302371efb0a3	no	16
Warnings:					
Information:					
		Total Files Size (in bytes):	44650609		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

February 5, 2008

WRITER'S DIRECT NUMBER:
(202) 772-8862
INTERNET ADDRESS:
LGORDON@SKGF.COM

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Art Unit 2142

Attn: Mail Stop Amendment

Re: U.S. Utility Patent Application
Application No. 10/995,159; Filed: November 24, 2004
For: **Network System Extensible By Users**
Inventors: LANGE *et al.*
Our Ref: 2222.0300002

Sir:

Transmitted herewith for appropriate action are the following documents:

1. First Supplemental Information Disclosure Statement;
2. Form PTO/SB/08A (2 sheets) listing 47 documents (US1-US43 and FP1-FP7);
3. Form PTO/SB/08B (5 sheets) listing 47 documents (NPL1-NPL47); and
4. Copies of cited documents (FP1-FP7 and NPL1-NPL47).

The above-listed documents are filed electronically through EFS-Web.

In the event that extensions of time are necessary to prevent abandonment of this patent application, then such extensions of time are hereby petitioned.

The U.S. Patent and Trademark Office is hereby authorized to charge any fee deficiency, or credit any overpayment, to our Deposit Account No. 19-0036.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Lori A. Gordon
Attorney for Applicants
Registration No. 50,633

LAG/MDW/cs
Enclosures

764581_1.DOC

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Confirmation No.: 5640
LANGE *et al.*	Art Unit: 2142
Appl. No.: 10/995,159	Examiner: Douglas B. Blair
Filed: November 24, 2004	Atty. Docket: 2222.0300002
For: **Network System Extensible By Users**	

First Supplemental Information Disclosure Statement

Mail Stop Amendment

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Sir:

Listed on accompanying IDS Forms are documents that may be considered material to the examination of this application, in compliance with the duty of disclosure requirements of 37 C.F.R. §§ 1.56, 1.97 and 1.98.

Applicants have listed publication dates on the attached IDS Forms based on information presently available to the undersigned. However, the listed publication dates should not be construed as an admission that the information was actually published on the date indicated.

Applicants reserve the right to establish the patentability of the claimed invention over any of the information provided herewith, and/or to prove that this information may not be prior art, and/or to prove that this information may not be enabling for the teachings purportedly offered.

This statement should not be construed as a representation that a search has been made, or that information more material to the examination of the present patent application does not exist. The Examiner is specifically requested not to rely solely on the material submitted herewith.

Applicants have checked the appropriate boxes below.

☐ 1. Statement under 37 C.F.R. 1.704(d). Each item of information contained in this Information Disclosure Statement was first cited in a communication from a foreign patent office in a counterpart application and this communication was not received by any individual designated in 37 C.F.R. § 1.56(c) more than thirty days prior to the filing of this information disclosure statement.

☒ 2. Filing under 37 C.F.R. § 1.97(b). This Information Disclosure Statement is being filed before the mailing date of a first Office Action on the merits. No statement or fee is required.

☐ 3. Filing under 37 C.F.R. § 1.97(c). This Information Disclosure Statement is being filed more than three months after the U.S. filing date AND after the mailing date of the first Office Action on the merits, but before the mailing date of a Final Rejection, or Notice of Allowance, or an action that otherwise closes prosecution in the application.

☐ a. Statement under 37 C.F.R. § 1.97(e)(1). I hereby state that each item of information contained in this Information Disclosure Statement was first cited in any communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this Information Disclosure Statement. 37 C.F.R. § 1.97(e)(1).

☐ b. Statement under 37 C.F.R. § 1.97(e)(2). I hereby state that no item of information in this Information Disclosure Statement was cited in a communication from a foreign patent office in a counterpart foreign application and, to my knowledge after making reasonable inquiry, was known to any individual designated in 37 C.F.R. § 1.56(c) more than three months prior to the filing of this Information Disclosure Statement. 37 C.F.R. § 1.97(e)(2).

☐ c. The required fee is provided through online credit card payment authorization in the amount of $0.00 in payment of the fee under 37 C.F.R. § 1.17(p).

☐ 4. Filing under 37 C.F.R. § 1.97(d) This Information Disclosure Statement is being filed more than three months after the U.S. filing date and after the mailing date of a Final Rejection or Notice of Allowance, but before payment of the Issue Fee. The required fee is provided through online credit card payment authorization in the amount of $0.00 in payment of the fee under 37 C.F.R. § 1.17(p); in addition:

☐ a. Statement under 37 C.F.R. § 1.97(e)(1). I hereby state that each item of information contained in this Information Disclosure Statement was first cited in any communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this Information Disclosure Statement. 37 C.F.R. § 1.97(e)(1).

☐ b. Statement under 37 C.F.R. § 1.97(e)(2). I hereby state that no item of information in this Information Disclosure Statement was cited in a communication from a foreign patent office in a counterpart foreign application and, to my knowledge after making reasonable inquiry, was known to any individual designated in 37 C.F.R. § 1.56(c) more than three months prior to the filing of this Information Disclosure Statement. 37 C.F.R. § 1.97(e)(2).

☐ 5. The document(s) was/were cited in a search report by a foreign patent office in a counterpart foreign application. Submission of an English language version of the search report that indicates the degree of relevance found by the foreign office is provided in satisfaction of the requirement for a concise explanation of relevance. 1138 OG 37, 38.

☐ 6. A concise explanation of the relevance of the non-English language document(s) appears below in accordance with 37 C.F.R. § 1.98(a)(3).

☒ 7. Copies of documents FP1-FP7 and NPL1-NPL47 are submitted. However, in accordance with 37 C.F.R. § 1.98(a)(2), no copies of U.S. patents and patent application publications cited on the attached IDS Forms are submitted.

☐ 8. Copies of the documents were cited by or submitted to the Office in an IDS that complies with 37 C.F.R. § 1.98(a)-(c) in Application No.__________, filed

____________, which is relied upon for an earlier filing date under 35 U.S.C. § 120. Thus, copies of these documents are not attached. 37 C.F.R. § 1.98(d).

☒ 9. It is expected that the examiner will review the prosecution and cited art in the parent application nos. 09/712,712 filed November 14, 2000 and 09/178,366 filed October 23, 1998 in accordance with MPEP 2001.06(b), and indicate in the next communication from the office that the art cited in the earlier prosecution history has been reviewed in connection with the present application.

It is respectfully requested that the Examiner initial and return a copy of the enclosed IDS Forms, and indicate in the official file wrapper of this patent application that the documents have been considered.

The U.S. Patent and Trademark Office is hereby authorized to charge any fee deficiency, or credit any overpayment, to our Deposit Account No. 19-0036.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Lori A. Gordon
Attorney for Applicants
Registration No. 50,633

Date: February 5, 2008

1100 New York Avenue, N.W.
Washington, D.C. 20005-3934
(202) 371-2600

DOC#720063_1.DOC

Atty. Dkt. No. 2222.0300002

Substitute for form 1449/PTO	***Complete if Known***	
FIRST SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT *(Use as many sheets as necessary)*	Application Number	10/995,159
	Filing Date	November 24, 2004
	First Named Inventor	Danny LANGE
	Art Unit	2142
	Examiner Name	Douglas B. Blair
Sheet 1 of 3	Attorney Docket Number	2222.0300002

U.S. PATENT DOCUMENTS

Examiner Initials*	Cite No.[1]	Document Number Number-Kind Code[2] (If Known)	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	US1	4,575,797	03-11-1986	Gruner et al.	
	US2	4,653,100	03-24-1987	Barnett et al.	
	US3	4,716,583	12-01-1987	Groner et al.	
	US4	4,974,254	11-01-1990	Perine et al.	
	US5	5,001,745	03-01-1991	Pollock	
	US6	5,079,695	01-01-1992	Dysart et al.	
	US7	5,093,914	03-01-1992	Coplien et al.	
	US8	5,129,083	07-01-1992	Cutler et al.	
	US9	5,129,084	07-01-1992	Kelly, Jr. et al.	
	US10	5,136,634	08-01-1992	Rae et al.	
	US11	5,187,790	02-01-1993	East et al.	
	US12	5,206,951	04-01-1993	Khoyi et al.	
	US13	5,261,080	11-01-1993	Khoyi et al.	
	US14	5,297,283	03-01-1994	Kelly, Jr. et al.	
	US15	5,303,375	04-01-1994	Collins et al.	
	US16	5,303,379	04-01-1994	Khoyi et al.	
	US17	5,307,490	04-01-1994	Davidson et al.	
	US18	5,321,841	06-01-1994	East et al.	
	US19	5,327,559	07-01-1994	Priven et al.	
	US20	5,339,430	08-01-1994	Lundin et al.	

FOREIGN PATENT DOCUMENTS

Examiner Initials*	Cite No.[1]	Foreign Patent Document Country Code[3] Number[4] Kind Code[5] (if known)	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear	T[6]
	FP1	WO 91/10191 A1	07-11-1991	Aoe et al.		
	FP2	WO 96/11542 A2	04-18-1996	Miner et al.		
	FP3	WO 97/33416 A1	09-12-1997	Taskett		
	FP4	EP 0 495310 A2	07-22-1992	Campbell et al.		
	FP5	EP 0 495319 A2	07-22-1992	Crossland et al.		
	FP6	EP 0 546809 A2	06-16-1993	Conner et al.		
	FP7	EP 0 697780 A2	02-21-1996	Martin et al.		
	FP8					
	FP9					

Examiner Signature		Date Considered	

Substitute for form 1449/PTO	Complete if Known	
FIRST SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT *(Use as many sheets as necessary)*	Application Number	10/995,159
	Filing Date	November 24, 2004
	First Named Inventor	Danny LANGE
	Art Unit	2142
	Examiner Name	Douglas B. Blair
Sheet 2 of 3	Attorney Docket Number	2222.0300002

U.S. PATENT DOCUMENTS

Examiner Initials*	Cite No.[1]	Document Number Number-Kind Code[2] (If Known)	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	US21	5,351,276	09-01-1994	Doll, Jr. et al.	
	US22	5,367,454	11-01-1994	Kawamoto et al.	
	US23	5,377,350	12-01-1994	Skinner	
	US24	5,379,426	01-01-1995	Foss et al.	
	US25	5,396,630	03-01-1995	Banda et al.	
	US26	5,414,852	05-01-1995	Kramer et al.	
	US27	5,421,013	05-01-1995	Smith	
	US28	5,421,015	05-01-1995	Khoyi et al.	
	US29	5,446,842	08-01-1995	Schaeffer et al.	
	US30	5,446,901	08-01-1995	Quicki et al.	
	US31	5,452,433	09-01-1995	Nihart et al.	
	US32	5,500,920	03-01-1996	Kupiec	
	US33	5,546,584	08-13-1996	Lundin et al.	
	US34	5,559,927	09-01-1996	Clynes	
	US35	5,608,786	03-04-1997	Gordon	
	US36	5,636,325	06-01-1997	Farrett	
	US37	5,860,064	01-01-1999	Henton	
	US38	5,873,057	02-01-1999	Eves et al.	
	US39	5,890,123	03-30-1999	Brown et al.	
	US40	5,987,415	11-01-1999	Breese et al.	

FOREIGN PATENT DOCUMENTS

Examiner Initials*	Cite No.[1]	Foreign Patent Document Country Code[3] Number[4] Kind Code[5] (if known)	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear	T[6]
	FP10					
	FP11					
	FP12					
	FP13					
	FP14					
	FP15					
	FP16					
	FP17					
	FP18					

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant. [1]Applicant's unique citation designation number (optional). [2] See Kinds Codes of USPTO Patent Documents at www.uspto.gov or MPEP 901.04. [3] Enter Office that issued the document, by the two-letter code (WIPO Standard ST.3). [4] For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. [5] Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST.16 if possible. [6] Applicant is to place a check mark here if English language Translation is attached.

Substitute for form 1449/PTO

FIRST SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT

(Use as many sheets as necessary)

Sheet 3 of 3

Complete if Known	
Application Number	10/995,159
Filing Date	November 24, 2004
First Named Inventor	Danny LANGE
Art Unit	2142
Examiner Name	Douglas B. Blair
Attorney Docket Number	2222.0300002

U.S. PATENT DOCUMENTS

Examiner Initials*	Cite No.[1]	Document Number Number-Kind Code[2] (If Known)	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	US41	6,144,938	11-07-2000	Surace et al.	
	US42	6,366,650 B1	04-02-2002	Rhie et al.	
	US43	6,839,733 B1	01-04-2005	Lange et al.	
	US44				
	US45				
	US46				
	US47				
	US48				
	US49				
	US50				
	US51				
	US52				
	US53				
	US54				
	US55				
	US56				
	US57				
	US58				
	US59				
	US60				

FOREIGN PATENT DOCUMENTS

Examiner Initials*	Cite No.[1]	Foreign Patent Document Country Code[3] Number[4] Kind Code[5] (if known)	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear	T[6]
	FP19					
	FP20					
	FP21					
	FP22					
	FP23					
	FP24					
	FP25					
	FP26					
	FP27					

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant. [1]Applicant's unique citation designation number (optional). [2] See Kinds Codes of USPTO Patent Documents at www.uspto.gov or MPEP 901.04. [3] Enter Office that issued the document, by the two-letter code (WIPO Standard ST.3). [4] For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. [5] Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST.16 if possible. [6] Applicant is to place a check mark here if English language Translation is attached.

Substitute for form 1449/PTO				*Complete if Known*	
FIRST SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT *(Use as many sheets as necessary)*				Application Number	10/995,159
				Filing Date	November 24, 2004
				First Named Inventor	Danny LANGE
				Art Unit	2142
				Examiner Name	Douglas B. Blair
Sheet	1	of	5	Attorney Docket Number	2222.0300002

NON PATENT LITERATURE DOCUMENTS

Examiner Initials*	Cite No.[1]	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published	T[2]
	NPL1	S Gibbs, "Class Management for Software Communities", Communications Of The Association For Computing Machinery, vol. 33, No. 9, 1 Sep. 1990, pp. 90-103, XP 000162393.	
	NPL2	K. Nielsen, et al., "Inter-Processor Communication and ADA in Distributed Real-Time Systems", Computer Communications, vol. 13, No. 8, 1 Oct. 1990, pp. 451-459, XP 000161020.	
	NPL3	W. Gentleman, et al., "Administrators and Multiprocessor Rendezvous Mechanisms", Software Practice & Experience, vol. 22, No. 1. Jan. 1992 Chichester GB.	
	NPL4	G. Welling, et al. "An Architecture of a Threaded Many-to-Many Remote Procedure Call", Proceedings Of The International Conference On Distributed Compution Systems, Yokohama, Jun. 9-12, 1992 No. Conf. 12, 9 Jun. 1992, Institute Of Electrical And Electronics Engineers, pp. 504-511, XP 000341046.	
	NPL5	U. Ramachandran, et al. "An Implementation of Distributed Shared Memory", Software Practice & Experience, vol. 21, No. 5, 1 May 1991, pp. 443-464, XP 000297178.	
	NPL6	H. Bruggemann, "Rights in an Object-Oriented Environment", Database Security V. Status And Prospects Results Of The IFIP WG 11.3 Workshop, 4 Nov. 1991, Shepherdstown, USA.	
	NPL7	M. Rottman and D. Thompson, "The Amcad Real-Time Multiprocessor Operating System", Proceedings of the IEEE 1989 National Aerospace and Electronics Conference NAECON 1989, pp. 1813-1818, (1989).	
	NPL8	A. Corradi, L. Leonardi and M. Zannini, "Distributed Environments Based on Objects: Upgrading Smalltalk Toward Distribution", Ninth Annual International Phoenix Conference On Computers And Communications, 21-23 Mar. 1990 Conference Proceedings, IEEE Computer Society, pp. 332-339, (1990).	
	NPL9	J. Padget, R. Bradford and J. Fitch, "Concurrent Object-Oriented Programming in LISP", Computer Journal, vol. 34, No. 4, Aug. 1991, pp. 311-319, (1991).	
	NPL10	L. Gunaseelan and R. LeBlanc, Jr., "Distributed Eiffel: A Language for Programming Multi-Granular Distributed Objects on the Clouds Operating System", Proceedings Of the 1992 International Conference on Computer Languages, IEEE Computer Society, pp. 331-340 (1992).	

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.
[1] Applicant's unique citation designation number (optional). [2] Applicant is to place a check mark here if English language Translation is attached.

DOC#719925_1.DOC

Substitute for form 1449/PTO	*Complete if Known*	
FIRST SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT *(Use as many sheets as necessary)*	*Application Number*	10/995,159
	Filing Date	November 24, 2004
	First Named Inventor	Danny LANGE
	Art Unit	2142
	Examiner Name	Douglas B. Blair
Sheet 2 of 5	*Attorney Docket Number*	2222.0300002

NON PATENT LITERATURE DOCUMENTS

Examiner Initials*	Cite No.[1]	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume number, publisher, city and/or country where published	T[2]
	NPL11	K. Ogata, S. Kurihara, M. Inari and N. Doi, "The Design and Implementation of HoME", ACM Sigplan '92 Conference On Programming Language Design And Implementation, San Francisco, CA 17-19 Jun. 1992, ACM Sigplan Notices, vol. 27, No. 7, pp. 44-54, (Jul. 1992).	
	NPL12	James W. Stamos and David K. Gifford, "Remote Evaluation", ACM Transctions on Programming Languages and Systems, vol. 12, No. 4, Oct. 1990, pp. 537-565.	
	NPL13	James W. Stamos and David K. Gifford, "Implementing Remote Evaluation", IEEE Transactions on Software Engineering, vol. 16, No. 7, Jul. 1990, pp. 710-722.	
	NPL14	Casais, Eduardo, "An Object Oriented System Implementing KNOs", Proceedings of the Conference on Office Information Systems, vol. 9, Nos 2-3, pp. 284-290 (1988).	
	NPL15	Kahn, Robert E., and Cerf, Vinton G., "The Digital Library Project: vol. 1: The World of Knowbots"; Corporation of National Research Initiatives (Draft) (1988).	
	NPL16	Borenstein, Nathaniel S., "Secure and Portable Active Messaging: A New Platform for Distributed Applications and Cooperative Work," was to be submitted to Communications of the ACM for publication (date unknown).	
	NPL17	Curtis, Pavel, "LambdaMOO Programmer's Manual", retrieved as /lambda/moo/gamma/ProgrammersManual.texinfo from the Internet network (Aug. 1991).	
	NPL18	Hutchinson, Norman C.; Raj, Rajendra K.; Black, Andrew P.; Levy, Henry M.; and Jul, Eric, "The Emerald Programming Language Report", Technical Report 87-10-07, Department of Computer Science, University of Washington (Oct. 1987).	
	NPL19	Jul, Eric; Levy, Henry; Hutchinson, Norman; and Black, Andrew, "Fine-Grained Mobility in the Emerald System", ACM Transactions on Computer Systems, vol. 6, No. 1, pp. 109-133 (Feb. 1988).	
	NPL20	Rashid, Richard F., and Robertson, George G., "Accent: A Communication Oriented Network Operating System Kernel", ACM document number 0-89791-062-1-12/81-0064, pp. 64-75 (1981).	

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.
[1] Applicant's unique citation designation number (optional). [2] Applicant is to place a check mark here if English language Translation is attached.

DOC#719925_1.DOC

Substitute for form 1449/PTO **FIRST SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT** *(Use as many sheets as necessary)*	***Complete if Known***
	Application Number — 10/995,159
	Filing Date — November 24, 2004
	First Named Inventor — Danny LANGE
	Art Unit — 2142
	Examiner Name — Douglas B. Blair
Sheet 3 of 5	*Attorney Docket Number* — 2222.0300002

NON PATENT LITERATURE DOCUMENTS

Examiner Initials*	Cite No.[1]	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume number, publisher, city and/or country where published	T[2]
	NPL21	Butterfield, David A., and Popek, Gerald J., "Network Tasking in the Locus Distributed Unix System", Locus Computing Corporation, Santa Monica, California, pp. 62-71 (date unknown).	
	NPL22	Douglis, Fred, "Process Migration in the Sprite Operating System", Report No. UCB/CSD 87/343, Computer Science Division (EECS), University of California, Berkeley (Feb. 1987).	
	NPL23	Powell, Michael L., and Miller, Burton P., "Process Migration in DEMOS/MP", ACM document number 0-89791-115-6/83/010/0110 pp. 110-119 (1983).	
	NPL24	Theimer, Marvin M.; Lantz, Keith A.; and Cheriton, David R., "Preemptable Remote Execution Facilities for the V-System", ACM document number 0-89791-174-1-12/85-0002 pp. 2-12 (1985).	
	NPL25	Borenstein, Nathaniel S., "Computational Mail as Network Infrastructure for Computer-Supported Cooperative Work,"CSCW 92 Proceedings, pp. 67-74 (Nov., 1992).	
	NPL26	Makoto, "TNG PhoneShell (part 2). A proposal and an implimentation of internet access method with telephones and facsimilies", JICST abstract 96A0053311, May 1995.	
	NPL27	PwWebSpeak Overview [online]. The Productivity Works, 1996-09-04, [retrieved on 1997-04-15]. Retrieved on the Internet <URL: http://www.prodworks.com/pwwwovw.htm.	
	NPL28	Hakkinen et al., "pwWebSpeak: User Interface Design of an Accessible Web Browser".	
	NPL29	"`WebSpeak` opens cyberspace to visually impaired," The Times, Trenton, NJ, 3 pages (Feb. 12, 1996). cited by other .	
	NPL30	Christodoulakis et al. "The Multimedia Object Presentation Manager of MINOS: A symmetric approach", SIGMOD vol. 15 No. 2 pp. 295-310, Jun. 1986.	

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.
[1] Applicant's unique citation designation number (optional). [2] Applicant is to place a check mark here if English language Translation is attached.

DOC#719925_1.DOC

Substitute for form 1449/PTO				*Complete if Known*	
FIRST SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT *(Use as many sheets as necessary)*				*Application Number*	10/995,159
				Filing Date	November 24, 2004
				First Named Inventor	Danny LANGE
				Art Unit	2142
				Examiner Name	Douglas B. Blair
Sheet	4	of	5	*Attorney Docket Number*	2222.0300002

NON PATENT LITERATURE DOCUMENTS

Examiner Initials*	Cite No.[1]	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume number, publisher, city and/or country where published	T[2]
	NPL31	Zue, "Navigating the Information Superhighway Using Spoken Language Interfaces" IEEE Expert pp. 39-43, Oct. 1995.	
	NPL32	Caldwell et al., "Project Echo--Telephonic Browser for the WWW", <http:www.cc.gatech.edu/people/home/tgay/echo.html> Apr. 15, 1997, undated.	
	NPL33	James, "Presenting HTML Structure in Audio: User Satisfaction with Audio Hypertext", <http://www-pcd.stanford.edu/.about.fjames/reports/pilot-tr/techrep-pilot. html> Apr. 14, 1997, undated.	
	NPL34	James, "AHA:Audio HTML Access" <http://www-pcd.stanford.edu/.about.fjames/aha/www6/PAPER296.htmll Apr. 14, 1997>, undated.	
	NPL35	Novick et al., "A multimodal browser for the World-Wide Web", undated.	
	NPL36	House, "Spoken-language Access to Multimedia (SLAM)", Master's Thesis, Oregon Graduate Institute, undated.	
	NPL37	Groner, "The telephone--the ultimate terminal", Telphony, pp. 34-40, Jun. 1984.	
	NPL38	Arita et al., "The Voice Browser--an Archetype-Based Dialog Model", NEC Res & Develop., vol. 36 No. 4. pp. 554-561, Oct. 1995.	
	NPL39	Hemphill et al., "(Surfing the Web by Voice", ACM 0-89791-751-0-95/11 pp. 215-222, Nov. 1995.	
	NPL40	Chin, John P., "Personality Trait Attributions to Voice Mail User Interfaces", Proceedings of the 1996 Conference on Human Factors in Computing Systems, CHI 96, Online! Apr. 13-18, 1996, pp. 248-249, XP002113878 Vancouver, BC, CA; retrieved from the Internet on 1999-09-96.	

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.
[1] Applicant's unique citation designation number (optional). [2] Applicant is to place a check mark here if English language Translation is attached.

DOC#719925_1.DOC

Substitute for form 1449/PTO				***Complete if Known***	
FIRST SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT *(Use as many sheets as necessary)*				*Application Number*	10/995,159
				Filing Date	November 24, 2004
				First Named Inventor	Danny LANGE
				Art Unit	2142
				Examiner Name	Douglas B. Blair
Sheet	5	of	5	*Attorney Docket Number*	2222.0300002

NON PATENT LITERATURE DOCUMENTS

Examiner Initials*	Cite No.[1]	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume number, publisher, city and/or country where published	T[2]
	NPL41	"Method for Appropriately Interfacing to User Characteristics in a Voice Interface System," IBM Technical Disclosure Bulletin, vol. 37, No. 3, pp. 307-308, XP000441484, New York, Mar. 1994.	
	NPL42	Database Inspec `Online` Institute of Electrical Engineers, Stevenage, GB, Trainer et al.: "The inclusion of personality trait based adaptive interfaces into computer based learning and training environments," Database accession No. 5193879 XP992113879, Abstract and Proceedings of the Thirty-First International Matador Conference, Apr. 20-21, 1995, pp. 195-200, Manchester, UKISBN: 0-333-64086-1.	
	NPL43	Reeves, B. and Nass, C., The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places, pp. 89-108, ISBN No. 1-57586-052-X, CSLI Publications (1996). cited by other .	
	NPL44	Dunlap, C. AT&T: Internet can talk, too. Computer Reseller News, Nov. 11, 1994, Iss. 607; p. 12 [retrieved on 200-07-06]. REtrieved from the Internet <URL: http://proquest.umi.com/-22 .	
	NPL45	WebSpeak Browser Guides Blind on to Internet. The Sunday Times, Feb. 25, 1996, [retrieved on 1997-04-97]. Retrieved from the Internet <URL: http://www.prodworks.com/st960225.htm>.	
	NPL46	Aguilar, R. Visually Impaired Get Talking Browser. News.com[online], 1996-02-12, [retrieved on 1997-02-12}. Retrieved from the Internet <URL: http://www.news.com/News/Item/0,4,642,00.htm>.	
	NPL47	"New Product Makes the Internet World Wide Web Usable by the Visually Impaired," at <http://www.prodworks.com/pwwovw.html>, pwWebSpeak Press Release, The Productivity Works, Inc., 2 pages (last updated Feb. 10, 1996). cited by other .	
	NPL48		
	NPL49		
	NPL50		

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.
[1] Applicant's unique citation designation number (optional). [2] Applicant is to place a check mark here if English language Translation is attached.

DOC#719925_1.DOC

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : **G06F 9/44, 9/46**	**A1**	**(11) International Publication Number:** **WO 91/10191** **(43) International Publication Date:** 11 July 1991 (11.07.91)

(21) International Application Number: PCT/JP90/01680

(22) International Filing Date: 21 December 1990 (21.12.90)

(30) Priority data:
1/337031 26 December 1989 (26.12.89) JP
1/337032 26 December 1989 (26.12.89) JP

(71) Applicant *(for all designated States except US):* FUJITSU LIMITED [JP/JP]; 1015, Kamikodanaka, Nakahara-ku, Kawasaki-shi, Kanagawa 211 (JP).

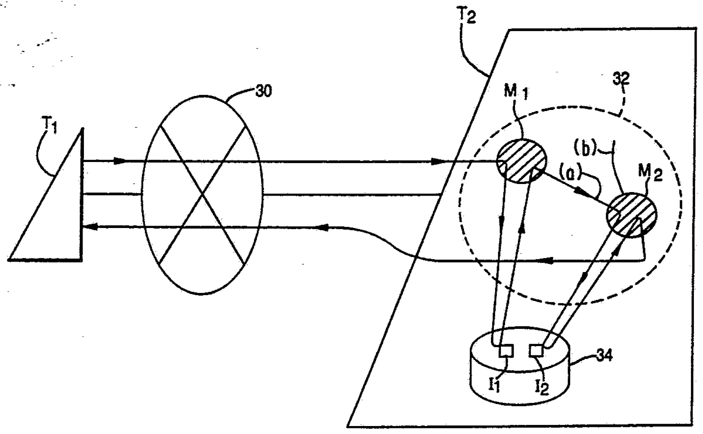
(72) Inventors; and
(75) Inventors/Applicants *(for US only) :* AOE, Shigeru [JP/JP]; 5-3-11-702, Yashio, Shinagawa-ku, Tokyo 140 (JP). KAKEHI, Gen [JP/JP]; 20-9-101, Edaminami 5-chome, Midori-ku, Yokohama-shi, Kanagawa 227 (JP). RYU, Tadamitsu [JP/JP]; 1-604, Konandai Kotohausu, 1151-121, Kamigocho, Sakae-ku, Yokohama-shi, Kanagawa 247 (JP).

(74) Agent: IGETA, Sadakazu; Fujitsu Limited, 1015, Kamikodanaka, Nakahara-ku, Kawasaki-shi, Kanagawa 211 (JP).

(81) Designated States: AT (European patent), BE (European patent), CA, CH (European patent), DE (European patent), DK (European patent), ES (European patent), FR (European patent), GB (European patent), GR (European patent), IT (European patent), JP, LU (European patent), NL (European patent), SE (European patent), US.

Published
With international search report.

(54) Title: OBJECT ORIENTED DISTRIBUTED PROCESSING SYSTEM



(57) Abstract

A system for distributing processing between terminals ($T_1 \sim T_n$) connected via a communication network (30). Each terminal (T_i) is provided with at least one method group (32) and a memory unit (34) to store data files. An originating terminal, e.g., terminal (T_1) accesses data elsewhere in the distributed system by generating a message. The message includes a terminal code identifying an object to access a terminal (T_2) to execute an object, a method code identifying a method for accessing the data and a command name containing or identifying the desired data. The terminal, e.g., terminal (T_2), containing the desired data, decodes the message and accesses a data file in the memory unit (34) containing the desired data identified by the command name. The message may also include a selector and reset conditions which control the sequence of data processing in the terminals (T_0 and T_n) so that the processing of the originating (T_1) and data accessing (T_2) terminals can be coordinated. When the processing identified by the method code is completed in the data accessing terminal (T_2), a message including the resulting data is sent from the data accessing terminal (T_2) to the originating terminal (T_1). Objects according to the present invention may contain a method without data and may be transmitted from one ternimal to another as data. Thus, any application or operating system program can be replaced or added by transmitting an object containing that program as data.

DESCRIPTION

OBJECT ORIENTED DISTRIBUTED PROCESSING SYSTEM

Technical Field

The present invention is directed to object oriented distributed processing in a system of computers at terminals connected by a communication network and, more particularly, to communication between objects stored and executed in the terminals to perform processing of software.

Background Art

In conventional distributed data processing systems, if a first terminal requires data stored in a second terminal, the first terminal requests the data and the second terminal transmits the data to the first terminal so that the first terminal can process the data. The transmission of raw data which is typically higher in volume than processed data requires a high capacity communication network and limits the number of terminals which can be connected together before significantly reducing overall throughput of the system.

Large distributed processing systems contain a large number of terminals, each having its own computer system. When changes are made to the operation of the distributed processing system, the computer system at each terminal must be updated. Most conventional distributed processing systems do not have the capability to update the operating system of the computer systems at the terminals via the communication network.

In computer systems using object oriented architecture, abstracted data (instance) and a program (method) specifying processing of the data are together treated as an object. Data processing is carried out by processing the methods in such objects and communicating messages therebetween. In object oriented processing, wherever the data is located, procedures for processing the data will be located also. Execution of these procedures is triggered by message transfer between objects, including the transmission of resulting data in messages after processing of data in an object is completed.

Applying object oriented techniques to a distributed data processing system is not easily accomplished due to the need to maintain the linkage between data and method of an object, to determine where an object is located and to communicate between objects. One way of implementing object oriented techniques in a distributed processing system is to maintain a database in each terminal of the objects in all of the terminals. In a large system, this requires a large amount of overhead due to the size of the memory required and the time and communication traffic required to update the object location database in each of the terminals.

An object of the present invention is to provide object oriented processing in all terminals of a distributed processing system, executed as if in a single processing system.

Another object of the present invention is to provide a system for distributed data processing in which objects can be obtained by one terminal from another terminal without maintaining an object location database in all terminals.

A further object of the present invention is to provide a distributed processing system capable of transmitting changes to the operating system programs of the distributed processing system to the terminals via the communication network of the distributed processing system.

Yet another object of the present invention is to provide a method for linking objects regardless of their location or content.

DISCLOSURE OF INVENTION

In a distributed processing system having a plurality of terminals connected via a communication network, object oriented processing is performed by communicating via messages containing a terminal code, a method code identifying processing of data and a command containing data or identifying data in an object to be processed. The method identified by the method code is executed by the terminal identified by the terminal code. Messages may contain multiple submessages, where each submessage includes a terminal code, a method code and a command. The command may identify an object in one terminal which is to be installed in the terminal identified by the terminal number. This object may be a new program or a replacement for an existing program in the terminal identified by the terminal number. The program may be an application program or an operating system program. Objects may contain methods without data, thereby permitting any type of program to be updated.

The terminal code may identify an object for transmitting the message to the terminal where the object identified by the object code is located. If the terminal code is zero or null, the object will be looked for in the terminal in which the object generating the message is executing. If the terminal number is unknown, the object identified by the terminal code will identify an object which will perform processing to locate the object in one of the other terminals on the system.

The command in the object code may include a selector condition and a reset condition for identifying how the method identified by the method code is to be initiated and how to proceed when the sequence starting with the method identified by the method code has completed execution, respectively. The command code may contain a message for identifying an object for obtaining data for the method identified by the method code to process in the terminal identified by the terminal code.

These objects, together with other objects and advantages which will be subsequently apparent, reside in the details of constitution and operation as more fully hereinafter described and claimed, reference being had to the accompanying drawings forming a part hereof, wherein like reference numerals refer to like parts throughout.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of a distributed processing system to which the present invention can be applied;

Fig. 2 is a block diagram of one of the terminals;

Fig. 3 is a chart of the command link and sequence files;

Fig. 4 is a flowchart of processing to locate an object according to the present invention;

Fig. 5 is a flowchart of processing to update a program in a remote terminal;

Fig. 6 is a flow diagram of object sequencing in the prior art;

Fig. 7 is a conceptual block diagram of processing in a terminal having an object required by another terminal and illustrating the use of common objects;

Fig. 8 is a flow diagram of object sequencing using common objects according to the present invention;

Fig. 9 is a flowchart of sequencing using common objects according to the present invention; and

Fig. 10 is a flowchart of processing using common objects according to the present invention.

BEST MODE OF CARRYING OUT THE INVENTION

As illustrated in Fig. 1, a plurality of terminals T_0 ~ T_n are connected via a communication network 30, such as a local area network (LAN), intelligent network (IN), integrated services digital network (ISDN). In the case of ISDN, a dedicated line is preferably used for carrying supervisory information, while object commands and responses thereto are communicated by ISDN. Each terminal T_i has program modules stored in method groups 32 and data stored in a data file 34. An object executes in one of the terminals by executing the program stored in one of the method groups and identified as being the method for that object. When the method is instructed to begin execution of another object, e.g., to obtain data from the second object, a message is generated having the general format G below

$$G = \{(T_i: M_j, I_k); (T : M_m, I_n); \ldots \} ,$$

where T_i is a terminal code, M_j a method code and I_k the name of an instance and any one or two of the (T_i, M_j and I_k) may be missing, as described below.

Each terminal T_i includes the components illustrated in Fig 2. A system interface 40 is connected to the communication network 30. A processor 42 receives input from an operator via an input device 44 which may be a keyboard or other peripheral, including graphic tablet, tape or floppy disk drive, etc. An output device 46, such as a CRT display, printer, the tape or floppy disk drive, etc., provides output to the operator. The objects (methods and instances) are stored in a memory unit 48, such as a hard disk drive, and are addressed by a system table 50 which may be stored in the memory unit 48 and in random access memory (not shown) in the processor 42. An example of a commercial embodiment of the terminal illustrated in Fig. 2 is a personal computer.

The system table 50 is one of the most important tables in each terminal T_i. As illustrated in Fig. 3, the system table includes part of a command link file 52 and all of a sequence file 54. The command link file 52 stores the location and size of all objects stored in the memory unit 48. The method entries 56 in the command link file 52 are included in the system table because they identify the programs which constitute and can be executed by the system controlling the operation of the terminal T_i. The command link file 52 also includes instance entries 58 which identify the location and size of data stored in the memory unit 48.

When the message G above is received by terminal T_i, the object may be interpreted method M00003 and instance I00003. The processor 42 can retrieve the method M00003 and instance I00003 by accessing the system table 50 to determine their address and size, as indicated by the dashed lines in Fig. 2.

The command link file 52 permits the same block of code or data to be used in more than one object. In the example illustrated in Fig. 3, methods M00001 and M00004 have the same address and size. However, the instances, I00001 and I00004, forming objects with these methods are stored in different locations, although they are the same size. Thus, the same method (stored at address A00007482) can be used to process different data, depending on which name (M00001 or M00004) is used. Similarly, the same data can form different objects by being combined with different methods. For example, one method may transfer data to or from the terminal T_i, while another method processes the same data in the terminal T_i.

When a new object is to begin control of the processor 42, the processing illustrated in Fig. 4 is executed. First, the new object name is stored 60 in an object management table. Next, the terminal number associated with this object in the message controlling the triggering of the new object, is compared 62 with the terminal number, e.g. T_1, of the terminal performing this process. If there is a match and the address in the command link file 52 indicates the method is stored in terminal T_1, the object is moved 64 from the memory unit 48 to the execution area (not shown) in the processor 42. When the terminal number associated in the message with the new object name does not match 62 the terminal number of the terminal performing this process or the address of the method and instance of an object refer to another terminal as indicated for object O00003 (method M00003 and instance I00003), a message is sent 66 to the terminal, e.g. T_2, identified in the message or address field of the command link file to trigger the object in that terminal (T_2). This is done by triggering an object with the same name as the terminal number (T_2) in the terminal T_1 executing the process illustrated in Fig. 4.

When the terminal number indicates that the terminal in which the object to be executed is unknown, first the system table 50 of the terminal (T_1) executing the process is checked 68 for the object name. If the object name is found 70, the object is moved 64 to the execution area of the processor 42. If the object name is not found 70, an object is triggered to access a master system file. The master system file keeps a record of the location of all objects. The master system file may be located in the communication network 30 or in one of the terminals (a server). Alternatively, a copy of the master system file may be located in each of the terminals. However, this last alternative requires a significant amount of overhead to maintain the master system file.

When an object is located in a different terminal, the object may be executed there or transferred to another terminal, including a terminal requesting its execution. In the preferred embodiment, when an object not presently stored in a terminal is desired to be executed, the object will be executed in the terminal in which it is stored. To provide other terminals with an object stored in terminal T_1, for example, the processing steps illustrated in Fig. 5 are executed. This process is termed a learning method and may be used to transfer any application programs or system programs when all are in the form of objects.

As illustrated in Fig. 5, when an object transfer request is received 74 by terminal T_1, e.g. via the input device 44, the requested object is accessed 76 in memory unit 48 via the system table 50. The object is decomposed 78 in byte data for transfer 80 as an entity to the second terminal, e.g. terminal T_2. Depending upon the type of transfer request received, the entire object may be transferred 80 after decomposition, or only the method or data in the object. The requested portion or entirety of the object is transferred 80 by transmitting a message including the object (or portion thereof) as entity data. Terminal T_2 receives 82 the message including the entity data, composes 84 the object for installation and stores 86 the object in the memory unit 48 in terminal T_2. In addition, the system table 50 in terminal T_2 is updated 86 with the object name. If the object name previously existed in the system table, the address and (if necessary) size of the object will be updated to reflect the object just stored.

The sequence file 54 (Fig. 3) enables predefined sequences of methods to be stored in the system table 50. As an alternative to the message format G above, a simplified message format G_1 may be used, where sf_i is

$$G_1 = \{T_i:O_i,\ sf_i;\ T_{i+1}:O_{i+1},\ sf_{i+1}\ \ldots\}$$

a sequence flag indicating that the preceding object name is a sequence number which should be looked for in the sequence file 54. For example, if O_i in G_1 is S00001, methods M00001, M00003 and M00005 will be executed in sequence. In a first alternative embodiment, the object name may identify a method and the command fields of the sequence file 54 will be searched for the corresponding method name when the sequence flag is set. In this embodiment, each method may be limited to a single entry in the sequence file or some rule, such as first occurrence in the file may be used to find the right sequence. In this case each method may appear in as many sequences as desired, but should start only one sequence. In a second alternative embodiment, the sequence flag may be a sequence number identifying the starting point of the sequence. In this embodiment, the object name can be used to access the first method and instance while the sequence is being determined.

Another way of sequencing objects according to the present invention is to use common objects. In the prior art, the only way to sequence objects was to link one object to the next as illustrated in Fig. 5. According to the present invention, messages may have the format G_2 including a selector condition s_i and a

$$G_2 = \{T_i:(O_i,\ s_i,\ r_i);\ T_{i+1}:(O_{i+1},\ s_{i+1},\ r_{i+1})\ldots\}$$

reset condition r_i. According to the present invention, common objects permit the same object to be used in different ways by different objects and to predefine a variety of sequences in the common objects themselves instead of in a sequence file. The selector condition

s_i defines an entry point in the common object O_i. The reset condition r_i defines how the sequence ends.

It should be noted that common objects are not limited to use in forming sequence objects. Sequence objects have a defined structure with a beginning and end. On the other hand, common objects have a defined function and can be executed whenever that function is required. For example, common objects can be used in the kernel of an operating system to perform functions whenever needed.

A very simple example of the use of common objects in sequence will be provided first with reference to Fig. 7 and message G_3 below. The beginning of message G_3 instructs

$$G_3 = \{T_2:O_1;\ T_2:(O_2,\ a,\ r)\}$$

terminal T_1 that objects are to be executed in terminal T_2 and so message G_3 is transmitted to terminal T_2 by executing an object called T_2 in terminal T_1. Terminal T_2 decodes the message G_3 to determine that for the first object O_1 method M_1 is to be executed using the data in instance I_1 and then a second object O_2 is to be performed using method M2, by entering at (a), as identified by the selector condition a. The data in instance I_2 is processed by method M_2 and then processing returns to terminal T_1, as identified by the reset condition r. No selector or reset condition is provided for object O_1, because there is a single entry and exit point for object O_1.

An example of a few sequence objects 90 and common objects 92 are illustrated in Fig. 8. Common objects O_A, O_B, and O_C have three entry points, a, b and c, and three exit points, but common object O_C has an extra entry point and common object O_D has only one entry and exit point. Processing of the common objects illustrated in Fig. 8 will be discussed with reference to the flowchart illustrated in Fig. 9.

First taking as an example the sequence initiated by sequence object O_1, common object O_A is triggered with selector condition a and a reset condition to return to the originating object (O_1). Since the selector condition is a, the selector condition matches in the test 94 for condition a, and the appropriate condition steps 96 are executed. After any common steps 98 are executed, it is determined 100 whether a new common object should be triggered. As indicated by the dashed line across object O_A from entry point a, in this example, common object O_B is to be triggered 102 with selector condition a. The reset condition is passed on in the message which triggers common object O_B. The same steps of Fig. 9 are executed in common object O_B and a message is generated to trigger common object O_C with selector condition b and the same reset condition.

In common object O_C, it will be determined 94 that the selector condition is not a, but a match will be found in the test 104 for selector condition b. Therefore, the steps 106 for condition b will be executed prior to the common steps 98. As indicated by the dashed line entering common object O_C at b, it will be determined 100 that there are no more common objects to be executed and so the reset condition will be tested 108. As noted above, the reset condition in the message generated by sequence object O_1 was to return to the originating object. This reset condition is passed to common object O_C and therefore processing will return 110 by referencing the object management table to determine the originating object.

Processing of the other sequences illustrated in Fig. 8 is similar. The sequence originating with sequence object O_2 uses selector condition b in common object O_B and proceeds to use the same selector condition in common object O_A. At the end of the sequence, common object O_C is triggered with selector condition a. The reset condition in the originating message from sequence object O_2 indicates that a new sequence object O_3 is to be triggered 112 by the final common object O_C in the sequence. The sequence originating in sequence object O_4 passes through common objects O_B and O_D before terminating 114 in common object O_C.

An example of the use of common objects is provided in the flowchart illustrated in Fig. 10. The operations of sequence objects appear on the left side of Fig. 10, while the operations of common objects appear on the right. One or more sequence objects are used to receive and initiate transfer 120 of data to a common object. Thus, unique input routines are used to interface with an operator or a peripheral device inputting application specific data. A common object is used to store 122 the data. The common object transfers 122 a command name, identifying the data, to a sequence object. The sequence object fetches the command and processes 124 as required by the application. If it is determined 126 that editing is necessary, general purpose editing routines stored as one or more common objects can be used to edit 128 the data.

After any necessary editing, it is determined 130 whether any existing entity data is needed. If so, a command name is transferred 132 to another common object which extracts 134 the entity data and converts command names to entity data. Processing is continued by a sequence object which fetches 136 the entity data and performs additional processing. According to the present invention, commonly executed routines can be stored as common objects, but flexibility in the order in which they are executed and details of how they are executed is provided.

INDUSTRIAL APPLICABILITY

The present invention relates to a distributed data processing system and particularly to a system for realizing distributed object oriented processing between terminals connected via a communication network. In a database system which is required to provide sophisticated functions for diversification of application modes in which a plurality of terminals are connected via a communication network, the man-hours required for system development increase as more sophisticated functions are added. A system with improved processing speed and reduced communication traffic requiring less development time is provided.

CLAIMS

What is claimed is:

1. A distributed processing system, comprising:

a communication network to transmit messages, including data; and

a plurality of terminals connected together by said communication network, each terminal capable of generating the messages, each of the messages having a terminal number for identifying one of the terminals to receive the message, a method code identifying processing of data and a command for identifying data to be processed.

2. A distributed processing system as recited in claim 1, wherein each of said terminals comprises:

a memory unit to store objects, including methods and instances;

a processor, operatively connected to said communication network, to execute the methods stored in said memory unit; and

a system table to store identifying information on the location and execution sequence of the methods stored in said memory unit.

3. A distributed processing system as recited in claim 1, wherein each of said terminals comprises:

a memory unit to store objects, including methods, at least one of the objects including a learning method;

a processor, operatively connected to said communication network, to execute the methods stored in said memory unit; and

a system table to store identifying information on the location and execution sequence of the methods stored in said memory unit, said processor updating said system table and the contents of said memory unit upon receipt of one of the messages from another terminal instructing said processor to execute the learning method stored in said memory unit.

4. A terminal in a distributed processing system, comprising:

object storage means for storing objects, including at least one object including a method;

system table means for storing identifying information on location and execution sequence of the objects stored in said object storage means; and

decomposing means for decomposing the method in one of the objects into byte data in response to a transfer request;

transfer means for transferring the byte data decomposed by said decomposing means to another terminal upon completion of the decomposing and for receiving an updated object requested to be transferred to said terminal; and

installation means for storing the updated object into said object storage means and for updating said system table means with an object name corresponding to the updated object.

5. A method for transferring programs in a distributed processing system executing objects in terminals, the objects inclusive of objects including a method, said method comprising:

(a) receiving a transfer request at a first terminal to transfer a selected object from the first terminal to a second terminal;

(b) decomposing the selected object into byte data in response to the transfer request;

(c) transferring the byte data decomposed in step (b) to the second terminal upon completion of said decomposing;

(d) receiving the byte data at the second terminal;

(e) composing the byte data to produce an updated object corresponding to the selected object requested to be transferred to the second terminal; and

(f) storing the updated object and an object name, corresponding to the updated object, in the second terminal.

6. A method as recited in claim 5, wherein said decomposing in step (b) comprises the step of selecting one of the method and data in the selected object for said transferring in step (c).

7. A method for processing messages in a distributed processing system having a plurality of terminals connected by a communication network, said method comprising the steps of:

(a) determining in a first terminal, for a first message identifying a first object, whether the first object is stored in the first terminal;

(b) triggering the first object in the first terminal when said determining in step (a) determines that the first object is stored in the first terminal; and

(c) generating a second message to a second terminal to locate the first object when said determining in step (a) determines that the first object is not stored in the first terminal.

8. A method as recited in claim 7, wherein all messages generated in the distributed processing system, including the second message generated in step (c), include a terminal number for identifying one of the terminals to receive the message, a method code

identifying processing of data and a command for identifying data to be processed.

9. A method as recited in claim 8, wherein said determining in step (a) comprises comparing the terminal number identifying the first terminal with the terminal number in the first message.

10. A method as recited in claim 9, wherein said generating in step (c) comprises setting the terminal number in the second message equal to the terminal number in the first message when said comparing in step (a) determines that the terminal in the first message is not equal to the terminal number of the first terminal.

11. A method as recited in claim 10, further comprising the steps of:

(d) storing method codes of each of the objects stored in each of the terminals in a command link file in each of the terminals, respectively, and all of the method codes for all of the objects stored in all of the terminals in a master system table;

(e) comparing the method code in the first message with the method codes in the command link file for the first terminal when said comparing in step (a) determines that the terminal number of the first message is unknown;

(f) executing step (b) when said comparing in step (e) determines that the method code in the first message is included in the message codes in the command link file for the first terminal; and

(g) generating a third message to access the master system table when said comparing in step (e) determines that the method code in the first message is excluded from the message codes in the command link file for the first terminal.

12. A method as recited in claim 7,
wherein said generating in step (c) comprises the step of (c1) generating the second message with a subset of a second object, and
wherein said method further comprises the steps of:
(d) storing command names associated with each of the objects stored in each of the terminals in a command link file in each of the terminals, respectively;
(e) storing the subset of the second object in the second terminal upon receipt of the second message at the second terminal; and
(f) storing a subset command name, corresponding to the subset of the second object, in the command link file of the second object.

13. A method as recited in claim 12, wherein said generating in step (c) further comprises the step of (c2) decomposing the second object into byte data prior to said generating in step (c1).

14. A method as recited in claim 13, wherein said generating in step (c2) comprises the step of transferring all of the byte data decomposed from the second object into the second message.

15. A method as recited in claim 13, wherein said generating in step (c2) comprises the step of transferring a method portion of the byte data decomposed from the second object into the second message.

16. A method as recited in claim 7,
wherein all messages generated in the distributed processing system, including the second message generated in step (c), include a terminal number

for identifying one of the terminals to receive the message and an object code, and

wherein said method further comprises the steps of:

(d) determining whether the object code includes a sequence file identifier; and

(e) executing sequence objects in a sequence determined by entries in a sequence file when said determining in step (d) determines that the object code includes the sequence file identifier.

17. A method as recited in claim 7,

wherein all messages generated in the distributed processing system, including the second message generated in step (c), include a terminal number for identifying one of the terminals to receive the message and an object code, and

wherein said method further comprises the steps of:

(d) storing common objects in at least one of the terminals;

(e) determining whether the object code includes a common object name, a selector condition and a reset condition;

(f) triggering a corresponding common object in dependence upon the common object name and the selector condition when said determining in step (e) determines that the object code includes the common object name, the selector condition and the reset condition; and

(g) ending execution of the common objects in dependence upon the reset condition when said determining in step (e) determines that the object code includes the common object name, the selector condition and the reset condition.

18. A method as recited in claim 17, wherein said triggering in step (f) comprises the steps of:

(f1) comparing the selector condition with predetermined selector conditions in the corresponding common object; and

(f2) executing condition steps in the corresponding common object identified by one of the predetermined selector conditions in dependence upon said comparing in step (f1).

19. A method as recited in claim 17, wherein said ending in step (g) comprises the steps of:

(g1) returning to the first object when the first message was determined in step (e) to contain the common object name of one of the common objects and the reset condition is set to return;

(g2) triggering a new sequence object when the reset condition is set to trigger; and

(g3) terminating execution of the common objects without returning or triggering when the reset condition is set to terminate.

20. A message transmitted between objects in an object oriented distributed processing system having a plurality of terminals connected by a communication network, said message comprising:

a terminal number for identifying one of the terminals to receive said message;

a method code identifying processing of data; and

a command for identifying data to be processed.

21. A message as recited in claim 20, wherein said command includes a sequence file identifier identifying a sequence defined in a sequence file for executing objects triggered by said message.

22. A message as recited in claim 20, wherein said command includes a selector condition and a reset condition when said method code identifies a common object, the selector condition determining how the common object executes and the reset condition determining how execution of the common object and any subsequent additional common objects ends.

23. A system table in an object oriented distributed processing system having a plurality of terminals connected by a communication network, said system table comprising:

a method code portion of a command link file including fields for method name, address and size; and

a sequence file including fields for sequence number, command name and next sequence number.

FIG. 1

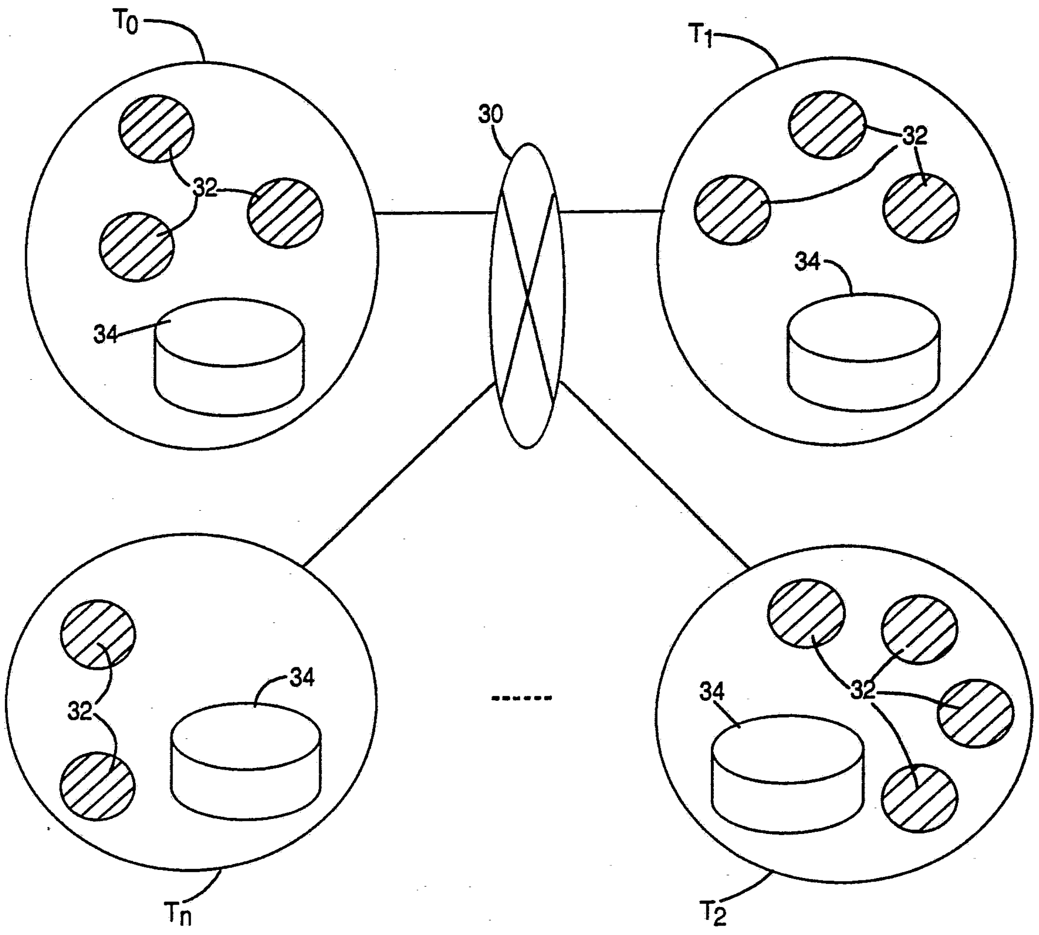


FIG. 2

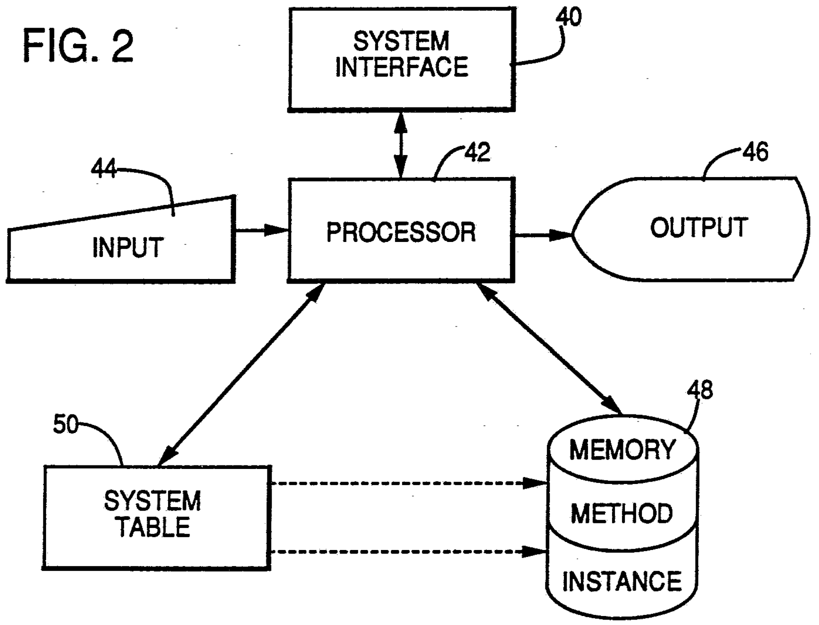
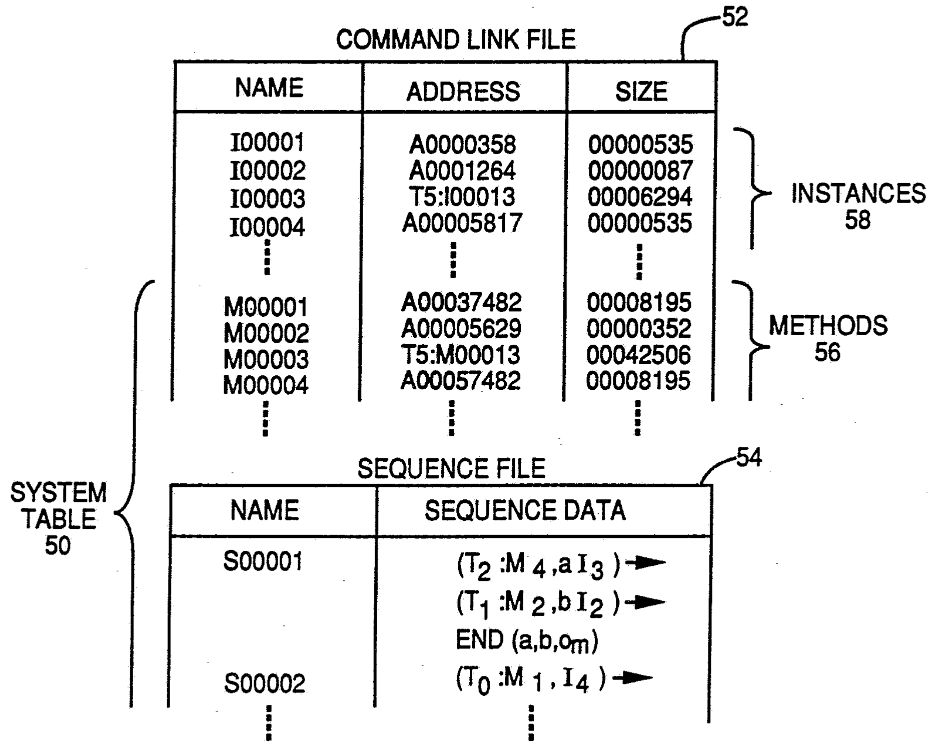


FIG. 3


52
INSTANCES 58
METHODS 56
54
SYSTEM TABLE 50

COMMAND LINK FILE

NAME	ADDRESS	SIZE
I00001	A0000358	00000535
I00002	A0001264	00000087
I00003	T5:I00013	00006294
I00004	A00005817	00000535
⋮	⋮	⋮
M00001	A00037482	00008195
M00002	A00005629	00000352
M00003	T5:M00013	00042506
M00004	A00057482	00008195
⋮	⋮	⋮

SEQUENCE FILE

NAME	SEQUENCE DATA
S00001	$(T_2 : M_4, a I_3) \rightarrow$
	$(T_1 : M_2, b I_2) \rightarrow$
	END (a, b, o_m)
S00002	$(T_0 : M_1, I_4) \rightarrow$
⋮	⋮

FIG. 4

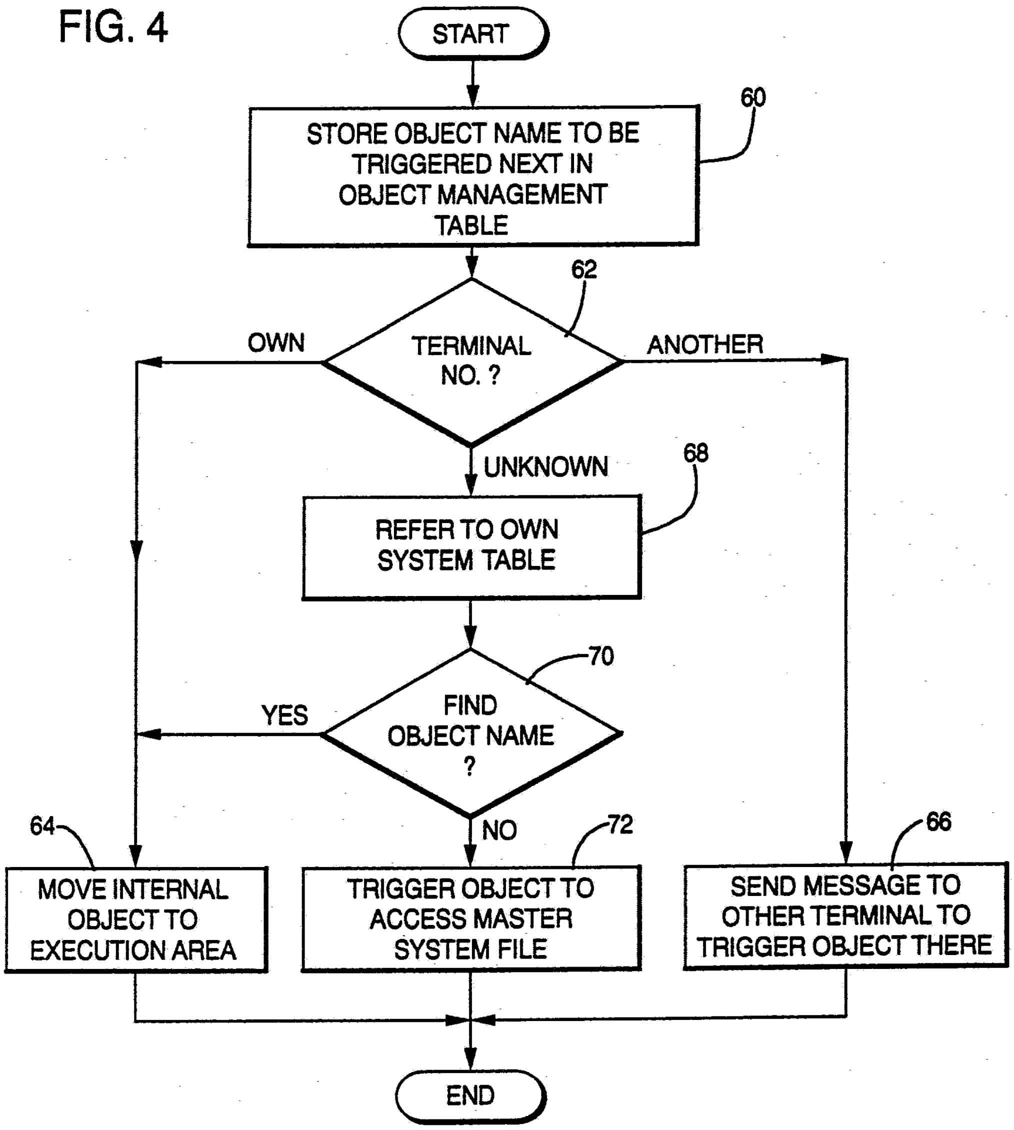
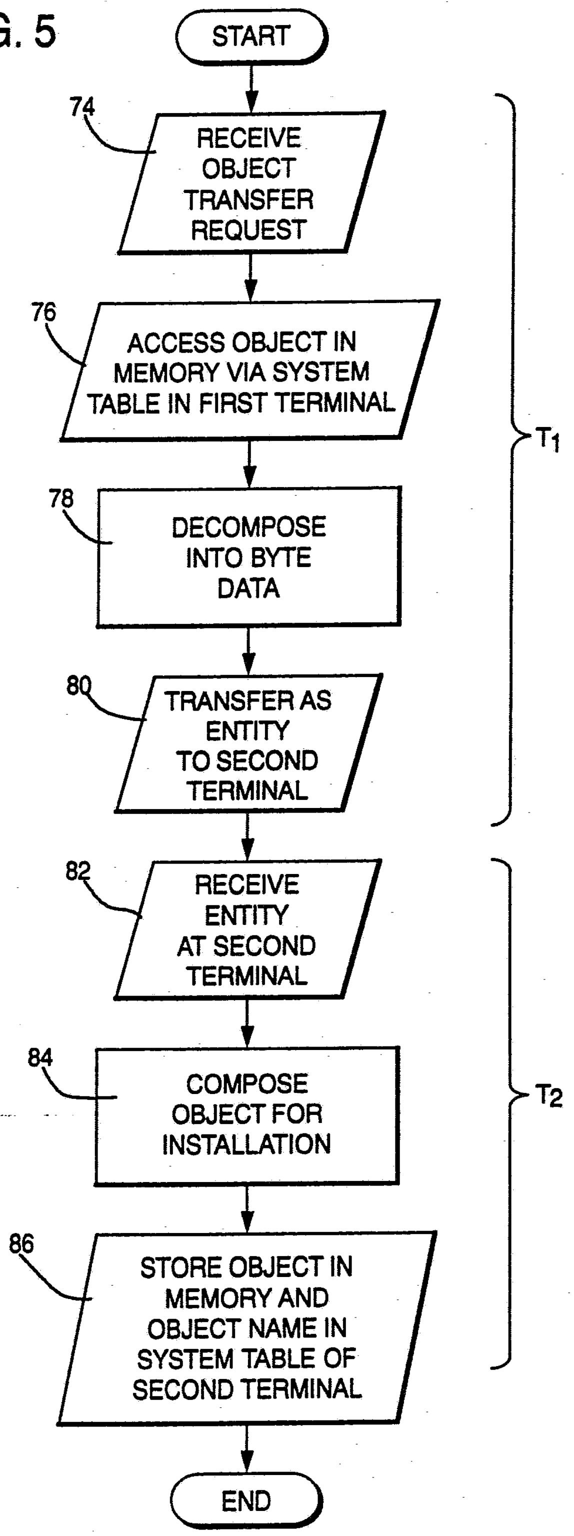

START
STORE OBJECT NAME TO BE TRIGGERED NEXT IN OBJECT MANAGEMENT TABLE
60
62
TERMINAL NO. ?
OWN
ANOTHER
UNKNOWN
68
REFER TO OWN SYSTEM TABLE
70
FIND OBJECT NAME ?
YES
NO
64
72
66
MOVE INTERNAL OBJECT TO EXECUTION AREA
TRIGGER OBJECT TO ACCESS MASTER SYSTEM FILE
SEND MESSAGE TO OTHER TERMINAL TO TRIGGER OBJECT THERE
END

FIG. 5


START
74
RECEIVE OBJECT TRANSFER REQUEST
76
ACCESS OBJECT IN MEMORY VIA SYSTEM TABLE IN FIRST TERMINAL
78
DECOMPOSE INTO BYTE DATA
80
TRANSFER AS ENTITY TO SECOND TERMINAL
T_1
82
RECEIVE ENTITY AT SECOND TERMINAL
84
COMPOSE OBJECT FOR INSTALLATION
86
STORE OBJECT IN MEMORY AND OBJECT NAME IN SYSTEM TABLE OF SECOND TERMINAL
T_2
END

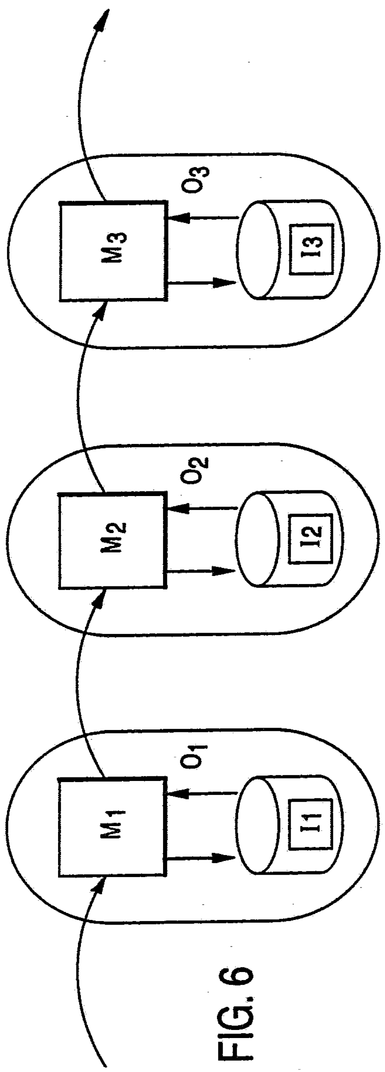


FIG. 6

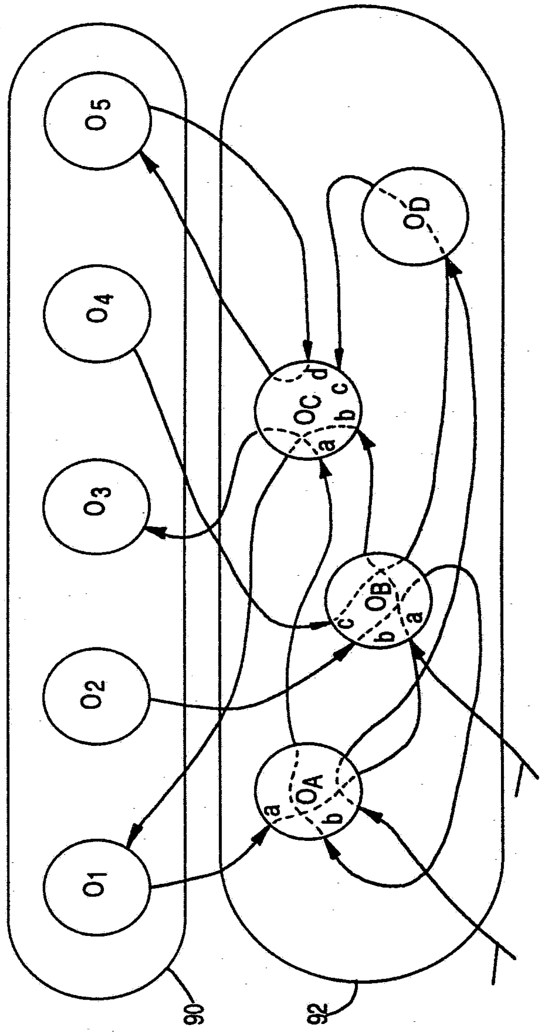


FIG. 8

FIG. 7

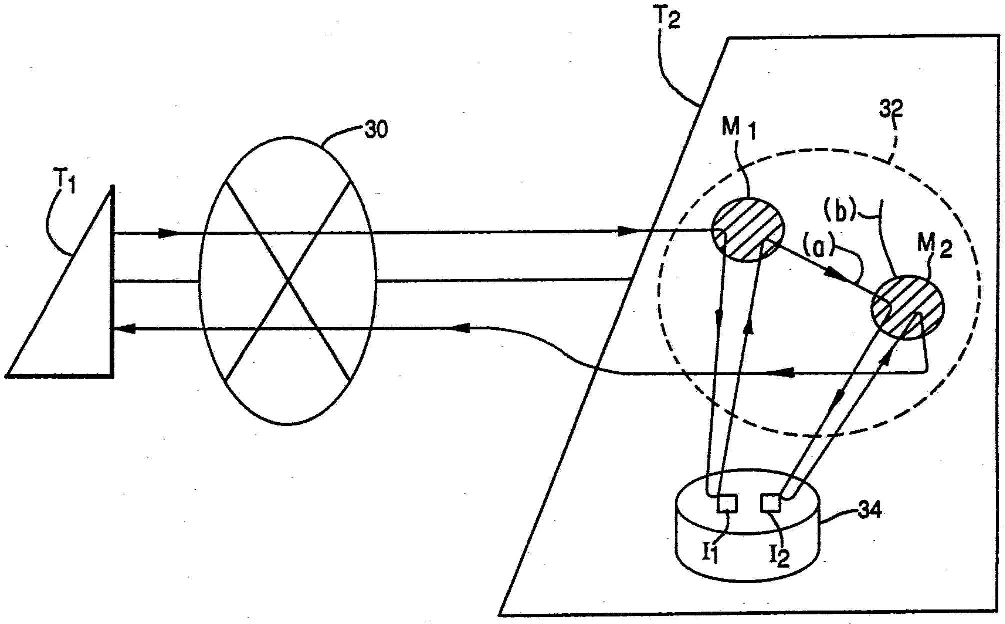


FIG. 9

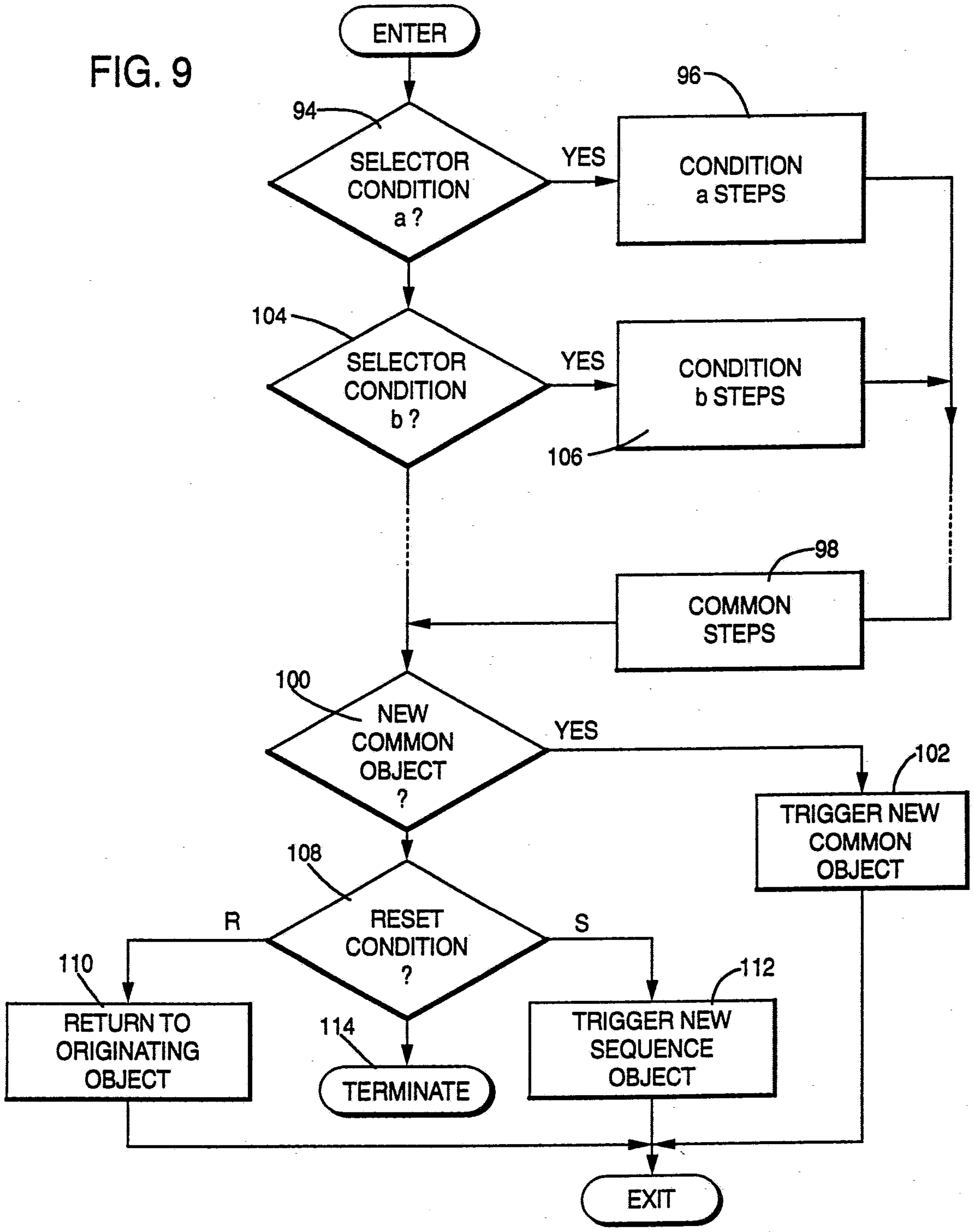
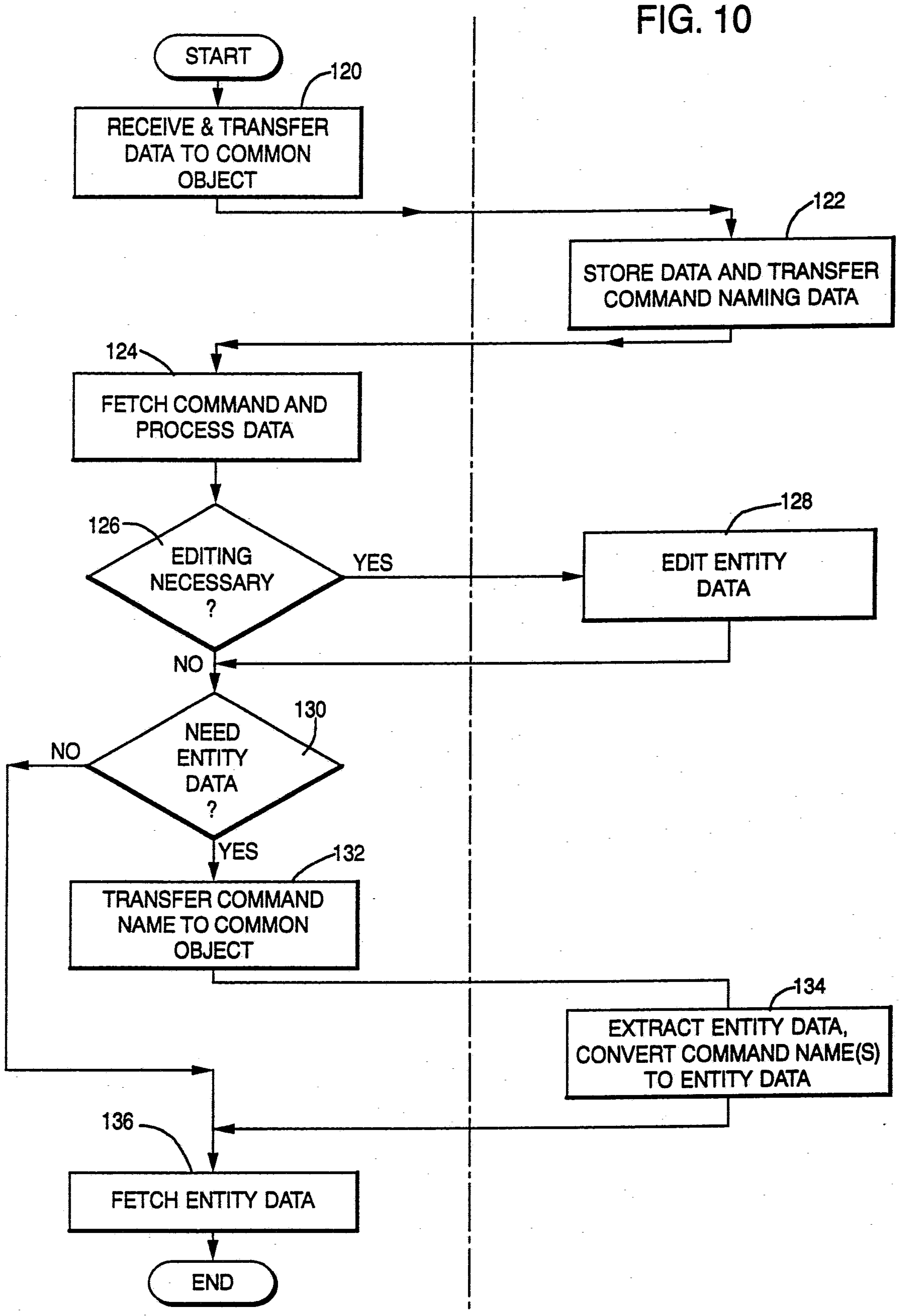

ENTER
94
SELECTOR CONDITION a ?
YES
96
CONDITION a STEPS
104
SELECTOR CONDITION b ?
YES
CONDITION b STEPS
106
98
COMMON STEPS
100
NEW COMMON OBJECT ?
YES
102
TRIGGER NEW COMMON OBJECT
108
RESET CONDITION ?
R
S
110
RETURN TO ORIGINATING OBJECT
114
TERMINATE
112
TRIGGER NEW SEQUENCE OBJECT
EXIT

FIG. 10


START
120
RECEIVE & TRANSFER DATA TO COMMON OBJECT
122
STORE DATA AND TRANSFER COMMAND NAMING DATA
124
FETCH COMMAND AND PROCESS DATA
126
EDITING NECESSARY ?
YES
128
EDIT ENTITY DATA
NO
130
NEED ENTITY DATA ?
NO
YES
132
TRANSFER COMMAND NAME TO COMMON OBJECT
134
EXTRACT ENTITY DATA, CONVERT COMMAND NAME(S) TO ENTITY DATA
136
FETCH ENTITY DATA
END

INTERNATIONAL SEARCH REPORT

International Application No PCT/JP 90/01680

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) [6]

According to International Patent Classification (IPC) or to both National Classification and IPC

IPC^5: G 06 F 9/44, G 06 F 9/46

II. FIELDS SEARCHED

Minimum Documentation Searched [7]

Classification System	Classification Symbols
IPC^5	G 06 F

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched [8]

III. DOCUMENTS CONSIDERED TO BE RELEVANT [9]

Category *	Citation of Document, [11] with indication, where appropriate, of the relevant passages [12]	Relevant to Claim No. [13]
Y	ACM Transactions on Computer Systems, vol. 6, no. 1, February 1988, ACM, (New York, NY, US), E. Jul et al.: "Fine-grained mobility in the Emerald System", pages 109-133, see page 111, lines 3-6; page 114, lines 1-19, section 3.2 --	1-5,7-8,20
Y	Informationstechnik IT, vol. 30, no. 6, December 1988, R. Oldenbourg Verlag, (München, DE), G. Barth et al.: "Objektorientierte Programmierung", pages 404-421, see page 405, right-hand column, lines 10-37 --	1-3,7-8

./.

* Special categories of cited documents: [10]
- "A" document defining the general state of the art which is not considered to be of particular relevance
- "E" earlier document but published on or after the international filing date
- "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- "O" document referring to an oral disclosure, use, exhibition or other means
- "P" document published prior to the international filing date but later than the priority date claimed
- "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step
- "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- "&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search: 19th March 1991	Date of Mailing of this International Search Report: 23.04.91
International Searching Authority: EUROPEAN PATENT OFFICE	Signature of Authorized Officer: miss T. MORTENSEN [illegible]

Form PCT/ISA/210 (second sheet) (January 1985)

III. DOCUMENTS CONSIDERED TO BE RELEVANT (CONTINUED FROM THE SECOND SHEET)

Category *	Citation of Document, ¹¹ with indication, where appropriate, of the relevant passages	Relevant to Claim No.
Y	Microprocessing and Microprogramming, vol. 24, no. 1-5, 'Supercomputers: Technology and Applications', Fourteenth EUROMICRO Symposium on Microprocessing and Microprogramming (EUROMICRO '88), Zurich, 29 August - 1 September 1988, edited by S. Winter et al., (North-Holland, Amsterdam, NL), S.T. Krolak et al.: "DEOS - A dynamically extendible object-oriented system", pages 241-248, see section 3.2	2-5
A		23
	--	
Y	Hewlett-Packard Journal, vol. 40, no. 4, August 1989, (Palo Alto, CA, US), J.A. Dysart: "The new wave object management facility", pages 17-23, see page 22, right-hand column, line 39 - page 23, left-hand column, line 3	4,5

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : **H04M**	A2	**(11) International Publication Number:** **WO 96/11542** **(43) International Publication Date:** 18 April 1996 (18.04.96)

(21) International Application Number: PCT/US95/11737

(22) International Filing Date: 13 September 1995 (13.09.95)

(30) Priority Data:
08/316,635 30 September 1994 (30.09.94) US

(71) Applicant: WILDFIRE COMMUNICATIONS, INC. [US/US]; 20 Maguire Road, Lexington, MA 02173 (US).

(72) Inventors: MINER, Richard, A.; Apartment 105, 129 Franklin Street, Cambridge, MA 02139 (US). WARNER, William, J.; 48 Bogle Street, Weston, MA 02193 (US). LOVELL, Anthony, M.; 94 Raymond Avenue, Somerville, MA 02144 (US). SHIENBROOD, Eric, R.; 1763 Main Street, Concord, MA 01742 (US). GABRYELSKI, Keith; Apartment 2, 2 Drummond Place, Cambridge, MA 02140 (US). ARNOLD, Kenneth, C.R.C.; 7 Moon Hill Road, Lexington, MA 02173 (US). D'ARBELOFF, Nicholas, C.; 345 Cross Street, Belmont, MA 02178 (US). HINCKLEY, Kee; 25 Forest Circle, Winchester, MA 01890 (US).

(74) Agent: PRAHL, Eric, L.; Fish & Richardson P.C., 225 Franklin Street, Boston, MA 02110-2804 (US).

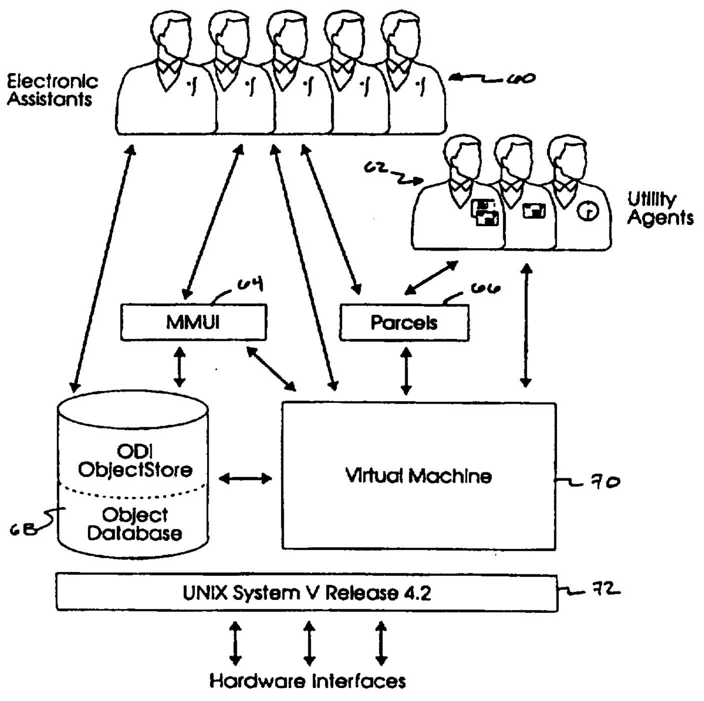
(81) Designated States: CA, JP, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: A NETWORK BASED KNOWLEDGEABLE ASSISTANT

(57) Abstract

A method implemented by a computer-based electronic assistant to receive and manage incoming calls to a subscriber including the steps of receiving an incoming call to the subscriber from a caller; establishing a first connection between the electronic assistant and the caller; establishing a second connection between the electronic assistant and the subscriber; over the second connection, electronically notifying the subscriber of the incoming call; in response to receiving a call accept command from the subscriber over the second connection, linking the caller and the subscriber so that they may communicate with each other; upon linking the subscriber to the caller, switching the electronic assistant to a background mode in which said electronic assistant continues to monitor the subscriber over the second connection while the subscriber is linked with the caller; and in response to receiving a summoning command, switching the electronic assitant into a foreground mode, wherein the electronic assistant when in its background mode responds to a first set of commands including at least the summoning command and when in its foreground mode responds to a second set of commands where the second set of commands is larger than the first set of commands.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgystan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo			SE	Sweden
CH	Switzerland	KR	Republic of Korea	SI	Slovenia
CI	Côte d'Ivoire	KZ	Kazakhstan	SK	Slovakia
CM	Cameroon	LI	Liechtenstein	SN	Senegal
CN	China	LK	Sri Lanka	TD	Chad
CS	Czechoslovakia	LU	Luxembourg	TG	Togo
CZ	Czech Republic	LV	Latvia	TJ	Tajikistan
DE	Germany	MC	Monaco	TT	Trinidad and Tobago
DK	Denmark	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	US	United States of America
FI	Finland	ML	Mali	UZ	Uzbekistan
FR	France	MN	Mongolia	VN	Viet Nam
GA	Gabon				

A NETWORK BASED KNOWLEDGEABLE ASSISTANT

Background of the Invention

The present invention relates to a computer-implemented method and apparatus for managing communications to and/or from a user over multiple media.

Today there are many different, commercially available devices that enable people to communicate with each other electronically. In addition to the ubiquitous telephone that has been around for decades, there now are cordless phones for the home, mobile phones for the car, handheld wireless phones which fit into a person's jacket pocket, pagers, local and wide area computer networks, and facsimile machines, to name a few. Undoubtedly, the number and type of devices and their sophistication will continue to increase over time. Indeed, it is likely that a day will soon arrive when it will be possible for everybody to conveniently and inexpensively be within arms reach of some communication device that enables them to communicate electronically with other people.

The proliferation of different types of communication devices and the increasing diversity of communications media present new challenges. How will communications among the different devices and over the different communications media be coordinated and managed so that people have truly effective access to each other? One challenge is associated with communicating information between and across different communications media. Another challenge is related to handling the inevitable increase in the number of calls so as to maintain accessibility of users. For example, as more people come to rely on their wireless phones to transact business while on the road or away from their offices, their phones are likely to be busy a larger percentage of the time. As a consequence, although a wireless phone can go anywhere with its owner, to the people trying to reach that owner when the phone is in use, the owner will still seem to be as inaccessible as when he did not carry a wireless phone. In addition, the more the owner of such a device uses it, the more likely it will be that he will not know that somebody else was trying to reach him and thus he may miss important calls.

An obvious advantage of many of the new commercially available communications devices is that they offer the possibility of greater mobility to the user. Unfortunately, however, it is not always having to be near the office telephone that ties a business person to the office. The office provides other services that are also important and may not be so mobile. Thus, to fully realize the greater mobility that is offered by the new communications devices and media, these new technologies must be provided in a way that takes into account the business person's dependance on other services besides communications.

Summary of the Invention

The invention described herein is referred to as an electronic assistant. It is a computer-implemented entity that assists a subscriber with his or her communications by carrying out tasks that are delegated to it. The electronic assistant, modeled to have human-like qualities, recognizes speech and performs functions within the familiar model of an office.

Each subscriber who has an account is assigned at least one electronic assistant which may be dedicated or shared. The electronic assistant offers services to both the subscriber and to any contacts or other callers trying to reach the subscriber. A contact is a person, place, or group that the subscriber has described for the electronic assistant. A contact can be another subscriber or an outside caller. Electronic assistants treat both subscribers and contacts as users of the system.

The electronic assistant offers a wide range of services to its subscriber, among which are the following. The electronic assistant can handle incoming calls from several of the subscriber's personal contacts while at the same time it is doing any of its tasks, including reviewing messages with the subscriber, managing information, etc. It can make logical decisions about how to manage, whether to forward, and where to forward the incoming calls from the subscriber's contacts. The electronic assistant can also communicate with the subscriber and the subscriber's contacts using a wide variety of different communication devices, such as telephones, fax machines, pagers, computer terminals, and communications enabled handheld devices (e.g. Personal Digital Assistants otherwise referred to as PDA's). When the subscriber tries to reach a contact or a contact tries to reach the subscriber, the electronic assistant mediates the connection and then remains available to add value to the session. The electronic assistant can schedule and manage reminders for its subscriber. When reminders come due, the electronic assistant notifies the subscriber.

In general, in one aspect, the invention is a method implemented by a computer-based electronic assistant to receive and manage incoming calls to a subscriber. The method includes the steps of: receiving an incoming call to the subscriber from a caller; establishing a first connection between the electronic assistant and the caller; establishing a second connection between the electronic assistant and the subscriber; over the second connection, electronically notifying the subscriber of the incoming call; in response to receiving a call accept command from the subscriber over the second connection, linking the caller and the subscriber so that they may communicate with each other; upon linking the subscriber to the caller, switching the electronic assistant to a background mode in which said electronic assistant continues to monitor the subscriber over the second connection while the subscriber is linked with the caller; and in response to receiving a summoning command, switching the electronic assistant into a foreground mode. When in the background mode, the electronic assistant responds to a first set of commands including at least the summoning command and when in the foreground mode, it responds to a second set of commands. The second set of commands is larger than the first set of commands.

In general, in another aspect, the invention is a computer-implemented method of processing an electronic reminder that is addressed to a subscriber. The electronic reminder includes subscriber-generated content and a specified time at which it is to be delivered to the subscriber. The method includes the steps of: storing the electronic reminder in an electronic database that is accessible to the electronic assistant; when current time coincides with the specified time, detecting that the stored electronic reminder has become due; in response to detecting that the stored electronic reminder has become due, identifying a communications device through which the subscriber can be reached at the specified time; establishing a connection to the communications device; upon reaching an answering party through the communications device, electronically notifying the answering party that the call is intended for the subscriber; electronically informing the answering party that the answering party may accept the call by issuing an accept reply; if the call is accepted by the answering party, electronically delivering the contents of the electronic reminder to the answering party through said communications device.

In general, in yet another aspect, the invention is a method implemented by a computer-based electronic assistant to receive and manage incoming calls to a subscriber. The method includes the steps of: receiving an incoming call to the subscriber from a caller; in response to receiving the incoming call, establishing a first connection between the electronic assistant and the caller; through a dialog between the electronic assistant and the caller over the first connection, determining the identity of the caller; detecting that the subscriber is presently interacting with the electronic assistant through a second separate connection; electronically alerting the subscriber over the second connection that there is an incoming call for the subscriber; electronically identifying to the subscriber the identity of the caller; monitoring the second connection for a response sent by the subscriber to the electronic assistant directing the electronic assistant how to process the incoming call.

In general, in still another aspect, the invention is a method implemented by a computer-based electronic assistant for managing information and connection resources for a plurality of subscribers including a first subscriber and a second subscriber. The method includes the steps of: receiving a call from the first subscriber to the electronic assistant over a communications media; establishing a first connection between the electronic assistant and the first subscriber; identifying the first subscriber as the source of the call; starting up a first session in said electronic assistant which is a thread of execution of code for managing data and performing functions on behalf of the first subscriber; within the first session, receiving a first command sent by the first subscriber to the electronic assistant over the first connection instructing the electronic assistant to perform a function relating to the second subscriber; responding to the first command by sending a first message addressed to the second subscriber and containing information relating to the first command; in response to the first message, starting up a second session which is a thread of execution of code for managing data and performing functions on behalf of the second subscriber and which is separate from the first session; within the second session, receiving the first message and performing a function that produces a result that is responsive to the first message.

In general, in another aspect, the invention is a computer-implemented method of processing communications through a multimedia interface that includes a plurality of interface devices and a plurality of input/output devices. Each of the interface devices is capable of connecting to a different one of a plurality of different communications networks, and each of the input/output devices is capable of processing a different one of a plurality of media types. The method includes the steps of: establishing a channel representing a physical connection to any selected one of the plurality of communications networks through the interface devices; attaching an appropriate subset of a plurality of ports to the channel, wherein each port represents a different one of the input/output devices and wherein the appropriate subset of ports includes ports which correspond to input/output devices that are capable of connecting to the selected communications network; executing an operation that generates an item of information that is to be communicated through the multimedia interface to at least one of the communications networks; retrieving from a memory a multi-media reference to the item of information, wherein the multi-media reference contains a plurality of references to the item of information, each of which refers to the item of information in a different one of a plurality of formats, each of which is of a different media type; passing the multi-media reference to the ports attached to the channel; in response to receiving the multi-media reference at the attached ports, retrieving the item of information from memory in a particular one of the formats identified in the multi-media reference; and passing the retrieved item of information to an input/output device for delivery over the connected communications network. The step of retrieving is performed by one of the attached ports that is capable of processing the format of the retrieved item of information and the input/output device to which the retrieved item is passed for delivery is the input/output device that is associated with the attached port that retrieved the item of information from memory.

In preferred embodiments, the electronic assistant enables the individual to manage and customize his availablilty to friends, family, business associates, customers and strnagers depending upon the time of day, day of the week and his or her needs. In addition, the electronic assistant has the subscriber's schedule and therefore knows where the subscriber is, what he is doing, what his availability is, and how to reach him (e.g. through what communications device). The schedule is used to manage the accessibility of the subscriber to others and his visibility to other subscribers on the system.

Other advantages and features will become apparent from the following description of the preferred embodiment and from the claims.

Brief Description of the Drawings

Fig. 1 shows the electronic assistant and office items;

Fig. 2 shows the hardware platform;

Fig. 3 is a functional overview of the software architecture;

Fig. 4 is a process view of the software architecture;

Fig. 5 is a functional block diagram of the system highlighting the hardware interface to the communications channels;

Figs. 6A-H illustrate the process of completing a task and the objects that are involved;

Fig. 7 illustrates the use of the box, the hand and the finger in manipulating items;

Fig. 8 shows a sample MMUI menu;

Fig. 9 shows a sample MMUI menu with moguls;

Figs. 10A-D show the relationship between memes, menus and moguls;

Fig. 11 illustrates the process of submitting and delivering a message and the objects that are involved;

Fig. 12 illustrates the process of delivering reminders and the objects that are involved;

Fig. 13 shows the contents of a box;

Fig. 14 shows the contents of a user object;

Fig. 15 shows the relationship between user objects and contacts;

Fig. 16 shows the flow of events between the VM and the agent sessions and hardware;

Fig. 17 shows the VM internal objects and their relationship to each other;

Figs. 18A-B illustrate the process of creating a channel and the objects that are involved;

Fig. 19 illustrates the operations for presenting memes and menus to ports;

Fig. 20 shows the functional relationships between the agents and the applications services;

Fig. 21 shows the flow of data in an assistant's session;

Fig. 22 shows the agent class hierarchy;

Figs. 23A-D present an example of a three-part handshake;

Figs. 24A-B is a flow chart of the answer call task;

Figs. 25A-D illustrate the use of a conference object in establishing a call between subscribers;

Fig. 26 is a flow chart of the locate-and-notify task;

Fig. 27 is a flow chart of the notify task;

Fig. 28 shows a "Create-A-Contact" dialog with the electronic assistant;

Fig. 29 shows a "Remind-Me" dialog with the electronic assistant;

Fig. 30 shows a "Remind-Me-To-Call" dialog with the electronic assistant;

Fig. 31 is a flow chart of the handle reminder task;

Fig. 32 shows a "Find" command dialog;

Fig. 33 shows an "I-Will-Be" dialog; and

Fig. 34 shows a "Create-An-Itinerary" dialog.

Description of the Preferred Embodiments

The Virtual Office:

As illustrated by Fig. 1, a subscriber works with an electronic assistant 10 using a "virtual office" as a model. The electronic assistant 10 works in an office containing the subscriber's objects, which are called "items". An item is a piece of information that the electronic assistant stores in a database and works on for the subscriber. The subscriber can use spoken or touch-tone commands to have the electronic assistant work on various items, and the electronic assistant then uses a dialog to gather the information it needs from the subscriber to complete the task.

An item may be any one of the following: a schedule 12, a contact 16, (e.g. a person 18, a place 22, or a group 20), a message 14, a reminder 24, a phone book 26, or trash 28. Messages can include other items, such as a contact, reminder, page, etc. In derivative implementations, a message could be any sort of multimedia or composite information.

A "schedule" lists where the subscriber can be reached and at what time. There is a default schedule and an override schedule. The default schedule is the subscribers daily or weekly schedule; it is entered by a system administrator. Subscribers can override the default schedule with a "I-Will-Be" or "Create-an-Itinerary" commands to create an override schedule (see Exhibit A at the end of the specification for a list of spoken and touch-tone commands that an electronic assistant implements). The "I-Will-Be" command enables the subscriber to indicate when and where he can be reached and his availability at those times.

A "contact" is the subscriber's view of a person, place, or group. The subscriber can use a contact as the recipient of a command, such as placing a call or sending voice mail. A contact is analogous to an entry in an address book file. An "outside contact" is a contact that is not a subscriber.

A "person" is a representation of a person, which describes another subscriber or someone outside the system. The description includes the spoken name, spelled name, priority, gender, and a list of places where the person can be reached.

A "place" is a location with a single address (e.g. a phone number, fax number, network id, etc.) associated with it. Each person can include a standard set of places: work, home, car, mobile, and other. A corporation, department, or other organization is also considered a place, and can include a phone number and fax number.

A "group" is a user-defined set of persons, places, and/or groups.

A "message" is a piece of information that is addressed to a person or group. The most common type of message is voice mail.

A "reminder" is a notice to be delivered at a future time. The reminder can be a notice to call a contact about a particular subject ("call reminder") or a notice containing a recorded audio message about another subject ("recorded reminder").

A "phone book" is a list of other system users that is published for easy access by subscribers.

The "trash" is a collection of all the items that the subscriber has thrown away.

In the described embodiment, the electronic assistant can recognize specific words or phrases, which are called utterances. To recognize a subscriber's speech, the electronic assistant gathers training utterances from the subscriber, which are recordings of the way the subscriber pronounces a word or phrase. The process of gathering these training utterances is sometimes referred to as training in that the subscriber is training the electronic assistant to recognize his voice. For example, the electronic assistant may ask the subscriber to say his full name several times or the name of a command several times. These utterances are then compiled into a compressed format known as a vocabulary. A "vocabulary" is a finite set of recorded words and phrases that can be used directly by the system's speech recognition hardware to recognize the subscriber's speech.

The electronic assistant uses two different kinds of vocabularies, namely, a speaker-dependent vocabulary and a speaker-independent vocabulary. The "speaker-dependent vocabulary is used to recognize the speech of users who have explicitly trained the electronic assistant with their own pronunciations. The speaker-independent vocabulary is used to recognize a multitude of different voices without requiring each user to train the system.

In addition to these different kinds of vocabularies, the electronic assistant uses two different methods to recognize speech, namely, continuous recognition and discrete recognition. Continuous recognition recognizes naturally spoken words or sequences of words, that is, words without artificial pauses between them. In the described embodiment, there are two vocabularies recognized with continuous recognition: numbers (the digits "one" through "nine", "zero", and "oh") and yes/no ("yes" and "no"). Continuous recognizers also discriminate against words like "um" and "ah". Discrete recognition recognizes discrete, isolated words or phrases, or sequences of distinctly spoken words that are separated by pauses.

The electronic assistant responds to a variety of commands (see Exhibit A). Some commands operate on the current context in the hand (described later), while others do not. Some commands start a dialog between the electronic assistant and the subscriber so the assistant can gather the information it needs to carry out the task. For example, to call a contact, the subscriber can use the "Call" command:

Subscriber	Assistant
Call	Call Whom?
Bill Bishop	Where?
Work	Dialing...

Functional Description of the User Interface

Before going into the details of the hardware and the software architecture of the system, a description of how the system handles an incoming call to a subscriber will first be presented. This will provide the context for then describing the underlying mechanisms that are implemented within the system to achieve their functionality.

Throughout the remainder of this specification except where noted, it will be assumed that the subscriber is named John Smith and the caller is named Bill Bishop.

When a caller calls into the system in an attempt to reach a particular subscriber (i.e., by calling a telephone number that is assigned to that subscriber), the system answers the call. By playing back audio signals that are stored in the system's database, the system announces to the caller:

> Good Morning, I'm the electronic assistant for John Smith. Please say your full name.

The caller replies by stating his name:

> Bill Bishop

The system records the caller's utterance and using a speaker dependent dictionary that is stored within the system's database, it attempts to recognize the caller. If the system cannot recognize the caller's name, it responds by playing the following message:

> Using touch-tones, enter your area code and phone number followed by the # key.

The user enters his telephone number as requested. The system using decodes the touch tones and then searches a contact list for the subscriber to find a contact with that telephone number.

If the system succeeds in recognizing the caller on the basis of his phone number, it then plays the following message to the caller:

> Would you like to take a moment to teach me how to recognize your name better?

The caller may accept this offer by pressing "9" on his touch-tone key pad. The system then responds by playing the following message:

> Please repeat your full name.

After repeating this operation a second time, the system stores the vocalizations with the identity of the contact. The next time the caller contacts the system, it will use the stored vocalization of the caller's name to recognize the caller.

Once the assistant either recognizes the caller either through a match with a stored vocalization or through the caller's phone number or labels the caller as unknown, it then attempts to locate the subscriber. It does this by carrying out a sequence of operations the first of which is to check the subscriber's status. If the subscriber currently has a connection established with his assistant (and he has not enabled a do not disturb function), then his status is available. If the subscriber is not connected, then the assistant may check a secondary information source (such as a cellular network) to determine the subscriber's availability. Finally, the assistant will check the subscriber's schedule. The subscriber can set his availability to indicate that he is accepting all calls, he is accepting no calls, or he is accepting only important calls.

If the subscriber is not accepting any calls, the system plays the following message to the caller:

> Sorry, he's not available. Please leave a message and then hang up or press the "#" key. Recording.

After the caller has finished, the system may send a non-interactive notification to the subscriber that the caller has just tried to reach him. It does this by, for example, sending a page through its pager interface or sending an E-Mail message to the subscriber's workstation. The notification identifies the caller and it indicates whether voice mail was left.

If the subscriber is accepting only important calls, the system checks the subscriber's contact list to determine what the caller's priority is. The subscriber can designate the contact as either high priority or normal priority and this information is stored with other information about the contact. If the caller has only normal priority, the system reports that the subscriber was not available and offers the caller the option to leave voice mail, as previously described. On the other hand, if the caller has high priority, the system continues its attempt to locate and notify the subscriber of the call.

In the event that the subscriber is accepting all calls, the system continues its attempt to locate and notify the subscriber of the call without regard to the caller's priority designation.

As a first step in locating the subscriber, the system determines whether the subscriber is already connected to the system, either through another call or through some other communications medium (e.g. logged into his computer). If the subscriber is on another call being handled by the system, the system briefly interrupts that call to notify the subscriber that he has a call waiting and it identifies the name of the caller. If the caller is also logged onto the system through his computer, the system may also send a visual message to the workstation notifying the subscriber of the call and identifying the caller.

The subscriber then has the option of accepting the call, asking the system to place the caller on hold while he completes his present call, or asking the system to take a message.

If the subscriber accepts the call, the electronic assistant responds by immediately establishing a connection between the caller and the subscriber. If the subscriber instructs the system to take a message, the system offers the caller the voice mail option previously mentioned. If the subscriber instructs the system to place the incoming call on hold, the system informs the caller that subscriber will be with him shortly. When the subscriber has completed his other call, using spoken commands, he instructs the system to then establish a direct connection with the new caller.

If the system determines that the subscriber is not presently on a call but is connected to the system or reachable through his workstation or other two-way data device, it may send a message that is displayed on the device's display screen. The message includes a menu which offers the subscriber the option to accept or not accept the call. If the subscriber accepts the call, the system notifies the caller that the subscriber will be with him shortly. In the meantime, the subscriber calls into the system over another phone and the system connects him with the caller.

If the subscriber does not accept the call or does not respond within some predetermined period of time, the system notifies the caller that it was unable to locate the caller and offers the caller the option to leave voice mail.

If the electronic assistant does not detect the presence of a subscriber on the system, the electronic assistant checks whether his location is indicated on one or more schedules that the system keeps for the subscriber. If there is a schedule that places the subscriber at a particular location at that moment and there are phone numbers identified with that location, the electronic assistant places a call to one of the phone numbers. Additionally, the assistant may check one or more additional sources of subscriber location information (such as cellular network databases).

If a party answers the call, the electronic assistant plays the following message to the answering party:

> Hello, I'm trying to reach Jim Smith. If he is available, press the 9 key. If he is not available, press the 6 key or hang up.

If the party answering the call indicates that Jim Smith is available, the electronic assistant then says:

> There is a call from Bill Bishop. Do you want to take the call? Indicate Yes by pressing the 9 key, indicate No by pressing the 6 key.

If the party indicates that they will accept the call, the electronic assistant connects the caller to the subscriber.

In the event that electronic assistant is unable to establish a connection with the subscriber, the electronic assistant may send a non-interactive notification to the subscriber indicating that Bill Bishop has called him at a specified time.

Hardware architecture

Fig. 2 shows the basic hardware components of the described embodiment. The system consists of a high-performance 486 computer equipped with an ISA bus 40 with a passive backplane. The computer includes a CPU card 42 and display adapters (not shown). The passive backplane is a standalone bus that is not part of the CPU card (i.e., the motherboard). A set of ISA adapters (not shown) plug into the passive backplane to form 486 computer system. Interface cards 44 and the CPU card connect directly to the ISA bus. The passive backplane can hold up to 20 interface cards.

The interface cards are special-purpose cards to support many different forms of connectivity and communication. They include network cards to connect with standard digital telephone lines as well as special-purpose adapters for recognizing speech, making phone calls, and sending and receiving faxes, etc.

A Multi-Vendor Integration Protocol (MVIP) bus 46 consisting of a 40-pin ribbon cable is connected to all of the interface cards providing telephone services. The MVIP bus is a high-speed communications channel that carries all audio traffic between interface cards and switches telephone lines.

Finally, the system also has fixed and removable storage including a set of high-capacity, high-speed disk drives 48 and a floppy drive 50.

A base system, supporting 8 ports, has 2GB of disk space and 32MB of memory with an additional 1GB of disk space and 8MB of memory for each additional set of 8 ports. The described embodiment, supports a maximum of 24 ports.

In the described embodiment, the following specific hardware is used. The line interface cards are either Natural MicroSystems DTI-48 T1 cards supporting connection of two T1 trunks or Voice Technology Group Voice Bridge PC PBX cards supporting 8 lines of PBX station set emulation. The line processing cards are Natural Microsystems AG-8 cards, each supporting 8 telephone channels. The ASR daughter cards are Natural MicroSystems DB-31's and there is one DB-31 card for each AG-8 line processing card. The daughter card performs speech recognition on names only. Another ASR card which is provided is a VPro-84 from Voice Processing Corporation. There is one VPro-84 card for each AG-8 line processing card. The VPro-84 card performs speech recognition on commands and digits and supports up to 8 discrete recognizers or 4 continuous recognizers.

The MVIP bus supports up to 256 full-duplex telephone connections. The 256 full-duplex connections are time-division multiplexed (TDM) so that only 32 separate signals are transmitted, with each signal divided into 8 time slots. Interconnections between two communications channels is accomplished by enabling through software control each of the channels to have access the time slots of the other channel. That is, the input of each channel is permitted to listen to the output of the other channel.

Other Possible Hardware Configurations

The description above presents one possible architecture which concentrates a number of hardware elements within a single computer chassis, including multiple special purpose interface cards connected together with a special purpose bus. Many other hardware organizations could also be used to support the functionality described. The basic abstract elements needed are: (1) one or more basic computer resources to support the program and data as described; (2) support for one or more incoming communications channels and support for receipt and generation of connections on the respective channels; and (3) a switching resource to switch multiple communications channels together

It is possible for these resources to be distributed across multiple systems. For example, in one possible implementation a separate switch resource could be connected to computers which contain hardware for managing communications channels. These computers which manage the communications channels are then connected via a network to larger systems which run the applications and provide database services. The applications and database services could be further split up across multiple systems.

In yet other hardware configurations, one might wish to consolidate the resources even more than is described for the present embodiment. For example, a single computer could have a plug in card, or support on the mother board, to handle the communications channels. Switching could be done between these channels in hardware or software. In this way the applications and database would run on a single computer with the necessary hardware support to manage all necessary communications channels.

Software architecture

This following description presents two different views of the system's software architecture: one emphasizing functional components and the other emphasizing processes and events.

Functional View:

Fig. 3 shows the high-level software architecture of the system. In this and subsequent illustrations of processes, the structure objects are represented symbolically by icons which have an appearance relating to the function of the entity being represented.

The system includes four primary components. One primary component includes assistants 60 and agents 62. The assistants carry out tasks on behalf of users and the agents carry out tasks on behalf of assistants. Another primary component is a communication mechanism including a Multimedia User Interface (MMUI) 64 and a parcel mechanism 66. These allow assistants and agents to communicate with other system components. A third primary component is an object database 68 which stores user information, such as contacts and messages, and system information, such as prompts. And the fourth primary component is a Virtual Machine (VM) 70 which services requests from assistants and agents as well as hardware devices. The arrows in Fig. 3 show the primary paths for the flow of interactions between the various components.

The described embodiment uses the Univel Unixware™ operating system 72 which is based on the UNIX System V Release 4.2 operating system. The object database is the ObjectStore™ object-oriented database by Object Design, Inc.

Other Possible Software Configurations

In the described embodiment, the VM, Assistants, Agents and Database all reside on one host computer. There is nothing in the architecture, however, that necessitates this. Other implementations could separate these components and have them run on separate computers using the previously mentioned remote procedure calls to communicate between the different processes.

Process View

Fig. 4 summarizes the process architecture of the system. The system consists of a Virtual Machine service process 80 (vmserver) and one process 82 for each instance of an electronic assistant or agent. Each session process communicates with the vmserver using RemoteProcedureCalls (RPCs) 84. The vmserver services these RPCs from the session processes and also services events 86 from hardware devices 88.

The Interfaces to the Communications Media:

Fig. 5 is a block diagram view of the system highlighting the hardware interface to various communications channels. A central office 90 which receives incoming calls from telephones 92 and fax machines 94 forwards them over a T1 line to a T1 interface card 96 in the system. The T1 interface cards are connected to the MVIP bus 46 along with other cards including a line processor card 98, a speech recognizer card 100 and a fax board card 102. The line processing card does coding and decoding of speech, i.e., it plays audio that is stored in the database and records audio for storage in the database. It also decodes touch-tone signals (DTMF). The speech recognition card performs the speech recognition function using a vocabulary supplied to it from the database. The vocabulary might be a speaker dependent vocabulary generated by the user or a speaker independent library.

There may be one or more T1 lines coming into the system. It is likely, however, that there will be many more phone numbers assigned to the system than there are T1 lines. Numbers are not mapped to particular lines. Rather, when the central office receives a call intended for one of those numbers, it simply selects from among whatever lines are available at the time and sends a ring signal to the system over the selected line. When the interface card for that line answers the incoming call (i.e., connects to the line), it receives from the central office a sequence of touch tones (four digits) identifying the extension that is being called. Using this four-digit sequence identifying the extension and the subscriber's numbers stored in its database, the system is able to identify the subscriber for whom the call is intended.

In the described embodiment, the system can also establish connections to a Wide Area Network (WAN) or a Local Area Network (LAN) 104 through an ethernet card 106. In addition, it can establish connections to various other communications devices through one or more serial interface cards 108. In Fig. 5, serial card 108 has two serial ports, one of which is connected to a SkyTel™ system 110 over one serial line and the other of which is connected to a two-way modem 112 over another serial line. The SkyTel™ system 110 transmits one way communications to pagers 114. The two-way modem 112 exchanges two-way communications with wireless devices 116 such as PDA's (personal digital assistants).

Fig. 5 also presents a logical view of the mechanisms which the system uses to connect to communications channels through the interface cards. To connect the system to different communications media, the system establishes various channels or communications paths including a phone line channel 120, a fax channel 122, a TTY line channel 124, a two-way data line channel 126, and a batch pager channel 128. For each channel, there is a set of ports that can be attached to it. The ports, which are represented in software by port objects, refer to input/output devices supported on the interface cards.

In the software, each of the line channels is represented by an object, i.e., a data structure which identifies the physical line to which that particular channel maps. The data structure also includes a list of the ports which can be attached to the channel. Each port is represented by a port object and is supported by the functionality found on the interface cards that are connected to the system. That is, each port represents a logical digital signal processor on the particular interface card which implements the functionality associated with that port.

Fig. 5 shows the ports that can be attached to the various channels. The phone-line channel can have a DTMF port 130, an audio-in port 132, an audio-out port 134, and an ASR (automatic speech recognition) port 136. A fax channel can have a fax-in port 138 and a fax-out port 140. Both the TTY line channel and the two-way data line channel can have a text-in port 142 and a text-out port 144. The batch pager channel can have a text-out port 144.

For the phone line channel, the DTMF port represents the capability of receiving and interpreting DTMF signals sent to the system over the T1 line by the caller. The audio-out and audio-in ports represent the capability on the line processor cards to both generate and record audio over the communications channel. The ASR port which is implemented by the speech recognition card performs the automatic speech recognition on the audio.

Note that throughout this description the term "call" is used in its most general sense. Not only does it include a call placed over the telephone lines but it also includes the initiation of any contact over any of the other communications media including wireless communication channels, computer networks, fax channels, etc. Thus, the concept of a call is not meant to be limited only to a telephone call.

While the described embodiment includes a limited number of channels and ports, the architecture can be expanded to handle new channels or ports providing for future forms of connections and capabilities.

The four primary software components will now be described in greater detail.

Assistants and agents

The system is designed to support many different kinds of agents. All agents and assistants are based on a generic agent object. Before exploring the specific agents used in the system, this generic agent will first be described.

Agents, sessions and gadgets:

An *agent* is a software entity that performs an action or brings about a certain result on behalf of a user or another agent. To communicate with users, an agent engages in dialogs using *gadgets*. A gadget is a representation of a communications device, such as a phone, fax machine or pager. A gadget includes a description of the communications device as well as its address. For example, a phone gadget contains the area code, number, and extension for a particular telephone. To communicate with a specific user over the telephone, an agent uses the phone gadget for that user.

To use a gadget, the agent adds it to a session. In the case of a phone gadget, the VM dials the phone number and returns an *active gadget* to the agent's session. An active gadget represents the connection from an agent's session through a communications network to a communications device. The active gadget represents a transient connection to the persistent gadget. An active gadget can be used for communicating with users, while a gadget itself cannot because it just stores the address and capabilities of the device. For example, a phone gadget specifies the number 617-555-1212, while the active gadget represents an active phone line on which the number has already been dialed. A session consists of a collection of zero or more active gadgets connected together.

To communicate with users, agents may need to add capabilities to active gadgets. A *capability* is a representation of an ability or feature of a device, such as the ability to recognize speech or play audio. For example, if an agent needs to fax materials to a user, it adds the fax capability to the active gadget. A discussion about how the VM handles capability requests from agents is presented later in the discussion of the VM. For other embodiments that include a graphical interface, a capability that outputs graphics and input capabilities that track the user's focus (i.e., mouse, eye, etc.) and gathers input from the user is envisioned.

Agents can work with many different kinds of active gadgets simultaneously. For example, an agent can be simultaneously placing an outgoing call, sending a message to a pager, and sending a message to a fax machine.

User agents, tasks, and presenters:

A *user agent* is an agent that is capable of communicating with humans as well as with gadgets. User agents use media-independent dialogs to communicate with users. These dialogs consist of a prompt from the user agent and a response from the user. When a user agent is communicating with a user, it directs its dialogs to an active gadget known as the *focus gadget*. For example, if a user agent is placing a call for the user with one active gadget and talking to the user with another active gadget, the latter gadget is the focus gadget. Other embodiments may group a set of gadgets to be used as the focus gadget. This would allow the assistant to interact with the gadget set and have the interaction span all gadgets in the set.

User agents can also connect active gadgets together for phone conversations. When the user agent places an outgoing call for the user, and the called party answers the phone, the user agent connects the two active gadgets together and then goes in the background. When the user agent is in the background, it is idle; to delegate additional tasks to the user agent, the user needs to bring the user agent back into the foreground.

A unique feature of user agents is their ability to handle tasks. A *task* is an action to be carried out by a user agent on behalf of a user. A task may consist of a form full of fields that the agent gives to a presenter. A *presenter* is an object that knows how to engage in a dialog with a user over a given medium as part of filling in the fields of a task. An agent gives each task to a presenter. The presenter gathers information for a field and then hands the task back to the agent. The agent looks to see if a field changed recently and examines it. For example, the agent may request information from the database and adjust the next field in the form. Then the agent hands the task back to the presenter, and it gathers information for the changed field. This activity of handing the task back and forth between the agent and the presenter continues until the agent is satisfied that all required fields are complete. Then, the agent executes the task.

Figs. 6A-H illustrate the process of assigning a task and monitoring its progress. In this example, a user agent 150 has obtained a focus active gadget 152 for its session 154 and a user 156 has issued the "call" voice command (Fig. 6A). In response, the agent assigns a call task 158 to a phone presenter 160 (Fig. 6B). The presenter interacts with the user over the focus gadget to fill in the fields of the task (Fig. 6C). The presenter gathers the name of the contact to call and places it in the first field before handing it back to the agent (Fig. 6D). The agent notices that this field changed and looks up the contact in the object database 68 (Fig. 6E). The agent adjusts the second field of the form to include the valid places defined for this contact and hands the task back to the presenter. The presenter gathers the location where to call the contact and places it in the second field before handing it back to the agent (Figs. 6E-F). Once the fields of the task are filled, the user agent places the call (Fig. 6H).

A user agent obtains the list of tasks to carry out from an object known as the task stack. When the agent receives parcels (to be described shortly), it may decide to place tasks on the task stack as a result of processing the contents of the parcel. The user agent gets the next task from the stack and carries it out. It repeats this process until there are no more tasks on the stack, and then it exits. In the process of carrying out a task, another task may be pushed on the stack.

A task is not the only way that an agent interacts with a user. For simple interactions, the agent uses a C++ method. Tasks are used primarily for complex interactions.

Assistants and Electronic Assistants:

An *assistant* is a user agent that is capable of making logical decisions and performing complex tasks on behalf of its users. The entity that will be referred to hereinafter as the *electronic assistant* is an assistant that assists users with their calls, messages, contacts, and schedule. Each subscriber has an electronic assistant. As an example of a complex task that distinguishes the Electronic Assistant from an ordinary user agent, the electronic assistant can locate a user by consulting the user's schedule and deciding which numbers to call.

Up to this point, three different classes of agents have been discussed: agents, user agents, and assistants. Table I summarizes the distinctions among them.

Distinctions Between Agents and Assistants

Agent Type	**Description**
Agent	A software entity that performs an action or brings about a certain result on behalf of a user or another agent.
User agent	An agent capable of communicating with humans using media-independent dialogs and carrying out tasks on a user's behalf.
Assistant	A user agent capable of making logical decisions and performing complex tasks.

TABLE I

Any agent can logically run several sessions simultaneously. Each time an agent handles a session, there is a separate process running for each session. The *master* *session* is the session in which the electronic assistant is talking to its subscriber or trying to locate its subscriber. There can never be more than one master session running per subscriber.

Box, Hand and Finger:

Referring to Fig. 7, the electronic assistant uses three different objects to manipulate its user's items, namely, a box 170, a hand 180, and a finger 182. The *box* is an object that contains all the items belonging to a user. The *hand* is an object that holds the items that the user is manipulating. The *finger* is a software pointer that marks the currently selected item in the hand.

When the subscriber asks the electronic assistant to "Find" items, such as contacts, the electronic assistant looks in the box for the items and then picks them up in its hand. The hand can pick up a subset of the items in a box according to certain criteria. For example, when the subscriber asks the electronic assistant to find saved messages, it only finds the messages that are marked with a saved flag.

When the user asks for the first item, the electronic assistant moves its finger to that item and selects it. If the user asks for the next item, the electronic assistant moves its finger to the next item in its hand. The items in the hand can be accessed in a circular fashion. If the finger points to the last item in the hand and the user asks for the next item, the finger moves around to the first item in the hand.

Sequential vs. Random Access to Lists of Information

Using first-item and next-item provides the ability to browse a set of items in a sequential fashion. So, if a subscriber instructed their assistant to Find New-Message, then using next item and previous item would give the ability to traverse through the new messages in forward or reverse order. First item would return the finger to point to the beginning of the list.

In addition to this form of sequential navigation through items in a list, the system also provides capabilities to access information randomly or based on more complex queries. For instance, a subscriber can ask to see all the new message from a particular contact. The following dialog:

```
Find                  <find what>
New-Messages-From     <new message from whom>
Bill-Bishop
```

would put into the hand all messages that had been received from the contact named Bill Bishop. This feature can also be extended to the group items which a subscriber can create. As a result, if a subscriber has a group called Hot-Prospects, the above dialog could be repeated as:

```
Find                  <find what>
New-Messages-From     <new message from whom>
Hot-Prospects
```

and the result would be a list of items that includes new messages from members of the group Hot-Prospects. In addition to searching for items from a particular contact, this technique can be used to fetch items that have been stamped as priority or urgent, etc. Finally, all of the items described herein can be randomly searched for and collected into the hand based on different attributes.

Context:

The ability to Find items and gather them into the hand introduces into the system a concept of state. This state includes the current contents of the hand and what is being pointed to. Since some number of commands may need to be issued to change the state, and since it is often desirable to go back to one of the previous state, the system supports an ability to "Go-Back" to the previous states. Each time a change in the state of the assistant occurs, the new state is stacked on top of the previous state. Issuing the Go-Back command pops the previous state of the stack.

For example, let's assume the assistant is holding and pointing to a contact named Bill Bishop and the subscriber issues the following commands:

```
Find            <find what>
New-Message     <one new message from Bill Bishop>
What's-It-Say

GoBack
```

In this example, the subscriber replaced the contents of the hand (which had been holding a contact for Bill Bishop) with a new message from Bill Bishop. After listening to the message (as a result of issuing the What's-It-Say command) the subscriber said Go-Back. The result of this command is that the previous state of the hand (the single contact for Bill Bishop) is restored. The subscriber can now manipulate this contact for Bill Bishop.

Another useful tool for managing state is the ability to refer to the current item being pointed to in dialogs. The utterance "This-One" is used to refer to an object currently being pointed to by the assistant in the hand. The item is also referred to as "it" so that commands such as "Describe-It", "Update-It", "Throw-It-Away" also refer to the current item. Finally, the commands "Send-A-Copy", "Send-A-Reply" and "Give-Them-A-Call" are operations on the current items to which the assistant's finger is pointing. These draw on the context and leverage on a shared understanding between the subscriber and the assistant of the context. The result is a set of condensed dialogs that do not need to explicitly refer to objects which can be inferred. Other embodiments can take advantage of pronouns such as "him", "her" and "them" to refer to objects that have recently been referenced.

Parcels

Agents can communicate with each other using parcels. A *parcel* contains the address of the sender (From), the address of the recipient (To), and contents (a persistent object). The contents of a parcel can also be another parcel.

The VM guarantees delivery of parcels. Any parcels addressed to a user are delivered to the master session for that user. If no master session exists, the VM starts one. Then, the electronic assistant tries to locate the user from within the master session. In other embodiments, parcel addresses can support distributed boxes and may support communication between remote agents.

Agent to Agent Communication

The fact that the VM acts as the routing and delivery mechanism for parcels is powerful. It allows all agent to agent communications to be mediated through a mechanism that guarantees the behavior that a recipient of a parcel will be either located or started to receive the parcel. In the described embodiment, the agent to agent communication exists as communication between two agents on a single box. In other embodiments which support distributed systems, the VM can use routing information embedded in the (To) address to route the parcel to the appropriate remote or clustered system. An agent running on one coast could check to see if an agent for a user across the country wants to accept a call, and all the messaging could be happening across SS7 (signaling system-7 - a phone network) or some other wide area network without the need to do a call setup.

MMUI:

The MMUI is a media-independent interface for communicating with users. The MMUI allows agents to focus on the content to be communicated rather than the format of the content and the details of using specific devices in presenting content.

The fundamental building block of the MMUI is an object referred to herein as a meme. A *meme* is a media-independent reference to a piece of information. The meme contains a set of media objects which store the information in a variety of different formats. A *media* object is a piece of information, such as a sound, text string, or DTMF sequence, that can be presented to a particular kind of communications device.

Agents use memes and menus as part of their dialogs with users. These dialogs can be part of a task or outside of a task, such as from a C++ method. For example, as part of a call task, an electronic assistant needs to ask the user who to call. Within a field of the call task is a "Call Whom?" meme, which the presenter gives to the active gadget. The "Call Whom?" meme consists of several different media objects: for example, an audio recording for use on the telephone, or a text string for displaying on a computer monitor. The active gadget passes the meme to the VM, which selects the type of media based on the capabilities of the active gadget. For example, if the active gadget is a telephone, the VM selects the audio recording in the meme, and the user hears something like "Call Whom?".

The type of gadget is the primary factor but not the only factor that affects what type of media is used from a given meme. Users can set preferences for their electronic assistants, and system administrators and system integrators can set system-wide preferences that apply to all electronic assistants. For example, a user may select verbose prompts and a male voice for an electronic assistant. The system integrator may have set up the system with Spanish as the default language. These preferences are known as attributes in the MMUI. An *attribute* is a name/value pair. Typically, an attribute is used to tailor the behavior of an active gadget and of all MMUI elements.

When an electronic assistant obtains an active gadget from the VM, it sets the attributes on it. Later on in the session, when the electronic assistant sends memes to the gadget (in this example, as part of a task), the attributes and the type of gadget determine what media object in a meme is used. For example, in a bilingual voice system, memes may have two audio media objects: one with the information recorded in French, and another with it recorded in English. If the user had set the language attribute to French and the gender attribute to male in the previous example, the resulting prompt would be spoken in French with a male voice.

Another important building block in the MMUI is the midget. A *midget* is an object used for constructing a multimedia dialog. A midget in a multimedia user interface is analogous to a widget in a graphical user interface. The most common type of midget is a menu. Referring to Fig. 8, a *menu* 180 is a set of choices that can be presented to a user using multiple media. A menu consists of rows describing each choice (a meme 182) and columns describing the media 184 that can be used to present the meme in each row. An additional column optionally stores a pointer 186 to the referenced data object in the database e.g. a contact object. In some cases, columns are blank if it is not possible or appropriate to present the meme using that type of media.

In a typical dialog, the electronic assistant passes a meme and a menu to the active gadget. The meme is presented to the user, the user's response is matched against the rows in the menu, and the menu row selected (known as the *menu pick*) and the data pointer (if present) are returned to the electronic assistant. For example, in the case of the call task, the electronic assistant passes a meme for presenting the question "Call Whom?" and a menu containing the user's contact list. In this example, the MMUI menu has four columns. The first column contains the text spelling of the contact name; the second column contains a string describing the DTMF sequence that can be used as a shortcut for the contact name; the third column contains the vocabulary used for recognizing the spoken contact name; and the fourth column contains a pointer to the contact's information in the database.

Although this example menu has four columns, menus can have more columns to accommodate different attributes. For example, in a command menu, there could be verbose and terse versions of the spoken command or versions spoken in different languages.

Some menus contain additional objects, called moguls, attached to them for managing complex media types. A *mogul* is an object that manages a particular type of media. A mogul can store media-specific information. For example, an audio mogul is attached to the audio media column of a menu and stores the speech-recognition vocabulary to be downloaded on the recognizer hardware. A mogul can update the content of certain media in the memes in a menu when updates occur to other media in the memes. for example, a DTMF mogul updates DTMF media objects whenever their text media counterparts change. In Fig. 8, if the subscriber changes the spelling of Susan Schmidt to Susan Smith, the DTMF mogul would change "773" in column two of the menu to "776". Media that are updated by a mogul instead of created by the user are called *mogul-generated media*. DTMF media is an example of mogul-generated media.

The algorithm to generate DTMF for a name or command is as follows: DTMF commands are usually at least three digits. If there is one word, then map the first three letters to the corresponding telephone key that has those letters. If there are two words then map the first letter of the first word and use the two first letters of the second word. If there are three words then take the first letter of each word. In the case where three letters is not enough because the command is still not unique, then continue to take the first letter of subsequent words until you have a unique sequence.

Fig. 9 shows the contact menu with the DTMF mogul 188 and audio mogul 190 attached to it. Moguls form a sort of third dimension to menus; they can be thought of as the depth of a menu.

Figs. 10A-D show the "Call Whom?" example previously discussed. First, the presenter 160 passes a meme 182 and a menu 194 to the active gadget 152 (Fig. 10A). The menu has two moguls attached to it: a DTMF mogul 196 and an audio mogul 198. When the VM receives the menu, it unpacks the vocabulary stored in the audio mogul and downloads it on the recognizer hardware. Since the active gadget is a telephone with speech recognition capabilities, the subscriber hears the electronic assistant say "Call Whom?" (Fig. 10B). When the subscriber responds with "Susan Schmidt" (Fig. 10C), the recognizer hardware uses the vocabulary to find a match in the contact menu and returns the menu pick to the electronic assistant (Fig. 10D).

Memes are often strung together in a *meme* list to form a complete statement. The agent or presenter can pass a single meme or a meme list to the active gadget. When a meme list is passed to the active gadget, the memes are played in the order in which they are listed.

Utility agents:

The parcel mechanism provides a way of communicating between agents and sessions. Utility agents carry out actions on behalf of the electronic assistant, such as delivering messages, scheduling reminders, and answering unassigned phone lines. There are several different types of utility agents including a postmaster agent, various courier agents, a cron agent and a secret agent.

The *postmaster agent* is an agent that receives messages from electronic assistants and distributes them to appropriate courier agents for delivery. The *courier agents* are agents that receive messages from the postmaster agent and deliver them to their destination. The *message store* is a portion of the object database containing the incoming and outgoing messages for all persons.

Referring to Fig. 11, when Susan's electronic assistant 156 submits a message 200 for delivery, the parcel mechanism packages a reference to the message in a parcel 202 and sends it to a postmaster agent 204. The parcel mechanism also inserts the message into the appropriate user's section of the message store 206. In return, the postmaster agent 204 packages a chit 208 in a parcel 210 and sends it to the electronic assistant. A *chit* is a receipt issued by the postmaster agent, which an electronic assistant can use to check on the delivery status of the message.

As the postmaster agent receives messages for delivery, it selects the appropriate courier agent 212 that should deliver it. Different gadgets require different types of courier agents. For example, the system courier agent delivers messages for electronic assistants, and a SkyTel™ courier agent delivers messages to SkyTel pagers. When the postmaster agent gives a message to the system courier agent, it sends a "wake up" parcel 214 to John's electronic assistant 216 to notify it about the new message. The user's electronic assistant retrieves the message from the message store.

If the courier agent could not deliver the message, it returns the parcel containing the reference to the message back to the postmaster. Depending on the number of retries permitted for the message, the postmaster may ask the courier agent to retry the delivery or mark the delivery as failed.

Cron agent:

The *cron agent* is an agent that receives reminders from agents, tracks them until they are scheduled for delivery, and then delivers them to agents. The cron agent may also track other items. Therefore, the parcel can actually contain any database object, not just a reminder.

Referring to Fig. 12, when an electronic assistant 150 submits a reminder for delivery, it creates a parcel 220 containing a reminder 222, and the parcel is both from the user and to the user. Next, it takes this parcel and places it in another parcel 224, which is addressed to the cron agent from the user. This latter parcel is submitted to a cron agent 226. When the cron agent receives this parcel, it unpacks the parcel inside it. It then places the parcel in a queue 228 in time-sorted order with other parcels and keeps track of when this reminder should be delivered. When the time comes, sends this inner parcel back to the electronic assistant.

Object database

The *object database* is an object-oriented database, i.e., it is a database that maintains object structures and relationships directly rather than flattening them and reconstructing them. The object database stores all the information that both users and the system need across sessions. For example, if a user creates a new contact, the electronic assistant stores it in the object database. When the electronic assistant speaks to a user, it is using prerecorded media objects that are stored in the object database.

The object database stores a user's information using three different objects: a box object, a message store object and a user object. The box object and the message store were previously mentioned. The *user* object describes a subscriber or person.

Referring to Fig. 13, for each subscriber's assistant, the object database stores a box 230 containing: a pointer 232 to the user object 234 describing that subscriber; a *contact list* 236 which is a MMUI menu containing all of the user's contacts; and a pointer 238 to the section of the message store 206 containing the user's messages. The contact list is used with electronic assistant commands, such as "Call" and "Send Voice Mail", which expect the user to specify a contact.

A stored contact can also include a pointer to a note (e.g. a voice message) which the subscriber can generate and attach to the contact. When the subscriber instructs his electronic assistant to call the contact, the subscriber can also have the electronic assistant play back the attached note while the electronic assistant is attempting to establish the connection. The note might include information about the contact which the subscriber wishes to be reminded of whenever he calls that contact. For example, he may wish to know the name of contact's secretary so that he can address her by name if she answers the phone.

Referring to Fig. 14, the database maintains a single definition for each user in the system, which is known as a user object 234. The user object specifies the user's password, gender, and schedule. It also contains a pointer 240 to the contact list 236 in the box 230, a list of places 242 (i.e., home, work, car, mobile, other, pager), a list of groups 244 and a list of any reminders 246 that the subscriber has generated.

In the database, a person contact consists of a reference to a user object plus local information, such as the way the user pronounces the contact's name and a priority. Therefore, not only does the box point to a user object, but each person contact in the contact list in a box points to a user object. A group contact consists of a set of object IDs for user objects. A place contact consists of a phone gadget and fax gadget.

Referring to Fig. 15, the object database also contains phone book objects 250. A *phone book* is a list of other subscribers on the system and consists of a set of pointers 252 to user objects. There may be one or more phone books, but any subscriber can access only one phone book. Each entry in a phone book is a reference to one of the user objects. Likewise, a user's contact 254 consists of a reference 256 to a user object plus some local information.

When a user changes the portion of a contact that is stored in the user object, such as the work or home phone number, all phone books and contacts referencing that user object show the new information.

For every user item, the object database also keeps general information necessary for accessing that item. This information includes the object ID, object label, owner, time of last modification, time of last access, and time of last change to ownership. The object label is a meme that can include the spoken and spelled names of the item. Each item also has a set of associated flags, such as read, unread, important, and not important.

Virtual Machine

The Virtual Machine is the system's operating system, it is a process that allocates and manages system resources for agents and assistants. The VM responds to requests from agent sessions (just as an operating system responds to system calls) and events generated by the hardware. In essence, the VM is a large event processor. The VM initiates all I/O operations, including input (recording memes), output (playing memes), and recognition (recognizing speech or DTMF). In addition, the VM initiates connections to gadgets, disconnects connections to gadgets, and handles incoming connections.

Handling events:

Referring to Fig. 16, the VM 70 responds to three kinds of events: interprocess communication (IPC) messages 260 from agent sessions 262; hardware events 264 from the communications cards and other hardware 266; and timer events 268 from its internal time queue.

Each action taken by an agent results in an IPC message to the VM. The VM receives this "event" and carries out the request. The VM may send IPC messages back to the agent session to communicate with it. The VM transmits parcels using IPC messages. For example, when an agent sends a parcel to another agent, the first agent sends an IPC message to the VM, and as a result, the VM forwards the IPC message on to the second agent.

Output to and input from the hardware causes a hardware event. For example, when a caller begins speaking a hardware event occurs.

Objects internal to the VM can set timeouts as part of handling other events. These events are placed on a time queue, and when the time expires, a timer event occurs that the VM handles like any other event. For example, when a user is pressing DTMF tones, a timer goes off if there is a long pause between tones.

The VM must handle events quickly and efficiently to provide fast response time to agent requests. The VM listens for events and when one occurs, it blocks all other activity and responds to it. When it finishes handling an event, it listens for the next event.

After the VM creates a session, the session sends an IPC connection request to the VM. A VMListener object 310 in the VM receives this connection request, and the VM creates a VMServer object 312 for that session to use in communicating with the VM. It is the VMServer object that receives the IPC messages from the agent's session, as shown in Fig. 17. The VMServer object takes the object ID and performs an algorithm on it to obtain the appropriate RPC target (VMSession 314 or VMChannel 280) in the VM. The VMServer object then passes on the message to the right VMSession or VMChannel. To do so, the VMServer calls the RPC target's do_rpc virtual member function, and the RPC target processes the message and carries out the operation.

Each agent session has a corresponding VMSession object 314 in the VM. The VMSession object stores the current state of the session. This object also queues parcels destined for the agent when there is no session running for the agent. Once a session is running again, it sends the parcels.

Any VM object can create a Timer object to set a timeout. The VM stores Timer objects on its timer queue 288, and when the timer goes off, a timer event occurs, which the VM handles like any other event.

Managing resources

The VM manages several kinds of resources: those that can be directly manipulated by agents, those that can be indirectly manipulated by agents, and those internal to the VM. Fig. 17 shows these resources in the VM where there are two active agent sessions and a conference between the sessions.

Agents can directly manipulate sessions, channels, conferences, and parcels. For each agent session, the VM maintains a communications channel for IPC messages and an object that keeps track of the state of the session.

Each active gadget in an agent session corresponds to a channel 280 in the VM. As previously noted, a channel can have a set of ports 282 attached to it, with each port providing one or more capabilities. A *capability* is a representation of an ability or feature or a channel, such as the ability to recognize speech or play audio. Agents request capabilities from the VM in order to communicate with their users. In response, the channel attaches one or more ports, where each capability corresponds to one or more ports.

A *conference* 284 is an object that can connect multiple channels in a single session or in multiple sessions. For example, to arrange a conference call, an electronic assistant places calls to each participant and then uses a conference object to connect the participants' channels into a single session.

For each parcel sent by an agent, the VM transmits it and if necessary, queues it.

Through the manipulation of capabilities agents can indirectly manipulate ports. A *port* is an object referring to an input/output device. A channel creates one or more ports for each capability requested by the electronic assistant and manages the ports throughout the connection. The port connects to the hardware device 286 that actually has the capability. Ports can also be thought of as filters. For example, a speech recognition port is not an audio port; it takes audio and then converts it (i.e., filters it) to a menu selection.

The VM also has internal resources including timers 288, a bus 300, and user info objects 302. A *timer* is an object that represents a timeout. For example, a timer object is set for the time between DTMF keys and the time between retrieving a parcel and actually receiving it. A *bus* is an object that represents a data and control hardware bus, such as an MVIP bus or a TCP/IP bus. All objects connected to the bus can communicate with each other using the same protocol. A *user info* object is a cached pointer to the User object in the object database. A user info object is created for each user that calls their assistants and logs in.

The VM allocates and deallocates sessions as they are needed. When an incoming call arrives or a parcel is delivered to a user that does not already have a master session, the VM starts a new session.

A resource manager in the VM performs resource management functions. For example, it keeps a pool of free and available recognizers and interactive-in and interactive-out resources 304. It manages the assignment and deallocation of these resources and it notifies requesters when resources (e.g. channels) are not available. It also holds onto reservations for resources when they are not currently available and as soon as a requested resource becomes available, it assigns that resource to the requesting session. It also deallocates the interactive-in and interactive-out ports and recognition ports when they are no longer needed for a channel, freeing them up for use by other channels.

Conference Objects:

Conference objects are used by an agent and the VM to connect together multiple gadgets and their respective channels. The agent and the VM have slightly different views of a conference object. The agents see only the active gadgets that the agent is in control of; while the VM sees all the active gadgets and their respective channels that are connected into a conference.

The methods on conference objects are listed and described in the following table.

Method	Description
Add	This is used to add an active gadget or a gadget into the conference. When a gadget is added, an active gadget is returned.
Remove	This removes an active gadget from a conference.
SwitchOut	This switches an active gadget out of the communications paths for the conference. The gadget remains part of the conference but it is not connected to the streams of communication with other participants. This allows an agent to carry on a dialog with a specific active gadget without other participants of the conference hearing.
SwitchIn	This is the complement of SwitchOut. It switches a switched-out active gadget back into the media streams of the conference from which the active gadget had previously been removed.
CreateConference	This creates a new conference.
ChangeConference	All conference operations work on a current active conference. ChangeConference changes the active conference.
DestroyConference	This destroys a conference and deallocates all resources associated with that conference.

In the described embodiment, active gadgets are always connected to a conference object. In the case that there is only one active gadget, such as when a subscriber or in caller are conversing with an electronic assistant, then the active gadget is connected to a special form of conference called the idle conference.

Allocating Channels and Ports:

To speed up the allocation of sessions, the VM keeps a *session pool* 308 containing sessions that are not yet assigned to users. As noted earlier, the electronic assistant handles an incoming caller using a new session. The VM allocated this session from its session pool. The VM creates the session pool at system startup time and refills it as necessary. Each session in the pool is for a specific type of agent.

The VM also maintains channel resources, which are pools of channels or ports that the VM can allocate to active gadgets. The channel resource or port resource is responsible for managing the allocations and deallocations from its pool.

In the described embodiment, upon start up of the system, the VM constructs the channels that are necessary to handle communications through the interfaces supported by the system. The channels which are constructed have three possible modes: a listen mode, an idle mode, or an outgoing call mode. In the listen mode, the associated interface card is monitoring the incoming line (e.g. the T1 line) for incoming calls. In the outgoing mode, the interface card is set up to initiate a connection to its communications channel. Idle mode is used during transitions. Upon start up, all of the channels that are constructed are put into a listen mode, ready to receive an incoming call.

As illustrated in Figs. 18A-B, when an agent 150 adds a gadget 336 to a session, the VM gets a channel 280 (Fig. 18A), makes a connection on the channel, and creates an active gadget 152 (Fig. 18B) that refers to the channel. The channel represents the actual connection used to communicate with the user.

Agents can add input, output, and recognition capabilities to gadgets, and the VM attaches ports with those capabilities to the channel. There are three types of ports: input ports, output ports, and recognition ports. An *input port* records audio into a meme. An *output port* plays audio from a meme. A *recognition port* uses input from the user to make a selection from a menu.

Each port represents a capability of a physical device, such as a line processing card or a voice recognition interface card. When the VM attaches ports for a given capability, it usually attaches more than one port. For example, in the described embodiment when an agent requests the recognition capability, the VM adds a voice recognition port for recognizing speech and a DTMF recognition port for recognizing DTMF tones. When the agent requests the output capability, the VM adds multiple output ports, with each port handling a different audio format. This allows transparent support for hardware which uses different audio formats.

Presenting Memes and Menus:

When the agent prompts the user with memes and menus, each port responds to the media or mogul that it knows how to play or recognize. For example, the output port plays any audio media for system prompts, while the recognize port performs speech recognition to determine the user's selection from a menu. For each channel, only one type of media is presented from each meme. When presenting memes, only one output port can present a meme at a time. When analyzing a response, however, multiple recognition ports can be listening at the same time; for example, one can be listening for speech and the other for DTMF tones. If DTMF tones occur, the DTMF recognition port cancels the speech recognition port that was listening for speech. Likewise, if speech occurs first, it cancels the DTMF recognition port.

For example, consider the "Call Whom?" meme previously discussed. Referring to Fig. 19, when the presenter 160 passes the "Call Whom" meme 192 and contact menu 194 to the active gadget 152, the VM presents the meme to the collection of ports 282 attached to the channel 280. The audio output port knows how to play this meme, so it takes the meme and plays it. Next, the VM presents the menu to the collection of ports attached to the channel. Both the DTMF and VPC recognition ports know how to use this menu, so they take it and begin listening for DTMF tones or speech (respectively). Once the VPC recognition port detects speech, it cancels the DTMF recognition port, matches the speech against the menu, and returns the menu pick to the presenter.

When the VM presents a meme list to a collection of ports, each output port looks at the first meme in the list and any successive memes that have the same audio format. If an output port can play the memes, it removes them from the meme list, plays the memes in sequence, and returns the remainder of the list to the VM. The VM presents the remainder of the list to the ports again, and the process continues as before until there are no memes left on the meme list.

Switching channels

The VM is also responsible for connecting and switching calls. It keeps track of all connections between gadgets and it uses the conference object to connect and switch calls. A *conference object* can connect multiple channels into a single session. For example, to arrange a conference call, an electronic assistant places calls to each participant and then uses the conference object to connect the participants' channels into a single session.

Detailed Architecture and Internal Design of Agents

The system agents use a variety of application services to carry out its duties. Fig. 20 shows the functional relationships between the agents and these application services.

Tasks 350 are part of an assistant layer 352 but are not used by the utility agents 354. The assistant uses the MMUI 356, parcels 358, database utilities 360, and miscellaneous utilities 362, which in turn depend on the VM 364 and object database 366. Database utilities are a set of macros for accessing the database, while miscellaneous utilities are a set of standard data structures and containers (such as strings and bit arrays). The utility agents use parcels and database utilities and other VM services (e.g. access to channels, gadgets and active gadgets) carry out their duties. The parcel mechanism is layered on top of both the VM and the database utilities. Although the functional diagram shows the agents relationship with the rest of the architecture, it does not show the dynamics of a live session.

Fig. 21 shows a sample assistant session. At the bottom of the session, the MMUI, VM and parcel libraries 368, 370 and 372 are linked in. The active gadget 374, which the assistant uses to communicate with the user, is layered directly on top of the channel 376 itself. When the assistant takes a task 377 off the task list 378, the task and presenter 380 may pass menus 382 and memes 384 to the active channel or send parcels. In either case, the VM library sends an IPC message 386 to the VM, and the VM carries out the action. As part of carrying out the assistant's actions, the VM may send IPC messages back to the session. The VM library converts these messages back to the objects familiar to the assistant. The task may return menu picks, memes, and parcels to the assistant.

As noted earlier, conceptually, there are three kinds of agents, namely, agents, user agents, and assistants. In the class hierarchy, however, only agents and assistants have classes; user agents do not have a class.

The Agent class 398 is the base class for all agents and assistants. Fig. 22 shows the Agent class and other closely related classes, including, Session, ActiveGadget, Parcel, and Task classes. Also, there is a PersistAgent class, which holds the persistent data for an agent. (In Fig. 22, solid lines represent subclasses and dashed lines represent friend classes.)

Generally, the most important operations that are carried out by an agent are performed on the Session, Agent, or ActiveGadget, as outlined in Table 3.

Class	Operations Performed
Session	Manipulating gadgets Handling parcels
Agent	Handling tasks Handling contents of parcels
ActiveGadget	Presentation of prompts Collection of responses Manipulating capabilities Training

TABLE 3.

The following subsections describe SessionAgent, ActiveGadget, PersistAgent, Task, Parcel, and ParcelContents classes in more detail.

The Session Class:

The Session class is a subclass of the Client class, which provides it with the capability of being an RPC client in the VM. The session provides the thread of execution for the agent and keeps track of the Interprocess Communication (IPC) connection to the vmserver process.

The Session class consists of the following data members:

- A socket, which is used for communicating with the vmserver process (inherited from Client class)
- An object ID (inherited from Client class)
- A pointer to itself (inherited from Client class)
- A reference to the user object
- A pointer to the focus gadget
- A list of gadgets
- A list of parcels

The Session class contains member functions for:

- Managing gadgets. The set includes functions for adding, removing, and connecting gadgets and getting and setting the focus gadget.
- Handling parcels. The set includes functions for sending parcels, retrieving parcels, replying to parcels, and queueing parcels.
- Making a master session out of the current session.

The Agent class:

The Agent class is a subclass of both the Typed and ParcelHandler classes. The Typed class is the primary bass class. It provides run-time checking and member functions for downcasting objects. The ParcelHandler class is a base dispatcher class for handling parcels and gives agents the ability to communicate with one another using parcels.

The Agent class consists of the following data members:

- A name (inherited from Typed)
- A type ID (inherited from Typed)
- A pointer to the session (inherited from ParcelHandler)
- A timeout (inherited from ParcelHandler)
- A task stack, which contains a linked list of task objects to be executed by the agent
- A pointer to the active gadget
- A pointer to the persistent agent

The Agent class contains member functions for:

- Handling tasks. This set includes functions for adding tasks, removing tasks, getting the next task from the task stack, performing a task, executing a task, and cancelling a task.
- Returning the active gadget and persistent agent.
- Handling each type of parcel contents (ParcelContents class). These member functions are inherited from the ParcelHandler class.

Each subclass of Agent contains any objects belonging to that type of agent. For example, the subclass for the assistant contains the hand and hand history, while the subclass for the courier agent contains the list of parcels to be delivered. In addition, each subclass of Agent contains methods for all actions carried out by the agent that are not tasks.

All agents follow the same basic skeleton of operations. Specifically, an agent carries out the following operations in its main processing loop:

- Creates a session
- Creates an agent session within the session
- Enters the perform method to perform tasks
- Continues performing tasks until there are no more tasks or an exception is received
- Deletes its session
- Exits

The Agent::perform() method gets the next task on the task stack and performs it until there are no more tasks on the task stack. Utility agents, such as the cron agent and postmaster agent, however, do not use the task stack or tasks. Each utility agent knows how to handle a single type of parcel, so it provides its own perform() method that waits to receive a parcel and then services it.

The main routine for a complex agent, such as the electronic assistant, has the basic operations just presented, but includes many additional operations, such as ones for initializing the memes in the database, looking up the slot of the vmserver process, looking up the subscriber to which it belongs, getting the focus gadget, and so on.

The ActiveGadget class:

The ActiveGadget class is derived from the DB class, although it is not actually stored in the database currently. The DB class is the primary subclass of Typed and it contains al of the objects stored in the database. The ActiveGadget class consists of the following data members:

- A pointer to the channel in the VM
- A pointer to the meme list
- A pointer to a menu

The ActiveGadget class contains member functions for:

- Presenting a meme and collecting a response from a menu.
- Adding and removing capabilities.
- Training utterances. The set includes functions for starting training, collecting utterances, adding training, completing training and aborting training.

The ActiveGadget class defines operators for:

- Presenting information to the user (<<)
- Recording information from the user (>>)

Gadgets store the "address" at which a user can be reached. There are the following types of gadgets:

PhoneGadget, which describes the address of a telephone -- the prefix, country code, city code, area code, number, and extension.

AgentGadget, which describes the address of an assistant -- a pointer to its subscriber's user object.

ArdisGadget, which describes the address of an Ardis pager.

SkyTel PagerGadget, which describes the address of a SkyTel pager -- a pager ID and security ID.

NumericPagerGadget, which describe the address of a numeric pager -- a phone number.

SocketGadget, which describes the address of a data network -- a hostname, service, and service type.

Theoretically, any agent can use any type of gadget. Certain agents, such as the courier agents, are specialized to deal with particular kinds of gadgets. For example, the system courier agent knows how to deal with the AgentGadget, while the Skytel courier agent and Numeric Pager courier agent know how to deal with the SkyTel PagerGadget and NumericPagerGadget respectively.

To use a gadget to communicate with a user, an agent adds it to the session to obtain an active gadget. Then, the agent can send and receive information over the active gadget.

The PersistAgent class:

The PersistAgent class has no data members or member functions. Each subclass of PersistAgent provides data members for its agent's persistent data and member functions for accessing the data. For example, the persistent assistant contains the subscriber's box, while the persistent cron agent contains a list of cron events.

The Task class:

The Task class is derived from the both the DB class and the FieldParent class. The FieldParent class gives a task a way of knowing whether it is changed, completed, or cancelled.

The Task class consists of the following data members:

- What has changed (inherited from FieldParent)
- Whether the task is complete (inherited from FieldParent)
- Whether the task has been cancelled (inherited from FieldParent)

- A pointer to the agent
- A pointer to the active gadget
- A pointer to the session

The Task class contains member functions for:

- Processing executing, and cancelling tasks. These functions are protected and accessed only by the Agent class.
- Returning and setting the active gadget.
- Returning and setting the agent.
- Returning and setting the session.

Each subclass of Task contains protected member functions for processing and executing that task.

The Parcel class:

The Parcel class is derived from the DB class and has the following data members:

- The address of the recipient
- The address of the sender
- A parcel ID
- A message ID
- A reference to the contents of the parcel

The Parcel class has member functions for accessing data members of the class. It also has operators for:

- Filling an RPC output buffer with the parcel (<<).
- Removing a parcel from an RPC input buffer (>>).

The ParcelContents class:

The ParcelContents class is derived from the DB class, and is an abstract class. The ParcelContents class contains no data members and a single member function for processing the contents of a parcel.

Each subclass of ParcelContents contains data members appropriate for the parcel type, such as a reminder or even another parcel. Each subclass also has a member function for processing its particular content.

Setting up the focus gadget:

The focus gadget is a specific active gadget that an assistant must use to communicate with a user. In the case of a phone gadget, the focus gadget has speech-recognition resources associated with it.

When the VM starts a session for an incoming call, it equips the session with a focus gadget, Focus, which is a data member of Session. For phone calls, this focus gadget is of type PhoneGadget. The assistant uses this focus gadget to communicate with the subscriber, contact, or non-contact that is calling. Before the assistant can use the focus gadget supplied with the session, it must take the following actions to set it up:

1. Obtain the focus gadget with the get_focus member function.
2. Set any attributes needed on the focus gadget.
3. Add any necessary capabilities to the focus gadget.
4. Answer the call on the focus gadget.

In the case where a session is started because of a parcel delivery, the VM does not create a focus gadget. In this situation, the assistant attempts to locate and notify the subscriber by making an outgoing call. Before the assistant can communicate with the person being called, it must take the following actions to set up a focus gadget:

1. Determine the gadget that corresponds to the subscriber's location.
2. Obtain a focus gadget by adding the gadget to the session.
3. Set any attributes needed on the focus gadget.
4. Add any necessary capabilities to the focus gadget.

Once the focus gadget is set up, the assistant can communicate with the user using memes and menus and manipulate the gadgets as necessary to make additional calls, place users on hold, and connect users.

Obtaining additional active gadgets:

Theoretically, agents can add any gadget to a session with the Session:add(Gadget) member function. When the agent adds the gadget, it receives an active gadget in return, and the focus is already set to that active gadget, making it the focus gadget.

Each standard place (home, work, car, mobile, other) for each contact is represented by an addressable gadget object (e.g.a PhoneGadget object) that can be added to a session. For example, if a subscriber asks the assistant to place an outgoing phone call to a contact, the assistant obtains the phone gadget for the correct place (such as work), and adds it to the session.

When the agent adds a gadget to a session, the VM obtains a phone line and dials the number specified in the gadget. The VM does not create an active gadget and give it to the agent until the call connects successfully. The criteria for successful connection is variable; it can mean the phone rang several times or that someone answered the phone. If the call fails to connect, the VM does not create an active gadget, and add operation fails for the agent.

Setting attributes and capabilities:

Before the agent communicates over the active gadget, it must set any attributes and capabilities that it needs for communicating with the user.

Attributes specify which media get presented in a meme and can be thought of as properties of MMUI objects. The language and type of prompts (brief, dialog, instructional) used by the assistant are examples of attributes. Most attributes are set as part of the user preferences mechanism in the assistant. When the assistant starts up, it sets attributes on the focus gadget for each user preference. Whenever the user changes a preference, the assistant sets the appropriate attribute.

The VM currently provides the following set of capabilities:

- WfInterOutCap, which provides audio output
- WfInterInCap, which provides audio output
- WfRecognizerCap, which provides speech and DTMF recognition
- WfFaxInCap, which provides fax input
- WfFaxOutCap, which provides fax output
- WfAllCaps, which provides all capabilities

The assistant can set capabilities when it sets up the focus gadget or anytime during the session with the user. For example, if the user asks to fax some information, the agent can add the fax capability to the active gadget for carrying out that user request.

Answering the focus gadget:

For incoming calls, the assistant must answer the focus gadget with the ActiveGadget::answer member function, which picks up the call and stops the ring tone. After the assistant answers the call, it can begin communicating with the user or callee (e.g. it could be a fax machine that called).

Connecting gadgets:

Agents can connect, disconnect, and reconnect gadgets as needed during a session. To connect gadgets that are located in separate sessions, agents use a separate object, the Conference object. Once a connection is established, agents can disconnect and reconnect gadgets to place users on hold and connect them back in.

Parcels:

Agents use parcels to communicate with one another. A parcel consists of a target (addressee), which can be a user or a session, and content (ParcelContents class). Parcels are delivered to the addressee's session. If the addressee is an agent that can have more than one session running, the parcel is always delivered to the master session.

Table 5 describes the parcels that are used by agents.

Parcel/Contents Types

ParcelContents	Description	Sending or Receiving Agents
ConnectParcel	Contains the caller requesting a connection and their name recording, a conference object, and the response from the callee and the caller. The assistant uses this parcel to negotiate connections between incoming callers and subscribers.	Assistant
CourierEvent	Contains a bond to the message to send, the list of addressees to which to send the message, and the number of retries left. The Postmaster agent uses this parcel to submit messages to courier agents for delivery.	Postmaster agent, SkyTel courier agent, system courier agent
CronEvent	Contains a parcel to be delivered and the time at which it should be delivered. The assistant uses this parcel to submit reminders (or any other items that need delivery at a specific time) to the Cron agent.	Assistant, Cron agent
MSFEvent	Contains a bond to a message. The assistant uses this parcel to submit messages to the Postmaster agent for delivery.	Assistant, Postmaster agent
NewMailEvent	The courier agents uses this parcel to indicate to the assistant that they have just delivered a message.	Assistant, system courier agent, SkyTel courier agent, Secret agent

ParcelContents	**Description**	**Sending or Receiving Agents**
PostalEvent	Contains a bond to the message to send, the list of addresses to which to send the message, and the number of retries left. Courier agents use this parcel to request that the Postmaster retry delivery of a message.	Postmaster agent, SkyTel Courier agent
ReminderEvent	Contains a reminder. The assistant uses this parcel to package up a reminder.	Assistant

Assembling parcels

To assemble a parcel, the agent uses the Parcel constructor and specifies the ParcelContents class as an argument. The agent can then fill in the appropriate fields in the parcel's contents.

Sending and receiving parcels

To send a parcel, reply to a parcel or retrieve a parcel, the Session class provides the following member functions:

send_parcel
reply_to_parcel
retrieve_parcel

When replying to a parcel, the VM automatically addresses the parcel to a specific session rather than a specific addressee.

Passing the same parcel back and forth:

Agents can pass a parcel back and forth, amending its contents each time, as a means of negotiation. As an example of using parcels in this manner, consider how the assistant handles calls. When a subscriber calls a contact, there is no negotiation involved in connecting the telephone call. There is a three-part handshake involved in connecting a contact to a subscriber in the following two situations:

An outside contact calls a subscriber;

A contact that happens to be a subscriber calls another subscriber who is logged into the system

Referring to Figs. 23A-D, the three-part handshake involves the following steps:

1. When a contact calls a subscriber, the electronic assistant 400 for the contact's session 402 sends a connect parcel 404 to the master session 406 (Fig. 23A). The parcel indicates that the caller is requesting a connection to the subscriber and contains a reference to the contact (or a dummy contact if the caller is not a known contact). If there is no master session, the VM starts one.
2. When the electronic assistant for the master session 406 receives the connect parcel 404, it checks to see if it has the focus gadget 408, which indicates it already has a session running with the subscriber. If the electronic assistant does not have the focus gadget, it means that the session was started because it received a parcel, and it needs to try to locate and notify the subscriber in order to get the focus gadget. Once the electronic assistant has the focus gadget, it asks the subscriber whether or not to connect to the contact. If the subscriber says yes, the electronic assistant for the master session sends back the connect parcel

404 and indicates within it that the connection request has been accepted (Fig. 23B).

3. When the electronic assistant for the contact's session receives the connect parcel, it sends it back with the indication to go ahead and make the connection (Fig. 23C). When the electronic assistant for the master session receives the connect parcel, it connects the two gadgets and moves the contact's active gadget into the master session (Fig. 23D).

The way in which conference objects are used handling communications is illustrated in Figs. 25A-D, which diagram the steps in creating a conference object for establishing a call from one subscriber to another subscriber. In this example, it is assumed that both subscribers are initially interacting only with their electronic assistants and thus are connected to idle conference objects. This is illustrated in session A by a focus gadget 401a connected to an idle conference object 403a and in session B by a focus gadget 401b connected to an idle conference object 403b.

Subscriber A initiates the call by instructing his electronic assistant to establish a connection to subscriber B. In response, the VM checks whether subscriber B is in session, i.e., whether a master session is running for subscriber B. If subscriber B is in session, the VM notifies subscriber A's electronic assistant that subscriber B is in a session on the system. This information is communicated to subscriber A, who then is given the opportunity to confirms his request for a connection. When subscriber A confirms his request for a connection, his electronic assistant sends a conference request 405 to subscriber B's electronic assistant and the VM creates a conference object 407 for the requested connection (Fig. 24A).

If subscriber B indicates to his electronic assistant that he will accept the requested connection, his electronic assistant sends an acknowledgment (ACK) back to the electronic assistant for subscriber A (Fig. 24B). In response to receiving the ACK from subscriber B, subscriber A's electronic assistant constructs a conference parcel 409 identifying the conference object 407 which has been created for the connection, connects itself into the conference object 407, and sends the conference parcel 409 to subscriber B's electronic assistant (Fig. 24C).

When the electronic assistant for subscriber B receives the conference parcel from subscriber A, it extracts the identity of the conference object from the parcel, removes its active gadget from its idle conference object, and adds its active gadget to the newly received conference object 407 (Fig. 24D). At this point, subscribers A and B are switched together and their electronic assistants let them know that they are connected by saying "Go ahead".

The Functionality of The Electronic Assistant

The set of commands that are accessible to the subscriber are presented in Exhibit A. The various tasks which the electronic assistant can execute implement the functionality associated with these commands.

The following are more detailed descriptions of various functions that the electronic assistant performs for its subscriber using the mechanisms described above. The functions that are described include among others: (1) handling an incoming call to a subscriber, (2) creating a contact, (3) creating a reminder, and (4) notifying the subscriber of a reminder.

Handling an Incoming Call:

First, we describe how the system handles an incoming call from a caller (i.e., Bill Bishop) to a subscriber (i.e., Jim Smith). The system's call handling functions are invoked when a caller places a call to the subscriber's phone number. The central office notifies the system of an incoming call by placing a ring signal on a T1 line into the system. The interface card that is monitoring that T1 line responds to the ring signal by picking up the line and acquiring a 4-digit sequence of touch tones identifying the extension that is being called. The interface card generates an event for the virtual machine. The virtual machine determines from the hardware what number was called and from that determines the identity of the subscriber being called.

Upon receiving an indication of an incoming call, the VM starts up a session, i.e., a single thread of execution, that will be assigned to handling that call. When the VM server spawns the session, a handshake occurs between the session and the VM server to enable the two entities to communicate with each other. As part of the handshake, the server process passes to the session an ID of the subscriber for whom the session is being started up. Thus, the session is able to immediately establish a reference to a user object in the database that contains the stored information for the subscriber.

In addition to receiving the ID of the user for which the session has been started, the VM also passes up the identity of the channel out of which it constructed an active gadget. The active gadget represents the communication path to the incoming caller. The active gadget represents the particular communications line that the call came in over, in this case a phone line. Until the capabilities are added, the active gadget simply represents a channel with no input or output capabilities. Therefore, to enable the system to play prompts, record incoming signals, and perform voice and DTMF recognition on the channel, the assistant adds capabilities to the active gadget. For example, the assistant adds interactive-out capability, interactive-in capability and recognition capability. The capabilities that are added to the active gadget map to ports which the VM attaches to the channel. For example, interactive-out capability maps to an audio-out port for playing audio, the interactive-in capability maps to audio-in port for recording speech), and the recognition capabilities map to speech recognition and DTMF ports.

Once the capabilities are set up, the electronic assistant invokes an answer call task to handle the incoming call. Referring to Figs. 24A-B, the answer call task engages in a dialogue with the caller to collect certain information which will be necessary to beginning execution of the task. In general terms, the answer call task initially has two fields of information that must be filled in: WHO is calling and WHAT does the caller would like the electronic assistant to do.

When an incoming call arrives for a given subscriber there are a number of ways in which the assistant might handle the call, depending on the preferences which the subscriber has previously selected. The assistant might directly forward the call to a telephone on the subscriber's desk phone, it might simply offer to take a message from the incoming caller, or it might attempt to locate the subscriber and offer to connect him to incoming call once he is located. In handling the call, the answer call task first checks the subscriber's status to determine which preference he has selected (step 500).

In the described embodiment, it is assumed that the caller wishes to have a connection established with the subscriber, thus the caller is not offered any options for responding to the second question. However, in alternative embodiments, the answer call task could be modified to offer the caller the option to leave a message, to specify the priority of the call, or to send a page, among other things.

Also note that if ANI (automatic number identification) is supported, the interface card could have stripped off the ANI signal that it received from the central office and supplied it to the electronic assistant for the session. If that number mapped to a contact in the subscriber's contact list, the electronic assistant could supply the "Who?" entry automatically without engaging in dialog with the caller. Even in that case, however, it may still be desirable to engage in a dialog with the caller to confirm the caller's identity.

After the answer call task determines that the subscriber wants calls screened, the answer call task attempts to identify the caller (step 504). The electronic assistant instruction passes an object (i.e., a meme object) to the active gadget instructing it to say "Please say your name." It also passes the subscriber's contact menu to the gadget so that the caller's reply can be matched against the subscriber's contacts to determine whether or not the caller is a known contact. The electronic assistant asks the gadget to return the identified name.

The gadget presents the "Please say your name" meme to the channel. The channel presents the meme to the collection of attached ports and the appropriate port plays the meme by retrieving the stored audio version from the systems database.

Upon receiving the caller's spoken response, the channel attempts to match the response against responses stored for known contacts in the contact menu. The result which is passed up to the electronic assistant will either be the identity of a known contact or an indication that its an unknown contact. If it is an unknown contact, the answer call task invokes a take a message task which offers the caller the option to leave voice mail for the subscriber (step 506). If the caller is a known contact, the electronic assistant determines whether the known contact is, in fact, the subscriber for whom the electronic assistant is acting (step 508). If the electronic assistant determines that the caller is the subscriber, the electronic assistant authenticates the subscriber by asking for a password before allowing him to have access to the system commands (step 510). Once the identify of the subscriber is verified, the electronic assistant notifies the subscriber of all new messages that have been stored since he last read them and it starts the command task which gives the subscriber access to the set of commands for controlling the electronic assistant (step 512).

If the caller is not the subscriber, the electronic assistant checks the subscriber's contact list to determine the priority of the caller (step 516). Recall that the subscriber's contact list includes the subscriber's designation of each contact's priority (i.e., normal or high). The priority indicates the importance of calls that are received from the caller.

While checking the priority of the caller, the electronic assistant also checks the data structure for the contact to determine whether there are any voice messages left for the caller by the subscriber (step 518). If there are voice messages, the electronic assistant invokes a deliver message task which asks the caller whether or not he wishes to hear the voice messages left by the subscriber (step 520).

The deliver message task performs the following sequence of events. First, the electronic assistant sends a meme to the channel instructing it to say to the caller "I have a message, would you like to receive it?". It also passes a pointer to the stored voice message in its message database. The caller responds by pressing the appropriate buttons on the touch-tone phone. If the caller presses "9" for "yes", the audio-out port retrieves and plays the appropriate media representation of the stored message. After the message is played, the electronic assistant checks whether there are any other unplayed messages, and if there are, it goes through the same sequence of operations to present the other messages to the caller.

Note that the deliver-message task of the described embodiment does not perform any verification of the caller's identity. It may, however, be desirable to do so. This can easily be done by modifying the task to request a verification such as a password.

After the deliver message task is complete or if there are no voice messages for the caller, the electronic assistant checks the availability of the subscriber (step 522). The subscriber has the ability to designate his availability through the "I Will Be" command (described elsewhere). This information is stored in the database as part of the subscriber's user object.

After identifying the subscriber's availability, the electronic assistant determines whether or not to attempt to establish a connection to the subscriber (step 524). In the described embodiment, the determination is based on whether the caller has high enough priority to meet the availability criteria established by the subscriber. For example, if the caller is identified in the database as a normal priority contact and the subscriber has indicated his availability to be "taking important calls", then the electronic assistant does not attempt to establish a connection to the subscriber for this particular call. On the other hand, if the contact's priority is designated as high priority, then the electronic assistant performs the operations necessary to establish a connection with the subscriber.

The decision algorithm in the described embodiment is a very simple one. Alternative embodiments might use much more elaborate algorithms to filter the caller's request for a connection to the subscriber. For example, the decision could take into account other information about the caller such as his job description, or the identity of the company from which he is calling. Also, it would be a simple matter to let the priority of the caller change to reflect the number of calls that the subscriber has initiated to that caller within some preselected period of time. A contact that the subscriber calls frequently would have a priority that is higher than a contact to whom the subscriber rarely places a call.

If the decision is made to not attempt a connection to the subscriber, the electronic assistant invokes a take message task (step 526). In general, the take message task reports to the caller that the subscriber is not available at that moment and asks the caller whether or not he wishes to leave voice mail for the subscriber. After the take-message task is complete, the electronic assistant performs whatever operations are necessary to disconnect from that channel and terminate the session for that caller. This includes, for example, freeing up the resources that were required to support the connection from a caller by stripping off the capabilities that are associated with the active gadget and deallocating the audio-in, audio-out, and recognition ports from the channel so that they may be used by other sessions.

If the subscriber has indicated that he is available to receive calls from that contact, the answer call task performs the steps that are necessary to establish the connection (step 528). To further illustrate the operation of parcel mechanism, the exchange of parcels that takes place to assist in establishing the connection will also be described.

The answer call task in the caller's session sends a connect parcel addressed to the subscriber. The connect parcel requests that a connection be established with the subscriber and it identifies the caller's session. The VM handles the delivery of the parcel. If the VM determines that no master session currently exists, it spawns a new session which it designates as the master session and then delivers the connect parcel to that master session.

Once the master session exists, it immediately checks whether any parcels have been delivered to it. At this point it will discover the connect parcel that was sent by the caller's session. In response to the connect parcel, the master session checks whether it has the focus gadget. If it does not, it invokes a locate-and-notify task the purpose of which is to attempt to establish a connection with the subscriber.

Referring to Fig. 26, the locate-and-notify task first attempts to determine where the subscriber can be reached (step 600). In the described embodiment, the assistant checks the subscriber's user object for stored schedules. There are two schedule data structures which are used to list the schedule for a user: a default schedule and an override schedule. The default schedule indicates where the subscriber can typically be found throughout a normal day or throughout the days of a normal week. The override schedule that the subscriber can generate through the "I-Will-Be" or "Create-an-Itinerary" commands, overrides the default schedule for the relevant periods of time. For example, the subscriber may usually be in his office on Mondays and his default schedule will reflect this. However, on a particular Monday he may have to drive elsewhere to visit a client. He can generate an override schedule to reflect this change from normal routine. The override schedule would indicate that he is reachable through his mobile phone for the period of time that he is expected to be traveling by car and that he is thereafter reachable at the client's business location for another period of time.

The subscriber could also set his location as being with a second subscriber whose schedule and whereabouts are known by the electronic assistant for the second subscriber. Once this is done, the schedule of the first subscriber will track the schedule for the second subscriber who the first subscriber is with.

The locate-and-notify task checks the subscriber's override schedule to determine whether the subscriber has an override for that time. If no override exists, the electronic assistant checks the default schedule. If the default schedule identifies a location for that time, the electronic assistant can find the telephone number for that location from the information stored in the subscriber's user object. For example, if the default schedule indicates that the subscriber is scheduled to be at home at that moment, the assistant looks in the user object to find the telephone number associated with the home location, assuming of course that the subscriber has provided one.

Once the appropriate communications device and its number (or address) is identified, the electronic assistant adds a gadget representing that device to the session (step 604). If the communications device is the subscriber's home phone, the gadget that is added to the session is a phone gadget that contains the phone number for the subscriber's home phone.

In response to receiving a gadget from the master session, the VM allocates a channel to that gadget. Since the gadget is a telephone, the VM causes the appropriate interface card to allocate a phone line channel connecting to a T1 line. The VM also passes the telephone number to the interface card and the interface card dials that number. If the called number is busy, the channel reports back to the master session that the attempt to connect failed. If the phone begins ringing, the channel returns an active focus gadget to the master session. The active gadget represents an actual connection to the subscriber's home phone.

Upon receiving the active gadget, the electronic assistant immediately assigns capabilities to the active gadget to enable it to communicate through the gadget and waits for somebody to pick up the phone (step 606). If a party answers, the electronic assistant determines whether the user is available (step 608). It does this through a dialog during which the answering party can indicate whether the subscriber is there to receive the call. In the described embodiment, the electronic assistant causes the audio-out port to play the previously described stored message:

> Hello, I'm trying to reach Jim Smith. If he is available, press the 9 key. If he is not available, press the 6 key or hang up.

It might be desirable to tailor the dialog to take into account the location of the phone that was just called. For example, one might wish to designate phones as being in friendly territory, questionable territory, or hostile territory. A phone that is in friendly territory might be any phone for which it is likely that the person answering it is familiar with interacting with the electronic assistant. A phone that in questionable territory might be is any phone for which there is a 50/50 chance that the answering party is not experienced in interacting with the electronic assistant. Phones in hostile territory might be any phone for which there is a reasonably high likelihood that the answering party is not experienced in interacting with the electronic assistant. By identifying the connection devices in this manner, an appropriate dialog for interacting with the answering party can be selected. For the inexperienced user, the electronic assistant might use a verbose dialog which explains how to interact with it more fully. Whereas, for the experienced user, it might use a terse dialog which assumes that the person knows how to respond without being told.

If the answering party indicates that the subscriber is not available, this is reported to the master session which then sends a reply parcel back to the caller's session indicating that the connection attempt failed (step 610). The master session then goes through the sequence of steps that is necessary to terminate the session. This includes stripping the capabilities from the active gadget.

If the answering party indicates that the subscriber is available, then the electronic assistant announces the identity of the caller to the subscriber (step 612):

> There is a call from Bill Bishop. Do you want to take the call? Indicate Yes by pressing the 9 key, indicate No by pressing the 6 key.

It then asks the subscriber whether he wishes to accept the call (step 614).

If the subscriber declines the call, the locate-and-notify task sends a failed connection reply parcel back to the caller's session (step 616). In response to the fail reply, the caller's session informs the caller that the subscriber could not be found and it offers the caller the option of leaving a recorded voice message for the subscriber. If the caller accepts the message option, the electronic assistant records the caller's voice message and stores it for the subscriber to play back at some later time.

While the connection to the subscriber still exists, the locate-and-notify task asks the subscriber whether there is anything it can do for the subscriber at this time (step 618). If the subscriber responds by saying no or pressing the "6" key, the locate-and-notify task says "Goodbye" to the subscriber, sends a hang-up command to the channel, and then terminates. On the other hand, if the subscriber indicates that he wants to access his electronic assistant's commands, he must first do additional authentication, then the electronic assistant starts a command task which enables the subscriber to access those capabilities either through voice or DTMF commands (step 620). The commands identified in Exhibit A are then available to the subscriber.

If the subscriber accepts the call, the locate-and-notify task sends a connect reply parcel to the caller's session. Referring to Fig. 24B, in response to receiving the connect reply parcel, the caller's session removes from its active gadget some of the capabilities that will no longer be necessary (e.g. interactive-in, interactive-out, and recognition capabilities) (step 536) and sends the gadget to the subscriber using the parcel mechanism (step 540). Upon receiving the active gadget from the caller's session, the electronic assistant for the master session adds that active gadget to its session and connects both the active gadget and the focus gadget together thereby enabling the subscriber and the caller to communicate with each other (step 622 in Fig. 26). The interface cards perform the switching function that connects the two gadgets together by allowing each gadget to listen to the output of the other channel (i.e., by allowing each channel to have access to the appropriate time slot on the MVIP bus line that carries the other party's signal).

When the connection is established between the caller and the subscriber, the electronic assistant switches into a background mode in which it monitors the subscriber's output for a command (i.e., "Wildfire") that will call it back into its foreground mode (step 624). While the electronic assistant is in the background, it only responds to the single command which calls it back into the foreground and it ignores all of the other commands in its command set.

Recall that it was assumed in the above discussion that a master session was not running when the VM wanted to deliver the request connect parcel from the caller's session. If, however, there is a master session running, the VM delivers the parcel to that master session.

When an already existing master session receives a request to connect parcel from another caller's session, it is stored in a queue. The master session repeatedly checks this queue for received parcels. When it detects the presence of a new parcel, it acts upon it immediately. In the case of the request to connect parcel, the master session initiates a notify task (see Fig. 27). The notify task checks whether the subscriber is interacting with the master session through a phone channel (step 630). If the subscriber is interacting with the session through communication channel other than a phone channel, it sends a notification to the subscriber which identifies the caller and notifies the subscriber that he has just received a call from that individual (step 632). The master session then sends a reply to the request to connect parcel back to the caller session indicating that the attempt to connect failed (step 634).

If the caller is communicating with the master session through a phone channel, the notify task first determines whether the subscriber has indicated that he is willing to accept an interruption to his call (step 636). For example, when the subscriber connected to the call he could have used a "Hold All Calls" command to indicate his preference to not be interrupted. The "Hold All Calls" command temporarily sets the subscriber's status as unavailable during a call being handled by the master session.

If the subscriber indicated that he did not want to be interrupted, the notify task sends a non-interactive notification to the subscriber of the caller's attempt to reach him and then reports the caller's session that the connect attempt failed (step 634).

If the subscriber did not set his status to block interruptions, the notify task notifies the subscriber of the caller on hold (step 638). The system has a form of call waiting in which it first plays a short tone that can be heard by both the subscriber and the party he is talking to and then it plays the name of the incaller, using the caller's spoken self identification. The interruption is handled so as to prevent the caller with whom the subscriber might be talking at that moment from hearing the message (referred to generally as the smart call waiting feature). On some phones this is done by placing the caller on hold for a short period of time while the announcement is made. On other phones, it is possible to send the message to the earpiece on the subscriber's phone without the caller hearing it.

The electronic assistant then gives the subscriber the option to reject the call or to accept the call. Current valid commands are Take-A-Message and I'll-Take-It. If the subscriber rejects the call or does not respond within a preselected period of time, the electronic assistant sends a fail reply to the caller's session (step 642). In response to the fail reply, the caller's session may simply inform the caller that the subscriber could not be found or it may offer to take a message from the caller. If the caller accepts the message option, the electronic assistant records the caller's voice message and store it for the subscriber to play back at some later time.

If the subscriber accepts the new call, the electronic assistant merges the caller's gadget into the master session (step 644) and places the first caller on hold.

While the subscriber is connected to the new caller, if yet another call comes, the electronic assistant handles the new call in the same manner. Thus, it is possible for more than one caller to be on hold at the same time.

To switch back to the first caller, the subscriber issues a "Press-the-hold-button" command. If there is only one call waiting, the electronic assistant puts the second caller on hold and switches to the first caller. Thus, by using the "Press-the-hold-button" the subscriber can toggle back and forth between the two parties. However, if there is more than one call waiting, the electronic assistant responds to the "Press-the-hold-button" by placing the caller with whom the subscriber is presently speaking on hold and saying: "Shall I connect you with <caller's name>?" If the subscriber responds by saying "yes", the electronic assistant connects the subscriber to the identified caller. If the subscriber responds with "No", the electronic assistant again says "Shall I connect you with <next caller's name>?" In this manner, the electronic assistant proceeds cycles the group of parties that are on hold until the subscriber indicates that he wishes to establish a connection with one of the parties.

Create Contact:

Among the other commands available to the subscriber are certain non-message object creation commands. These are used to create contacts which are added to the subscribers contact list, and to create call and recorded reminders, both of which are held by the cron agent for delivery to the subscriber at the appropriate time. Through a phone conversation the subscriber invokes these commands by saying: "Create-A-Contact", Remind-Me-To-Call" and "Remind-Me". An example dialogue which results from invoking the "Create-A-Contact" command is shown in Fig. 28. The mechanisms for implementing this dialog through the channel (and other dialogues which are described below) are the same as were described earlier and will not be repeated in the following discussions.

In response to receiving a "Create-A-Contact" command, the electronic assistant asks the subscriber "What kind - person, place or group?". The subscriber responds by identifying one of these types. In the illustrated dialogue of Fig. 28, the subscriber responds by saying "Person". The electronic assistant then asks for the person's name. Since the electronic assistant will use the subscriber's vocalization of the contact's name, it asks the subscriber to repeat the contact's name a second time to improve the quality of the stored vocalization.

After the subscriber has identified the contact's name, the electronic assistant requires the subscriber to add a phone number by asking: "Which phone number?". In response, the subscriber can identify home, work, mobile or other. In the illustrated example, the subscriber responds by saying "Work". The electronic assistant then asks for the phone number and the subscriber enters the number followed by a # key. The electronic assistant acknowledges receipt of the number by saying "Got it. Now what?"

If the subscriber wants to enter additional information for that contact, he uses the "Update-It" command. The electronic assistant responds by asking "What?". To this the subscriber responds by identifying the particular item of information which he wishes to add or update. If the subscriber says "Priority", the electronic assistant asks "Normal or high". After the subscriber selects one of the available priorities, the electronic assistant indicates it is ready for another command by saying "Got it". If the subscriber wishes to add or modify further information for that contact, he again uses the "Update-It" command followed by an indication of what information is to be added or changed. In the illustrated example, the next item selected for update by the subscriber is the spelling of the contact's name. In response, the electronic assistant notifies subscriber that it is ready to accept the new information by saying "Begin spelling now".

After the spelling has been entered, the electronic assistant indicates its readiness for the next command from the subscriber. If no command is forthcoming within a predetermined period of time, the electronic assistant prompts the subscriber for a command. If the subscriber indicates that he has no further additions or changes at this time, the electronic assistant indicates that the "Create-A-Contact" is over by saying the "Done".

The subscriber can also create new contacts by copying them from the phone book. Phone book entries, however, do not have voice identifications associated with them. Thus, if the subscriber copies an entry from the phone book, he must add a voice identification to make it part of his contact list.

Finally, a contact can be created as a result of receiving a message with an enclosed contact. When reviewing a message which contains a contact, the assistant asks the subscriber if he wishes to transfer the contact out of the message and into the subscriber's contact list. If the subscriber responds by saying "yes", then an procedure that is similar to the one for saving phone book listings is executed.

It should be noted that the system is capable of capturing phone numbers under other circumstances and later use this captured information to assist the subscriber. For example, when the assistant responds to a call and offers the caller the option of leaving a message for the subscriber, it first asks the caller to enter his phone number. After the phone number has been entered, the electronic assistant then records the caller's voice message. Also when the subscriber dials another number through the electronic assistant, that dialed phone number is remembered. This remembered information can later be used in response to a "Give-Them-A-Call" command or to create a contact for the person or place that is represented by the remembered phone number.

Remind-me

As indicated above, the subscriber can use the commands "Remind-Me" and "Remind-Me-To-Call" to create reminders. The dialogs associated with these tasks are shown in Figs. 29 and 30, respectively.

To create a reminder, the subscriber issues the "Remind-Me" command. This causes the assistant to invoke the reminder task with an attribute to indicate the creation of a recorded reminder. At this point the assistant prompts for what the user would like to be reminded about. At that point it starts recording as a message whatever the subscriber says into the phone, to be played when the reminder comes due. The subscriber indicates the end of the message by pressing the "#" key and the electronic assistant stores the recorded message in its database.

After the recording is complete, the electronic assistant engages in further dialogue with the subscriber to establish a time at which the reminder is to be delivered. The electronic assistant asks "When?". The subscriber responds using one of several conventions that are available in the system for specifying a time. The conventions are shown in Table III. In the described example, the presenter responds by saying "Today". The electronic assistant then asks the subscriber for a time. Using touch tone signals

DTMF, the subscriber enters a specific time (e.g. 530 for 5:30). The electronic assistant responds by restating the time which the subscriber has entered and asks for a confirmation from the subscriber. If the subscriber confirms the time, the electronic assistant indicates that the dialogue has concluded successfully by saying "Got it, I'll set my watch alarm.". If the subscriber does not confirm the time (either because the electronic assistant made a recognition error or because he changed his mind), the electronic assistant then repeats the above described sequence of operations to obtain a new time.

If the subscriber issues the "Remind-me-to-call" command (see Fig. 30), the create-reminder task asks the subscriber "Call whom?". The subscriber's response "John Smith" is recognized by the voice recognition capabilities of the ASR card and matched against utterances stored with the subscriber's contact list that identify the subscriber's contacts. If a match is found (i.e., if a contact from the subscriber's contact list is identified), a pointer to that contact is saved as part of this call reminder.

The electronic assistant then engages in the previously described dialogue with the subscriber to obtain a delivery time for the reminder. After the delivery time has been entered, the electronic assistant asks the subscriber: "Should I know the topic for the call?". If the subscriber responds in the affirmative, the electronic assistant says "Recording... " to notify the subscriber that he may begin recording. As before, the subscriber ends the recording by pressing the # key. In response, the electronic assistant confirms that the command has been received by saying "Got it, I'll set my watch alarm".

The electronic assistant sends the completed reminder to the Cron agent via the parcel mechanism. First, it packages the reminder in a parcel that is addressed to the subscriber and includes its desired delivery time. It then places that parcel in a second parcel that is addressed to the Cron agent. The VM delivers the outer parcel to the Cron agent. Upon receipt, the Cron agent opens the parcel and pulls out the parcel that is inside; it checks the delivery time for that parcel and places it in a time ordered queue. The Cron agent keeps track of the delivery time of the top parcel.

Handle Reminder Task:

At the delivery time, the Cron agent wakes up and sends the reminder parcel to the indicated address. As described earlier, the VM handles the delivery of the reminder parcel. If there is no master session running, the VM starts up a session and delivers the parcel to that new session. If there is a master session running, the VM delivers the parcel to the master session.

Referring to Fig. 31, when the session that receives the parcel detects that it has received a reminder parcel, it starts up a handle reminder task. The handle reminder task checks whether the current session is a master session (i.e. whether the session has the focus gadget) (step 700). If the session is communicating with the subscriber, the electronic assistant notifies the subscriber of the reminder using the existing focus gadget (step 702). The method used for notifying the subscriber depends upon the channel through which the subscriber is connected to the session. If the subscriber is connected through a phone channel, the electronic assistant briefly interrupts his call, reports to him that a reminder has come due. After notifying the subscriber of the reminder, the electronic assistant places the reminder in an active reminder list in the box, where the subscriber can review it during the current or any subsequent sessions (step 703). The user can then invoke a "Find-Active-Reminders" command to deal with the recently received reminder.

If the subscriber is not connected to the session, the handle reminder task initiates a locate-and-notify task. The locate-and-notify task operates in the manner previously described except that instead of notifying the subscriber of a caller's attempt to reach him, it notifies the subscriber of the reminder. If the locate-and-notify task is unable to establish a connection with the subscriber, it sends a failed-to-connect reply to the handle reminder task. In response, the handle reminder task checks whether the subscriber can be reached through a non-interactive means by, for example, checking for a pager assigned to the subscriber. If no alternative means for notifying the subscriber exists, the electronic assistant places the reminder on the non-scheduled reminder stack (step 710). If an alternative means for notifying the subscriber does exist, the electronic assistant sends a notification to the alternative gadget (step 708) before placing it on the non-scheduled reminder stack.

If the locate-and-notify task establishes a connection with the subscriber, the reminder is played back to the subscriber (step 720). After that, the electronic assistant asks the subscriber if he wants to reschedule the reminder (step 722). If he elects to reschedule it, a rescheduling task is executed that enables the subscriber to change the delivery time for the reminder (step 724). The modified reminder is then sent back to the cron agent for delivery at the rescheduled time. The handle reminder task then starts the command task loop running which enables the subscriber to access the full set of commands (step 726).

Additionally, at any time during a session, a subscriber can "Find" and reschedule pending reminders using the "Update-it" task. (See Exhibit A).

If the subscriber elects not to reschedule the reminder, the reminder is placed on a non-scheduled reminder stack (step 725) and the handle reminder task starts up the command task loop (step 724). If the call reminder identifies a contact, the subscriber can use the "Call Them" command at this point. Since the call reminder includes a pointer to the contact in the subscriber's contact list, the electronic assistant interprets the "Give-Them-A-Call" command as though it was equivalent to a "Call" command where the contact is the one in hand. The electronic assistant uses the gadget identified in the contact's object to place the call.

If the locate-and-notify task reaches the subscriber's gadget but the connection to the subscriber is not accepted, the electronic assistant places the reminder in the active reminder stack (step 703).

Other tasks that are implemented by the system are best described through examples of the commands and the dialogs that the system supports when interacting with the subscriber. A summary of the commands is presented in Exhibit A at the end of this specification.

Find Command:

Referring to Fig. 32, the subscriber can retrieve and review access various items of information that are stored for him by using the Find command. When the subscriber says "Find" while the electronic assistant is running the command task loop, the electronic assistant replies by asking "Find what?". At this point the subscriber can identify one of seven different items, namely, contact, all-the-contacts, phonebook-listings, messages, new messages, messages from, saved messages, reminders, and trash. The operation of the Find command will be illustrated using the contacts as the item which the subscriber wishes to manipulate.

When the subscriber is asked for what it is that he wants to find, he replies by stating "All-The-Contacts". The electronic assistant then accesses the subscriber's contact list and reports to the subscriber how many contacts are on the list. Then the electronic assistant waits for a command from the subscriber. The subscriber can search the contact list for a given name by issuing the Find "Contact" command. In response, the electronic assistant asks "Contact name?". The subscriber responds with a name, e.g. John Smith. The electronic assistant acknowledges by replying "Contact name, John Smith" to indicate it has the user object for that contact in its hand. The subscriber can then instruct the electronic assistant to: (1) describe it by issuing the describe command; (2) update it by issuing update command; (3) discard it by issuing the "Throw-It-Away" command; or (4) call them using the "Give-Them-A-Call" command.

If the subscriber issues the describe command, the electronic assistant replies by reporting to the subscriber the information that is stored for that contact.

If the subscriber issues the update command, the electronic assistant initiates a dialog similar to that previously described with the Create command which enables the subscriber to modify or add to the information stored for that contact.

The subscriber can instruct the electronic assistant to dial a number by saying "Call" which causes the electronic assistant to invoke the call task. The call task asks the subscriber to provide the number, which may be provided either as a reference to a contact, by entering a specific number or by saying "this-one", referring to a contact message or call reminder being pointed to; then, it dials the number. As soon as the ringing begins, the electronic assistant establishes a connection is between the subscriber's line and the outgoing call line so that the subscriber can complete the call. When that connection is made, the electronic assistant automatically switches into its background mode in which it will only respond to a particular command (e.g. "Wildfire") which causes it to switch back into the foreground. In its background mode, the electronic assistant disables its full command set so that the utterance of the command words during the course of the conversation will not unintentionally invoke a command task.

Anytime the electronic assistant is running its command task which give the subscriber full access to its command set, the subscriber can instruct the electronic assistant to go into its background mode. He does this by saying the "That will be all for now". In response the electronic assistant replies, "Say Wildfire when you need me" and switches into its background mode. When the subscriber needs to access the full command set of the electronic assistant, he says "Wildfire". In response the electronic assistant moves back into its foreground mode and replies "Here I am", confirming that the electronic assistant is again fully active and that the subscriber may now access the full set of commands that are supported by the electronic assistant.

Generating an Override Schedule:

As indicated above, the "I-Will-Be" and "Create-An-Itinerary" commands enable the subscriber to generate an override schedule. A typical dialog for each of these commands is shown in Figs. 33 and 34, respectively. The dialog for each of these commands is very similar. The "I-Will-Be" command is used for generating a single modification to the subscriber's schedule; the "Create-An-Itinerary" comand is used for generating more complex override schedules.

When the subsrciber utters "I-Will-Be", the electronic assistant responds by asking "Doing What?" In the described embodiment, the subscriber may respond in one of the four following ways:

Taking Calls
Only Taking Important Calls
Unavailable
Running on Schedule

"Taking Calls" indicates to the electronic assistant that all calls should be forwarded to the subscriber; "Only Taking Important Calls" indicates that only calls from contacts that are designated as high priority should be forwarded; "Unavailable" indicates that no calls should be forwarded; and "Running on Schedule" indicates that the override schedule is being cancelled in favor of the default schedule.

In the illustrated example, the subscriber reponds by saying "Taking Calls." Next, the electronic assistant asks "Where?" To this the subscriber may respond with one of the following designations: work, home, car, mobile or other, for each of which it is assumed that a correspodning address or phone number exists.

After the subscriber has indicated how he will be reachable, the electronic assistant prompts him to indicate for how many hours to which the subscriber responds with a number. After the subscriber has indicated the time, the electronic assistant acknowledges his response by saying "Done".

The "Create-An-Itinerary" command, a sample dialog of which is shown in Fig. 34, operates in a similar manner excpet that it allows the subscriber to use voice commands to build as large a override schedule as is desired. In other words, after the first entry has been completed, the electronic assistant asks "And then you'll be". In response the subscriber can enter more schedule information or can terminate the command routine by saying "Back on schedule."

Calling Commands:

A subscriber can ask his electronic assistant to place a phone call in one of three ways. If the party to be called is a known contact, the subscriber can identify the contact's name and the electronic assistant will obtain the information necessary to place the call from the subscriber's contact list. If the party is not a known contact, the subscriber can identify the party and provide, either by voice or by DTMF, a phone number to call. The third way is by relying on information available from context. That is, if the electronic assistant has just retrieved a voice mail message for the subscriber, the subscriber can say "Give-Them-A-Call" and the electronic assistant will call the party that left the voice mail message. Additionally you could say "Call" "This-One". This is possible because the stored message identifies the caller either as a contact or by a telephone that the caller was asked to leave.

The stored objects for the contacts also include a pointer to a note (e.g. a voice message) which the subscriber can generate and attach to the object. When the subscriber instructs his electronic assistant to call the contact, the electronic assistant plays the stored message to the subscriber while it is attempting to establish the connection with the contact. The note might include information about the contact which the subscriber wishes to be reminded of whenever he calls that contact. For example, he may wish to know the name of contact's secretary so that he can address her by name if she answers the phone.

Request Connection:

It is possible for a subscriber to infer to his assistant that he would like to talk with another subscriber without actually placing a call to her. This is referred to as requesting a connection with another user. One mechanism for accomplishing this is for the subscriber to explicitly request a connection. This causes the subscriber's assistant to inform the electronic assistant to whom the request is being made that a connection at some point is desired. When this request is received, the receiving assistant holds onto the request until it is communicating with its subscriber. During the next session with the subscriber, the receiving assistant informs its subscriber of the requested connection and the availability of the party requesting the connection. The subscriber can then decide to connect with the other party or ignore the request.

Feature Phone:

The feature phone is an object within the system that builds smart-phone like functionality on top of the conference object. The feature phone allows the user to manage multiple calls. The user can request that their assistant place different callers on hold, hang up on certain callers, and place calls to other contacts through the feature phone. When a subscriber receives a call using the system's smart call-waiting, the call is managed by the feature phone. Current implemented functionality includes the user commands: call, hang-up, and press-the-hold button. These commands are used to manipulate an unlimited number of simultaneous calls.

Virtual Hallway:

As indicated above, the subscriber can ask his electronic assistant what is in the virtual office, what it is holding in its hand or pointing to, what is in the "trash", what new messages have come in, etc. However, a concept of a virtual hallway is also supported in the described embodiment. The virtual hallway is made up of the collection of virtual offices. The subscriber can ask his electronic assistant what other subscribers are presently communicating with their electronic assistants by using a "Who-Else-Is-Around" command. This feature can be "filtered" so that the subscriber will only see people who are in the subscriber's contact list, or who are members of certain specified groups, or who are part of a particular phone book.

In the described embodiment, which supports internal switching, and given that the virtual hallway feature enables the subscriber to see other subscribers who are on the system, the system also offers the capability to connect and share information between distributed or travelling groups or subscribers more quickly than is possible by messaging. In addition, "visibility" in the hallway can also include the ability to see what the other person is doing (e.g. on the phone, reading a message, etc.). Depending upon what the subscriber has told her assistant about her availability and accessibility, the electronic assistant will control how visible the subscriber is (or how visible she will be) in the virtual hallway.

Implemented Assistants

The described embodiment implements two types of assistants, namely, the electronic assistant described above and a reception assistant. The reception assistant is much simpler than the electronic assistant, its job is to answer a central phone number (for all of the subscribers on a particular system) and route the call to the assistant for a specific subscriber. This avoids using the DID line and allows one number to handle a large number of subscribers. The basic operation of this assistant is to ask for the extension of the subscriber being dialed and then start up that subscriber's assistant to handle the rest of the call. At this point the call can be handled in a similar manner to the answer call task previously described. In the described embodiment, it is assumed that only subscribers (not their contacts) use the reception assistant. As such, once the extension for the subscriber is entered the subscribers assistant immediately asks for the subscriber's passcode. Since the assistant can assume it is the subscriber it does not need to ask for the name of the person calling, as previously described.

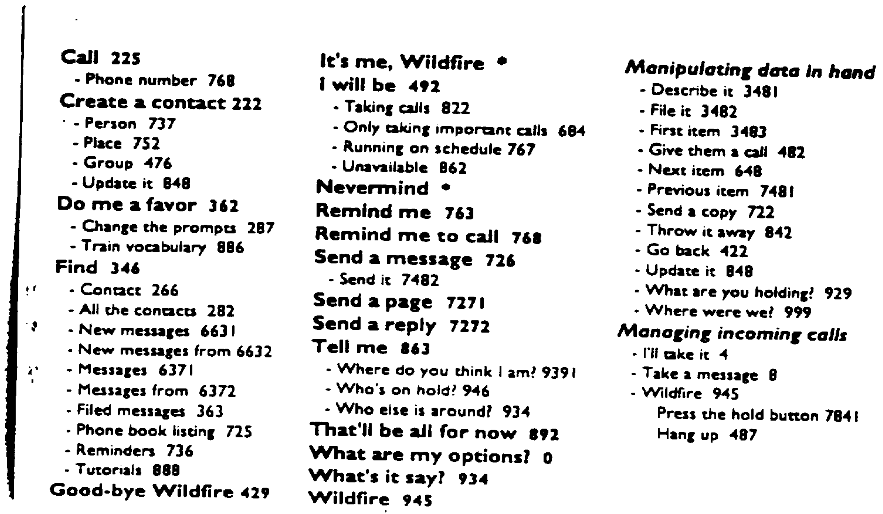
It is possible to also include electronic assistants with different "personalities", i.e., assistants whose functionality is tailored to the particular subscriber for whom that assistant will be providing services. For example, there could be an electronic assistant for customer service representatives. That electronic assistant would implement a set of commands and tasks that are more appropriate to the role of a customer service representative or even a group of customer service representatives. It might handle an incoming call by asking the caller a sequence of questions designed to gather information relating to the particular customer service problem. That information might be used by the electronic assistant to determine the most appropriate person to receive the call and it might also be used to assist the customer representative in responding to the customers concerns by retrieving relevant stored data from a database. The electronic assistant might place the caller in a queue with other callers having customer service questions and then connect them as resources become available.

In addition to "customizing" the personality and skills of a subscriber's electronic assistant other specialized electronic assistants can be added to the system. For instance, a system may have, or be able to connect with, a stockbroker assistant to check on particular investments.

Other embodiments are within the following claims.

What is claimed is:

EXHIBIT A



Call 225
- Phone number 768

Create a contact 222
- Person 737
- Place 752
- Group 476
- Update it 848

Do me a favor 362
- Change the prompts 287
- Train vocabulary 886

Find 346
- Contact 266
- All the contacts 282
- New messages 6631
- New messages from 6632
- Messages 6371
- Messages from 6372
- Filed messages 363
- Phone book listing 725
- Reminders 736
- Tutorials 888

Good-bye Wildfire 429

It's me, Wildfire •

I will be 492
- Taking calls 822
- Only taking important calls 684
- Running on schedule 767
- Unavailable 862

Nevermind •

Remind me 763

Remind me to call 768

Send a message 726
- Send it 7482

Send a page 7271

Send a reply 7272

Tell me 863
- Where do you think I am? 9391
- Who's on hold? 946
- Who else is around? 934

That'll be all for now 892

What are my options? 0

What's it say? 934

Wildfire 945

Manipulating data in hand
- Describe it 3481
- File it 3482
- First item 3483
- Give them a call 482
- Next item 648
- Previous item 7481
- Send a copy 722
- Throw it away 842
- Go back 422
- Update it 848
- What are you holding? 929
- Where were we? 999

Managing incoming calls
- I'll take it 4
- Take a message 8
- Wildfire 945
  - Press the hold button 7841
  - Hang up 487

Claims:

1. A method implemented by a computer-based electronic assistant to receive and manage incoming calls to a subscriber, said method comprising:

receiving an incoming call to the subscriber from a caller;

in response to receiving the incoming call, establishing a first connection between the electronic assistant and the caller;

establishing a second connection between the electronic assistant and the subscriber;

over the second connection, electronically notifying the subscriber of the incoming call;

in response to receiving a call accept command from the subscriber over the second connection, linking the caller and the subscriber so that they may communicate with each other;

upon linking the subscriber to the caller, switching the electronic assistant to a background mode in which said electronic assistant continues to monitor the subscriber over the second connection while the subscriber is linked with the caller; and

in response to receiving a summoning command, switching the electronic assistant into a foreground mode,

wherein the electronic assistant when in its background mode responds to a first set of commands including at least the summoning command and when in its foreground mode responds to a second set of commands, said second set of commands being larger than said first set of commands.

2. The method of claim 1 wherein said second set of commands includes commands that are unavailable in the first set including a command that causes the electronic assistant to terminate the first connection.

3. The method of claim 1 wherein said second set of commands includes commands that are unavailable in the first set including a command that causes the electronic assistant to establish a connection between the subscriber and another party.

4. The method of claim 1 wherein said second set of commands includes commands that are unavailable in the first set including a command that causes the electronic assistant to send a message to another party.

5. A method of processing an electronic reminder that is addressed to a subscriber, said electronic reminder including subscriber-generated content and a specified time at which it is to be delivered to the subscriber, said method implemented by a computer-controlled electronic assistant and comprising:

storing the electronic reminder in an electronic database that is accessible to the electronic assistant;

when current time coincides with the specified time, detecting that the stored electronic reminder has become due;

in response to detecting that the stored electronic reminder has become due, identifying a communications device through which the subscriber can be reached at the specified time;

establishing a connection to the communications device;

upon reaching an answering party through the communications device, electronically notifying the answering party that the call is intended for the subscriber;

electronically informing the answering party that the answering party may accept the call by issuing an accept reply;

if the call is accepted by the answering party, electronically delivering the contents of the electronic reminder to the answering party through said communications device.

6. The method of claim 5 wherein the step of identifying the communications device comprises:

in response to detecting that the stored electronic reminder has become due, recognizing that the addressee on the stored electronic reminder is the subscriber;

executing a locator algorithm to determine the likely whereabouts of the subscriber at the specified time; and

if the locator algorithm determines a location at which the subscriber is likely to be at the specified time, identifying a communications device that is associated with that location,

wherein the communications device that is associated with that location is the communications device to which said connection is established.

7. The method of claim 5 wherein the electronic assistant is electronically connected to a plurality of different communications media and wherein the step of establishing the connection to the communications device comprises:

selecting the most appropriate one of said plurality of different communications media for placing a call to the communications device;

establishing a channel to a selected communications medium; and

establishing a connection to the communications device through said channel.

8. A method of processing an electronic reminder that is addressed to a subscriber, said electronic reminder including subscriber-generated content and a specified time at which it is to be delivered to the subscriber, said method implemented by a computer-controlled electronic assistant and comprising:

storing the electronic reminder in an electronic database accessible to the electronic assistant;

when current time coincides with the specified time, detecting that the stored electronic reminder has become due;

in response to detecting that the stored electronic reminder has become due, recognizing that the addressee on the stored electronic reminder is the subscriber;

executing a locator algorithm to determine the likely whereabouts of the subscriber at the specified time;

if the locator algorithm determines a location at which the subscriber is likely to be at the specified time, identifying a communications device that is associated with that location;

establishing a channel to a communication medium that is appropriate for placing a call to the communications device;

establishing a connection to the communications device through the channel;

upon reaching an answering party through the communications device, notifying the answering party that the call is intended for the subscriber;

notifying the answering party that the answering party may accept the call by issuing an accept reply;

if the call is accepted by the answering party, delivering the contents of the electronic reminder to the answering party.

9. The computer-implemented method of claim 8 further comprising:

after delivering the contents of electronic reminder to the answering party, maintaining the connection to said communications device so that the subscriber can issue commands to the electronic assistant;

monitoring the connection for subsequently subscriber-issued commands;

in response to receiving the subsequently subscriber-issued commands, performing other operations for the subscriber.

10. A method implemented by a computer-based electronic assistant to receive and manage incoming calls to a subscriber, said method comprising:

receiving an incoming call to the subscriber from a caller;

in response to receiving the incoming call, establishing a first connection between the electronic assistant and the caller;

through a dialogue between the electronic assistant and the caller over the first connection, determining the identity of the caller;

detecting that the subscriber is presently interacting with the electronic assistant through a second separate connection;

electronically alerting the subscriber over the second connection that there is an incoming call for the subscriber;

electronically identifying to the subscriber the identity of the caller;

monitoring the second connection for a response sent by the subscriber to the electronic assistant directing the electronic assistant how to process the incoming call.

11. A method implemented by a computer-based electronic assistant for managing information and connection resources for a plurality of subscribers including a first subscriber and a second subscriber, said method comprising:

receiving a call from the first subscriber to the electronic assistant over a communications media;

in response to receiving the call from the subscriber, establishing a first connection between the electronic assistant and the first subscriber;

identifying the first subscriber as the source of the call;

starting up a first session in said electronic assistant, said first session being a thread of execution of code for managing data and performing functions on behalf of said first subscriber;

within said first session, receiving a first command sent by the first subscriber to the electronic assistant over the first connection, said first command instructing the electronic assistant to perform a function relating to the second subscriber;

within said first session, responding to said first command by sending a first message addressed to the second subscriber, said first message containing information relating to the first command;

in response to said first message, starting up a second session, said second session being a thread of execution of code for managing data and performing functions on behalf of said second subscriber, said second session being separate from said first session;

within said second session, receiving the first message; and

within said second session and in response to receiving said first message, performing a function that produces a result that is responsive to the first message.

12. The method of claim 11 further comprising from within said second session, sending a reply message to the first session reporting the result.

13. The method of claim 12 further comprising within said first session, receiving the reply message from the second session and in response thereto executing some function which uses information in said reply message and that is responsive to said first command.

14. A computer-implemented method of processing communications through a multimedia interface that includes a plurality of interface devices and a plurality of input/output devices, each of said plurality of interface device being capable of connecting to a different one of a plurality of different communications networks, each of said plurality of input/output devices capable of processing a

different one of a plurality of media types, said method comprising:

establishing a channel representing a physical connection to any selected one of the plurality of communications networks through said interface devices;

attaching an appropriate subset of a plurality of ports to the channel, wherein each port of said plurality of ports represents a different one of said plurality of input/output devices and wherein the appropriate subset of said plurality of ports includes ports which correspond to input/output devices that are capable of connecting to the selected communications network;

executing an operation that generates an item of information that is to be communicated through the multimedia interface to at least one of said plurality of communications networks;

retrieving from a memory a multi-media reference to said item of information, said multi-media reference to said item of information containing a plurality of references to said item of information, each of which refers to said item of information in a different one of a plurality of formats, each of said plurality of formats being for a different media type;

passing said multi-media reference to the ports attached to said channel;

in response to receiving said multi-media reference at the attached ports, retrieving the item of information from memory in a particular one of the formats identified in the multi-media reference, wherein the step of retrieving is performed by one of the attached ports that is capable of processing the format of the retrieved item of information; and passing the retrieved item of information to an input/output device for delivery over the connected communications network, the input/output device to which the retrieved item is passed for delivery being the input/output device that is associated with the attached port that retrieved the item of information from memory.

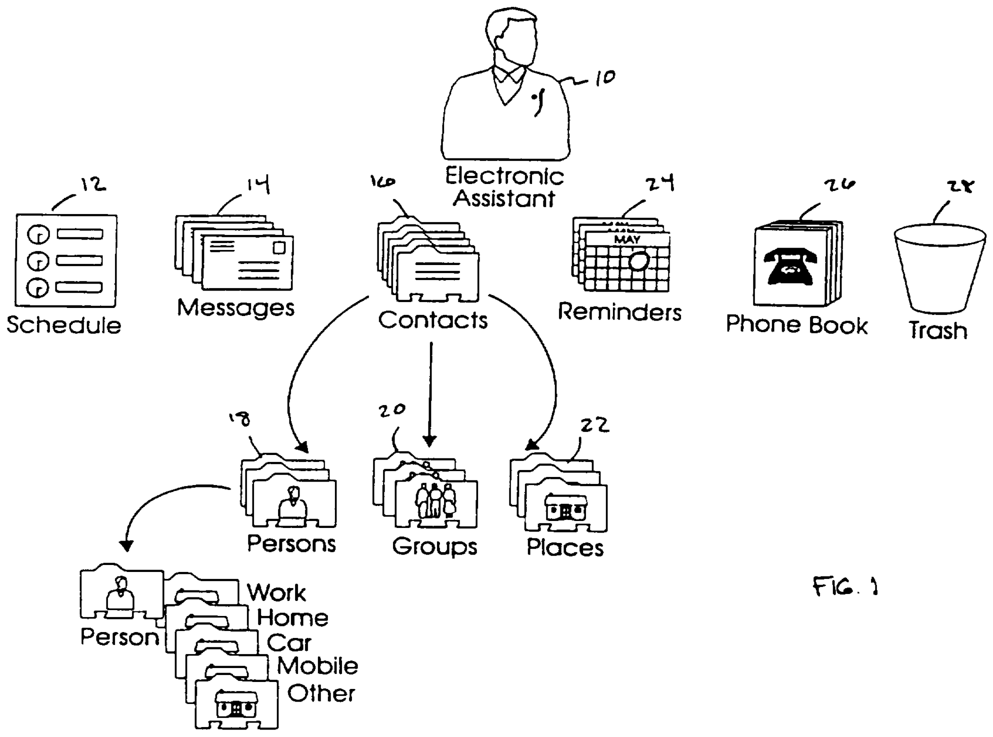

10
Electronic Assistant
12
Schedule
14
Messages
16
Contacts
24
MAY
Reminders
26
Phone Book
28
Trash
18
Persons
20
Groups
22
Places
Person
Work
Home
Car
Mobile
Other

FIG. 1

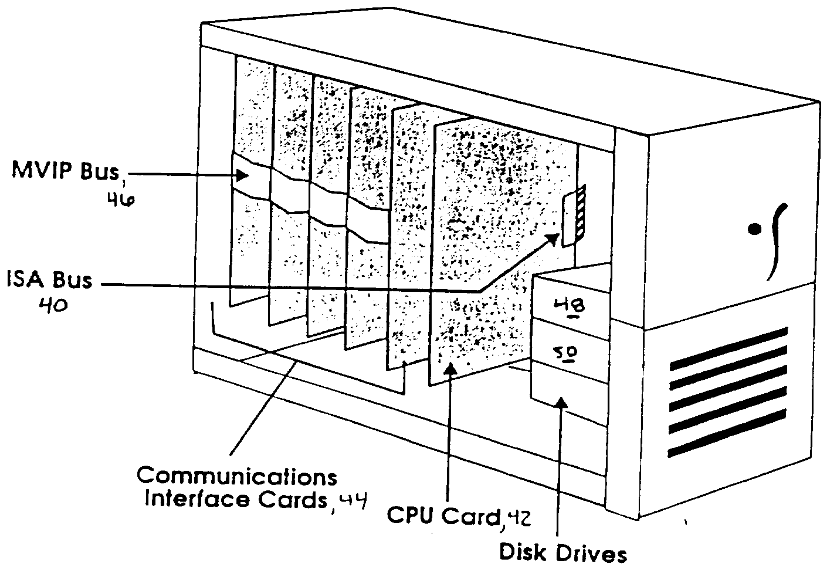

MVIP Bus, 46
ISA Bus 40
48
50
Communications Interface Cards, 44
CPU Card, 42
Disk Drives

FIG 2

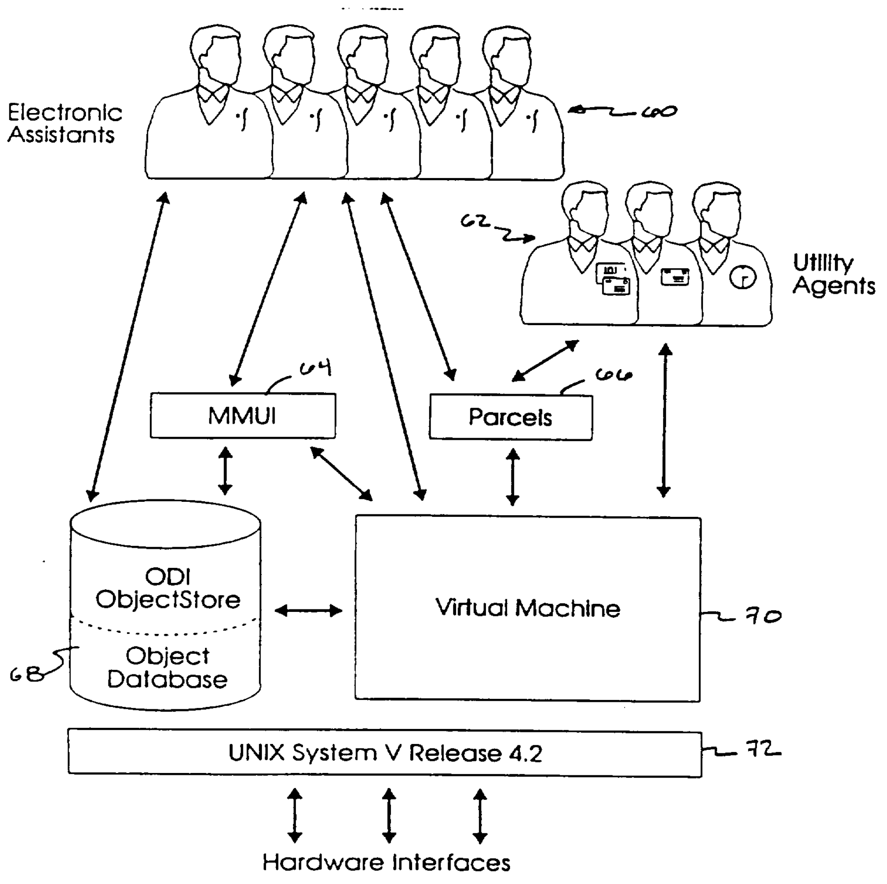

Electronic Assistants
60
62
Utility Agents
64
MMUI
66
Parcels
ODI ObjectStore
Object Database
68
Virtual Machine
70
UNIX System V Release 4.2
72
Hardware Interfaces

FIG 3

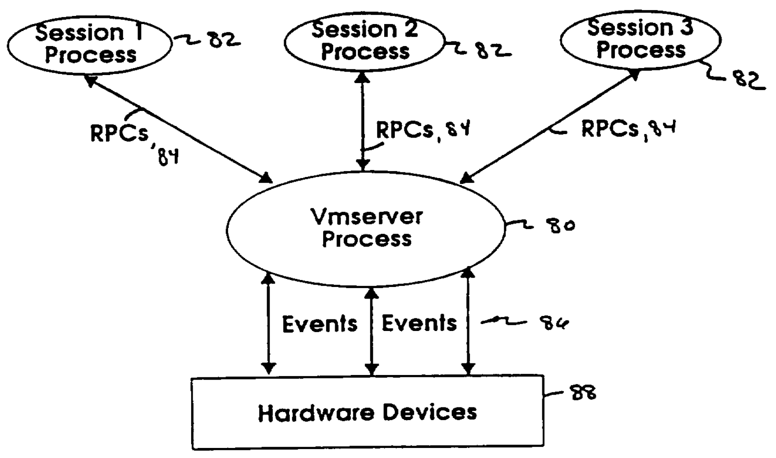

Session 1 Process
82
Session 2 Process
82
Session 3 Process
82
RPCs, 84
RPCs, 84
RPCs, 84
Vmserver Process
80
Events
Events
84
Hardware Devices
88

FIG. 4

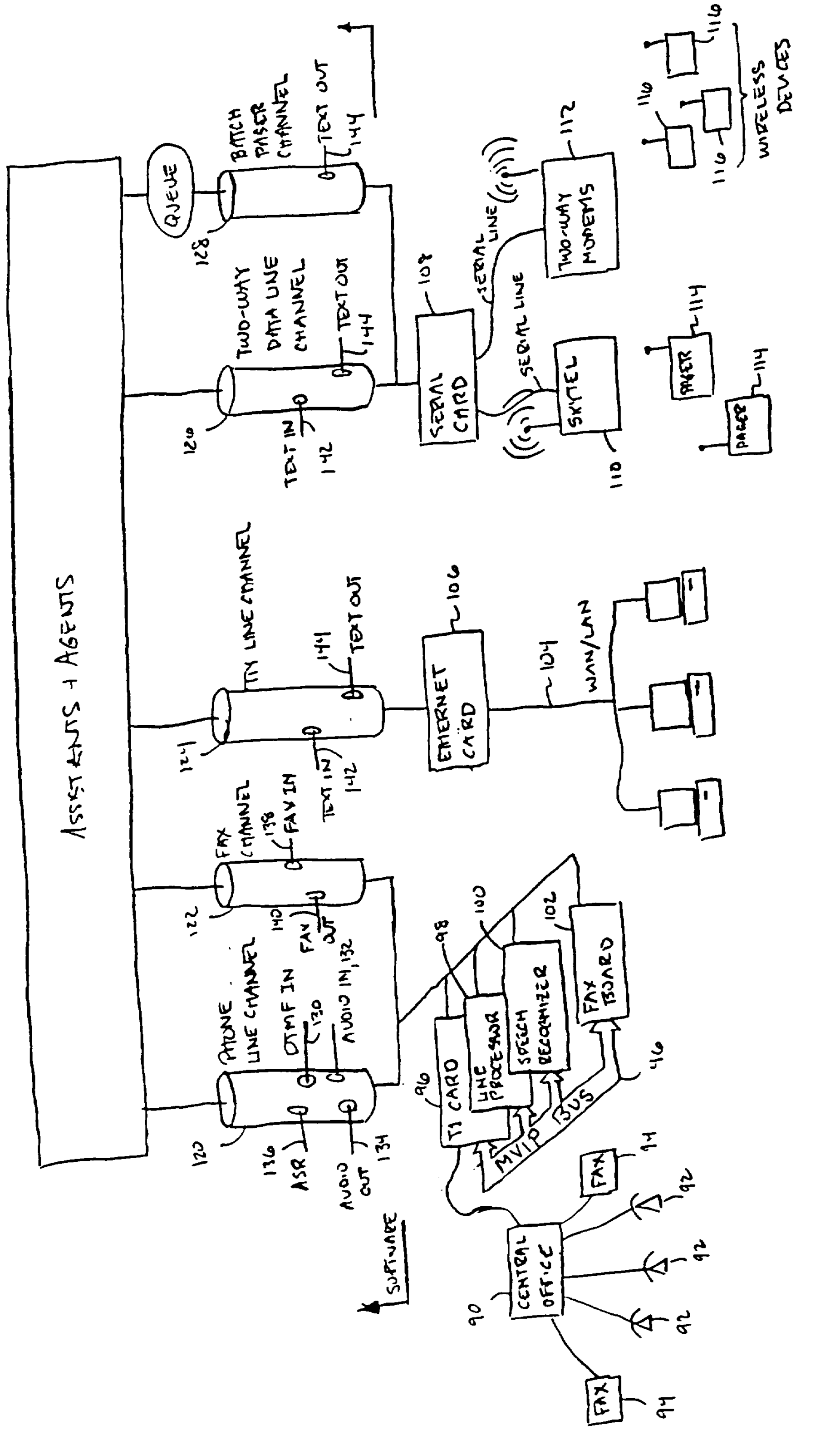

ASSISTANTS + AGENTS
QUEUE
128
BATCH PAGER CHANNEL
TEXT OUT
144
TWO-WAY DATA LINE CHANNEL
126
TEXT IN
142
TTY LINE CHANNEL
124
141
FAX CHANNEL
122
138
FAX IN
140
FAX OUT
PHONE LINE CHANNEL
120
DTMF IN
130
AUDIO IN, 132
ASR
136
AUDIO OUT
134
SOFTWARE
SERIAL CARD
108
SERIAL LINE
TWO-WAY MODEMS
112
SKYTEL
110
PAGER
114
WIRELESS DEVICES
116
ETHERNET CARD
106
WAN/LAN
104
T1 CARD
96
LINE PROCESSOR
98
SPEECH RECOGNIZER
100
FAX BOARD
102
MVIP BUS
46
CENTRAL OFFICE
90
FAX
94
92

FIG 5

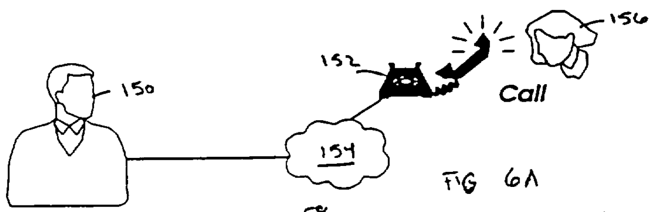


FIG 6A

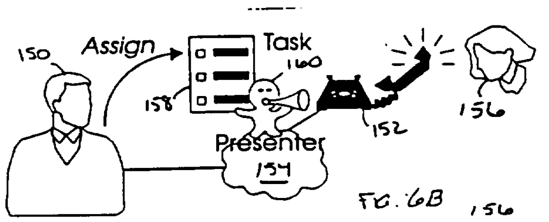


FIG. 6B

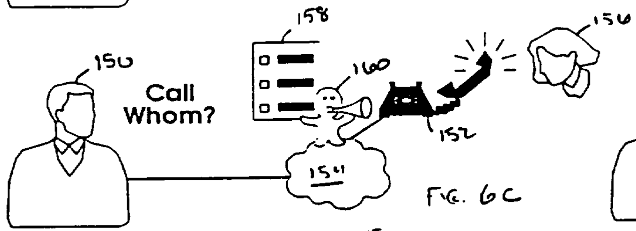


FIG. 6C

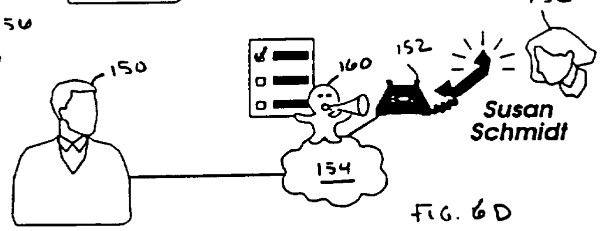


FIG. 6D

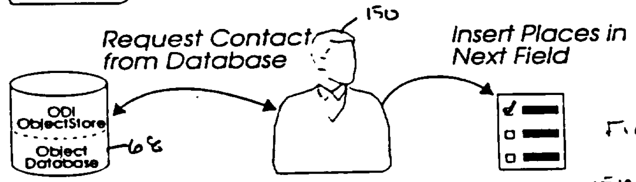


FIG. 6E

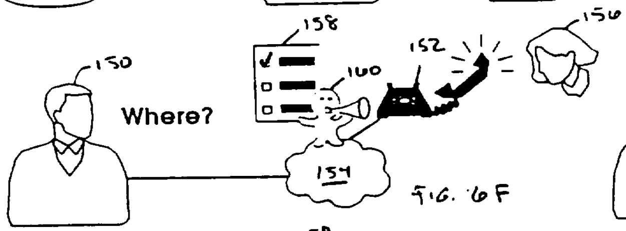


FIG. 6F

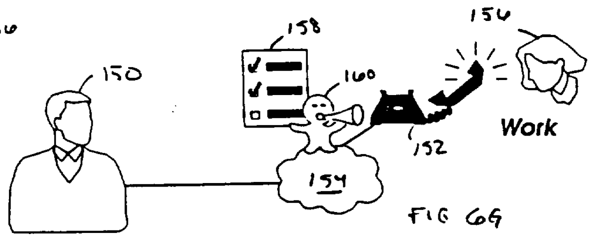


FIG 6G

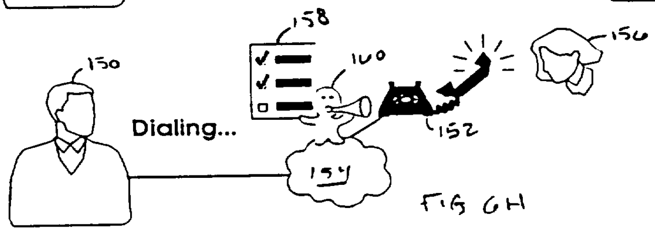
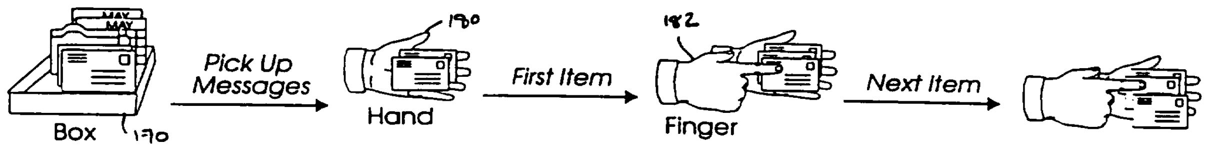
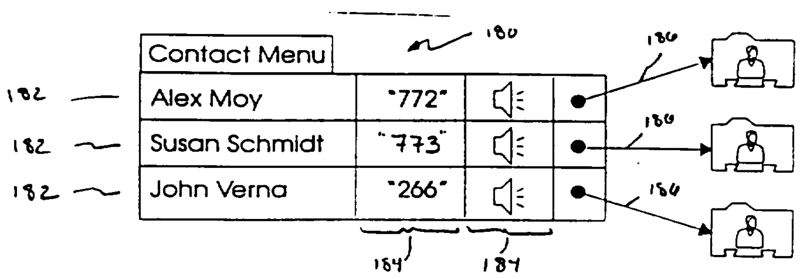


FIG 6H



Box
170
Pick Up Messages
Hand
190
First Item
182
Finger
Next Item

FIG. 7



Contact Menu
180
182
Alex Moy
"772"
182
Susan Schmidt
"773"
182
John Verna
"266"
184
184
186
186
186

FIG. 8

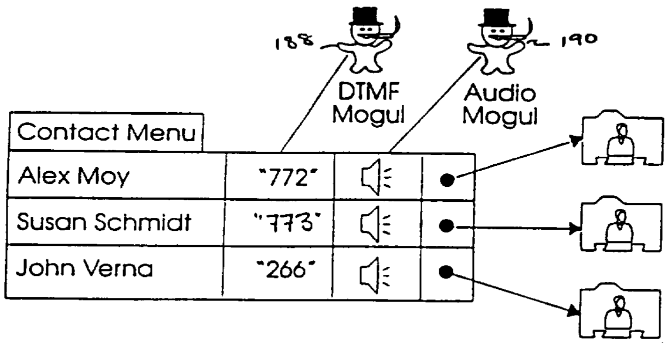


FIG. 9

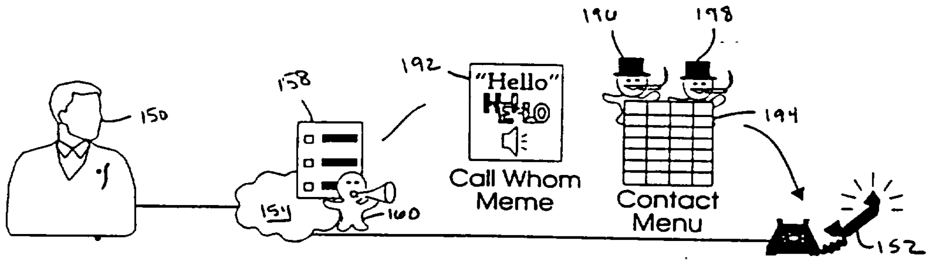


FIG 10A

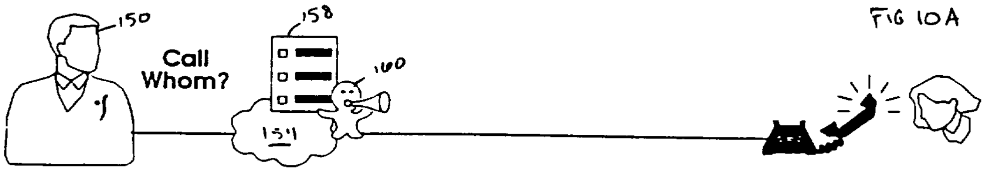


FIG. 10 B

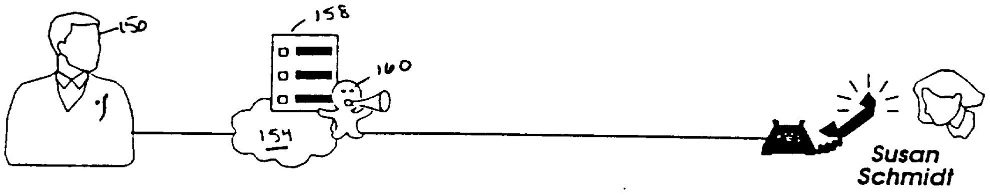


FIG 10C

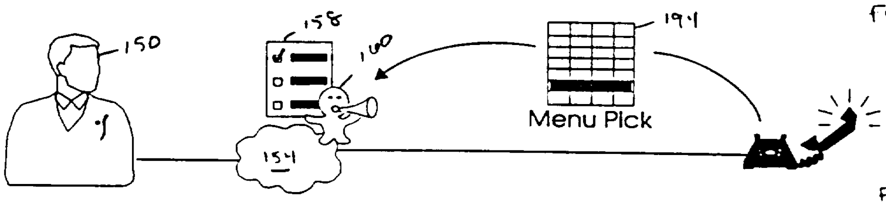


FIG. 10D

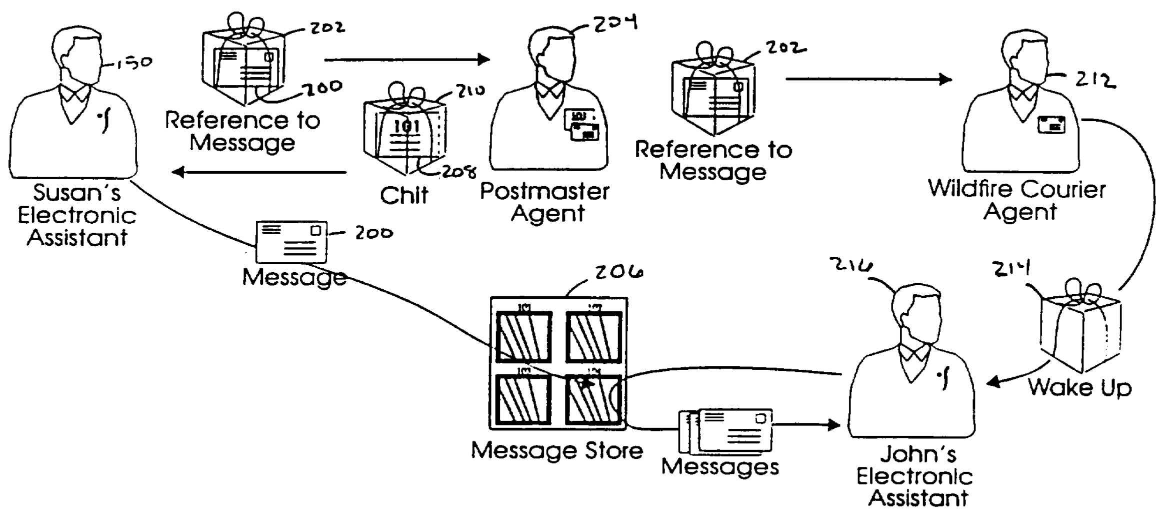

150
202
200
Reference to Message
210
101
208
Chit
Susan's Electronic Assistant
200
Message
204
Postmaster Agent
202
Reference to Message
212
Wildfire Courier Agent
206
Message Store
Messages
216
214
Wake Up
John's Electronic Assistant

FIG. 11

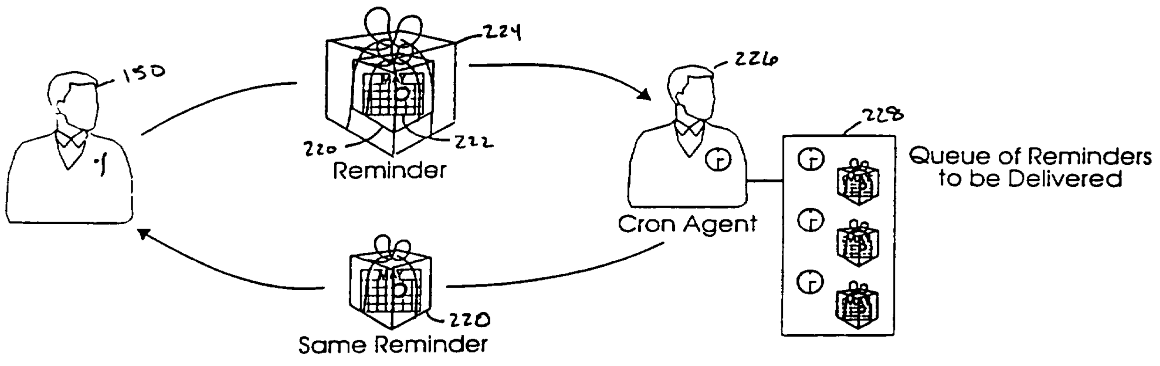


FIG. 12

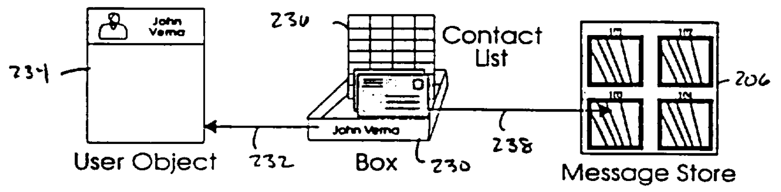


FIG. 13

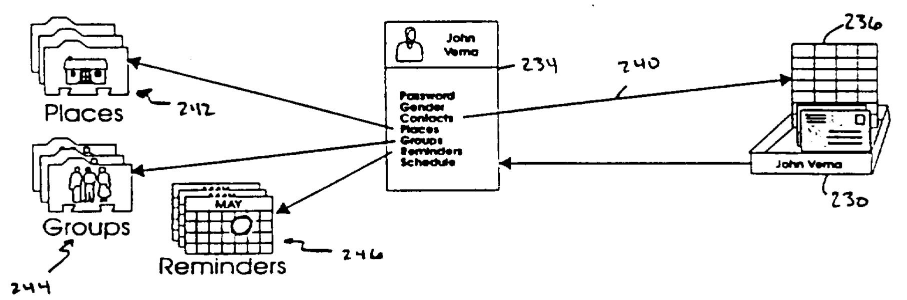


FIG. 14

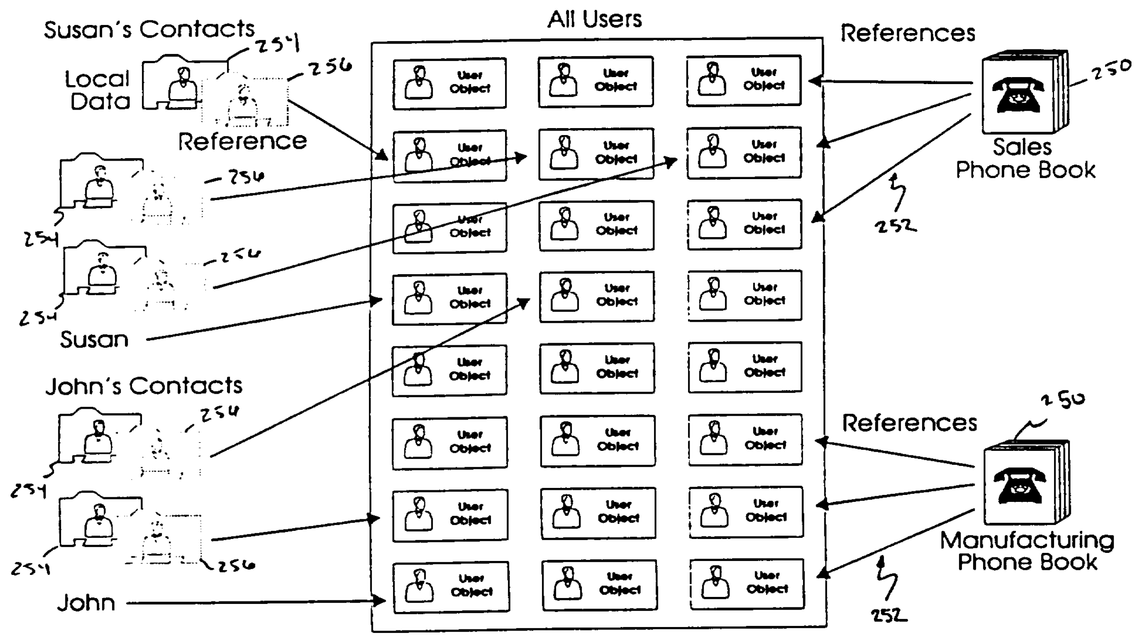

Susan's Contacts
254
256
Local Data
Reference
All Users
User Object
References
250
Sales Phone Book
252
254
256
Susan
John's Contacts
John
Manufacturing Phone Book

FIG 15

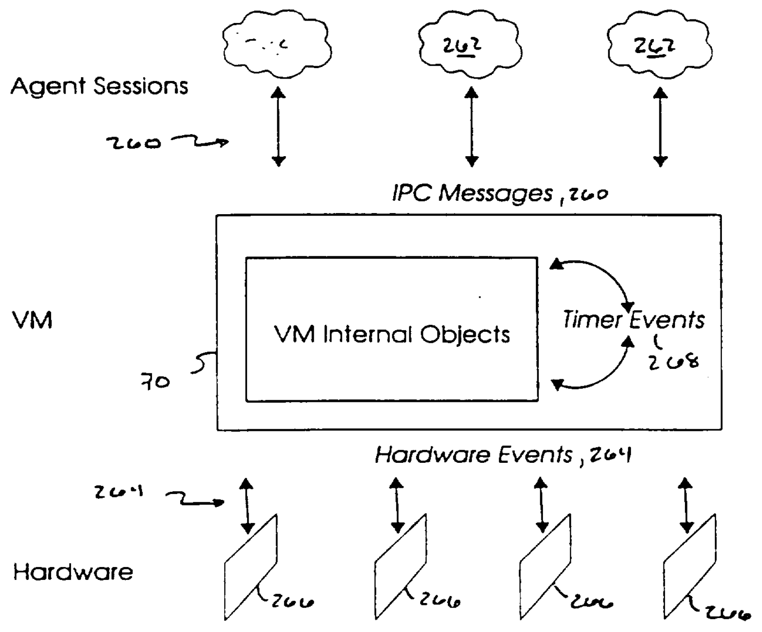

Agent Sessions
262
260
IPC Messages, 260
VM
VM Internal Objects
Timer Events
268
70
Hardware Events, 264
264
Hardware
266

FIG 16

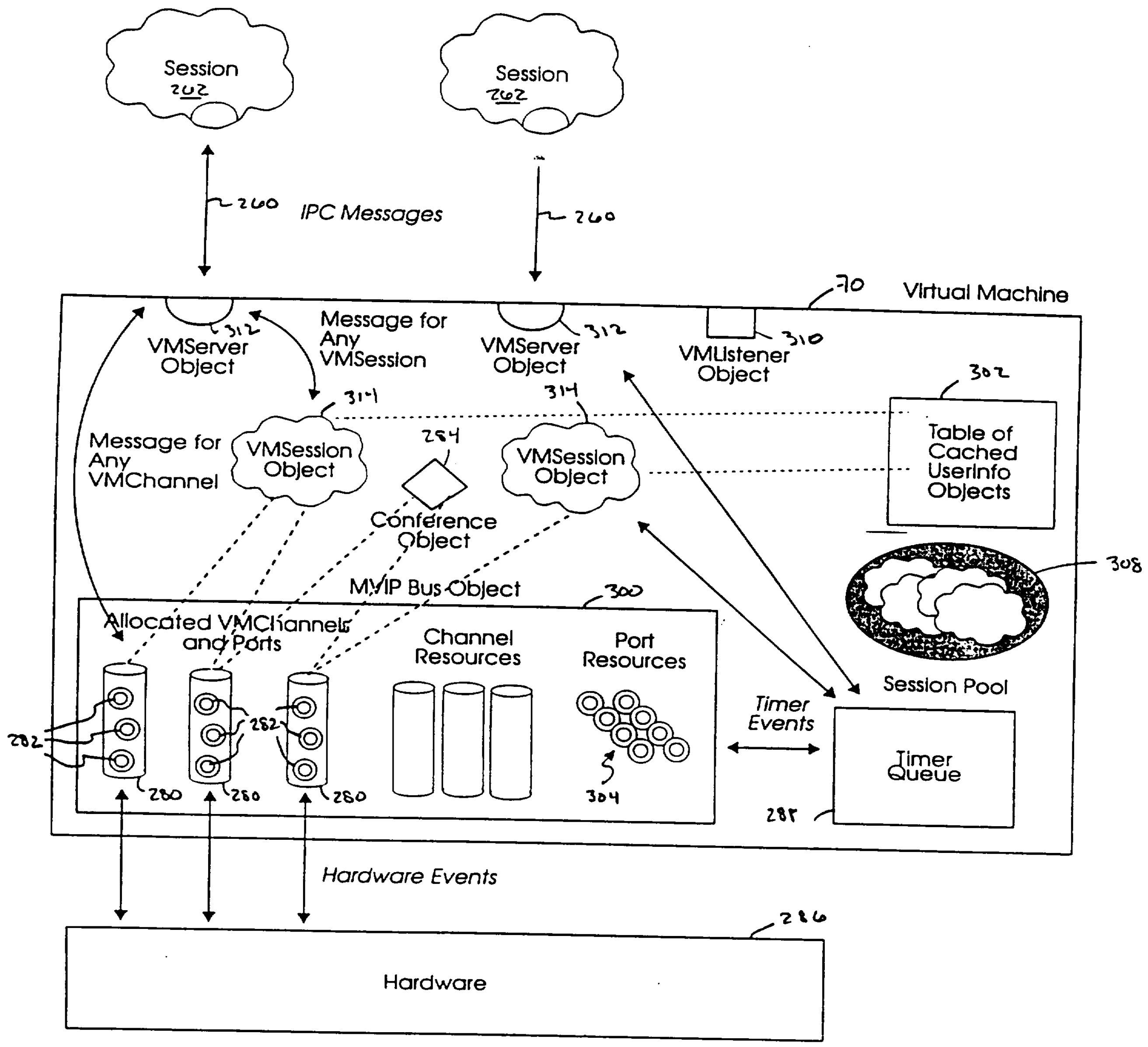

Session
262
Session
262
260
IPC Messages
260
70
Virtual Machine
312
VMServer Object
Message for Any VMSession
312
VMServer Object
310
VMListener Object
302
Table of Cached UserInfo Objects
314
314
Message for Any VMChannel
VMSession Object
284
VMSession Object
Conference Object
308
MVIP Bus Object
300
Allocated VMChannels and Ports
Channel Resources
Port Resources
Session Pool
282
282
Timer Events
Timer Queue
280
280
280
304
288
Hardware Events
286
Hardware

FIG. 17

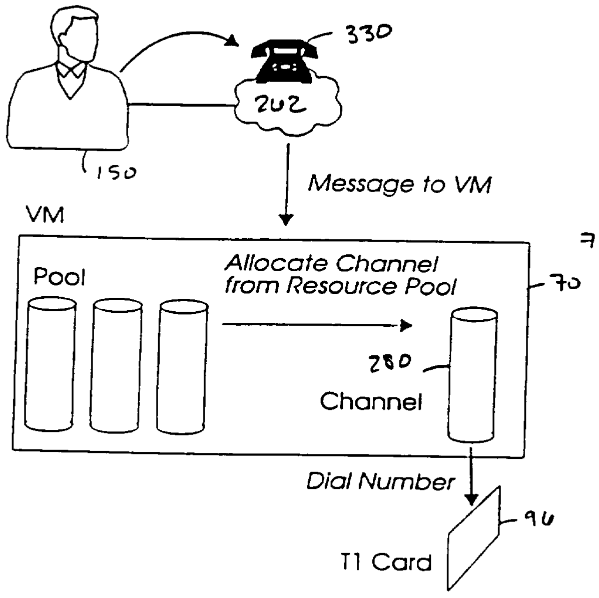


FIG. 18A

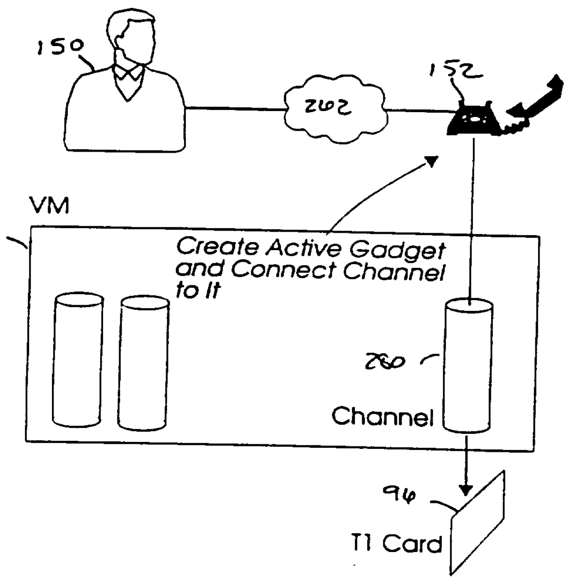


FIG 18B

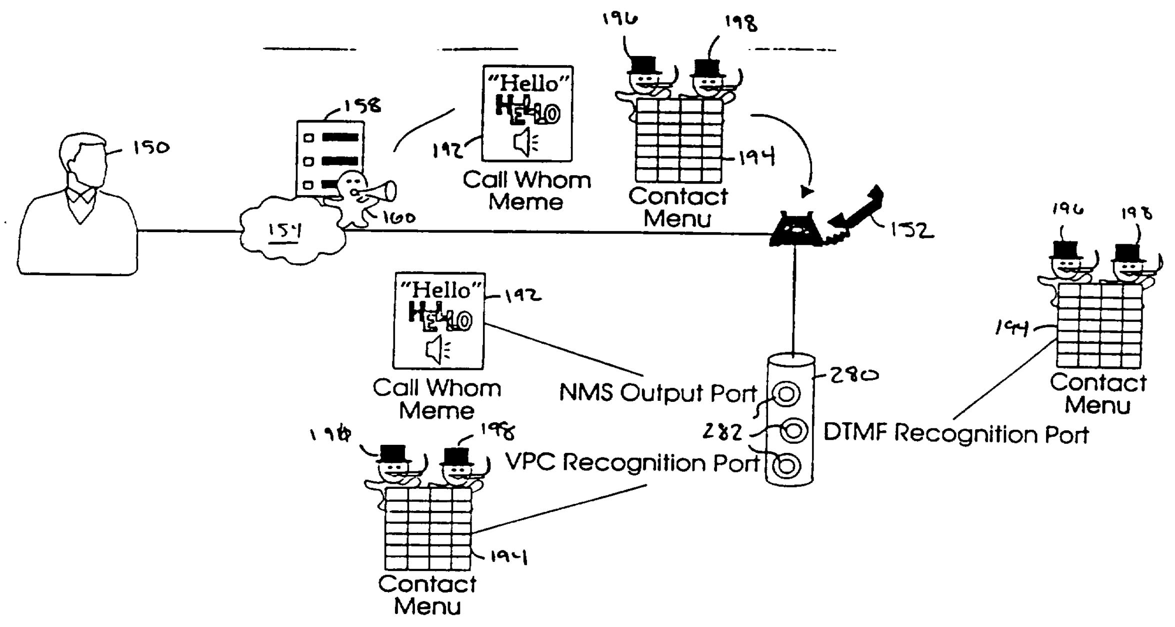

150
158
160
154
"Hello"
HELLO
192
Call Whom Meme
196
198
194
Contact Menu
152
"Hello"
HELLO
192
Call Whom Meme
NMS Output Port
280
282
DTMF Recognition Port
VPC Recognition Port
196
198
194
Contact Menu
196
198
194
Contact Menu

FIG. 19

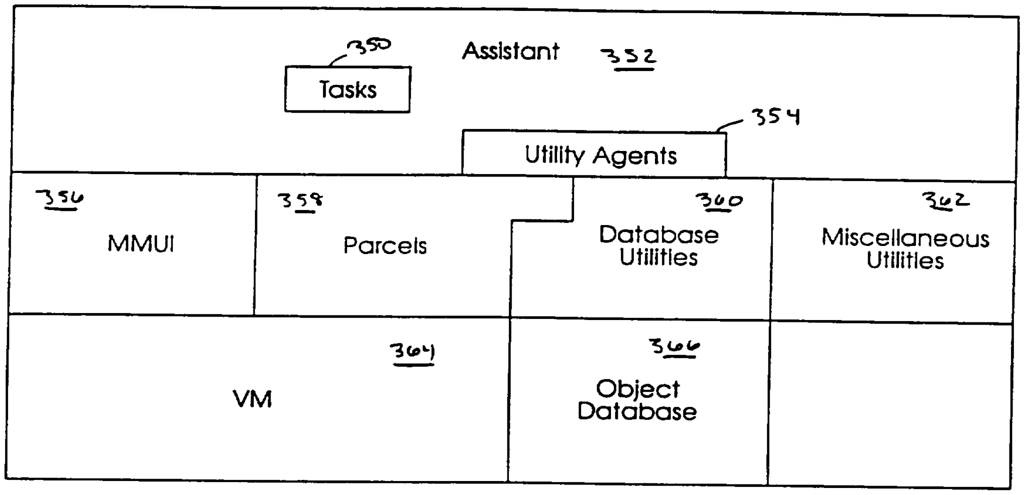

350
Tasks
Assistant 352
354
Utility Agents
356
MMUI
358
Parcels
360
Database Utilities
362
Miscellaneous Utilities
364
VM
366
Object Database

FIG. 20

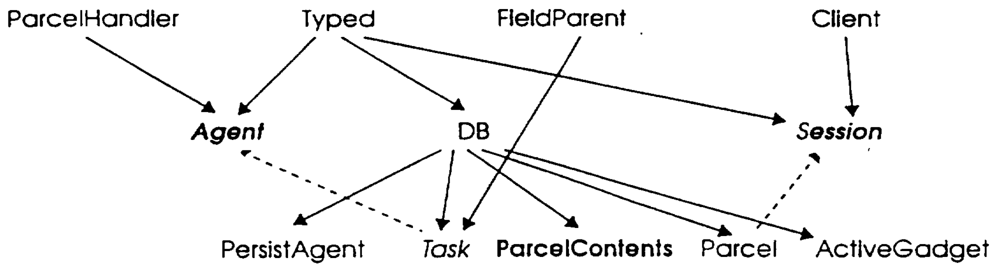

ParcelHandler
Typed
FieldParent
Client
Agent
DB
Session
PersistAgent
Task
ParcelContents
Parcel
ActiveGadget

FIG. 22

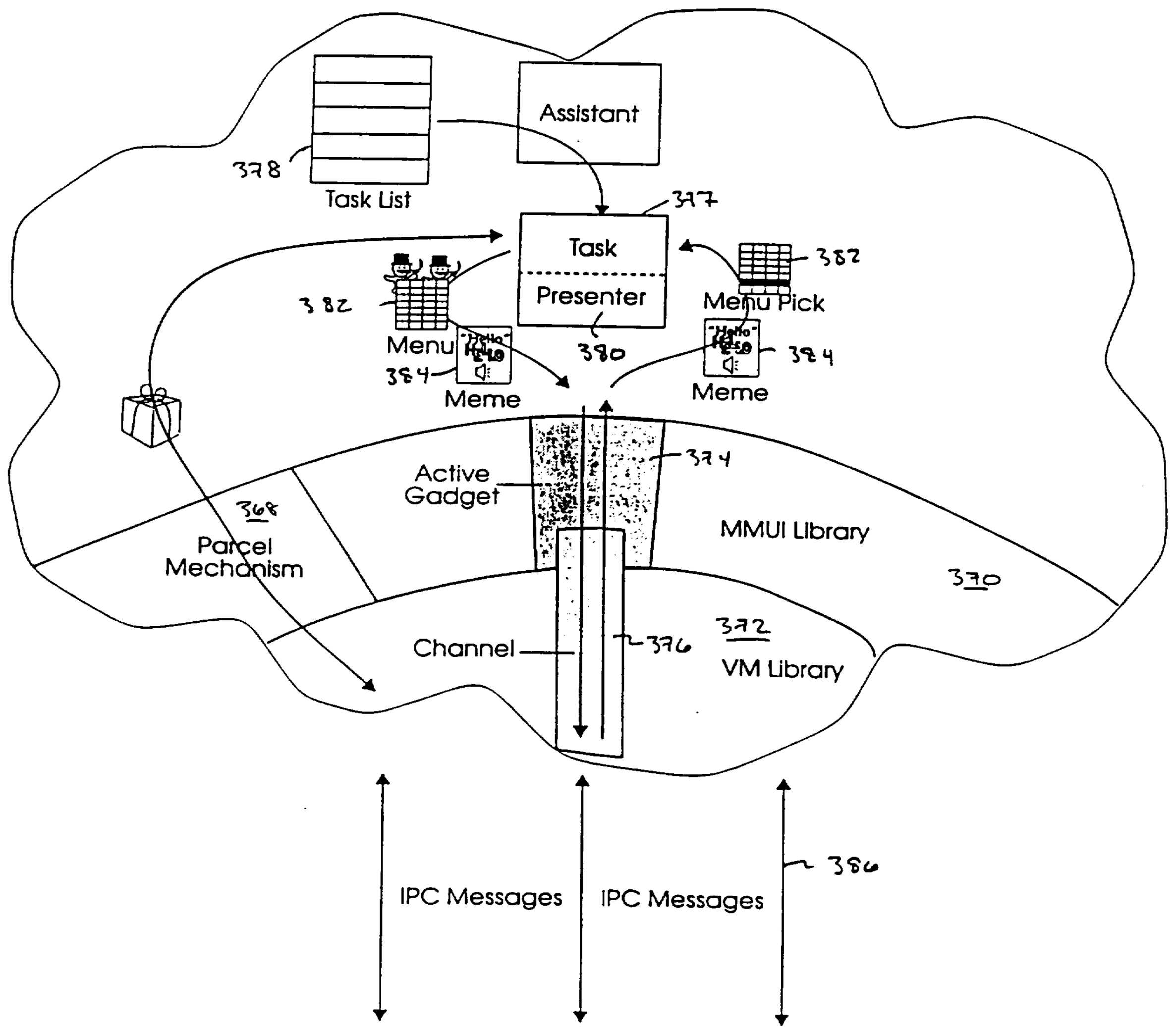

Task List
378
Assistant
Task
Presenter
377
380
382
Menu
384
Meme
382
Menu Pick
384
Meme
Active Gadget
374
368
Parcel Mechanism
MMUI Library
370
372
Channel
376
VM Library
IPC Messages
IPC Messages
386

FIG. 21

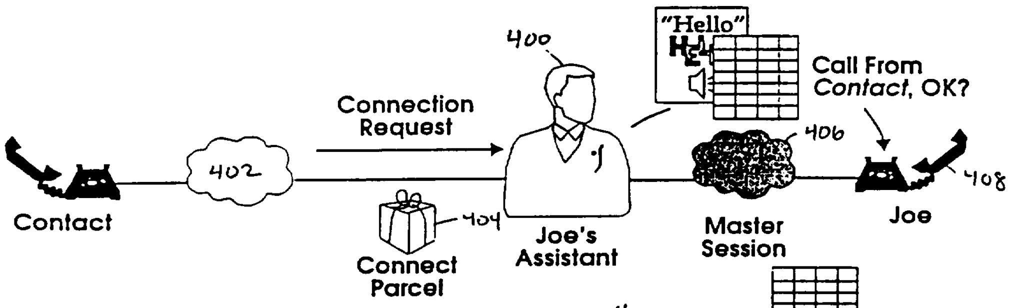


FIG. 23A

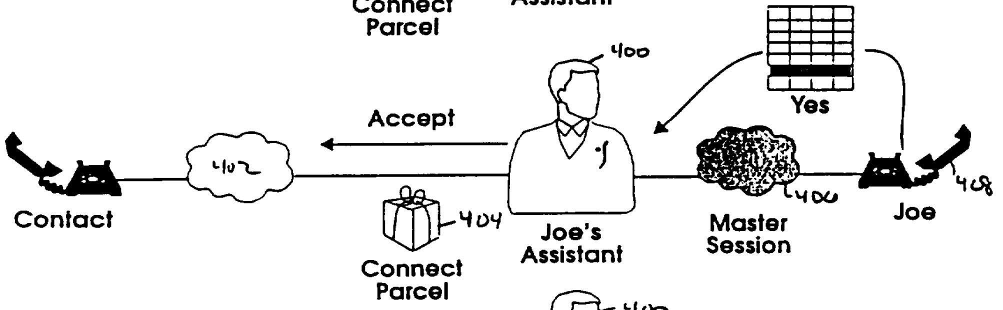


FIG. 23B

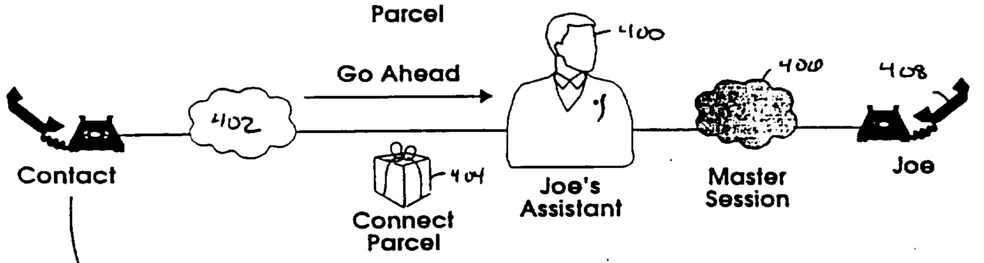


FIG. 23C

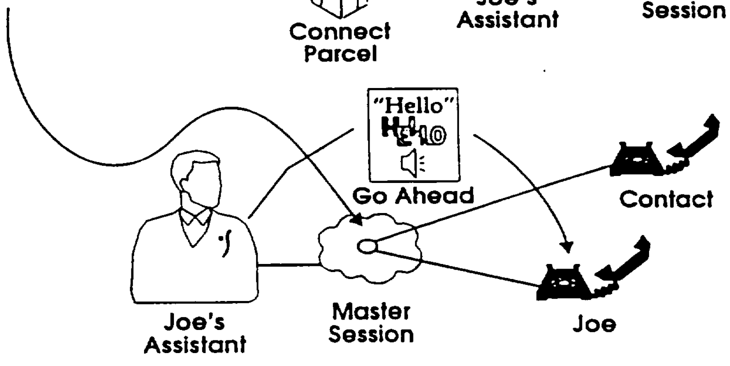


FIG. 23D

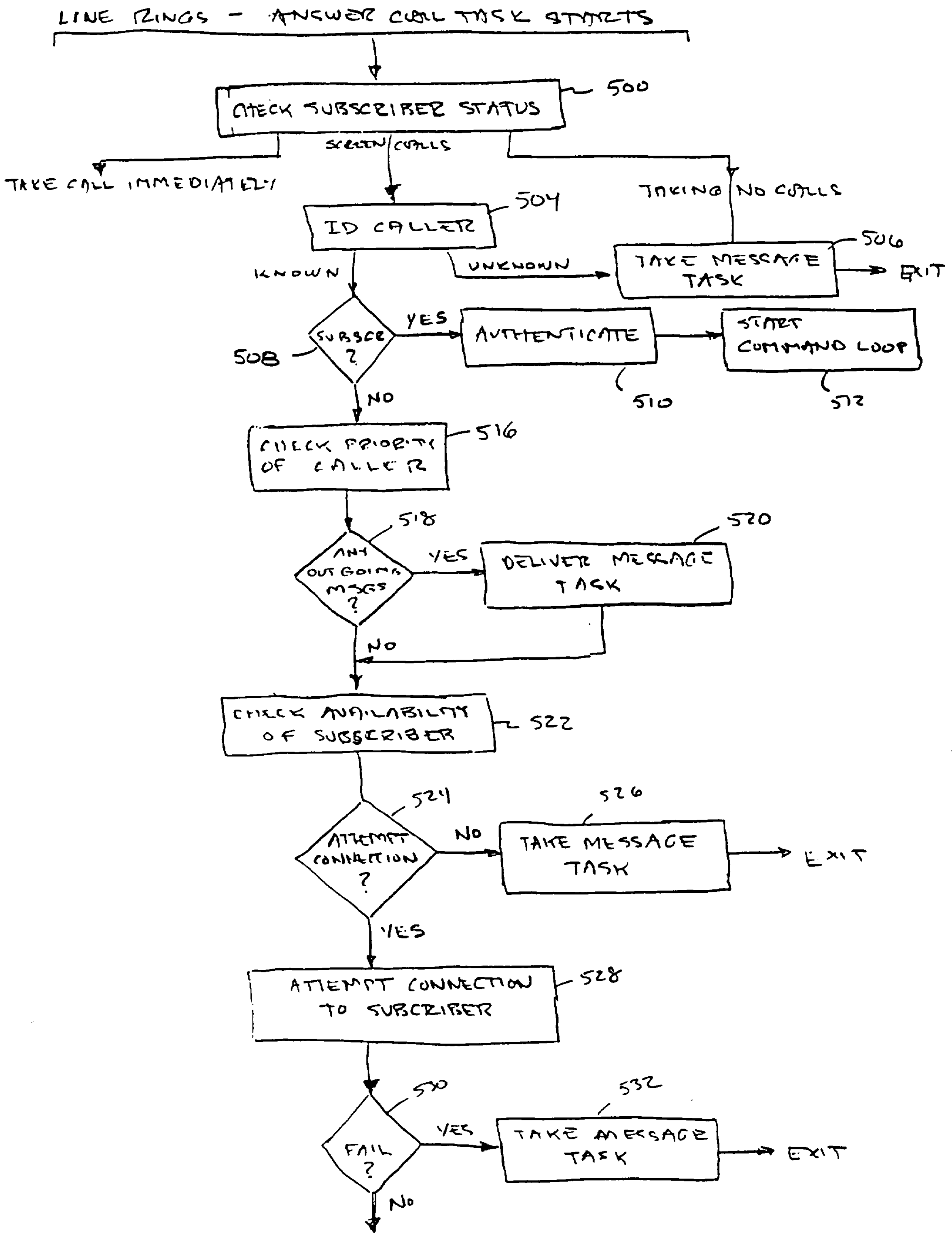

LINE RINGS - ANSWER CALL TASK STARTS
CHECK SUBSCRIBER STATUS
500
SCREEN CALLS
TAKE CALL IMMEDIATELY
TAKING NO CALLS
ID CALLER
504
UNKNOWN
TAKE MESSAGE TASK
506
EXIT
KNOWN
SUBSCR ?
508
YES
AUTHENTICATE
510
START COMMAND LOOP
512
NO
CHECK PRIORITY OF CALLER
516
518
ANY OUTGOING MSGS ?
YES
DELIVER MESSAGE TASK
520
NO
CHECK AVAILABILITY OF SUBSCRIBER
522
524
ATTEMPT CONNECTION ?
NO
TAKE MESSAGE TASK
526
EXIT
YES
ATTEMPT CONNECTION TO SUBSCRIBER
528
530
FAIL ?
YES
TAKE MESSAGE TASK
532
EXIT
NO

FIG. 24A

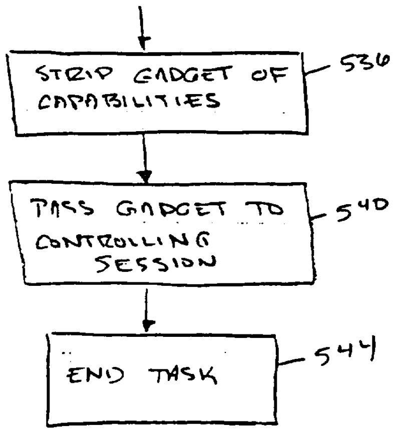

STRIP GADGET OF CAPABILITIES
536
PASS GADGET TO CONTROLLING SESSION
540
END TASK
544

FIG. 24B

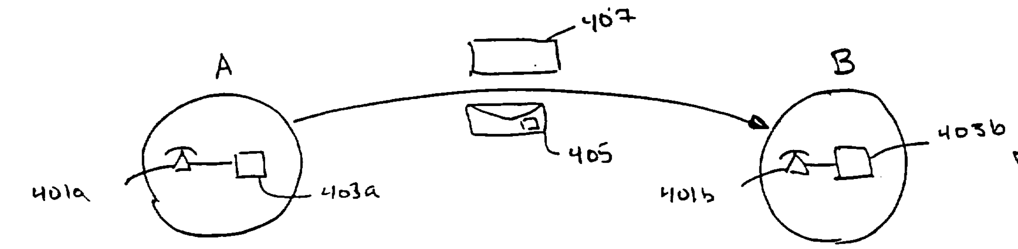


FIG. 25A

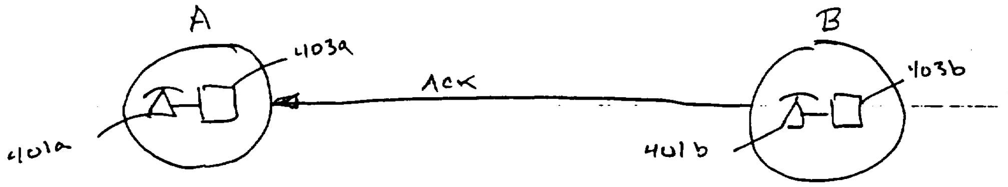


FIG. 25B

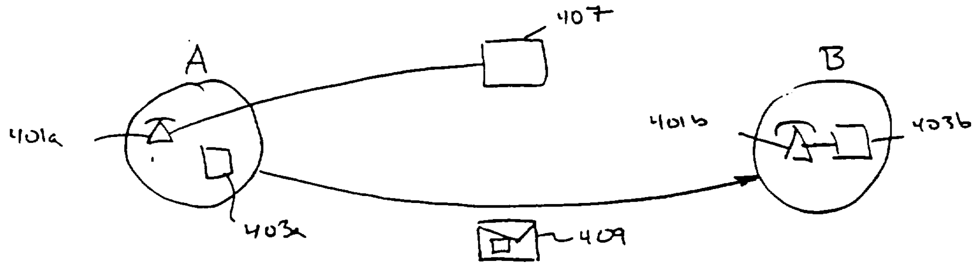


FIG. 25C

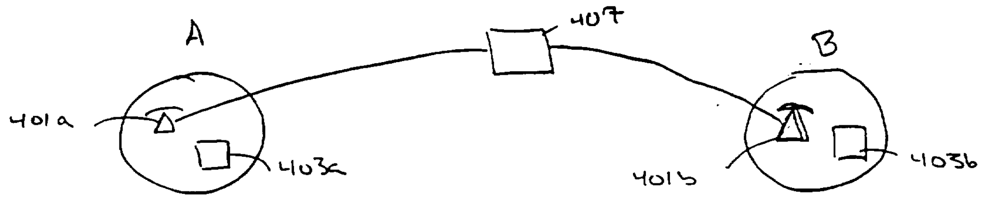


FIG. 25D

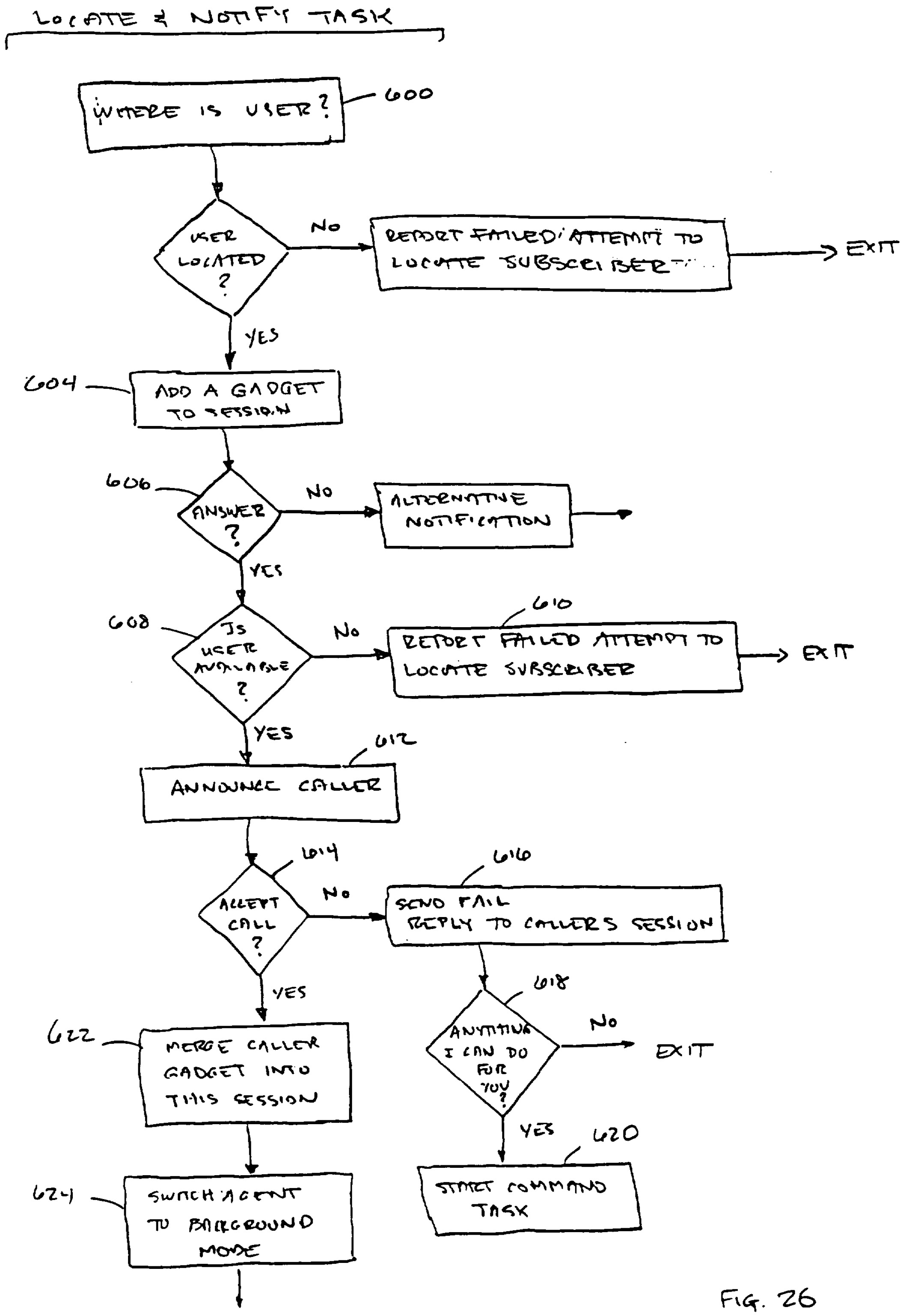

LOCATE & NOTIFY TASK
WHERE IS USER?
600
USER LOCATED ?
NO
REPORT FAILED ATTEMPT TO LOCATE SUBSCRIBER
EXIT
YES
604
ADD A GADGET TO SESSION
606
ANSWER ?
NO
ALTERNATIVE NOTIFICATION
YES
608
IS USER AVAILABLE ?
NO
610
REPORT FAILED ATTEMPT TO LOCATE SUBSCRIBER
EXIT
YES
612
ANNOUNCE CALLER
614
ACCEPT CALL ?
NO
616
SEND FAIL REPLY TO CALLERS SESSION
618
ANYTHING I CAN DO FOR YOU ?
NO
EXIT
YES
620
START COMMAND TASK
YES
622
MERGE CALLER GADGET INTO THIS SESSION
624
SWITCH AGENT TO BACKGROUND MODE

FIG. 26

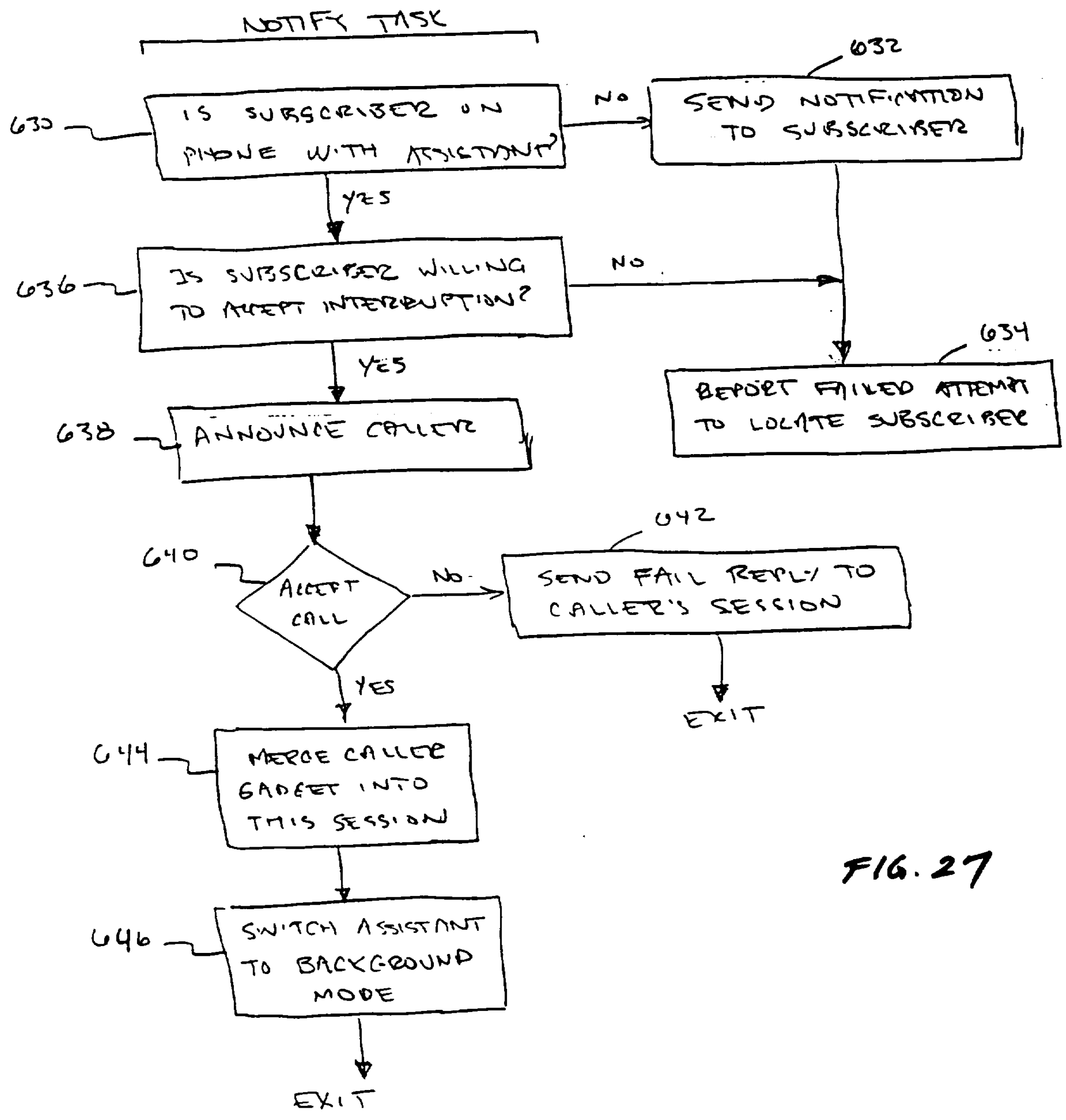

NOTIFY TASK
630
IS SUBSCRIBER ON PHONE WITH ASSISTANT?
NO
632
SEND NOTIFICATION TO SUBSCRIBER
YES
636
IS SUBSCRIBER WILLING TO ACCEPT INTERRUPTION?
NO
634
REPORT FAILED ATTEMPT TO LOCATE SUBSCRIBER
YES
638
ANNOUNCE CALLER
640
ACCEPT CALL
NO
642
SEND FAIL REPLY TO CALLER'S SESSION
EXIT
YES
644
MERGE CALLER GADGET INTO THIS SESSION
646
SWITCH ASSISTANT TO BACKGROUND MODE
EXIT

FIG. 27

SUBSCRIBER	ASSISTANT
Create-a-contact	WHAT KIND, PERSON, PLACE or GROUP?
Person	PERSON NAME?
John Smith	AGAIN PLEASE
John Smith	ADD WHICH NUMBER?
Work	PHONE NUMBER?
555-1212 #	GOT IT. NOW WHAT?
Update-it	UPDATE WHAT?
Phone number	WHICH PHONE NUMBER?
Home	PHONE NUMBER?
555-2121 #	GOT IT.
Update-it	UPDATE WHAT?
Priority	NORMAL OR HIGH?
Normal	GOT IT.
Update-it	UPDATE WHAT?
Spelling	BEGIN SPELLING NOW
j-o-h-n-- s-m-i-t-h #	GOT IT.
	WOULD YOU LIKE TO UPDATE THIS ITEM FURTHER?
No	DONE.

FIG. 28

SUBSCRIBER	ASSISTANT
Remind-me	ABOUT WHAT?
voice message delivered by subscriber	
Enter #	WHEN?
Today	TIME?
530 (using touch tone)	TODAY AT 5:30 IN THE AFTERNNON - IS THAT CORRECT?
Yes	GOT IT, I'LL SET MY WATCH ALARM.

FIG. 29

SUBSCRIBER	ASSISTANT
Remind-me-to-call	CALL WHOM?
John Smith	WHEN?
In 3 hours	IN 3 HOURS. IS THAT CORRECT?
Yes	SHOULD I KNOW THE TOPIC FOR THE CALL?
Yes	RECORDING...
voice message delivered by subscriber	
Enter #	GOT IT, I'LL SET MY WATCH ALARM.

FIG. 30

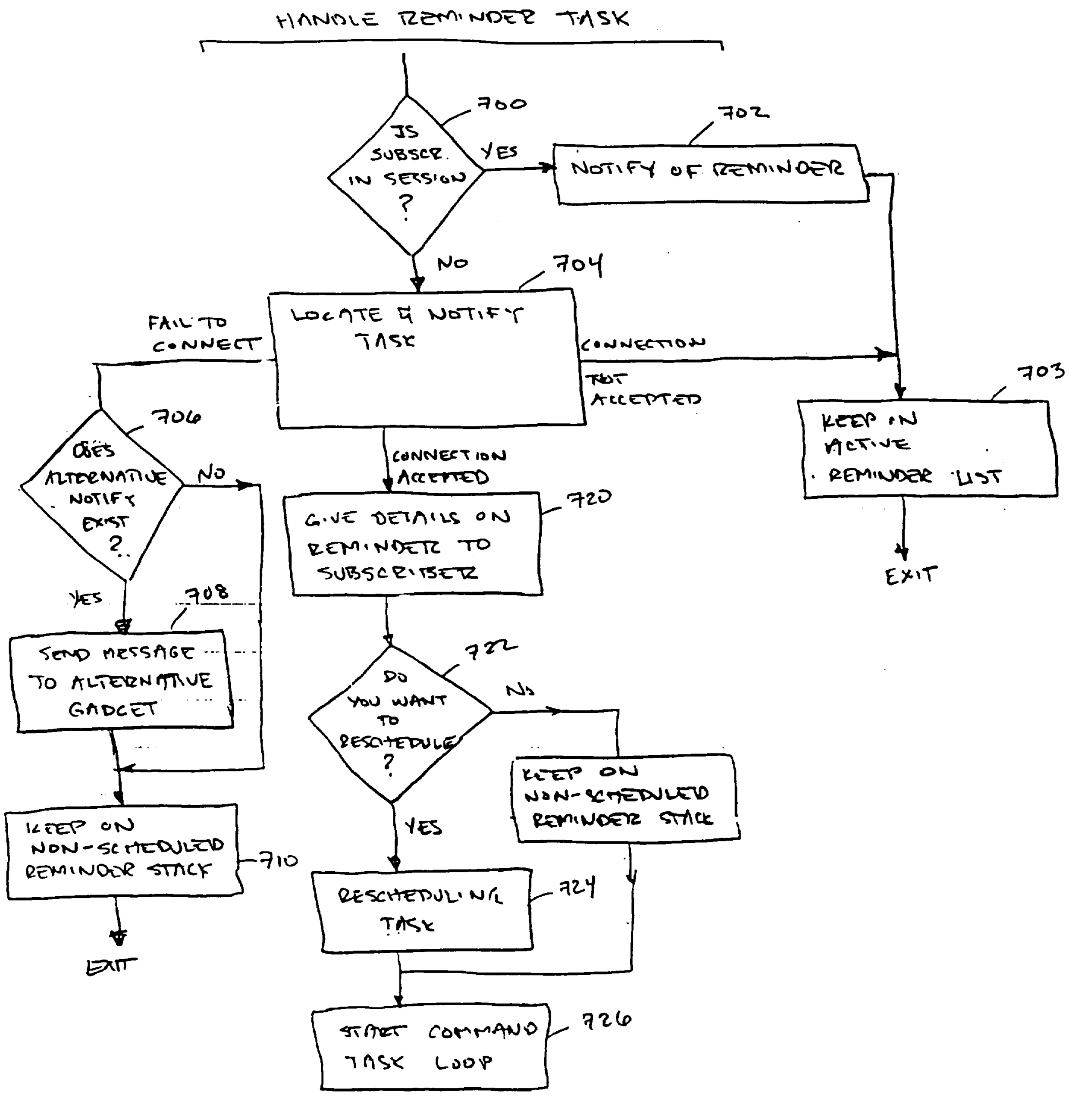

HANDLE REMINDER TASK
700
IS SUBSCR. IN SESSION ?
YES
702
NOTIFY OF REMINDER
NO
704
LOCATE & NOTIFY TASK
FAIL TO CONNECT
CONNECTION NOT ACCEPTED
703
KEEP IN ACTIVE REMINDER LIST
EXIT
706
DOES ALTERNATIVE NOTIFY EXIST ?
NO
YES
708
SEND MESSAGE TO ALTERNATIVE GADGET
KEEP ON NON-SCHEDULED REMINDER STACK
710
EXIT
CONNECTION ACCEPTED
GIVE DETAILS ON REMINDER TO SUBSCRIBER
720
722
DO YOU WANT TO RESCHEDULE ?
NO
KEEP ON NON-SCHEDULED REMINDER STACK
YES
RESCHEDULING TASK
724
START COMMAND TASK LOOP
726

FIG. 31

SUBSCRIBER	ASSISTANT
Find	FIND WHAT?
Contact	CONTACT NAME
John Smith	CONTACT NAMED JOHN SMITH
Describe	JOHN SMITH IS A PERSON SPELLED J-O-H-N S-M-I-T-H HOME PHONE NUMBER IS 555-1212 WORK PHONE NUMBER IS 555-2121
Update	WHAT?
Phone number	WHICH?
Work	I ALREADY HAVE THAT INFORMATION. THROW IT AWAY OR UPDATE IT.
Update it	PHONE NUMBER?
555-1313 #	OKAY. ANY MORE PHONE NUMBERS?
No	

FIG. 32

SUBSCRIBER	ASSISTANT
I-Will-Be	DOING WHAT?
Taking Calls	WHERE?
Mobile	FOR HOW MANY HOURS
Two	OK

FIG. 33

SUBSCRIBER	ASSISTANT
Create-an-Itinerary	DOING WHAT?
Taking Calls	WHERE?
Mobile	STARTING NOW?
Yes	UNTIL FURTHER NOTICE?
No	WHEN?
For · 2 · hours	FOR TWO HOURS, OKAY?
Yes	AND THEN YOU'LL BE?
Back on schdeule	

FIG. 34

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : **H04M 1/64, 3/42, 3/44, 3/50, 3/56, 11/06, 15/06, G06F 13/10, 13/14**	A3	(11) International Publication Number: **WO 96/11542** (43) International Publication Date: 18 April 1996 (18.04.96)

(21) International Application Number: PCT/US95/11737

(22) International Filing Date: 13 September 1995 (13.09.95)

(30) Priority Data:
08/316,635 30 September 1994 (30.09.94) US

(71) Applicant: WILDFIRE COMMUNICATIONS, INC. [US/US]; 20 Maguire Road, Lexington, MA 02173 (US).

(72) Inventors: MINER, Richard, A.; Apartment 105, 129 Franklin Street, Cambridge, MA 02139 (US). WARNER, William, J.; 48 Bogle Street, Weston, MA 02193 (US). LOVELL, Anthony, M.; 94 Raymond Avenue, Somerville, MA 02144 (US). SHIENBROOD, Eric, R.; 1763 Main Street, Concord, MA 01742 (US). GABRYELSKI, Keith; Apartment 2, 2 Drummond Place, Cambridge, MA 02140 (US). ARNOLD, Kenneth, C.R.C.; 7 Moon Hill Road, Lexington, MA 02173 (US). D'ARBELOFF, Nicholas, C.; 345 Cross Street, Belmont, MA 02178 (US). HINCKLEY, Kee; 25 Forest Circle, Winchester, MA 01890 (US).

(74) Agent: PRAHL, Eric, L.; Fish & Richardson P.C., 225 Franklin Street, Boston, MA 02110-2804 (US).

(81) Designated States: CA, JP, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

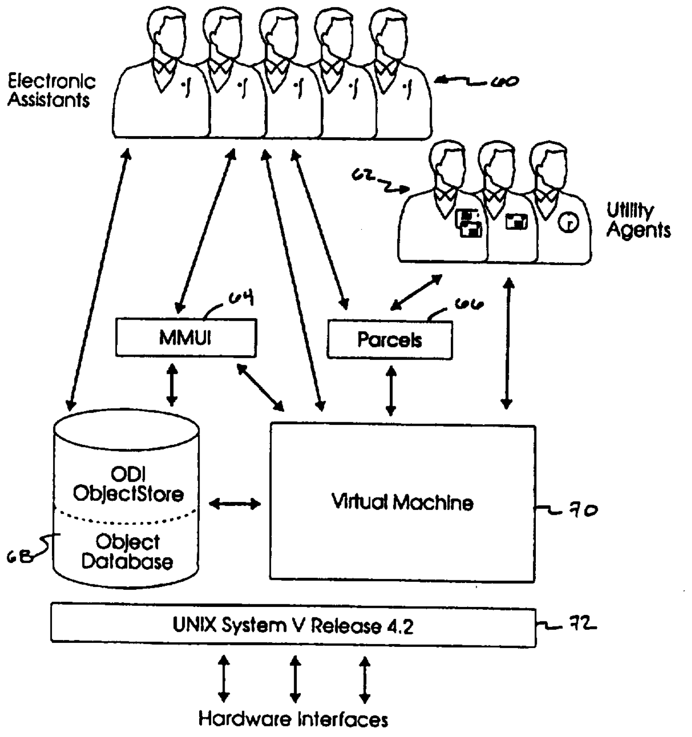
Published
With international search report.

(88) Date of publication of the international search report: 13 June 1996 (13.06.96)

(54) Title: A NETWORK BASED KNOWLEDGEABLE ASSISTANT

(57) Abstract

Methods implemented by a computer-based electronic assistant (60) include receiving and managing incoming calls to a subscriber. Methods include receiving and incoming call to the subscriber, establishing a connection between assistant (60) and subscriber, linking the caller and subscriber in response to receiving a call accept command, and switching the assistant (60) to a background mode. The assistant (60) monitors the subscriber for a summoning command while in the background mode and in response to receiving a summoning command, switches the assistant into a foreground mode which enables the subscriber to initiate conferencing features. Other methods include providing an electronic reminder service to the subscriber via use of personal locator and multimedia communication type determination, identification of a third party caller to subscriber while in an established connection, and providing multiple media type interaction capabilities.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgystan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo			SE	Sweden
CH	Switzerland	KR	Republic of Korea	SI	Slovenia
CI	Côte d'Ivoire	KZ	Kazakhstan	SK	Slovakia
CM	Cameroon	LI	Liechtenstein	SN	Senegal
CN	China	LK	Sri Lanka	TD	Chad
CS	Czechoslovakia	LU	Luxembourg	TG	Togo
CZ	Czech Republic	LV	Latvia	TJ	Tajikistan
DE	Germany	MC	Monaco	TT	Trinidad and Tobago
DK	Denmark	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	US	United States of America
FI	Finland	ML	Mali	UZ	Uzbekistan
FR	France	MN	Mongolia	VN	Viet Nam
GA	Gabon				

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US95/11737

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :H04M 1/64, 3/42, 3/44, 3/50, 3/56, 11/06, 15/06; G06F 13/10, 13/14
US CL :379/67, 102, 214, 215; 395/200
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 379/67, 102, 214, 215, 74, 87, 88, 89, 100, 101, 142, 167, 170, 201, 202, 203, 206, 210, 211, ;212, 213, 243, 244, 245, 258; 395/200

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X --- Y	✓US, A, 5,131,024 (Pugh et al.) 14 July 1992, col.1,ln.57-col.2,ln.30; col2,ln.45-col.4,ln.19; col.5,ln.11-64; col.6,ln.33-54.	1, 3 ------ 2, 4
X, P ----- Y, P	✓US, A, 5,414,754 (Pugh et al.) 09 May 1995, col.2,ln.37-53; col.2,ln.57-col.3,ln.9; col.4,ln.34-62, col.5,ln.9-34; col.6,ln.38-col.7,ln.22.	1, 3 ------- 2, 4
Y, P	✓"The Electronic Receptionist, A Knowledge-Based Approach to Personal Communications", Bellcore 1994, pages 1-8, especially page 3 through page 5.	1-4, 10-13
Y, P	✓US, A, 5,408,526 (McFarland et al.) 18 April 1995, Abstract; col.1,ln.61-col.2,ln.18; col.3,ln.36-col.4,ln.28; col.4,ln.44-59.	1-4, 14

[✓] Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance		
"E"	earlier document published on or after the international filing date	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search 17 FEBRUARY 1996	Date of mailing of the international search report 20 MAR 1996
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer [illegible] JEFFERY A. HOFSASS for Telephone No. (703) 305-4701

Form PCT/ISA/210 (second sheet)(July 1992)★

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US95/11737

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A, P	US, A, 5,436,963 (Fitzpatrick et al.) 25 July 1995, col.3,ln.36-col.5,ln.18.	6, 8
A --- Y	US, A, 5,329,578 (Brennan et al.) 12 July 1994, col.1,ln.66-col.2,ln.2; col.2,ln.17-33; col.2,ln.45-49; col.2,ln.66-col.3,ln.2; col.4,ln.19-35; col.6,ln.47-col.7,ln.55; col.8,ln.40-42 col.9,ln.7-17; col.12,ln.38-55; col.12,ln.64-col.13,ln.3; col.13,ln.57-col.14,ln.12.	8 ---- 10
Y	US, A, 5,195,086 (Baumgartner et al.) 16 March 1993, col.2,ln.32-col.3,ln.51; col.15,ln.62-col.17,ln.46, col.19,ln.62-col.20,ln.30.	2, 4, 14
Y	"WordPerfect: New Telephony Features Boost Office", WordPerfect Office TECHBRIEF, 20 January 1994, Info-World Publishing Co., Vol.10, Issue 2, pages 2-3.	6, 8
A	"Phoneshell: the Telephone as Computer Terminal", Chris Schmandt, ACM Multimedia '93 Conference, 1993 p.374-377	11-14
Y	"A Conversational Telephone Messaging System", Schmandt et al., IEEE Transactions on Consumer Electronics, August 1984, Vol.CE-30, No.3, pp.xxi-xxiv, especially page xxii, right column through page xxiii second paragraph.	11-14
X, P	US, A, 5,355,403 (Richardson, Jr. et al.) 11 October 1994, col.2,ln.31-51; col.3,ln.8-17; col.6,ln.27-col.9,ln.2.	11-13
Y	US, A, 4,873,719 (Reese) 10 October 1989, col.3,ln.1-68; col.4,ln.25-68; col.5,ln.60-col.6,ln.35; claim 1.	10
Y	US, A, 5,263,084 (Chaput et al.) 16 November 1993, col.2,ln.36-51; col.4,ln.14-col.5,ln.26.	10
Y	US, A, 5,333,266 (Boaz et al.) 26 July 1994, col.2,ln.34-col.3,ln.24; col.4,ln.1-col.7,ln.17; col.7,ln.43-64; col.10,ln.23-col.11,ln.53; col.13,ln.61-col.14,ln.60; col.19,ln.9-60.	11-14
Y, P	US, A, 5,384,771 (Isidoro et al.) 24 January 1995, col.2,ln.39-59; col.3,ln.23-44; col.4,ln.14-32; col.4,ln.45-col.5,ln.46.	14

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US95/11737

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This international report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

Please See Extra Sheet.

1. ☒ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest
☐ The additional search fees were accompanied by the applicant's protest.
☒ No protest accompanied the payment of additional search fees.

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

APS
search terms: Electronic Receptionist, assistant, secretary; personal locator, PCS, call transfer, call waiting, call forwarding, conferencing, three way call, message notification, reminder service, message delivery, schedule, calendar, multimedia, media type, channel selection

BOX II. OBSERVATIONS WHERE UNITY OF INVENTION WAS LACKING
This ISA found multiple inventions as follows:

I. Claims 1-4 drawn to a method performed by computer based system which receives incoming calls and directs incoming calls to a called party who then has control to invoke the system to perform various services, claims 1-4 present special technical feature A.
II. Claims 5-9 drawn to method performed by computer based system which method provides service via use of personal locator information and time dependency with message delivery service to an accepting answering party, claims 5-9 present special technical feature B.
III. Claim 10 drawn to method performed by computer based system which requires answering of incoming calls, dialogue between system and caller to identify caller, determination of busy status of called party and provision of call waiting alert with identification of caller to called party, claim 10 presents special technical feature C.
IV. Claims 11-13 drawn to method performed by computer based system a session is established from a first subscriber on behalf of a second subscriber and involves messaging between a first session on a first system and sending of a message to a second subscriber wherein the receiving of the first message by the second system causes a function to be performed and wherein the result of the function is reported to the first session thus providing remote programming feature, claims 11-13 present special technical feature D.
V. Claim 14 drawn to computer implemented method for processing communications through multimedia interface which includes a plurality of linked I/O devices and interface devices with capability to connect to various different networks, retrieval of an associated media type information and delivery of the information over a respective network to a requesting multimedia device, claim 14 presents special technical feature E.

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : **H04M 1/00, 1/64**	**A1**	**(11) International Publication Number:** **WO 97/33416** **(43) International Publication Date:** 12 September 1997 (12.09.97)

(21) International Application Number: PCT/US97/03414

(22) International Filing Date: 7 March 1997 (07.03.97)

(30) Priority Data:
08/612,256 7 March 1996 (07.03.96) US

(71) Applicant *(for all designated States except US)*: AMERICAN EXPRESS TRAVEL RELATED SERVICES COMPANY, INC. [US/US]; c/o General Counsel's Office, American Express Tower, World Financial Center, New York, NY 10285 (US).

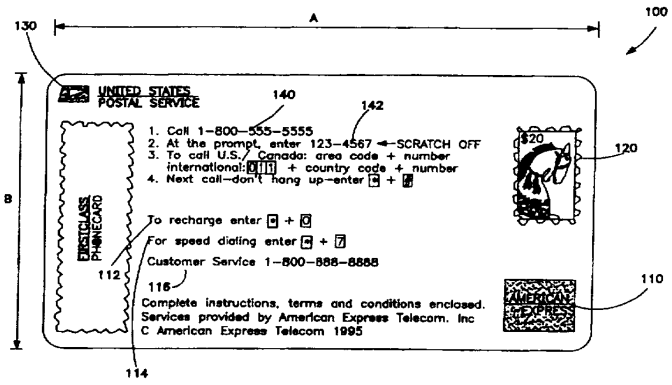
(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: TASKETT, John, M. [US/US]; 2673 East Coquina Court, Salt Lake City, UT 84121 (US).

(74) Agents: KELLY, Michael, K. et al.; Snell & Wilmer, One Arizona Center, 400 East Van Buren Street, Phoenix, AZ 85004-0001 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: METHODS AND APPARATUS FOR PROVIDING A PREPAID, REMOTE MEMORY TRANSACTION ACCOUNT WITH VOICE INDICIA


130
A
100
B
UNITED STATES POSTAL SERVICE
140
142
1. Call 1-800-555-5555
2. At the prompt, enter 123-4567 ← SCRATCH OFF
3. To call U.S./ Canada: area code + number
international: 011 + country code + number
4. Next call—don't hang up—enter * + #
$20
120
FIRSTCLASS PHONECARD
To recharge enter * + 0
For speed dialing enter * + 7
Customer Service 1-800-888-8888
112
116
Complete instructions, terms and conditions enclosed.
Services provided by American Express Telecom. Inc
C American Express Telecom 1995
AMERICAN EXPRESS
110
114

(57) Abstract

An improved distributed system for effectuating commercial transactions by a consumer is provided. The system includes a service provider network or host system (308) which comprises an account database (310) of prepaid accounts and an audio library containing audio indicia or sound bites. A prepaid instrument (100) is issued to a consumer for providing access to the service provider network, the audio library, and the account database (310). The prepaid instrument (100) has two sides. On one side of the instrument is a visual indicia such as a photograph, drawing, picture, or other image. On the other side of the instrument is a telephone number for accessing the host system and the audio library containing audio indicia that relates to the visual indicia on the prepaid instrument (100), and a predetermined authorization code that allows access to the account database. While connected to the service provider network, audio indicia relating to the visual indicia on the prepaid instrument (100) is suitably communicated to the consumer at appropriate times or intervals.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	GB	United Kingdom	MW	Malawi
AT	Austria	GE	Georgia	MX	Mexico
AU	Australia	GN	Guinea	NE	Niger
BB	Barbados	GR	Greece	NL	Netherlands
BE	Belgium	HU	Hungary	NO	Norway
BF	Burkina Faso	IE	Ireland	NZ	New Zealand
BG	Bulgaria	IT	Italy	PL	Poland
BJ	Benin	JP	Japan	PT	Portugal
BR	Brazil	KE	Kenya	RO	Romania
BY	Belarus	KG	Kyrgystan	RU	Russian Federation
CA	Canada	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic	KR	Republic of Korea	SE	Sweden
CG	Congo	KZ	Kazakhstan	SG	Singapore
CH	Switzerland	LI	Liechtenstein	SI	Slovenia
CI	Côte d'Ivoire	LK	Sri Lanka	SK	Slovakia
CM	Cameroon	LR	Liberia	SN	Senegal
CN	China	LT	Lithuania	SZ	Swaziland
CS	Czechoslovakia	LU	Luxembourg	TD	Chad
CZ	Czech Republic	LV	Latvia	TG	Togo
DE	Germany	MC	Monaco	TJ	Tajikistan
DK	Denmark	MD	Republic of Moldova	TT	Trinidad and Tobago
EE	Estonia	MG	Madagascar	UA	Ukraine
ES	Spain	ML	Mali	UG	Uganda
FI	Finland	MN	Mongolia	US	United States of America
FR	France	MR	Mauritania	UZ	Uzbekistan
GA	Gabon			VN	Viet Nam

METHODS AND APPARATUS FOR PROVIDING A PREPAID, REMOTE MEMORY TRANSACTION ACCOUNT WITH VOICE INDICIA

Technical Field

The present invention relates, generally, to prepaid, remote memory accounts used by a consumer with a transaction card to purchase goods and services and, more particularly, to methods and apparatus which permit the remote memory account to communicate to the consumer using prerecorded voice prompts associated with famous personalities and related to the transaction card.

Background Art and Technical Problems

Remote memory, prepaid accounts for use in purchasing goods and services are generally well known. Presently known schemes typically involve a printed transaction card, for example a wallet-sized plastic or cardboard card, which bears on one side a unique authorization or account number and instructions for access to funds, services, and the like. Often, the other side of the transaction card includes a photograph, drawing or other aesthetically pleasing image. Such prepaid cards have been used extensively throughout the world. One such example is the use of these remote memory cards as prepaid long distance telephone calling cards.

In contrast to stored value cards (*e.g.*, "smart cards") wherein a remaining account balance is stored in a memory circuit resident in the card, remote memory systems typically store information pertaining to a prepaid account at a central host computer. The host computer typically stores information relating to the available balance remaining in the account, as well as information pertaining to past activity associated with the account. In particular, the host computer may store transaction data relating to various goods or services purchased using the card. In the context of a prepaid telephone calling card, the host computer may store call record data, including the date, time, duration, and various other parameters relating to calls which were placed using the prepaid telephone card corresponding to the account. The host computer may be accessed via a telephone or data line by the consumer through the use of an authorization code, Personal Identification Number (PIN), or the like.

The use of prepaid remote memory telephone cards is particularly prevalent in the United States. A typical prepaid telephone calling card includes a toll-free telephone number used by a consumer to access a host computer system, a unique authorization code associated with the card (and, hence, the corresponding remote account), and dialing instructions. When a customer desires to use the card to place a long distance call, he dials the toll-free number, thereby accessing the host system which manages the remote accounts. The consumer then enters a predetermined authorization number for allowing access to the database. Next, the consumer then enters the desired long distance telephone number, and the system connects the consumer with the desired calling destination. Long distance telephone charges attributed to the telephone call are deducted from the remaining balance, and the call is terminated when the account is fully consumed. A call history may also be maintained by the host for each account, which call history includes information pertaining to the calls charged to the various accounts.

Prepaid calling cards are employed in a wide variety of applications for both personal and professional use. For example, various governmental entities and other employers often issue per diem cards to employees to accommodate employee travel; the per diem cards may be issued in predetermined amounts (*e.g.*, $100), and permit a traveling employee to charge gasoline, rental cars, hotels, meals, and telephone calls to the prepaid account. When the charges equal the prepaid limit, the card may simply be discarded; alternatively, the card may be "recharged" by the employer, as desired. In addition, various prepaid instruments are becoming increasingly popular as gift cards, on college campuses for purchasing school and living supplies, and for use at resorts, vacation areas, theme parks, sports stadiums, and the like.

As the number of card issuers increases and the calling schemes become more standardized, it becomes more challenging for an issuer of transaction cards to distinguish his product from others. A system and method is thus needed which permits an issuer to market his cards in a manner which distinguishes over the standard transaction card.

Summary of the Invention

The present invention provides an improved distributed system for effectuating commercial transactions by a consumer. This system includes a service provider network or host system which comprises an account database of prepaid accounts and an audio database containing audio indicia or sound bites. A prepaid instrument (*e.g.*, a phone card) is issued to a consumer for providing access to the service provider network, the audio database, and the account database. In a preferred embodiment, the prepaid instrument has two sides. One side displays a photograph or other visual indicia such as a printed drawing or other aesthetically pleasing image. The other side of the instrument provides a telephone number for accessing the host system and a portion of the audio database containing audio indicia which is related to the visual indicia on the prepaid instrument, and a predetermined authorization code that allows access to the account database. While connected to the service provider network, audio indicia corresponding to the visual indicia on the prepaid instrument is communicated over the telephone to the user at appropriate times or intervals.

In accordance with one embodiment of the invention, the prepaid transaction instrument comprises a prepaid telephone card. On the front side of the prepaid telephone card is an image or photograph of a living or deceased person such as a famous celebrity. The back side of the telephone card comprises an access telephone number and a predetermined authorization code. A service provider network or host system comprises an account database of prepaid accounts and a voice library or storage facility containing voice snippets of persons pictured on the front of prepaid telephone cards. Upon dialing the access telephone number, a reproduction (*e.g.*, recording) of the voice of the person pictured on the telephone card is communicated over the telephone line to the user of the card. The voice may provide instructions on using the telephone card or may prompt the user for additional information such as the predetermined authorization code for accessing the account database and the desired long distance phone number.

In accordance with another embodiment of the present invention, an image such as an animal, musical instrument, automobile or the like may be displayed on the front side of a telephone card. An audio sound bite relating to (or, alternatively, independent of) the image on the telephone card is then relayed to the user of the card upon connection to the service provider network. For example, in the case of an antique automobile pictured on the front side of the telephone card, sounds associated with that automobile may be communicated to the user of the phone card when the service provider network is accessed.

In accordance with another aspect of the present invention, an ensemble or "package" comprising a transaction instrument (*e.g.*, a calling card) and various related paraphernalia may be sold together in an envelope, for example a trading card, a postage stamp, or series of postage stamps having an image substantially similar or identical to the photograph or other visual indicia on the front side of the telephone card.

Brief Description of the Drawing Figures

The present invention will hereinafter be described in conjunction with the appended drawing figures, wherein:

Figure 1 is a view of one side of an exemplary telephone card in accordance with the present invention;

Figure 2 is a view of the other side of the prepaid telephone card of

Figure 1;

Figure 3 is a schematic block diagram wherein the telephone card of FIGS. 1 and 2 is used in the context of a long distance telephone service system;

Figure 4 is a schematic diagram of an exemplary host computer system in accordance with the present invention;

Figure 5 is a schematic representation of a voice library for storing voice reproductions or recordings of people pictured on corresponding telephone cards; and

Figure 6 is a series of postage stamps issued with the telephone card of FIGS. 1 and 2 in accordance with the present invention.

Detailed Description of Preferred Exemplary Embodiments

A preferred exemplary embodiment of the present invention surrounds a prepaid telephone card; however, it will be understood that the invention is not so limited. In particular, it will be appreciated that the present invention broadly contemplates virtually any type of prepaid transaction card instrument or methodology for virtually any type of goods or services of whatever kind or nature. Although the present invention contemplates stored value cards (*e.g.*, smart cards), a preferred exemplary embodiment described herein surrounds a prepaid, remote memory account telephone card which provides dialing instructions for permitting the holder of the card to make toll-free telephone calls from virtually any telephone extension.

Prepaid calling cards may be purchased from a variety of retail outlets, for example, convenience stores, drug stores, gas stations, supermarkets, and the like. For a more thorough discussion of prepaid instruments and telephone cards, see United States patent application serial number 08/456,525 entitled *Methods and Apparatus for Providing a Prepaid, Remote Memory Customer Account*, and serial number 08/458,715 entitled *Refundable Prepaid Telephone Card*, both filed June 1, 1995, by John

Taskett; serial number 08/510,590 entitled *Methods and Apparatus for Providing a Prepaid, Remote Entry Customer Account for the Visually Impaired*, filed in the names of John Taskett and Barbara Piernot on August 2, 1995; serial number 08/510,196, entitled *Methods and Apparatus for Providing a Prepaid, Remote Entry Customer Account for the Hearing Impaired*, filed in the name of John Taskett on August 2, 1995; and serial number (not yet assigned) entitled *Methods and Apparatus for Providing a Prepaid, Remote Entry Customer Account Having Multiple Language Capability*, filed in the name of John Taskett on November 7, 1995. The entire disclosures of these patent applications are incorporated herein by this reference.

Referring now to Figure 1, an exemplary telephone card 100 is analogous in its physical embodiment to a credit card-like instrument, being comprised of paper, plastic, cardboard or any other convenient material. In a preferred embodiment, card 100 comprises plastic and has a length on the order of 8.5-8.7 cm (dimension "A"), a height on the order of 5.3-5.5 cm (dimension "B"), and a thickness on the order of 0.65-0.85 mm.

With continued reference to Figure 1, telephone card 100 comprises an access telephone number 140 for accessing a service provider network including an audio database, and an authorization code 142 for accessing an account database. The service provider network, audio database, and account database are described in greater detail below.

Access telephone number 140 may be a toll-free 800 number, a 900 number, a local phone number or the like. In a preferred embodiment, access phone number 140 is assigned according to the image or photograph pictured on the other side of phone card 100; that is, a unique, corresponding access phone number is established for a particular famous person shown on the card. Alternatively, a single access hone number may be employed for a plurality of personalities, whereby the caller selects from a menu of personalities once connected to the service provider.

Authorization code 142 is a number which is preferably unique to a particular telephone card. In the illustrated embodiment, authorization code 142 is a unique predetermined number that is printed on telephone card 100 before it is issued to the consumer. However, in accordance with another aspect of the invention, code 142 may be a number that is selected by the individual user by having the user inform the service provider of the selected code so that a corresponding account may be set up in the account database of the host computer.

In accordance with the illustrated embodiment, user-friendly instructions may be set forth on one side of card 100 to explain to the consumer how to place phone calls. For example, card 100 may instruct the user to dial access telephone number 140 to access the host computer which maintains the account from which "funds" are "withdrawn" or otherwise consumed as a consequence of the long distance telephone calls made in accordance with instrument 100 (line 1). Thereafter, instrument 100 instructs the user (line 2) to enter authorization code 142. In accordance with a further aspect of the present invention, a bar code or other suitable indicia (*e.g.*, magnetic strip) of authorization code 142 may also be exhibited on instrument 100, for allowing authorization code 142 to be read by a bar code reader or other scanning device. This would alleviate the need for manual entry of authorization code 142 by the user.

The user is then instructed to dial a desired destination telephone number (line 3) either in his local country (*e.g.*, U.S./Canada) and/or internationally. Information may also include instruction for placing additional calls (line 4).

With continued reference to Figure 1, exemplary telephone card 100 provides an instruction 112 on how to "recharge" card 100 with additional telephone time, an instruction 114 on how to use a speed-dialing feature, and an instruction 116 on how to contact customer service.

If desired, additional information may be suitably printed on card 100 such as one or more trademarks or service marks 110 of the issuer of card 100 or trademarks or service marks 130 for advertising various related or unrelated products. In addition, a photograph or image 120 configured to resemble a postage stamp may be appropriately placed on the instruction side of card 100 so that telephone card 100 resembles a picture postcard. In a preferred embodiment, image 120 is reduced in size but is otherwise identical to the image on the reverse side of card 100 as well as postage stamps issued with telephone card 100 (see Figure 6).

It is noted that Figure 1 is illustrative only and that given the robust configuration of the present invention, virtually any additional information or data may be set forth on card 100.

Referring now to Figure 2 and as mentioned above, the opposite side of telephone card 100 comprises suitable visual indicia 202 of the any desired image which may include a photograph, reproduction, drawing or sketch of virtually any subject matter such as a living or deceased person(s) (*e.g.*, actor, artist, political or military figure, or the like), animals, scenic locations, musical instruments, automobiles, or the like. When a user connects to the service provider using phone number 140, portions of audio indicia stored in an audio database or library and relating to visual indicia 202 are selectively communicated to the caller at appropriate times or intervals. For example, in the case when visual indicia 202 is a picture of a horse, sounds associated with a horse may be communicated to the user of card 100 when the service provider is accessed.

In a preferred embodiment, visual indicia 202 comprises a photograph or drawing of a living or deceased person or celebrity such as Elvis Presley, Marilyn Monroe, James Dean, or the like. Upon dialing access phone number 140 and accessing the service provider network, a reproduction (*e.g.*, recording) of the voice of the celebrity stored in the voice library or voice storage facility is played back to the user of card 100. The voice of the celebrity may provide instructions on using telephone card 100 or may prompt the user for additional information such as authorization code 142 and the desired long distance phone number. In the case of deceased celebrities, the voice for the instructions may be obtained by taking snippets or sound bites from taped interviews, movies, records, and other recordings. These voice snippets are then sorted and stored in the voice library until retrieved by the host computer.

With continued reference to Figure 2, an initial stored value number 204 may be printed on transaction card 100 to indicate the value or initial amount associated with telephone card 100. Typically, the initial prepaid amount of card 100 ranges from one dollar to several hundred dollars or more (or foreign currency equivalent). Of course, once card 100 is used, the value of telephone card 100 may fluctuate as long distance telephone charges reduce the remaining balance in the charge, and as the user "recharges" the balance in the charge account. In accordance with another aspect of the invention, the host computer may inform the user of card 100 of the current balance of the account when accessing the service provider network.

Referring now to Figure 3, a distributed transaction system suitably comprises a network service provider (host computer network) 308 having a database 310 associated therewith and a calling party module 306 configured to communicate with host computer network 308 via a communications link 322. In a preferred embodiment where caller module 306 comprises a telephone, the caller is suitably routed to service provider 308 by way of a Local Exchange Carrier (LEC) 320.

In the embodiment shown in Figure 3, service provider network 308 comprises telephone switching equipment suitable for connecting long distance telephone calls. Alternatively, service provider 308 may be connected electronically to a remote long distance carrier (not shown) in order to facilitate the completion of long distance telephone calls.

In accordance with one aspect of the present invention, the functions of service provider 308 may suitably be performed by a financial institution, credit card issuer, telephone company or other entity issuing telephone cards 100. Those skilled in the art will appreciate that service provider 308 includes suitable computing hardware for effecting the functions set forth herein.

Referring now to Figure 4, host system 308 suitably comprises an incoming call trunk 410, a controller 406, and a ROM 409. Trunk 410 suitably comprises one or more incoming telephone lines 412(a)-412(c). In a preferred embodiment, first incoming line 412(a) corresponds to a first predetermined telephone number (*e.g.*, all Elvis Presley telephone cards). Similarly, second incoming line 412(b) corresponds to a second predetermined telephone number (*e.g.*, all Marilyn Monroe phone cards), third incoming line 412(c) corresponds to a third predetermined telephone number (*e.g.*, all antique automobile phone cards) and so on. In an alternate embodiment, each incoming line (*e.g.*, 412(a)) may correspond to a particular "class" of phone cars, *e.g.*, famous athletes, actors, scenic locations, or the like. Upon accessing a particular class, service provider 308 may be configured to accept a selection from the caller (*e.g.*, in response to a prompt from provider 308) from among members of the class. For example, a single predetermined access telephone number associated with incoming line 412(a) may correspond to "actors". Upon accessing the system, the caller may enter (*e.g.*, via the caller's telephone keypad) a first code to select Humphrey Bogart voice prompts, a second code to select John Wayne voice prompts, and so on. Thus, service provider 308 may be configured to accommodate a large number of incoming telephone lines to thereby facilitate the communication of audio indicia from the audio library managed by processor 406.

With continued reference to Figure 4, processor 406 is suitably configured to retrieve appropriate software modules from ROM 409 to appropriately access audio indicia from the audio library, to access prepaid accounts in database 310, or to process phone calls.

With continued reference to Figure 4, in a preferred embodiment of the present invention, database 310 suitably comprises an account database 411 for storing and managing various aspects of the prepaid accounts and an audio database or voice library 420 for storing audio indicia (*e.g.*, prerecorded voice message) or sound bites corresponding to the images associated with various telephone cards. Audio library 420 may be stored in any suitable medium that may be conveniently accessed by processor 406, such as read only memory (ROM), random access memory (RAM), compact disc-read only memory (CD-ROM), tape drives, a digital "voice mailbox" matrix, or the like.

In accordance with a further aspect of the present invention, when a user accesses service provider 308 using a particular access telephone number 140, processor 406 suitably retrieves appropriate audio indicia or sound bites from audio storage facility 420. Processor 406 then suitably directs the sound bites to the user via hardware link 322 to telephone 306.

Referring now to FIGS. 3 and 4, a consumer may use telephone card 100 to place long distance telephone calls in the context of the distributed processing system in the following manner.

In a preferred embodiment, data entry module 306 suitably comprises a conventional touch tone telephone. The consumer in possession of (or otherwise knowing the contents of) instrument 100 executes the instructions set forth thereon through an appropriate keypad 316 associated with telephone 306. In an alternate embodiment, it may be desirable to enter certain data into data entry terminal 306 through an alternative input/output modality via module 318, for example by reading bar code data, magnetic stripe data, voice recognition, or any other suitable medium.

With continued reference to FIGS. 3 and 4, upon dialing access number 140, the telephone call is routed by the Local Exchange Carrier (LEC) 320 to service provider 308 associated with the aforementioned toll free telephone number. In a preferred embodiment of the invention, service provider network 308 determines which image is on card 100 based upon, for example, the access phone number 140 dialed by the user. For example, the Elvis Presley phone card suitably has an access phone number specific to the Elvis card; the Marilyn Monroe phone card suitably has an access phone number specific to the Marilyn card; and a horse phone card has an access phone number corresponding to the horse card. Other methods, of course, may also be employed for determining which picture phone card is being used. For example, in an alternate embodiment, a similar access phone number 140 is issued for all the picture phone cards. Upon access by the user, service provider network 308 then prompts the user for which card is being used (*e.g.*, "Press 1 to speak with Elvis, press 2 to speak with Marilyn Monroe; press 3 if using the antique automobile card...").

After determining which picture phone card 100 is being used, controller 406 then retrieves the appropriate portion of audio indicia (*e.g.*, voice recordings) from audio library 420 and relays it to the user over telephone line 322. Additional information is then requested from the user (*e.g.*, using Elvis' voice if an Elvis phone card was used) such as access code 142 and a destination phone number.

In a preferred embodiment, the user enters data into data entry module 306 through keypad 316; however, it may also be desirable under certain circumstance to employ voice recognition circuitry within service provider 308 to permit the consumer to "speak" information into data entry terminal 306. In any event, once the information pertaining to authorization code 142 is received by service provider 308, a correlation is made between authorization code 142 and a particular account resident in account database 411. Upon determining the current available "balance" in the account, the service provider may inform the consumer (*e.g.*, using Elvis' voice) of the amount of long distance time available in the account or any other relevant parameter.

Upon entering the desired destination telephone number into data entry terminal 306, service provider 308 connects or otherwise permits the connection of telephone 306 with a requested destination telephone extension 312. In the illustrated embodiment, destination 312 suitably corresponds to the telephone extension of the area code and phone number entered into data entry module 306 by the consumer. During the course of successive long distance telephone calls, service provider 308 incrementally decreases the available balance in the consumer's account until the account is fully withdrawn, at which time service provider 308 may inform the calling party (*e.g.*, using Elvis' voice) that the account is fully withdrawn and either invite the calling party or the called party to make other arrangements for payment in order to continue the call, or simply terminate the call.

Referring now to Figure 5, in a preferred embodiment of the present invention, voice library 420 comprises a plurality of records 502, 504, 506, 508, etc., each record comprising a suite of audio indicia or sound bites corresponding to a particular image on transaction card 100. For example, record 502 may comprise all the voice recordings of Elvis Presley for playback to Elvis card users, record 504 may comprise all the voice recordings of Marilyn Monroe for playback to Marilyn Monroe card users, and so on.

In accordance with the illustrated embodiment of Figure 5, each record suitably comprises a plurality of fields, *e.g.*, the Elvis record comprises fields 502A, 502B, 502C, 502D, and so on. Each field in a record comprises a sound bite (*e.g.*, a sound bite may be comprised of a plurality of sound snippets taken from a movie, interview, etc.) that may be suitably retrieved by processor 406 and relayed to the telephone card user over telephone line 322 at appropriate times during the placing of a long distance call. Alternatively, certain snippets may also be at least partially synthesized (*e.g.*, digitally) or generate by an impersonator.

With continued reference to Figure 5, if record 502 comprises the Elvis record, each field of record 502 may contain a sound bite (*i.e.*, recording of

Elvis' voice). For example, Elvis sound bites may include "I'm glad you called" (field 502A); "Punch in your special code now" (field 502B); "You have ____ dollars left on your phone card" (field 502C); "Can I have your phone number now" (field 502D); "Your call cannot be completed" (field 502E); "One minute remaining" (field 502F); "Thank you for calling, goodbye" (field 502N); and so on.

In accordance with one aspect of the present invention, when service provider 308 receives a call from an Elvis telephone card user, instructions resident in ROM 409 direct microprocessor 406 to access field 502A. The sound bite in field 502A of Elvis' voice is then redirected through microprocessor 406 and communicated to the Elvis phone card user. Microprocessor 406, in accordance with software in ROM 409, then retrieves the sound bite from field 502B which instructs the card user to enter authorization code 142. After a code is entered, processor 406 searches the Elvis authorization codes in account database 411. If a number is entered that does not match a list of Elvis authorization codes, controller 406 is redirected to field 502B and the user is again prompted for the authorization code. If a correct authorization code is not entered by the card user within a predetermined number (*e.g.*, three) of attempts, controller 406 is directed to field 502E which informs the caller that the call cannot be completed. Controller 406 is then directed to field 502N where the phone card user is thanked (*e.g.*, by Elvis) for using the phone card system and the call terminated. On the other hand, if a correct authorization code is entered in response to field 502B's request, microprocessor 406 "speaks" to the user the sound bite from field 502C which informs the user of how many units or how much time is remaining in the card. Next, a sound bite from field 502D requesting the desired phone number is relayed by the microprocessor to the user. Once the call connection is made, time paid for by the user will incrementally decrease. When the point is reached where there is only one minute remaining in the user's account, controller 406 is directed to field 502F and plays the message "One minute remaining." If the call is not terminated by the user within a minute, microprocessor 406 reads from field 502N the voice message "Thank you for using American Express, goodbye" and then terminates the call.

It will be appreciated that various other fields or subfields may also be incorporated into record 502 in addition to, or in lieu of, one or more of the above-described fields.

Sound bites for other people, animals, automobiles, or the like may be stored in records in audio library 420 in a similar manner. Host computer 308 may determine which picture telephone card is being used by a variety of methods such as allocating specific phone numbers for each set of picture phone cards or prompting the user at the time of access which picture card is being used.

Turning now to Figure 6, in accordance with another aspect of the invention, a single postage stamp 606 or a series of postage stamps 600 may be distributed to a purchaser of transaction card 100 in a telephone card set or ensemble. These telephone card sets may be distributed through, for example, post offices, stationary stores, souvenir shops, or other stores.

Each postage stamp 606 suitably comprises an image or drawing 602 that is substantially similar or identical to image 202 on transaction card 100 (of course, the size of image 602 and image 202 may be different). A denomination 604 of stamp 606 may be of any variety, but is preferably set at a first class mail rate or a post card rate.

Although the present invention is set forth herein in the context of the appended drawing figures, it should be appreciated that the invention is not limited to the specific forms shown. Various other modifications, variations, and enhancements in the design and arrangement of the host computer, audio library, telephone card, and the like as set forth herein may be made without departing from the spirit and scope of the present invention as set forth in the appended claims.

CLAIMS

1. A telephone card for use with a prepaid telephone service provider having a computer database, comprising:

an access telephone number printed on said card for accessing the telephone service provider;

indicia of a predetermined authorization code on said card for accessing the computer database; and

an image on a front side of the card, said image being related to a sound bite stored in the computer database that is communicated to the user of the prepaid telephone card when said service provider network is accessed.

2. The prepaid telephone card of claim 1 further comprising instructions printed on said card setting forth a method for accessing said service provider.

3. The prepaid telephone card of claim 2 wherein said image on said card is substantially similar to an image on a postage stamp issued with said prepaid telephone card.

4. The prepaid telephone card of claim 2 wherein said image on said card is of a famous person.

5. The prepaid telephone card of claim 4 wherein said sound bite is a reproduction of a voice of said famous person.

6. The prepaid telephone card of claim 5 wherein said voice instructs the user on making a telephone call.

7. A distributed system for effectuating commercial transactions by a consumer, comprising:

a service provider network including an account database and an audio library;

a prepaid instrument for providing access to said account database having a first side and a second side;

an access telephone number printed on said first side of said prepaid instrument for accessing said service provider network;

a visual indicia on said second side of said prepaid instrument; and

an audio indicia related to said visual indicia stored in a record in said audio library, said audio indicia communicated to the consumer when the consumer accesses said service provider network.

8. The distributed system of claim 7 wherein said first side of said instrument further comprises instructions setting forth a method for accessing said service provider network and said account database.

9. The distributed system of claim 8 wherein said first side of said instrument further comprises a predetermined authorization code for accessing a predetermined credit account associated with said account database.

10. The distributed system of claim 9 further comprising:

a second audio indicia stored in a second record in said audio library;

a second prepaid instrument;

a second visual indicia printed on said second prepaid instrument, said second visual indicia relating to said second audio indicia; and a second access telephone number printed on said second prepaid instrument for use in accessing said second audio indicia.

11. The distributed system of claim 7 wherein said visual indicia is of a famous person.

12. The distributed system of claim 11 wherein said audio indicia is a reproduction of a voice of said famous person.

13. The distributed system of claim 12 wherein said voice instructs the consumer on making a commercial transaction.

14. A method for effectuating commercial transactions between a consumer and a host computer comprising:

(a) providing a host computer having an account database and a database comprising audio indicia;

(b) providing a prepaid transaction instrument for use by the consumer having a first side and a second side, said first side comprising an access telephone number for accessing the host computer and a predetermined authorization code for accessing said account database, said second side comprising a visual indicia related to said audio indicia; and

(c) communicating to the consumer said audio indicia when said host computer is accessed by the user.

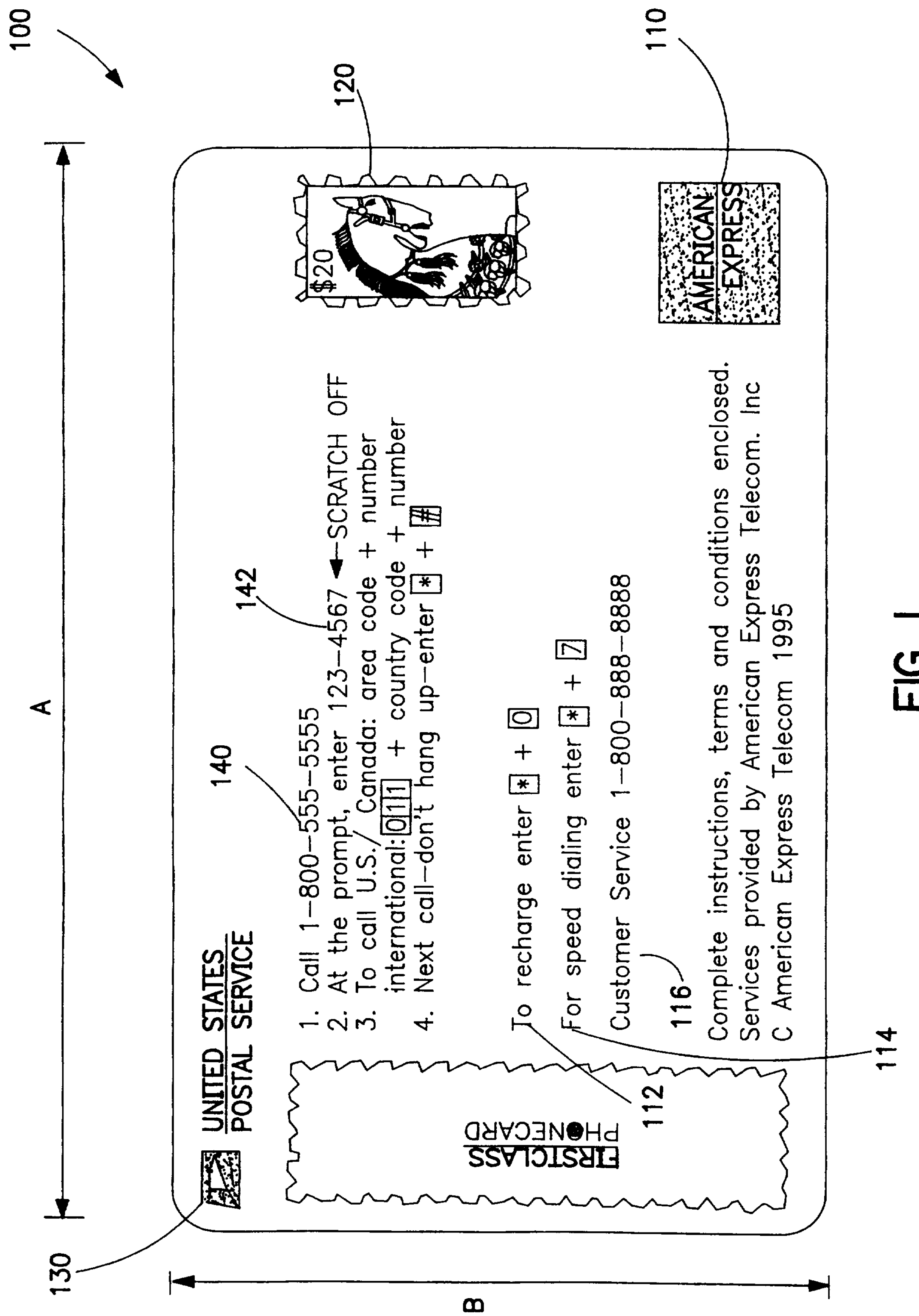

100
110
120
130
140
142
112
114
116
A
B
$20
AMERICAN EXPRESS
UNITED STATES POSTAL SERVICE
FIRSTCLASS PHONECARD
1. Call 1-800-555-5555
2. At the prompt, enter 123-4567 ←SCRATCH OFF
3. To call U.S./ Canada: area code + number
international: 011 + country code + number
4. Next call-don't hang up-enter * + #
To recharge enter * + 0
For speed dialing enter * + 7
Customer Service 1-800-888-8888
Complete instructions, terms and conditions enclosed.
Services provided by American Express Telecom. Inc
C American Express Telecom 1995

FIG. I

SUBSTITUTE SHEET (RULE 26)

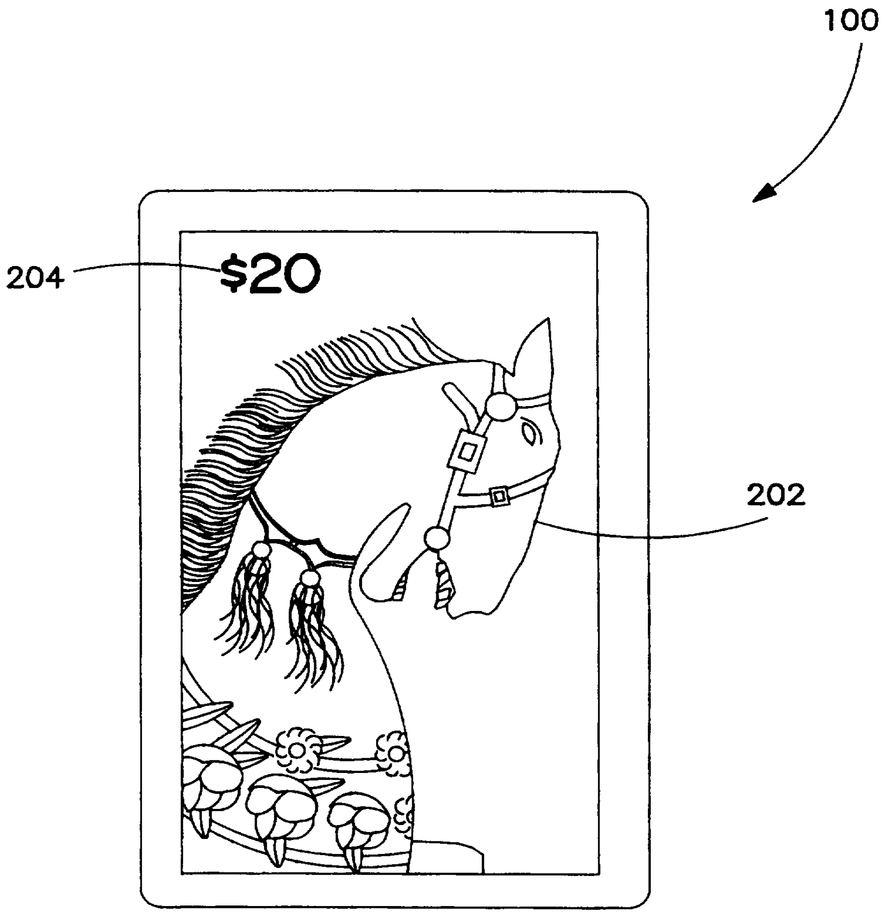


FIG. 2

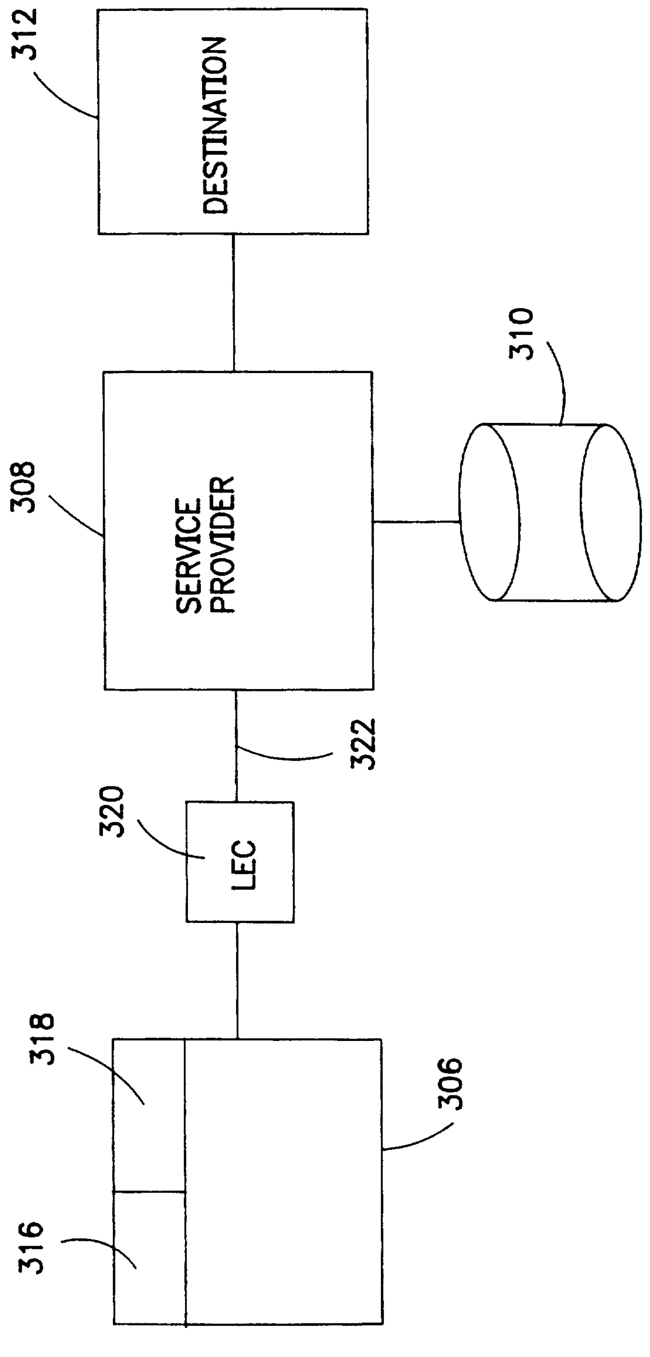

312
DESTINATION
310
308
SERVICE PROVIDER
322
320
LEC
318
306
316

FIG. 3

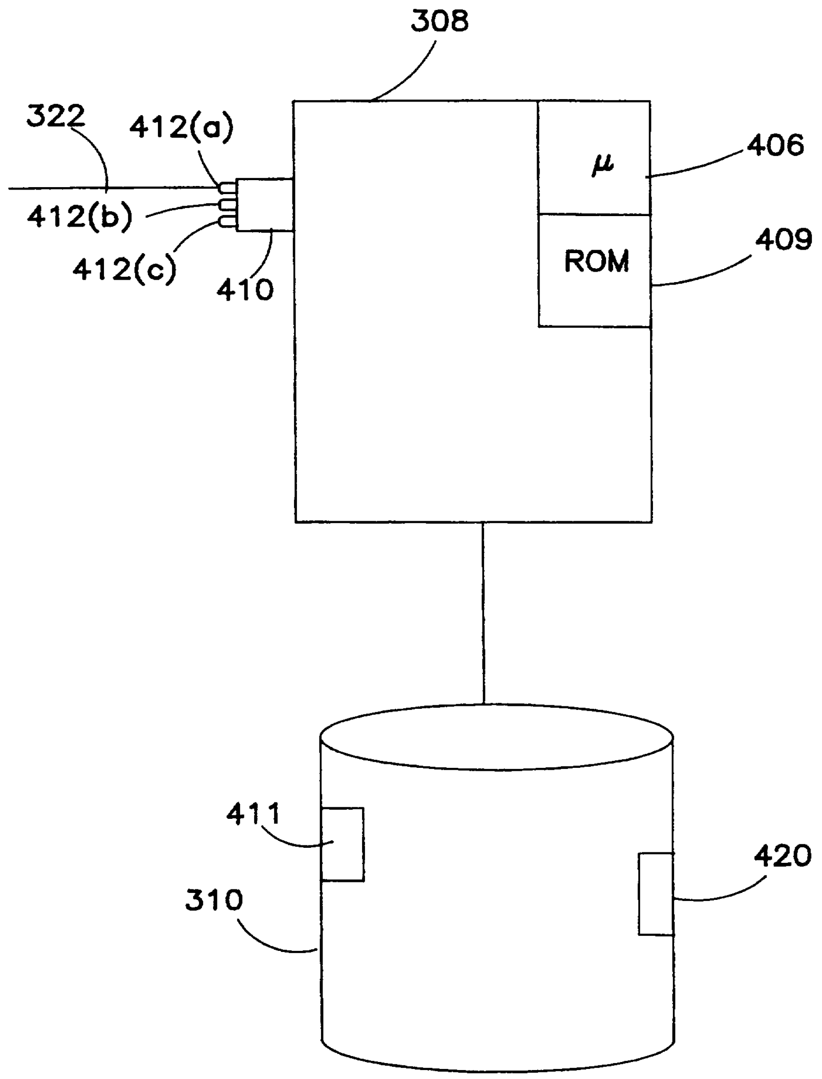


FIG. 4

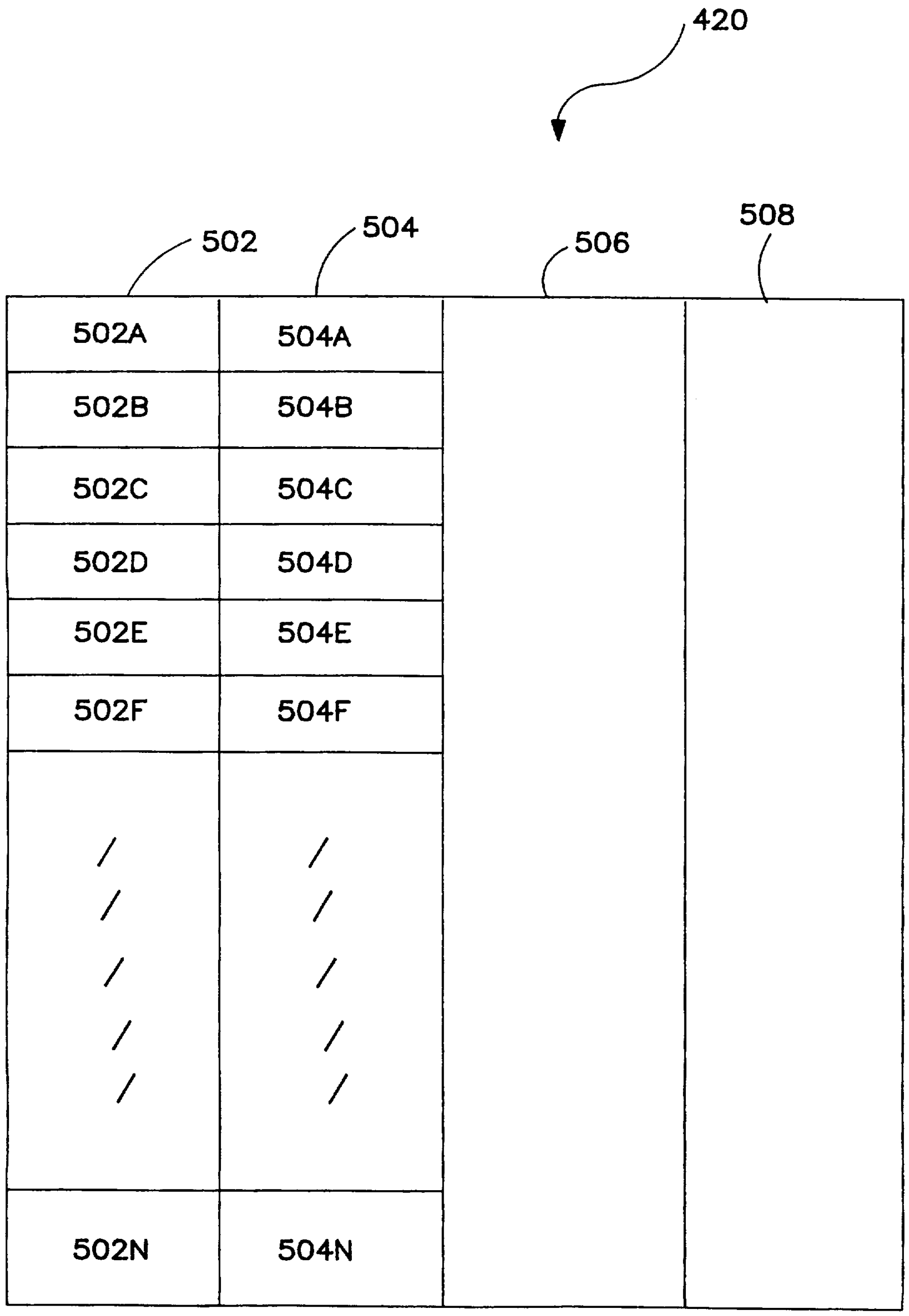


FIG. 5

SUBSTITUTE SHEET (RULE 26)

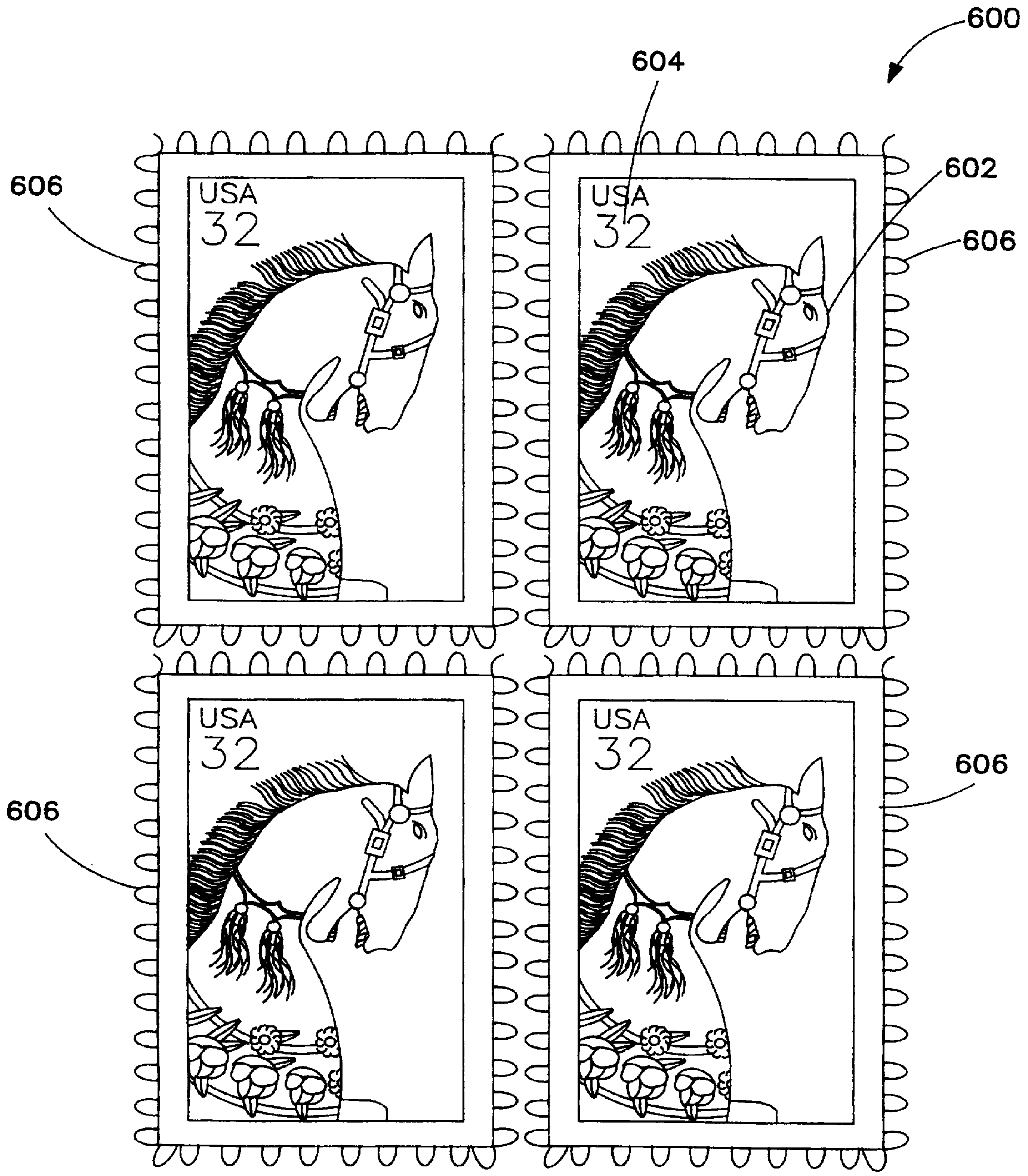


FIG. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/03414

A. CLASSIFICATION OF SUBJECT MATTER
IPC(6) :H04M 1/00, 1/64
US CL : 379/357, 67
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED
Minimum documentation searched (classification system followed by classification symbols)
U.S. : 379/357, 67, 88, 89, 111, 112, 144

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US, 5,287,403 A (ATKINS ET AL) 15 February 1994, column I, line 52 and columns 4-6.	1-6
Y	US, 5,251,251 A (BARBER ET AL) 05 October 1993 fig.3(b) and cols. 4-5	1-14

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:	"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A" document defining the general state of the art which is not considered to be of particular relevance	"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E" earlier document published on or after the international filing date	"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L" document which may thr w doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&" document member of the same patent family
"O" document referring to an oral disclosure, use, exhibition or other means	
"P" document published prior to the international filing date but later than the priority date claimed	

Date of the actual completion of the international search: 16 APRIL 1997	Date of mailing of the international search report: 14 MAY 1997
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer [illegible] FAN TSANG Telephone No. (703)3054895

Form PCT/ISA/210 (second sheet)(July 1992)★

INTERNATIONAL SEARCH REPORT	International application No. PCT/US97/03414

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This international report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

Please See Extra Sheet.

1. [✓] As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest
[✓] The additional search fees were accompanied by the applicant's protest.
☐ No protest accompanied the payment of additional search fees.

Form PCT/ISA/210 (continuation of first sheet(1))(July 1992)★

BOX II. OBSERVATIONS WHERE UNITY OF INVENTION WAS LACKING
This ISA found multiple inventions as follows:

This application contains the following inventions or groups of inventions which are not so linked as to form a single inventive concept under PCT Rule 13.1. In order for all inventions to be examined, the appropriate additional examination fees must be paid.
Group I, claim(s) 1-6, drawn to a telephone card.
Group II, claim(s) 7-14, drawn to a system for effectuating commercial transactions between a consumer and a computer.
The inventions listed as Groups I and II do not relate to a single inventive concept under PCT Rule 13.1 because, under PCT Rule 13.2, they lack the same or corresponding special technical features for the following reasons:
A telephone card, which does not have any electronic elements, is just a printed card for assisting a caller to use a telephone system. However, a system for effectuating commercial transactions between a consumer and a computer involves a computer for performing the transactions. Thus, it is clear that a telephone card is completely different from a system for performing the commercial transactions.

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 495 310 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **91311870.9**

(51) Int. Cl.^{5}: **G06F 9/44**

(22) Date of filing: **20.12.91**

(30) Priority: **17.01.91 US 642745**

(43) Date of publication of application:
22.07.92 Bulletin 92/30

(84) Designated Contracting States:
DE FR GB

(71) Applicant: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: **Campbell, David Lunn**
1501 Fairfield Drive
Austin, Texas 78758(US)
Inventor: **Cordell II, Robert Quentin**
1331 Sugar Creek Boulevard
Sugar Land, Texas 77478(US)
Inventor: **Hambrick, Geoffrey Martin**
2502 Chowan Way
Round Rock, Texas 78681(US)
Inventor: **Vakharia, Dinesh Javantilal**
801 Bluff Drive
Round Rock, Texas 78681(US)

(74) Representative: **Tomlinson, Kerry John**
Frank B. Dehn & Co. European Patent Attorneys Imperial House 15-19 Kingsway
London WC2B 6UZ(GB)

(54) **A process control method and apparatus.**

(57) A Process Control Framework is disclosed as an environment in which processes and their products can be formally defined as objects in a machine readable form, called a Process Control File. The Process Control File is used by a Process Control to determine the next action to be performed on the set of objects selected by the user. Where the action is machine executable, the Process Control is able to actually invoke the system command itself to perform the task. This gives rise to a unified user interface where the user can work exclusively with high level actions to Process Control File objects instead of having to remember the specific system level commands. Further, the objects have attributes associated therewith which are stored separate from the objects themselves. This allows for a single object to be accessed and used by multiple processes, both internal and external to the disclosed Process Control Framework. The object-oriented approach allows for access privileges for users, roles, and access rights for object classes to be defined and maintained by sending messages to retrieve or set the properties of these object classes. An object-oriented user paradigm for viewing and manipulating data and tools is reinforced with a true object-oriented system managing the data and tools.

EP 0 495 310 A2

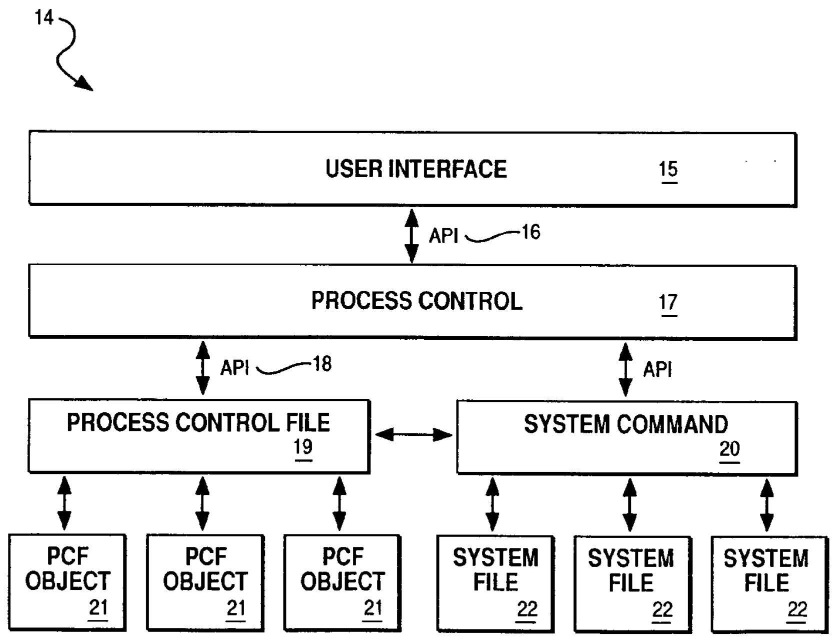

14
USER INTERFACE 15
API 16
PROCESS CONTROL 17
API 18
API
PROCESS CONTROL FILE 19
SYSTEM COMMAND 20
PCF OBJECT 21
PCF OBJECT 21
PCF OBJECT 21
SYSTEM FILE 22
SYSTEM FILE 22
SYSTEM FILE 22

FIG. 2

The present invention relates to a method and apparatus for managing processes and maintaining process integrity, and more specifically to a method and apparatus for providing a computer assisted process development system.

Every creative venture, from writing a great work of literature to coding a world class computer operating system, involves a process. Conversely, every process, formal or informal, is composed of a certain number of steps geared towards creating a product.

The tight coupling between process and product means that the first step towards developing a quality product is to define a "quality" process. In fact, having a quality process is essential to having a high quality product, since a quality process not only produces high quality products, but is also both repeatable and measurable.

Repeatable means that the process is understood well enough that the steps can be carried out again and again (and still produce the same quality products). Measurable implies that the qualities of the products can be stated in objective rather than subjective terms such that clear progress with respect to the goal can be determined.

Thus, a quality process is a well defined process. One way to formally define a process is to describe it in terms of a state transition diagram, where states represent process stages and transitions represent flow of control from one state to another. A state can be precisely described as a condition based on the current values of attributes associated with objects known to the process. Transitions are actions that use and modify the attributes in the course of their execution, thus changing the state.

There is an interesting side effect of having a well defined process: defects in the product can be traced back to defects in the process. This is true because either the process is not properly defined in the first place, or the process is not controlled. Therefore, we need some automated way to ensure that the process is properly defined and being followed.

An example of process definition and control is the procedure of building a computer program (software) capable of running or executing on a computer (hardware). The end result of this process, the computer program, is generally derived from numerous source code modules, each module being capable of performing a specific, or logically related group of, task(s). These individual source code modules, which are generally written by human beings in a high level computer programming language, such as C or Pascal, must be further refined into a series of computer commands and data which can be understood by the computer hardware. After this refinement step, also known as compilation and/or assembly, the individual modules must be combined into a machine executable program through a step known as linking or binding. This linking (binding), besides creating a machine executable program, further attempts to resolve any external addressing references which were not known to the compiler or assembler during the prior compilation or assembly step.

The management of this process becomes increasingly difficult as the number of individual programmers, whose source code modules are being used in this build process, increases due to increased functionality/complexity being placed in the resultant computer program. If a subsequent change is made to a source code module (e.g. to fix an error in the program), that module, and possibly other related modules, will have to be recompiled and relinked. However, some modules do not require the recompilation step and can be used, unmodified, in the relink step. Defining and controlling these module interrelationships can be tedious and time consuming. Computer tools exist in the marketplace today which assist in this process definition and control. "Make" is a utility program which controls the build process of computer programs. "Makefiles" are files used by Make containing specific build instructions for the computer program which is being generated. However, the tools available today do not provide for a general purpose process control framework, which is required in light of the emphasis today on object-oriented programming.

Object-oriented programming, and its related support tools, further exemplifies and amplifies the need for a system of controlling a process. Before examining this need for process control, we must first understand what object-oriented programming is all about. Conventional software programming techniques generally have a series of procedure, or subroutine, calls to perform a given task. Communication between these procedure calls is achieved by passing parameters or data structures which contain the information which the computer program is to manipulate. The problem with this technique is that there can be too much data transmission between the procedure routines. If this occurs, one solution is to instead put these routines into the data itself. Instead of building modules around computer tasks and distributing data structures between the resultant routines, object-oriented design does the reverse. It uses the most important data structures as the basis for modularization and attaches each routine to the data structures to which it most closely relates.

Object-oriented technology is the construction of software systems as structured collections of abstract data type implementations. In terms of an object-oriented approach, an integrated project support environment can be defined as follows:

"Classes" represent an abstract data type. For an integrated project support environment classes may be the definition for source files, compilers, documentation, data flow diagrams, and the like.

"Objects" are the instance of a class. Objects represent real entities within the environment. Objects in an integrated project support environment may be users, groups, data consisting of requirements, specifications, designs, code renderings, images, reports, publications, etc., tools, applications, and methods. In fact, everything can be defined as an object within the environment, including the environment itself.

"Methods" define the operations that can be performed on objects. Methods sometimes are called "actors" and are normally associated with classes. For an integrated project support environment, methods are essentially the tools that operate on the data objects. Methods specify the rules by which they operate. For example, the rule for a C compiler is that it will only accept C source file objects. All other objects would be rejected. The definition of a method in an integrated project support environment is called the tool's schema definition set.

"Messages" are the mechanism by which methods are invoked in an object-oriented environment. Messages are passed to the method and the method determines if the message is appropriate, e.g., can be executed, by that particular method.

Classes are organized into a "hierarchy". This means that classes are defined as belonging to other classes ("superclasses"). This is an important concept in an integrated project support environment because it allows the definition of components of a system to be decomposed into their constituent parts. Given a type hierarchy of object classes, "inheritance" provides the ability for one class to inherit properties of its superior class.

"Composite objects" are objects that are composed of more than one object. In an integrated project support environment, composite objects can be thought of as the components of the system that are made up of other objects.

The "object manager" is the mechanism by which all objects are controlled and maintained in the system. This is the basis on which an object-oriented integrated project support environment is constructed.

The emphasis is on structuring a system around the classes of objects it manipulates rather than the functions it performs on them, and on re-using whole data structures, together with the associated operations, rather than isolated routines.

The environment can be viewed as a set of objects that can perform operations or upon which operations can be performed. Classes and type hierarchies for objects can be defined to simplify the instantiation of a set of objects. Behavioral characteristics of objects can be defined as methods and operations can be mapped to appropriate methods for a particular class of objects. Thus, the object-oriented approach provides a high-level of commonality and consistent sharing of services through inheritance. This programming methodology is also discussed in Meyer, B., "Reusability: The Case for Object-Oriented Design", IEEE Software, Vol. 4, Number 2, pp 50-64, March, 1987 and Stroustrup, B., "What is Object-Oriented Programming?", IEEE Software, pp 10-20, May, 1988.

Using these object-oriented systems with the tools and methods previously discussed is not feasible due to inherent architectural differences in the methods of maintaining the system. The following discussion details the deficiencies which exist in today's computer-assisted software engineering (CASE) environment.

In a system using Make to manage its processes, once access is gained to the Makefile, then access to every object listed therein is provided. No concurrency controls are provided, i.e., Make does not prevent simultaneous access and/or update to Makefiles.

Make does not dictate the tools that are used in the environment, as long as they are executable AIX (a type of computer operating system: Trademark of IBM Corporation) commands. However, Make cannot accommodate any arbitrary process (one representable by a state transition diagram as described previously). This is because the only attribute considered for the "state" is the last update to the file representing the object.

Makefiles can be considered to be object oriented. However, only the source list and build process associated with an object are stored. Controlling other processes and storing other attributes either requires multiple Makefiles or multiple "list" variables in a single file. In either case, information associated with a given object is spread out (across files or lists) rather than consolidated in one place.

Makefiles cannot be updated under the control of Make. Therefore, Make does not address data integrity or error recovery issues in any way. Every process that needs to be controlled must be "recast" in terms of dependencies of actual operating system files. Therefore, Make is not easy to integrate.

Makefile syntax is positional and contains many special purpose macro symbols. Not all symbols are valid in all three "sections" of a Makefile (the suffix rules, macro and dependency definitions). For example, "$@" represents the current target name; "$$@" has the same meaning except it is only valid on dependency lines. The symbol "$?" stands for the "list of source files that have been modified more recently than the target". The symbols "$*" and "$<" only have meaning in suffix rules. The net result is that Makefiles are often hard to read and understand and therefore, hard to use.

If one process has already been integrated using Make, then controlling subsequent processes requires shifting to multiple Makefiles or list variables. Either way requires that another utility (such as libmake, an AIX operating system utility) control the processing.

Make only works with ASCII makefiles. There is no way to "compile" Makefiles or otherwise store them in a database for more efficient execution. The user interface is always command line driven. Make was not written using coding principles that allow changes to the data storage facilities and/or user interface look-and-feel.

The build process associated with a target object can be explicitly coded such that it sends mail, (or any other action). However, Make does not support a facility for automatic invocation of events (such as sending mail, etc.) when a Makefile object is updated.

Make does support limited reusability. For example, "include" files can be reused across several makefiles. "Macro definitions" allow for complex strings to be encoded once and substituted wherever needed. "Rules" provide for "attribute inheritance". However, the only attributes that can be "inherited" are the dependencies and build process. Further, inheritance is based on the filename suffix only. Therefore, Make does not fully support reusability of object attributes.

Since Make does not support controlled access to the associated Makefiles, it cannot be used by other tools to access Makefile information. Any tool can read a Makefile, but it must provide its own parser. Therefore, visibility to external tools is not fully met.

In summary, the Make utility cannot be considered to provide even full process control, due to the difficulty involved in integrating processes other than the one for which it was originally intended (i.e., executing commands to bring out-of-date targets up-to-date).

The Atherton Software BackPlane (hereafter called Atherton) provides an object oriented "integration" framework where attributes as well as methods can be associated with objects to provide process control. A summary of its major shortcomings are the failure to adequately provide for the following:

(i) Accommodation of processes and tools, (ii) Independent of platforms, (iii) Independent of technology, (iv) Notification and triggers, (v) Reusability of object attributes, and (vi) Scalable across a wide range of project sizes.

Once an object comes under control of Atherton, no outside access is permitted. The reason outside access is disabled is that Atherton provides its own built-in version control and problem tracking sub-systems. By providing this functionality, Atherton dictates some of the tools that must be used and some of the processes that are controlled; therefore it does not fully accommodate all processes and tools.

Since Atherton provides its own proprietary object-oriented database and its own user interface "look-and-feel", it limits a user's ability to take advantage of advanced technologies on the horizon. For example, windowing systems such as X-Windows (Trademark Of Massachusettes Institute of Technology) may give way to a more advanced user interface; "semantic networks" may become the database technology of the future. It is thus not independent of platforms (computer hardware) or technology.

Atherton does support a notification facility where mail can be sent to the owners of two objects that are "linked". However, since Atherton does not support the invocation of any arbitrary method associated with the linked objects, it does not fully meet the requirement of notification and triggers.

Atherton supports logical attribute inheritance to provide reusability. For example, a "c program" object can be defined that provides default attributes for "instances" of c program objects (like foo.c). However, Atherton only supports a single hierarchy. It does not support multiple inheritance, where an instance can inherit attributes from many hierarchies. In a true object-oriented system, multiple inheritance is needed to define classes in terms of more than one parent class, and hence the term multiple inheritance. Multiple inheritance would require rules for both parents having the same property. A single inheritance model such as Atherton does not require this level of sophistication and complexity, since there is only a single parent in the inheritance tree. Therefore, Atherton does not fully support reusability of object attributes.

In summary, Atherton provides far more functionality than Make and facilitates some degree of general purpose process control. However, the shortcomings summarized in this section show that Atherton does not provide an adequate solution to process control.

Other similar products exist on the market. For example, Cadre has a configuration management toolkit called Teamwork/MCM. Sage has its own Make and version control system called Polymake and PVCS, respectively. Each of these offerings has similar serious shortcomings.

A Process Control Framework is required to fully achieve the benefits of process control and overcome the deficiencies in existence today.

According to the invention, a Process Control Framework is an environment in which processes and their products can be formally defined as entities, or objects, in a machine readable form (hereafter called a Process Control File). The Process Control File (PCF) is an internal structure used by a Process Control to determine the next action to be performed on the set of objects selected by the user. Where the action is machine executable, the Process Control is able to actually invoke the system command itself to perform the task.

This gives rise to a unified user interface where the user can work exclusively with high level actions, which in turn interface to Process Control File objects, instead of having to remember the specific system level commands. More importantly, the Process Control is simple minded. It can only follow a well defined process that is both repeatable and measurable. Therefore, it can support a quality process. The aforementioned user interface, process control, and internal structure comprise three levels of integration in the Process Control Framework.

Gaining control of a process is itself a process called integration. The first step is to prioritize the processes that will be integrated. Then, for each process, the following steps are taken:

1) determine the objects and associated attributes to be maintained in the Process Control File,

2) migrate the data from its current form into a Process Control File, and

3) invoke tools, by the Process Control, to act on the Process Control File objects and attributes.

The present invention provides solutions for the following requirements:

1. Access and concurrency control: Concurrent access to Process Control Files and associated objects provides access only to authorized users.
2. Accommodation of processes and tools: The Process Controller does not dictate the processes controlled nor the tools that can be used.
3. Consolidation of object attributes: Unique attributes associated with an object are consolidated in one place within the Process Control File.
4. Data integrity and error recovery: Process Control File data is protected from loss due to unforeseen circumstances as much as is possible.
5. Distribution of objects: Objects are maintainable in separate Process Control Files.
6. Ease of integration: The Process Control Files support a wide variety of attributes that enable new tools to be quickly developed and new processes to be controlled. The Process Control Framework minimizes the amount of learning necessary to integrate sophisticated tools and processes.
7. Ease of use: End users are able to easily encode and read the attributes of objects in the Process Control Files. The Process Control Framework minimizes the amount of learning necessary to use sophisticated tools and processes.
8. Efficiency of execution and storage: The Process Controller is efficient; the Process Control Files are concise.
9. Independent of integration: Controlling new processes, supporting new tools and adding new attributes to objects stored in Process Control Files have no effect on existing tools and processes.
10. Independent of platforms: Processes controlled are not limited based on platform. Manual tasks are also "controllable" through instructions and support for status modification.
11. Independent of technology: The Process Controller is independent of the underlying database format or user interface "look and feel".
12. Notification and triggers: The Process Controller provides for actions (including interprocess communications) to be invoked when objects change states.
13. Reusability of object attributes: The Process Control File architecture supports sharing attribute values across objects.
14. Scalable across a wide range of project sizes: The Process Control framework imposes minimal overhead on small projects, while maintaining performance on large ones.
15. Visibility of external data: Tools in the Process Control Framework have access to data stored outside of the Process Control Files (subject to access controls).
16. Visibility to external tools: Other tools external to the Process Control Framework have access to objects stored in the Process Control Files (subject to access controls).

A Software IC concept relevant to the background of the present invention is discussed by Brad Cox in his book Object Oriented Programming: An Evolutionary Approach, Addison-Wesley, Reading, MA., 1986. Using this model, the Process Control requires interfaces (Application Programming Interfaces, or API's; see Figure 2) to the Process Control Files, System Commands and User Interface. However, there is not enough detail given by Cox to determine how this architecture could provide all the requirements of a Process Control Framework, as previously outlined.

Accordingly, we provide an architecture to achieve these requirements. An integrated framework architecture has been developed with three distinct components that provide the three previously described three levels of integration. These three components are attributes, methods, and links. The attributes of an object describe product and control information used to drive the development process, where methods are the actual programs that can be invoked against the internal objects. Links associate an internal object accessible only with the object manager to an external object accessible to other processes within the system. Methods interact with both the attributes of internal objects and/or links to external objects, and serve as the "glue" that ties the object manager to the resident system.

The Process Control File component makes direct use of System Commands. A Process Control File, as previously stated, is a collection of objects. An object is a named collection of meaningful information called attributes, which associate a name with a value. An attribute value can be derived and interpreted in many ways, from a simple string to an embedded object definition. The derivation and interpretation of an attribute actually requires the services of the underlying system, as will be more fully described hereafter.

Attributes can be explicitly encoded within the object itself or inherited from another object. There are also many ways in which an object can specify the other objects from which it will inherit attributes. Inheritance is but one of many reasons to reference another object. The reference interpretation of an attribute is derived from three components, thus providing a high degree of flexibility in how a reference may be accessed within the three levels of integration.

It is therefore an object of the present invention to provide a process control system.

It is a further object of the present invention to provide process control in a computer assisted software engineering environment.

It is yet another object of the present invention to provide a process control system that is flexible.

It is yet another object of the present invention to provide a process control system that is reusable.

It is yet another object of the present invention to provide a process control system that is interoperable with external processes.

The foregoing and other objects, aspects and advantages of the invention will be better understood from the following description of a preferred embodiment of the invention with reference to the figures listed below, in which:

Figure 1 is a high level illustration of the process control system.

Figure 2 is an illustration of the three levels of integration contained within the process control system.

Figure 3 is an illustration of the three components contained within the process control system.

Figure 4 shows object selection through boolean expression selection.

Figure 5a-5f details the overall system syntax.

Figure 6 shows various types of variable assignments.

Figure 7 shows attributes having multiple values.

Figure 8 shows environment variables and object attributes.

Figure 9 details the interrelationship between shell scripts and executable methods.

Figure 10 exemplifies embedded object attributes.

Figure 11a-11b is an example of how policies govern and control processes.

Figure 12 shows the interrelationship between internal and external objects.

Figure 13 is a system level diagram showing a CASE environment model.

Fig. 1 shows the system overview of the Process Control Framework 14. This Framework 14 has three levels of integration, the user interface level 15, the process control level 17, and the internal structure level 23. The Framework 14 further has three types of components defined therein, comprising attributes 30, methods 31, and links 32. The specifics of the integration levels and types of components are further shown in Figures 2 and 3, which will now be described.

Referring to the Process Control Framework 14 of Fig. 2, a user interface level 15 communicates with a process control level 17 through an application programming interface, or API, 16. The Process Control 17 operates on Process Control Files (PCF) 19 using a separate API interface 18. Each PCF 19 is just that: a file (or files) that is (are) uniquely addressable. These Process Control Files are used, as well as the system commands 20, in building the internal structure 23 of Fig. 1. The system can store the detailed structure of the data associated with an object 21 within the Process Control File 19 of Fig. 2. Although the object and associated data structure is defined later, it is key to understand that the PCF 19 can contain objects, attributes associated with these objects (as exemplified in Fig. 3 at 30), and detailed data structures, thus allowing and supporting the three levels of integration.

File permissions are used to provide PCF level access control. System level file locking is used to provide PCF level concurrency control. The access and locking provided depends on whether the PCF is opened for read or write. If access is denied at the PCF level, then it is denied for all object information stored within the PCF.

The PCF component provides data integrity and error recovery through standard database techniques, for example, two phase commits with transaction logging. This implies that there is a function to recover a PCF from an older "uncorrupted" version and one or more transaction logs.

The primary unit of access in the Process Control Framework is the object 21. Users communicate with objects through messages. Normally in object oriented systems, a message consists of an object and an associated action. The disclosed architecture is unique in that a message can be passed to a single object or an entire set of objects. Likewise, a message can specify a single action or a set of actions that is/are defined for object(s) accessed. We will now describe the features of objects and how they will meet various requirements.

Referring to Fig. 3, an object 21 is a named collection of attributes 30 that is stored in a Process Control File (see also Fig. 2 at 19 and 21). An object name must be unique within the PCF (although there is no requirement that the name be meaningful). Thus, the unique "address" of an object is formed by combining the PCF name and the object name. Knowing the object address allows direct access, which simply locates the object and performs the specified action(s). Note that an action is a list of commands, and a method is a named action.

However, objects can also be accessed indirectly, assuming the user specifies a conditional expression that describes the state of the object's attribute values. Indirect access means that the PCF is scanned by conventional software programming techniques to find objects that match a specified selection criteria (a Boolean expression based on attribute values). However, this does not imply that objects are scanned in any particular order. What indirect access provides is ease of use, since an entire collection of objects can be selected using an expression. As illustrated in Figure 4, objects 21 having attributes 30 indicating the status, type, etc. for each object would be scanned when a message 80 is encountered by the system. This message 80 will 'build' (the specified action 82) all 'c_programs' or 'shells' that are 'dropped' (the attributes 30 for each respective object 21), as defined in the selection expression 84 contained within the message 80. The associated action(s) 82 can be executed in the background to allow the user to perform other tasks.

The aforementioned build process serves as an excellent example of how integration works: First, the information needed to control the build is determined. This includes the list of source files upon which the target objects are dependent and the actual build process that is to be executed when a source file is determined to be out of date with respect to the target. This information would become attributes of the target objects. Then, since there is a large number of existing Makefiles, a tool converts the Makefile data into the PCF format. Finally, a tool scans the source list of a target object, "builds" each source object that is not already built, and checks the target against each source object to see if the target's build process needs to be invoked. At this point, the Process Control Framework is being used to control the build process.

File level access and locking may be too large a granularity, since multiple objects will probably reside within a single PCF. The PCF architecture allows for two types of object level access that are based on the reserved attributes, access permissions and access lists.

The access permissions attribute is functionally equivalent to AIX file permissions (i.e. user, group and others), except it is at the object level. When access permissions do not provide enough explicit control, specific userids and/or groups are associated with read/write/execute permissions. These are explicitly listed in the access list attribute to facilitate more fine-grained access control. There are performance reasons for providing both access permissions and access lists. Access permissions are efficient but provide only three predefined "classes" of users. Access lists are flexible, but can be inefficient if long lists are provided (and the current user's userid happens to be listed at the end).

Concurrency control at the object level is provided by special lock owner attributes that are maintained within the object. These attributes describe the process that currently has the object locked and for what purpose it is locked (read, exclusive read or read/write). Locks can be explicitly released by appropriate setting of the lock owner attributes; they can also be implicitly released, such as when the owning process no longer exists.

Triggers are events (messages) that get invoked whenever a given object is modified. For example, the owner of an object may want to be notified by mail whenever it is checked out (or in, etc.) of a master program/data repository. Special notification list attributes are used to record and handle the notification process. A given entry in the list consists of a message and a trigger expression. A trigger expression is a condition based on attributes associated with the object. When an object is modified and a trigger expression in the notification list is met, then the associated message is executed. Triggers can be explicitly removed through modification of the notification list. They can also be implicitly removed, such as when the object and/or action associated with the message is not found.

To provide the same explicit control for an entire PCF where the filesystem does not provide access lists, the PCF itself is considered to be an object and can be encoded within the PCF file. Its name is equal to the PCF name itself. If present, its access permissions and access list attributes can override those from the filesystem (but only if access to the PCF is allowed in the first place). Therefore, logical access to the PCF is contingent on the object representing the PCF, even though physical access is allowed by the operating system.

The file level locks can be provided for systems that do not support them directly by setting and clearing the lock owner attributes in the PCF object. However, this does not of itself prevent access by external tools. In any case, PCF and object level access, locking and notification depend on accessing attributes associated with an object. The next section will explain this concept in more detail.

An attribute 30 of Fig. 3 is always associated with an object 21. It can be equated with a field in a record. All access to an attribute has to come through the object itself. If a user does not have access to an object, then he cannot access any of the attributes contained within the object. Attributes can be accessed either implicitly through a "next attribute" function, which is a sequential access to attributes without needing to know the attribute name, or explicitly by name (recall that an attribute consists of a name/value pair). Attribute level access control is not directly supported by the Process Control File architecture. However, it can be "simulated;" i.e., an attribute can be treated as if it were an embedded object (i.e. a list of attribute value pairs that is wholly contained in the value itself -- this subject will be later described in more detail). In any case, it is the value of an attribute that conveys the real information. There are two ways to look at a given attribute value. First is how the value is derived; second is how it is used. There are many ways to both derive and use attribute values. Both concepts fall under the System Command component of the architecture shown at 20 in Fig. 2, since the system is required to execute certain tasks in order to help derive and use the attribute values.

The Command architecture is geared toward treating attributes like variable assignments in most shell languages (hereafter called the shell), such as the Bourne, C, or Korne shell. Within the shell, there are three ways that the "righthandside" of the assignment (=) can be evaluated in order to determine the effective value of the variable:

1) as the literal string represented by the value (i.e. what you see is what you get), as shown in Fig. 6 at 90 and 96;
2) as the string that results from substitution of any embedded variable expressions in the value, as shown in Fig. 6 at 92;
3) as the "standard output" string that results from executing the value, as shown in Fig. 6 at 94.

The Command architecture disclosed herein has added a fourth derivation as an extension: where variables in the shell are always single valued, attributes can be multivalued. In other words, each attribute value can be a list of values. The list can be treated as a whole, or each element can be separately derived (and/or interpreted as explained below). Referring now to Fig 7, prior methods of attempting to assign multiple values to an attribute would result in only the first value in the list actually being assigned, as shown at 98 where the resulting value of A is 1. By providing list support, the resulting assignment to A shown at 100 results in the entire list being assigned.

Supporting lists means that the shell assignment and substitution mechanisms are extended, not only to allow attributes to be treated as variables, but also to allow a variable to be multivalued (and either be treated as a whole or as individual elements). An additional extension is added to allow attributes in another object to be directly referenced for substitution (a detailed explanation of referencing is provided below).

Another effect of adding lists is that the normal shell quoting mechanisms used to encode lists ("" or '') will almost always be unnecessary. This is accomplished by not using whitespace to terminate a value, as does the prior art, but rather using either a 'newline' character or a subsequent attribute assignment to terminate a value. Let us now turn our attention to how a value can be used after it is derived.

The Command architecture supports seven ways that a value resulting from the assignment discussed above can be used:

1) as the literal string;
2) as the string after substitution;
3) as the standard output of the result of executing the value;
4) as a list of values;
5) as a list of executable commands;
6) as an embedded object definition;
7) as a reference to another object.

Note that the first four are exactly the same as the value assignment described above. The difference is that the seven interpretations found here are based on the usage of the attribute and require that a tool properly interpret the value.

One kind of "tool" is a special kind of executable attribute known as a "method" (those from "interpretations 3 and 5 above). Methods can be composed of four distinct kinds of commands to allow for varying degrees of flexibility and efficiency:

1. System commands are command strings that are directly executed by the operating system.
2. Internal commands are functions that are provided as part of the Process Control and as such can be more efficiently executed within the current environment. Examples would be commands such as destroy, create, or commit (see also Fig. 5c at 88 for the specific command syntax).
3. Interpreted commands are references to other attributes that are to be treated like shell functions; i.e. they are interpreted as executable commands (and may contain any of these four types of commands).
4. Compiled commands are references to other attributes that have been previously "compiled" and "linked" (presumably from an interpreted command) for more efficient execution.

Note that the Process Control File itself cannot determine if a command is valid when it is stored. That is the responsibility of the System Command component 20 at Fig. 2 when the value is derived or used.

Embedded objects are important in that they allow a user to encode complex information inside of an attribute value without resorting to positional syntactic devices (like "colon separated" values). One of the best features of objects is that they maintain the attribute name along with the value so that the information is more meaningful.

The Command architecture provides embedded objects for another reason: the "object" that is embedded within the attribute value is only accessible through the parent object. Further, it can override the access permissions and access list attributes of the object itself, providing some measure of attribute level access control (but not concurrency control). Since overriding attribute values is usually associated with the concept of inheritance, we leave it to a later section to explain this concept more fully (and return to the PCF architecture).

At this point, we know how an attribute value is derived and how it might be used, but not how it gets associated with an object in the first place. This subsection will describe two ways to associate attributes with objects that are supported within the framework. Both of these are described in terms of "inheritance" of attributes from some other object.

The most obvious way to associate attributes with an object is to encode them right along with the object definition. Since we are describing these methods in terms of inheritance then it makes sense to call this type: "none", since (so far) no attributes were inherited from any other object. If no inheritance mechanism was available, then it would get rather tedious to encode objects. Every single attribute would need to be specified every time, even if other objects had exactly the same list. For example, it seems reasonable that all C program objects share attributes such as compile process. It is also logical that many objects will share similar source code control attributes such as lpp, node, platform, command, etc.

The PCF architecture allows for logical inheritance. Logical inheritance is achieved by declaring a reserved attribute called "type" that is a list of object references (interpretation 7 as listed above) from which the target object will inherit attributes. Whenever an object is retrieved using logical inheritance, each object that is referenced in the type attribute is also retrieved. Attributes from the type object that are not explicitly encoded or already inherited (i.e. overridden) become part of the current object definition. Since a "type" object can also specify a type attribute, and since more than one object can be referenced, the PCF architecture supports multiple hierarchical inheritance. The benefits of logical inheritance are important, because as many groupings as desired can be created to minimize redundancy within a PCF. Only the attributes that are unique to a given object need be explicitly encoded. However, what about between Process Control Files? Luckily, an object reference can address objects in other Process Control Files. The next subsection will describe references in more detail.

Although the concept of reference makes the most sense in the context of inheritance through the type attribute, there are other good examples (for example, the build process interprets the source list attribute as a list of references to other objects). In general, a reference implies accessing the object specified for some purpose (like getting a specific attribute value for variable substitution). However, there is a high degree of flexibility in how a reference may be accessed. For example, there may be a need to disable the inheritance of attributes for the object referenced. Further, a specific PCF (or list of them) may need to be searched in order to locate the object. All of these possibilities add power, but they also add complexity and create additional keystrokes for the user.

To help ease the burden to the user, there is a well defined set of default conditions for references. These defaults are based on the referencing object and the reference itself. In general, a reference has three components:

1) the object name;

2) the inheritance status (which controls whether inheritance is enabled or disabled); and,

3) the list of PCF names to be searched (in the specified order) in which the object should be found.

The simplest reference is simply the object name itself. In this case, the inheritance status is taken from the referencing object itself. The list of Process Control Files that are searched defaults to the list of names specified in a special environment variable (that can be overridden within the current object). Note that when the PCF list is specified, only the named Process Control Files are searched and this overrides the environment variable for "recursive" references.

Support for references means that objects can be distributed across different Process Control Files that may reside on separate machines. Further, this implies support for scalability, since large projects can be broken up into manageable "pieces." Each "piece" is represented by a separate PCF. Small projects do not suffer performance degradation that can be caused when coexisting in the framework with a large project. Still, each reference can refer to common objects (stored in yet another separate PCF).

The details of the Process Control File syntax in the preferred embodiment is shown in Figures 5a-5f, using a Backus Naur form(BNF) definition, the BNF grammar being commonly known in the computer arts.

The objects defined can be described as stanzas in PCF format. Stanzas are nothing more than a named collection of attribute-value pairs. The values depend on the particular object that is represented by the stanza. For example, if a user was responsible for a file called foo.c, she or he might encode it in a PCF file as follows:

```
foo.c
        type=c_program
        lpp=bos
        node=3.1
        path=R2/cmd/foo
        owner=myuserid
        component=cmdfoo
        delta=""
        problems=""


           reason=""
           status=""
```

Remember that the filename attribute serves as the stanza name. Also note that the delta, problems, reason and status attributes are not persistent (i.e., they change every time the files are dropped). These values only need be filled in when the file is actively under development (in which case the status would be non null).

There are several reasons for settling on this PCF syntax, five of these will be discussed below. In summary, PCF files are:

1) readable;
2) concise;
3) maintainable;
4) extendable;
5) compilable.

Attribute files consist of named stanzas that group related attribute-value pairs. The attribute name serves to document the meaning of the associated value, and makes them far more understandable than the "equivalent" colon separated form.

To illustrate, the following example shows the above stanza in a colon separated form:
foo.c:c_program:bos:3.1:R2/cms/foo:myuserid:cmdfoo::::
Notice that it is almost impossible to understand the meaning of this entry without a good memory of the stanza format.

The keyword, rather than positional nature of PCF files also means that null valued attributes need not be encoded at all, reducing the number of keystrokes required to enter a stanza. Colon separated files still require a colon (to mark the position, as seen above for the delta, problems, reason and status attributes); fixed format databases will still use the space, where variable length records will at least carry a pointer to the (nonexistent) string associated with the field.

PCF files also make use of a default stanza concept to help minimize the number of keystrokes associated with entering both the attribute name and the value. The user can encode common values as attributes within a default stanza that are inherited by stanzas that follow in the file, unless they specifically override the value.

For example, if the user owning foo.c was also responsible for foo.h and fee.c that were stored in the same source control directory, etc., then it would get quite tedious to repeat all the attribute-value pairs. Using the default stanza concept, the file might look like the following:

```
default:
      type=c_program
      lpp=bos
      node=3.1
      path=R2/cmd/foo
      owner=myuserid
      component=cmdfoo
      delta=""
      problems=""
      reason=""
      status=""
foo.c:

foo.h:
      type="c_header"
fee.c:
```

Neither colon separated files nor database formats support this default concept.

A stanza groups related attributes together in one place. Therefore, when a user needs to alter, delete or update information concerning a given object, she or he has only to find the corresponding stanza in the file. Taking advantage of the default stanza concept increases the maintainability even more, since changes to a single default stanza can affect a large number of "regular" stanzas.

Since attributes within a stanza are order independent, adding new attributes to drive another process does not affect existing processes at all. This is true even if the new fields are added between existing attributes. For example, suppose another toolkit operating on files was developed, such as a makefile generator. They hypothetical attributes derived (e.g., from source__list and build__process) could be placed anywhere in the stanza without affecting the original Toolkit. Further, deleting an attribute only affects those processes that use it. Even those processes that do use it might remain unaffected, if they can handle a null valued attribute (since nonexistent attributes return null when queried).

The fixed nature of colon separated files and database formats means that most changes to the entry require update to all processes using the file. The exception is colon separated files where new attributes are added to the end of the entry.

Even though stanzas are somewhat free form, they are still highly structured. A stanza can be equated with a record in a database, and the attributes as the fields. Every attribute carries its own "field name," meaning that attribute files can be easily translated or compiled into other forms.

The result is that attribute files can be used to provide the information storage component of toolkits that are easily converted to a more efficient form. Also, few (if any) user interface tools need to be written, since standard AIX utilities (like vi and pg) can be used to edit and display the attribute file format.

By treating attributes defined in a stanza as shell variables, any process executed by the Process Control can use these variables just as they would any other. All of the enhanced quoting mechanisms described above (except the single quote) substitute variables. Therefore, one attribute can refer to the value in another. A side benefit is that the action parameter of a Process Control command can use standard shell commands.

One problem with the default stanza concept known in the prior art is that it introduces a positional semantics, since all stanzas following a default stanza inherit its attributes (unless they are explicitly overridden). This means a misplaced stanza can be unintended attributes defined, sometimes with disastrous results.

Also, if a new default stanza is encoded, all of the attributes defined in the previous one are unset. This introduces a source of redundancy, especially when only one attribute needs to change (all of the others would still need to be encoded).

Default stanzas encourage reuse of attributes and make the file much more concise, but they do not support the kind of reusability that an inheritance hierarchy can provide. Therefore, the Process Control recognizes a special attribute called type that refers (optionally) to one or more other stanzas that will be used to supply values not explicitly declared in the stanza itself. Since more than one stanza name can be listed, multiple inheritance is supported.

The ability to interpret attributes like shell variables within an attribute value means that a stanza can not only override the value of its "parent," but also can edit it. This feature provides an extremely high degree of potential for reusability.

The Process Control provides stanza update capability by comparing the environment variables corresponding to attributes to the pre-execution values. If they have changed, then the value within the file will be updated. If the $@ variable is set to null within the command script, then the stanza is deleted.

As more and more processes are controlled by stanzas in attribute files, there will be an increasing amount of contention for an individual file (especially with update support). This means that some method to break the attribute files up into smaller components must be provided, in order to allow for distributed file support.

Breaking up the files has the unwanted side effect of separating stanzas representing classes in the inheritance hierarchy from the attribute files that refer to them. Thus, a stanza must be able to refer to definitions in other files. Stanzas refer to those in other files by:

1) explicitly naming the file in which it is contained (<stanza>(<file>));
2) looking in other files listed on the command line, in order;
3) checking in files named attrscan.pcf in the filesystem at the current directory level and "higher" (limited by an environment variable: $ATTRSCAN_TOP).

The order listed is the order in which the files are searched. The first matching stanza is used.

As previously mentioned, the definition of the PCF architecture led to the definition of the Command component. Each Command component will be summarized separately. The features of the PCF architecture can be summarized as follows:

1. A PCF is a system file that can be opened for read or write; it is composed of a set of objects
2. An object is a named collection of attribute value pairs; they can be retrieved either "directly" by a reference or "indirectly" through a selection expression.
3. The selection expression can be based on the object name or other attributes; an unspecified expression selects all objects in the Process Control File.
4. An object can be retrieved for read or for read/write, depending on the state of the access permissions and access list attributes.
5. Encoding an object in a Process Control File with the same name as the PCF allows its access permissions and access list attributes to override equivalent filesystem values for the whole file.
6. Attributes can be inherited through references in the type attribute; inheritance can be multiple and/or hierarchical.
7. A reference requires the object name, a list of PCF names and an inheritance status.
8. The inheritance status can be: a) enabled, b) disabled, or c) unspecified (in which case it defaults to that used to retrieve the referencing object).
9. The PCF name list of a reference can be left unspecified, in which case it defaults to a special environment variable.

The features of the Command architecture are described as follows:

1. Environment variables and attributes are equivalent (hereafter, they will be called variables). Both have

a name associated with a value.

2. Variable values can be derived in four ways: (i) as literals, (ii) from variable substitution, (iii) by executing the value and returning standard output, and (iv) as a list or an element of a list.

3. Variable substitution is otherwise treated like most shell language substitutions.

4. Values can be used in any of the four ways specified in "feature" 2; or they can be used as: (i) an executable method, (ii) an embedded object, or (iii) a reference to another object.

5. Interpreted methods are either system commands or other executable variables.

6. Executable values can contain either interpreted or compiled methods; they return an exit code.

7. Embedded object attributes provide for complex attributes and enables attribute level access control.

These features will meet every other requirement listed in the Background section that is not covered by the high level architecture itself. Features 2-4 above have been previously described. Feature 1 is shown in Fig. 8, which shows an environment 106 having variables associated therewith, and an object 108 having attributes associated therewith. The object attributes are searched first and if a match is not found, the variables within the environment are searched. The net effect of this approach is that the system operates as if the environment itself has been inherited.

Feature 5 is shown in Fig. 9, and implemented in the syntax notation as shown in Fig. 5a-5f. Interpreted methods 124 are either system commands 122 of the shell script script 120, or are other executable variables at 110. As shown at 124, 'a=hello world' is an executable variable which is interpreted by the interpreter, and '(echo $a)' escapes to the shell script 120 to execute a system command at 122 of 'echo'.

Feature 6 is also shown in Fig. 9, where the executable values at 110 can contain either interpreted methods 116 or compiled methods 114. Here, the resulting effect is that the system is easily extensible.

Feature 7 is exemplified in Fig. 10, showing that the object attribute 'access_list' 130 allows users george and john to access 'foo'. However, the embedded object attribute at 132 only allows john to access the underlying structures 136 within the object 'a' 134. Other embedded object attributes are shown at 136. The next section details the architecture of the Process Control and User Interface that together make the Process Control Framework even easier to integrate and use.

Referring again to Fig. 2, the User Interface 15 is, in a sense, the Process Control 17 since what the end user sees is what he will perceive the Process Control Framework to be. Therefore, it is extremely important that the User Interface Architecture provide transparent access to the underlying functions, including System Commands 20 and Process Control Files 19, where required. This goes against the grain of most "friendly" user interface architectures, since they often try to "hide" as much of the system as they can from the user.

There are often good reasons to hide functionality, but there are equally good reasons to make the function obvious. To understand these reasons and the functionality that should be visible, we must understand the user community.

The Process Control Framework is envisioned to have three kinds of users: integrators, developers and managers, each of which will be interested in the Process Control for a different reason. Let us look at the needs of each separately.

The first user of the Process Control Framework will be the toolmaker, whose job it is to set up the processes that are going to be controlled. Remember from the introduction that this requires developing a machine readable state transition diagram of the process from object definitions. Therefore, the product related objects 21 of Fig. 3 and their associated attributes 30 must be determined, along with the tools, such as version control 34, source preparation 36, development build 38, and unit test 40, that move objects from state to state (or, more precisely, use and act upon the attribute values themselves).

An integrator will need to read and write product related objects 21 into Process Control Files 19. Some objects will be high level "parents" in the inheritance hierarchy, hereafter called product class objects or simply class objects. For example, an integrator for the build process 38 of Fig. 3 will create classes such as c_program, c_header, linkable_object and executable_object.

Commonly used attributes 30, such as standard libraries to use when linking, default dependencies (or tools to execute to derive the dependencies) and default build processes will all be set up for the developer to use. Other attributes 30, like functions to compute lines of code or test coverage, would be set up for the managers. Note that the attributes 30 for the developers insure that the process is repeatable; those for the managers make sure that the process is measurable.

To help set up new classes, the integrator will need facilities to copy attributes from a similar class object, and "merge" these into a higher level class object if appropriate. For example, many of the attributes that exist for c_program objects might be shared by a new pascal object (except for the compiler, of course). An astute integrator might create a superclass and call it source_code. The merge routine automatically determines the set of unique attributes for each object in the superclass, using a standard compare and merge programming technique to eliminate attributes that are in common with the union of objects comprising the superclass.

Once the process has been integrated, the developer will become the second user of the Process Control framework. The developer is the one who uses the process to do real work; i.e., produce a product. Where the process is software development and most of the tools are machine executable, the Process Control Framework will make life much easier. Instead of needing to know how to invoke the specific tools, the developer selects objects and executes associated actions.

Selecting objects and executing actions requires the services of both Process Control Files 19 and System Commands 20 of Fig. 2. The actions are much simpler than those required by the integrator, usually just the name of an executable attribute associated with the object(s) selected. For example, a developer would indicate the objects to build and then invoke the build_process attribute.

One major distinction exists between a developer and an integrator. Where the integrator works with classes of objects, the developer works with particular members of the class (also called an instance). For example, a file called "foo.c" is an instance of a c_program class object. The integrator is mainly interested in editing, compiling, linking and testing in the abstract, where the developer wants to do these things for real. Note also that the developer only refers to the classes, and never updates them. The developer will need to create and maintain the instances; therefore, she needs a way to create, update and destroy instance objects as well as select class objects from which to inherit. This means that there are two separate lists of files with which a developer works: those which are reference only (that contain classes) and those which are actively being scanned for selection and action (that contain instances). The User Interface Architecture 15 at Fig. 1 supports modification, storage and selection of these lists by treating the lists exactly like any other PCF and object within the system.

Developers usually invoke actions in one of three ways:

1. They select an object (or set of objects) and attempt to invoke a specific action (or set of actions). This could return an error, not just because the action fails, but also because the object is not ready for that action, since the user asked the system to: "Do these actions to these specific objects."
2. They select either a specific set of objects or an expression for the scan list and invoke the next (default) set of actions. This only returns an error if an action fails, since it is like saying: "Do all the actions that you can to all the objects in this set." Presumably, the objects will reach a state where some external checkpoint terminates automated activity.
3. They invoke a specific set of actions on objects matching an expression (or the entire scan list). This shouldn't return an error either, except when an action fails, since this is the equivalent of saying "Do these actions to every object in this set that is ready."

The Process Control User Interface Architecture supports these three methods. It checks to see that the object is in the proper state for the action, or vice versa. Menus are accessible that show both views: the actions selectable for a set of selected objects, and/or the objects selectable for a set of selected actions.

With these three toolkits contained within 15 of Fig. 2 and 70 of Fig. 13, the developer has actually begun producing some products, or at least has gotten them to a "checkpoint" that requires some outside review. This is where the manager comes in.

Tradition puts a manager somewhere outside of the process looking in. To some degree, this is an accurate assessment, since the manager does not seem to create any product related objects. Instead, a manager seems concerned with accessing attributes associated with measurability to determine if it is proper to go on to the next step. For example, a manager might need to "sign off" before the fix to a source file can be forced into the build.

The problem with this view is that this "manager" does not have to be a separate person. She may be the developer herself, or not even be a person at all! For example, lets say that when a program fix compiles successfully and over eighty percent of its code is covered by successful execution of regression testing, then it is OK to include it in the build. In this case, the test and coverage analysis processes help automate the "sign off" function of the manager.

Whether the "sign off" is manual or automatic, it is a policy decision that leads to the definition of the criteria used to determine whether an action is "valid." This always requires an actual person. Thus, the true role of a manager is to create and maintain the policies that govern the process.

These policies, like everything else having to do with the process, are objects 21 that are maintained in the Process Control Framework. They are "transitions" in the state transition diagram of the process. Basically, they are conditions based on the attributes 30 of product related objects 21. Each condition is mapped to a single action that is permitted when the condition occurs. Referring to Fig. 11a and 11b to detail how this works, the policies are defined within 'issue' 144. 'Menu' 142 echoes to the screen those methods valid for a particular system user. 'Execute' 140 actually performs the state transition if the requested action is valid. Fig 11b shows sample invocation and output for 'execute' at 146 and 'menu' at 148, when issue4 has been defined as shown at 145.

Where the integrator is concerned with maintaining classes of product related objects, the developer is concerned with maintaining instances of product level objects that refer to the classes. A manager is concerned with maintaining policies that refer to both. Once the integrator develops a class with a certain set of attributes and associated actions, the manager can develop a policy that governs when those actions come into play. The policy is activated by a developer during the operations as previously described. Therefore, the manager's job is quite different than that of an integrator or developer. The manager needs a specific set of User Interface Tools 70 of Fig. 13.

The policy objects are similar to tools developed by the integrator - they need to be edited and tested. However, they operate at the same high level as those expressions and actions for a developer. Conditions are based on expressions of attributes; an associated action is usually a single executable attribute from some object. Therefore, a manager needs tools similar to a developer, that allow him to construct expressions and select associated actions. Like an integrator, a manager will need tools to allow him to test the policies developed. The manager needs tools that allow status reports to be taken at any point in the process based on the "measurement" attributes. Actually, the objects that create reports are almost the same as policies, where a selection expression is associated with an action that handles report formatting (presumably developed by the integrator). The only difference is that a report action will not change the state. Once developed, these "reports" are persistent objects as well. Like any other program, they will need both a development and a test environment.

To summarize the user interface, the following is a list of functions that are supported by the User Interface Architecture and the user group with access to that function. It not only summarizes the features, but gives a better picture of how to limit visibility based on the user type:

FUNCTION	Integrator	Developer	Manager
1. Maintain product classes	yes	no	no
2. Edit and Test "low level" tools	yes	no	no
3. Select/refer to product classes	yes	yes	yes
4. Create/delete product instances	no	yes	no
5. Update product instances	no	yes	yes
6. Execute product actions	no	yes	no
7. Access product instances	no	yes	yes
8. Maintain/select scan PCF lists	yes	yes	yes
9. Maintain/select reference PCF lists	yes	yes	yes
10. Maintain/select selection expressions	yes	yes	yes
11. Maintain/select object references	yes	yes	yes
12. Maintain/select action lists	yes	yes	yes
13. Maintain/select Policies	no	no	yes
14. Refer to policies	yes	yes	yes
15. Maintain status report	no	no	yes
16. Execute status reports	yes	yes	yes

At some point, each user type can be considered a "developer." For example, an integrator "develops" classes and low level tools. A "normal" developer works with product related instances. A manager "develops" policies and status reports. Therefore, the developer function is the "real" function always available in any view. The difference is in the objects that are visible and available for execution of product actions (number 6) at any given point.

This idea implies one additional feature: a user is able to create/maintain and select a view that represents the user view of the particular "products" (or groups of products) plus the following objects: a) scan PCF lists, b) reference PCF lists, c) selection expressions, d) object references, e) action lists and f) status reports to execute. To "change hats", a user need only select a different view (each of which is maintained as an object in a user defined PCF file).

The objects visible from a given view can be customized to only those needed for a given task or project. Further, since a view is an object, views can be classed and subclassed as well. The net result is that the Process Control is extremely easy to use, increasing both productivity and quality.

Fig. 12 further illustrates how an entity, or object, manager 60 allows for both internal and external access to an object. Remember that an entity or object 62 has associated therewith both attributes 30, which contain product and control information, and methods 31(e.g. programs to run against an object). Fig. 12 illustrates that an internal object 62 has its own attributes and methods, maintained by the object manager 60, which can be, and in the preferred embodiment are, different from an external object's 64 attributes 66 and methods 68. Communicating or accessing an object is achieved by links 32.

The above description provides three levels of integration. At the user interface level 15 of Fig. 1, standard methods can be invoked on any object registered to the system. The registration process creates the object with attributes that associate an interface representation to the external object. This allows the object to be opened (if editable) or executed (if executable). This representation could include text, graphics, or auditory information.

The second level of integration is achieved by recording process control attributes about the target object in the Process Control File. This allows the user to build and execute custom methods associated with the object(s). The format of an object's internal data structure is not registered or maintained within the object. Therefore, only processes that operate on the PCF object as a whole can be developed.

The third level of integration is achieved when the internal data structure associated with a given target object is stored in the PCF. Then, the Process Control Framework can be used to access specific data structures within the target object. This allows custom methods to be created that bring together related information from multiple target objects in a seamless application. An object manager, to be described hereafter, is used to maintain the internal attributes and data structures necessary to provide each level of integration.

The above described Framework provides the ability to use external objects, in that objects at any level of integration can still be maintained externally to allow free access to other processes not a part of the Framework. Further, as the attribute and method components of objects are separable, multiple implementation approaches can be used for each one. For example, the entity/object manager that maintains the attributes can be implemented using the PCF syntax as disclosed herein, a semantic network, or an object management system. The methods can be implemented via interpreted commands, compiled languages, or a language built into the Framework object manager itself.

Fig. 13 shows a physical model of the AIX CASE environment, which can be viewed as a set of interrelated components that provides base and extended levels of functions for a true integrated project support environment. All these components have an exposed interface for the tools and applications to use. Minimally, this should include C language bindings, but other language binding may be appropriate. The major components of this architectural model will now be discussed in sufficient detail to see how the Process Control Framework would become a part of the larger CASE Environment.

The object management system 160 is the heart of the AIX CASE Environment. It provides the storage, integrity and control that the rest of the environment requires. The object management system can be viewed as consisting of the following components:

- An object manager.
- A link manager.
- A type manager.
- An access manager.
- A version manager.

These components of the AIX CASE Environment object management system are described individually below.

The object manager 162 provides the storage of objects in the AIX CASE Environment object management system. There are two types of objects that the object manager recognizes -- internal objects and external objects. Internal objects are objects that are stored and managed directly by the object manager. External objects are objects that are stored and managed outside of the object manager domain. Specifically, external objects are those objects that reside as separate entities in the AIX file system, including existing AIX Source Code Control Systems. The object manager guarantees the integrity for only internal objects. One of the key aspects of the object manager is object naming. The object naming rules must be constructed carefully to support very large object sets (10^6 objects) and, additionally, support object naming across a distributed network environment. Another key aspect of the object manager is performance. Finally, the object manager must view the environment as distributed. The approach selected is to provide a client/server model that uses a message passing scheme through the distribution manager to converse with the other object managers in this distributed environment. The object manager satisfies the requirements for a persistent storage for project information and environment preservation in the AIX CASE Environment.

The link manager 166 manages the relationships between objects in the object management system. The link manager provides facilities to link one object to another object, one object to many other objects, and many objects to one object. The link manager must support a very large number of object relationships within a project's environment (10^9 links) and uses the services of the object manager to create, update or delete links. The link manager satisfies the requirement for providing object relationships in the AIX CASE Environment.

The type manager 168 manages the type hierarchy in the AIX CASE Environment. It insures that the specified type hierarchy is enforced and is consistent at all times for all defined objects in the project's environment. The type manager's responsibility is to provide integrity constraints for a specified type hierarchy, automatic object linking (via the link manager), authorization control (via the access manager) and version control (via the version manager) for objects defined in the specified object class type hierarchy. The type manager can be thought of as the "traffic cop" for the object management system. The type manager satisfies the requirements for integrity constraints and environment extendibility (through the addition of new object classes and type hierarchies) in the AIX CASE Environment.

The access manager 170 provides the security authorization control in the AIX CASE Environment. The access manager has the notion of users and groups of users with differing access rights to objects -- none, read-only, read-write, execute, etc. Since users are defined as objects in the AIX CASE Environment, these access rights are properties in the type hierarchy of this object class. The access manager provides the locking mechanism on objects. With the object-oriented model, this results in linking user objects with the physical objects "checked out" to these users. Additionally, the access manager manages the view of mutable and immutable objects in the environment again as specified in the properties of the type hierarchy for the object class. Therefore, the access manager provides the integrity constraints, multiuser access and administrative control requirements for the AIX CASE Environment.

The version manager 172 is required to record the history of change and represent the effect of time for a project's environment. The version manager is based upon the concept of mutable objects. However, as mutable objects change over time, their descendants become mutable objects. A branch type abstract sequence provides this concept of multiple versions of objects. Such a sequence is known as a line of descent. Branches provide support for alternate or variant lines of descent. The main line of descent is the root branch. New branches can be started from any immutable version of an object. A merge operation is also supported to converge two branches into a single line of descent. For standard text objects, the version manager stores these branches as deltas to the lines of descent. However, for other object classes, e.g., graphics, images, binary encoded data, audio, etc., the delta approach is not conducive to conserving storage. Therefore, the version manager has the concept of n-level deep backup copies for these object classes, where n can be from 0 to i, as specified by the user. This maintains the notion of immutable objects and branching. The version manager satisfies the requirements of environment evolution, multiuser access and reproduction and replication of a project's environment for the AIX CASE Environment.

The user interface component 70 provides the common "look and feel" for the tools, applications and the AIX CASE Environment, itself. The user interface component is OSF's MOTIF (Trademark of Open Systems Foundation) user interface visualized through the X-11 client/server model. This user interface component satisfies the requirement for visual integration in the AIX CASE Environment.

The distribution manager 164 is responsible for supporting the AIX CASE Environment in a distributed workstation environment. The distribution manager is a queued message-driven interface and operates in a TCP/IP environment through a standard socket interface. In other words, there are cooperating distribution managers that communicate with each other in this AIX CASE Environment. The distribution manager provides a means to notify the requester on the status of queued messages to guarantee integrity and serialization for the environment. The distribution manager satisfies the environment extendibility and the ability to distribute the persistent storage for project information requirements for the AIX CASE Environment.

The event manager 174 provides the notification mechanism to the AIX CASE Environment. It supports single event to single action model, single event to multiple action model and blocking on multiple events. The event manager is implemented as a message passing system. This allows the event manager to send textual messages to users or data messages to other tools. The event manager 174 is invoked by the type manager 168 on some action when an object's type hierarchy specification requires this notification. The event manager interfaces with the distribution manager 164 to properly send the message(s) to the appropriate destinations. The event manager provides the block for multiple events by instantiating a lock against those objects waiting for an action to complete. The event manager satisfies the requirement for providing event notification in the AIX CASE Environment.

The schema definer 176 is the process by which the system administrator defines the project's environment. With the schema definer, the system administrator defines the object classes, the visible type hierarchy and enrolls tools and applications as objects to the AIX CASE Environment. The schema definer really is a set of "menus" that allow the system administrator to initially define a project's environment and to tailor this environment throughout the life of the project. The schema definer specifies the properties of objects in the environment to the type manager 168, access manager 170 and version manager 172 in the object management system 160. Additionally, the schema definer must recognize the existence of external objects (tools and data) and permit their definition within a project's environment as well. The schema definer satisfies the requirements of integrity constraints, environment preservation, environment extendibility, environment customization and control integration for the AIX CASE Environment.

The "browser" 178 is the means by which the user can view object relationships managed by the object manager 162. It uses the relationships (links) between objects to permit a user to traverse the relationships between objects. The "browser" has a MOTIF graphical user interface where objects are represented by icons and the relationships between objects shown by arcs (lines). The "browser" satisfies the requirement of visually showing object relationships to the user.

The administrative services manager 180 provides the capability to install, backup, restore and maintain the AIX CASE Environment. These facilities can be implemented as a set of utilities that are available to the system administrator. These utilities are constructed in such a manner that they can operate as standalone AIX commands or through a MOTIF user interface with "buttons" and "pull downs" representing the options to be selected. The administrative services manager satisfies the requirement of providing administrative control for the integrated project software environment.

The query manager 182 provides the user the ability to ask questions about the information content of the data maintained by the object manager. The query manager provides the ability to interrogate the object manager 162 regarding object classes, objects, object relationships, type hierarchies and schemas, as well as the overall status of the project. The query manager is simple to use with a MOTIF graphical interface. It allows for the ability to store both the target and the results of a query for the user. The target and results can be viewed as objects in the environment. The query manager satisfies the requirement of providing access to project information.

The work flow manager 184 provides the enforcement of project methodologies and policies. The work flow manager can be viewed as a tool that interacts with the system administrator in setting up and modifying the work flow for the project. The work flow manager interfaces with the type manager 168 in the specification of the methods (tools) to be used in the type hierarchy for object classes. For example, the work flow manager may specify in the object class of C source objects that in order to "check in" a C source object into the object store that it first must compile correctly and pass lint successfully. The work flow manager can be tailored to each project's environment. It supports the range from stringent DoD standards to very relaxed work flow methodologies. The work flow manager satisfies the requirements environment customization, environment evolution, control integration, administrative control and availability.

The configuration manager 186 provides the methods for grouping objects into a configuration or collection that can then be manipulated as a single entity. A MOTIF graphical representation of objects represented by icons that are related together through arcs that define the methods (tools) provide the configuration for this collection of objects. The collection can be treated as an object itself. Collections are a subtype of version in the type hierarchy and can, therefore, support versions on collections. Further, collections can be considered mutable or immutable. Immutable collections contain only immutable objects. Mutable collections can contain one or more mutable objects. The configuration manager satisfies part of the requirements for reproduction and replication of the software project and project reusability.

The method sets 188 are a set of standardized interfaces that provide the mechanism for tool-to-tool data integration. These method sets will grow over time as standards evolve and new data interchange methods are established. Each method can be represented as an object in the object class of methods and typed with the type hierarchy. These method sets satisfy the requirement for data integration between tools and applications.

While we have illustrated and described the preferred embodiments of this invention, it is to be understood that we do not limit ourselves to the precise constructions herein disclosed and the right is reserved to all changes and modifications coming within the scope of the invention as described in the appended claims interpreted by the description.

Claims

1. A method for maintaining entities of information in a computer process control environment, said entities having process control attributes and data structures, comprising:

recording said process control attributes by operation of an entity manager;

recording said data structures separate from said process control attributes by operation of said entity manager; and

processing said entities in said process control environment, said processing further comprising using said recorded data structures and said recorded process control attributes when accessing said entities.

2. The method of Claim 1 further comprising:

storing said entities of information in a file,

referencing said entities of information by multiple processes, wherein only a single copy of said entity of information is maintained for reference by said multiple processes.

3. The method of claim 2 wherein said multiple processes customize their respective views of said single copy of said entity of information by saving process specific process control attributes.

4. The method of Claim 3 wherein said process specific process control attributes which are saved by each respective multiple process are limited to only those process specific process control attributes which are different from said process control attributes of said entities of information.

5. A method for allowing access to an object in a computer environment by both an internal and an external process, said object having internal attributes and external attributes associated with each respective internal and external process, comprising:

recording said internal and external attributes associated with said object apart from said object;

maintaining an internal link between said object and said internal attributes;

maintaining an external link between said object and said external attributes;

accessing said object by said internal process using said internal link; and

accessing said object by said external process using said external link.

6. The method of Claim 5 wherein said internal attributes are inherited from another object.

7. The method of Claim 5 or 6 wherein said external attributes are inherited from another object.

8. The method of Claim 5, 6 or 7 wherein said objects are maintained in a file.

9. The method of Claim 8 wherein said file is machine readable.

10. A computer system for incorporating software development tools and data into a multi-leveled integrated environment, comprising means providing each of:

a user interface level having objects therein, said objects having attributes, methods, links, and data structures;

a process control level in communication with said user interface level and having an object manager which records process control attributes relating to said objects; and

an internal structure level in communication with said process control level which records said object data structures by the operation of said object manager, wherein said objects at any level of integration can be maintained externally by one or more external processes to allow free access of said objects to said external processes.

11. A computer system for incorporating software development tools and data into a multi-leveled integrated environment, comprising means providing each of:

a user interface level having objects therein, said objects having attributes, methods, links, and data structures;

a process control level in communication with said user interface level and having an object manager which records process control attributes relating to said objects;

a client level in communication with said user interface level;

a kernel level having a kernel ipc, a communication subsystem, a file subsystem, a process manager, and a display subsystem, all of which interface with a computer hardware system; and

a communication level in communication with and between said client level and said kernel level, for providing communications between said client level and said kernel level.

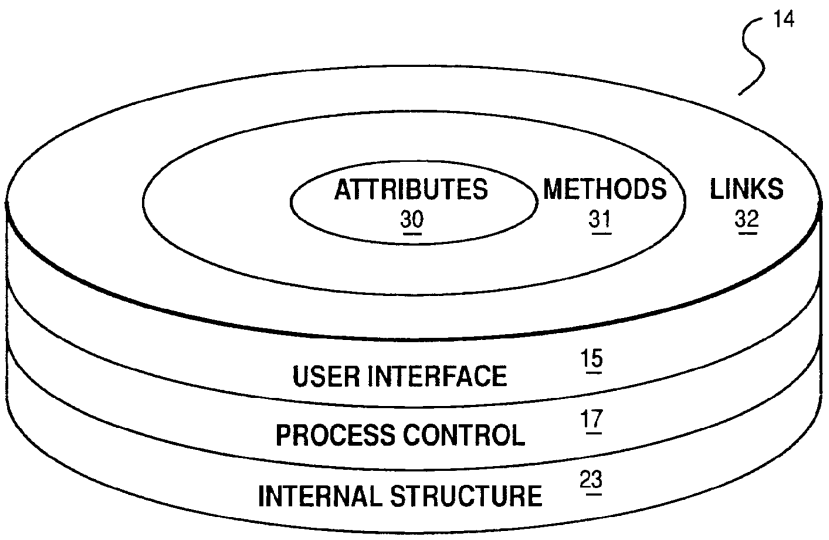

14
ATTRIBUTES 30
METHODS 31
LINKS 32
USER INTERFACE 15
PROCESS CONTROL 17
INTERNAL STRUCTURE 23

FIG. 1

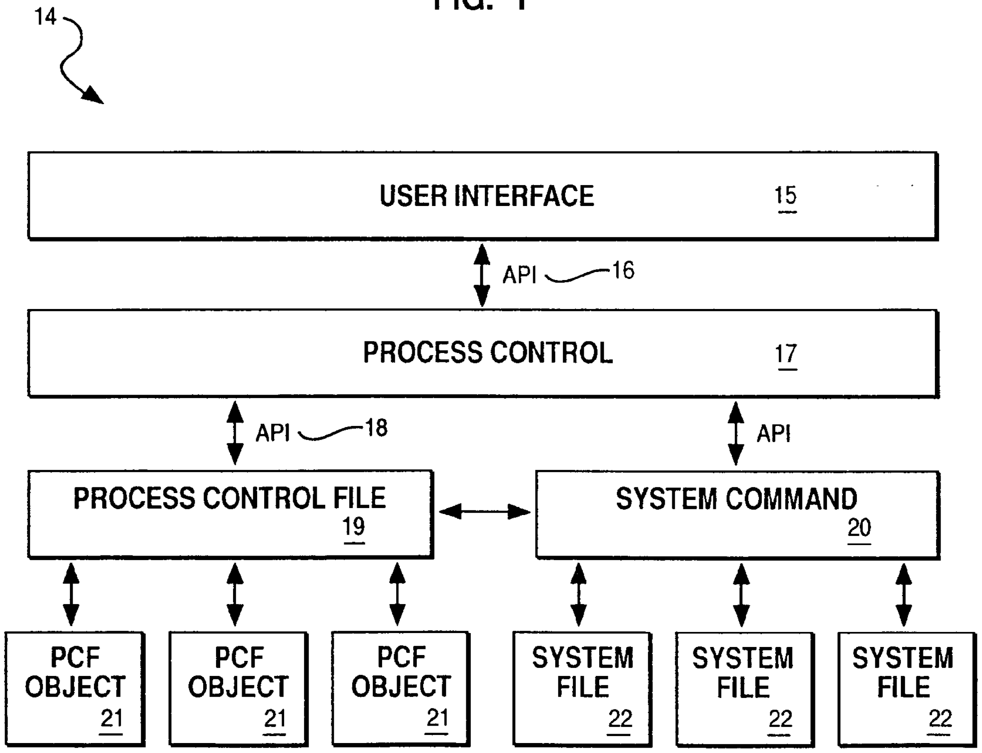

14
USER INTERFACE 15
API 16
PROCESS CONTROL 17
API 18
API
PROCESS CONTROL FILE 19
SYSTEM COMMAND 20
PCF OBJECT 21
PCF OBJECT 21
PCF OBJECT 21
SYSTEM FILE 22
SYSTEM FILE 22
SYSTEM FILE 22

FIG. 2

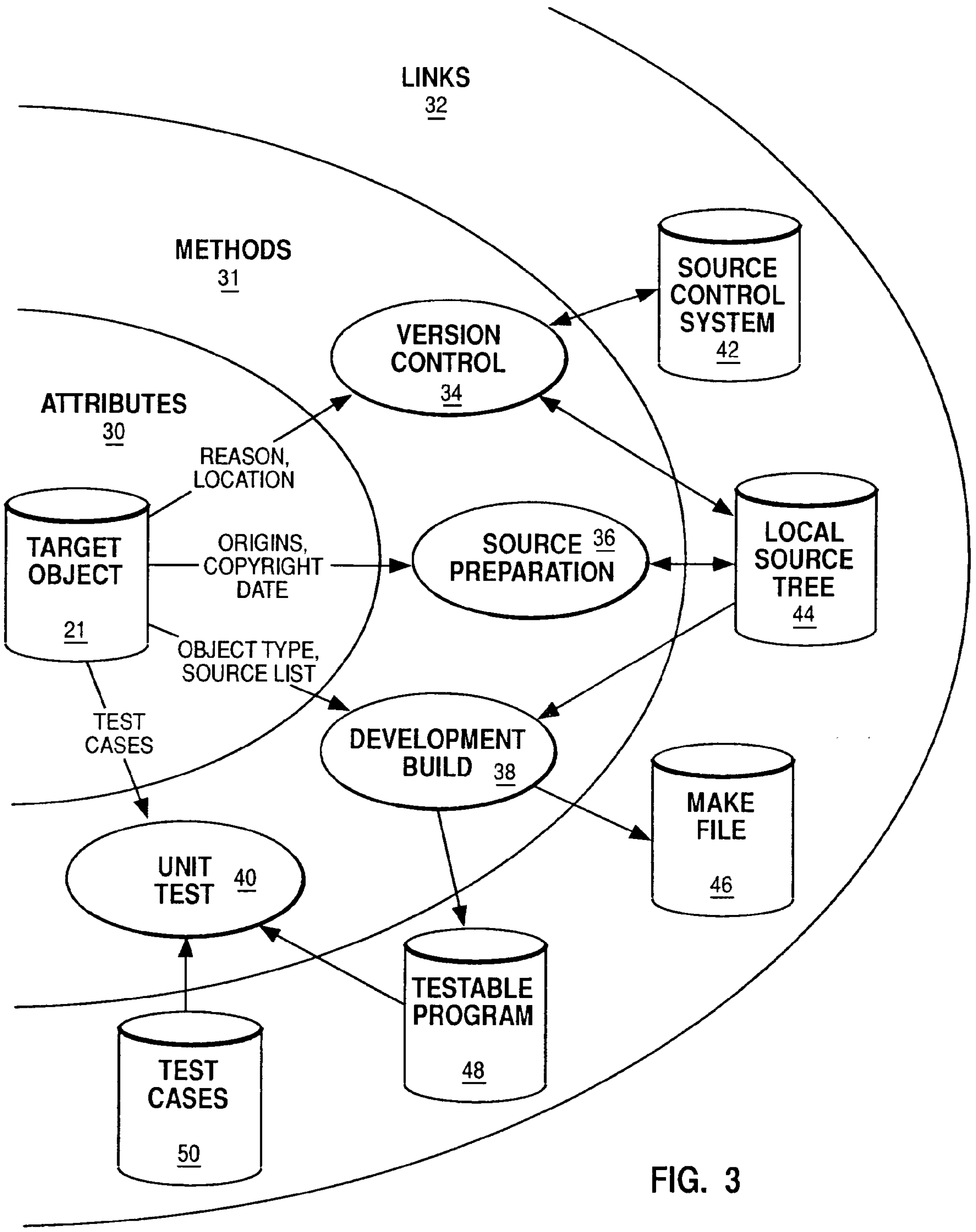

LINKS
32
METHODS
31
ATTRIBUTES
30
SOURCE CONTROL SYSTEM
42
VERSION CONTROL
34
REASON, LOCATION
TARGET OBJECT
21
ORIGINS, COPYRIGHT DATE
SOURCE 36 PREPARATION
LOCAL SOURCE TREE
44
OBJECT TYPE, SOURCE LIST
TEST CASES
DEVELOPMENT BUILD 38
MAKE FILE
46
UNIT TEST 40
TESTABLE PROGRAM
48
TEST CASES
50

FIG. 3

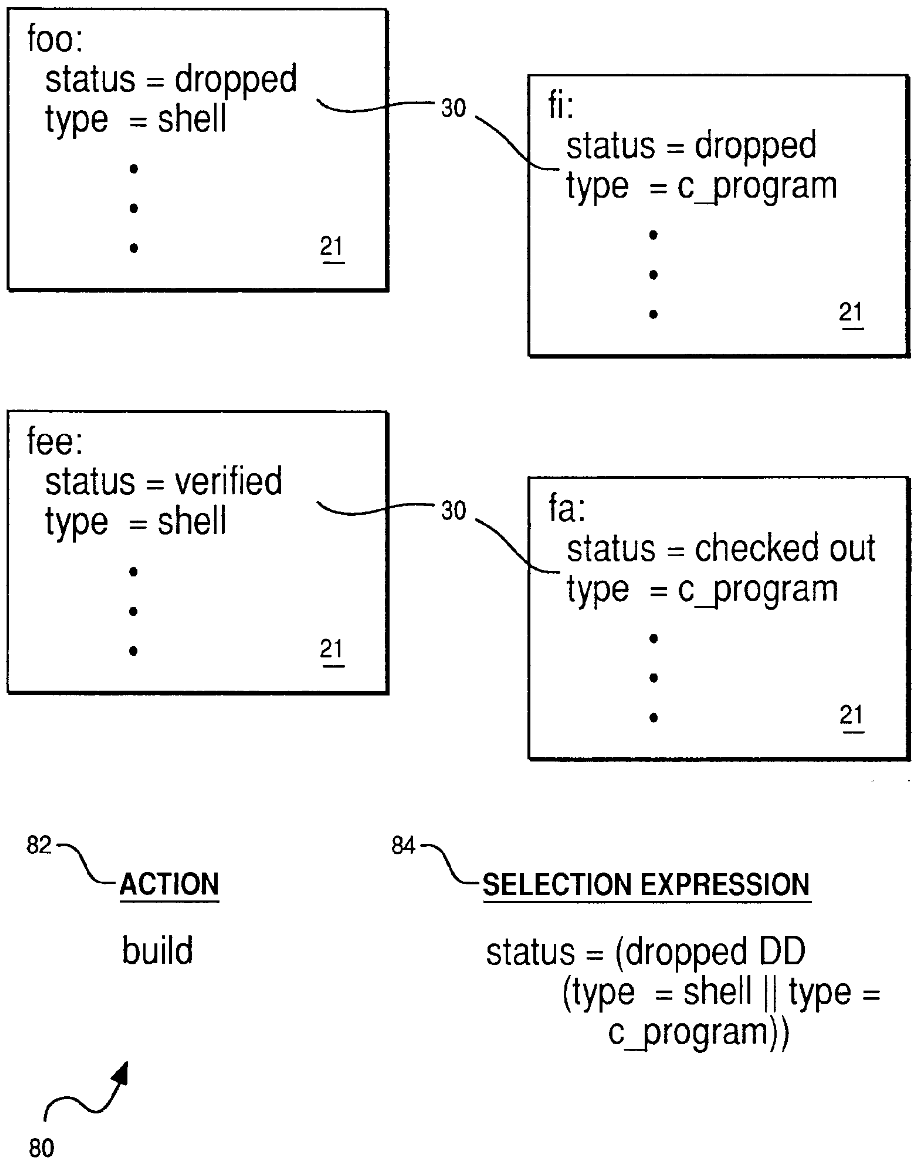

foo:
status = dropped
type = shell
21
30
fi:
status = dropped
type = c_program
21
fee:
status = verified
type = shell
21
30
fa:
status = checked out
type = c_program
21
82
ACTION
build
84
SELECTION EXPRESSION
status = (dropped DD
(type = shell || type =
c_program))
80

FIG. 4

```
SYNTAX:
   process_control_file :: =
       <optional_any_spaces> ¦
       <optional_any_spaces> <object> <process_control_file>

   object :: =
       <comment> <newline> ¦
       <name> <attribute_list>

   name :: =
       <name_string> <optional_any_spaces> ' : '

   attribute_list :: =
       <optional_any_spaces> ¦
       <optional_any_spaces> <attribute> <attribute_list>

   attribute :: =
       <comment> <newline> ¦
       <attribute_name_string> <optional_any_spaces> '=' <value_list>

   value_list :: =
       <optional_any_spaces> ¦
       <optional_any_spaces> <value> <any_space> <value_list>

   value :: =
       ' [ ' <value_list> ' ] '   ¦
       ' ( ' <embedded_object> ' ) '   ¦
       ' { ' <embedded_method> ' } '   ¦
       <simple_value>

   embedded_object :: =
       <object> ¦
       <attribute_list>

   embedded_method :: =
       <statement_line> ¦
       <statement_line> <newline> <embedded_method>
```

FIG. 5a

```
statement_line :: =
    <optional_any_spaces> ¦
    <optional_any_spaces> <comment> ¦
    <optional_any_spaces> <statement> <optional_spaces> ¦
    <optional_any_spaces> <statement> <optional_spaces>
          <comment>

statement :: =
    <if_statement> ¦
    <while_statement> ¦
    <for_statement> ¦
    <visit_statement> ¦
    <break_statement> ¦
    <continue_statement> ¦
    <exit_statement> ¦
    <return_statement> ¦
    <create_statement> ¦
    <commit_statement> ¦
    <destroy_statement> ¦
    <assignment_statement> ¦
    <action_statement>

if_statement :: =
    ' if ' <then_part>
    <elif_list>
    <else_part>
    ' fi '

elif_list :: =
    NULL ¦
    ' elif ' <then_part>
    <elif_list>

then_part :: =
    <optional_spaces> <action_statement> <end_line>
    ' then ' <endline>
    <embedded_method> <end_line>
```

FIG. 5b

```
else_part :: =
    NULL ¦
    ' else '  <end_line>
    <embedded_method>  <end_line>

while_statement :: =
    ' while ' <optional_spaces>  <action_statement>  <end_line>
    ' do '  <end_line>
    <embedded_method>  <end_line>
    ' done '

for_statement :: =
    ' for ' <spaces>  <attribute_name_string>  <spaces>
    ' in '  <spaces>  <line_list>  <end_line>
    ' do '  <end_line>
    <embedded_method>  <end_line>
    ' done '

visit_statement :: =
    ' visit ' <spaces>  <attribute_name_string>  <spaces>
    ' in '  <spaces>  <line_list>  <end_line>
    ' do '  <end_line>
    <embedded_method>  <end_line>
    ' done '

break_statement :: =
    ' break '  ¦
    ' break '  <spaces>  <simple_value>

continue_statement :: =
    ' continue '  ¦
    ' continue '  <spaces>  <simple_value>

exit_statement :: =
    ' exit '     ¦
    ' exit '  <spaces>  <simple_value>

return_statement :: =
    ' return '  ¦
    ' return '  <spaces>  <line_list>
```

FIG. 5c

```
   create_statement :: =
       <object_specifier>  ' platoCreate ' <spaces>  <line_list>
88             <end_line> ¦
       <object_specifier>  ' platoCreate ' <spaces>  <line_list>
               <end_line>
88     ' with '  <line_list>
   commit_statement :: =
       <object_specifier>  ' platoCommit '¦
88     <object_specifier>  ' platoCommit ' <spaces>  <line_list>
   destroy_statement :: =
       <object_specifier>  ' platoDestroy '
   object_specifier :: =
       NULL ¦
       <attribute_name_string>  ' . '
   assignment_statement :: =
       <address_string>  <optional_spaces>  ' = '  ¦
       <address_string>  <optional_spaces>  ' = ' <optional_spaces>
                 <line_list>
   action_statement :: =
       <not_statement> ¦
       <condition_statement>  ¦
       <shell_statement>  ¦
       <method_statement>  ¦
   not_statement :: =
       ' ! ' <optional_spaces>  <action_statement>
   condition_statement :: =
       ' [ ' <condition_expression>  ' ] '
   condition_expression :: =
       <condition_clause>  ¦
       <condition_clause>  ' ¦¦ '  <condition_expression>
   condition_clause :: =
       <sub_expression>  ¦
       <sub_expression>  ' && '  <condition_clause>
```

FIG. 5d

```
sub_expression :: =
    ' ! ' <sub_expression> ¦
    ' ( ' <condition_expression>  ' ) '  ¦
    <simple_value>  <optional_spaces>  <relop> <optional_spaces>
            <simple_value>
relop :: =
    ' = '   ¦
    ' < '   ¦
    ' > '   ¦
    ' != '   ¦
    ' >= '   ¦
    ' <= '   ¦
    ' in '
shell_statement :: =
    ' ( '  <shell_string_list>  ' ) '

shell_string_list :: =
    NULL ¦
    <shell_string>  <shell_string_list>

shell_string :: =
    <reference_string> ¦
    <shell_statement>
    <any arbitrary characters not including the " ( " or " ) ">

method_statement :: =
    <address_string>  ¦
    <address_string>  <optional_spaces>  <line_list>

end_line :: =
    <optional_spaces>  <newline>  <any_optional_spaces>  ¦
    <optional_spaces>  <comment>  <newline> <any_optional_spaces>

comment :: =
    ' # ' <any characters except newlines>

line_list :: =
    <value>  ¦
    <value>  <spaces>  <line_list>
```

FIG. 5e

```
simple_value :: =
    <simple_string>  ¦
    <simple_string>  <simple_value>

simple_string :: =
    <reference_string>  ¦
    <literal_string>  ¦
    <command_string>  ¦
    <quoted_string>  ¦
    <object_reference_string>

reference_string :: =
    ' $ '  <address_string>  ¦
    ' $$ '  <address_string>

address_string :: =
    <address_list>  ¦
    ' { ' <address_token>  <optional_any_spaces>  ' } '
    ' { ' <address_token>  <any_space>  <value_list>  ' } '

address_token :: =
    <optional_any_spaces>  <address_list>

address_list :: =
    <address>  ¦
    <address>  ' . '  <address_list>

address :: =
    <attribute_name_string>  <index_list>  ¦
    <attribute_name_string>  <index_list>  ' ( '  <index_value>  ' ) '

attribute_name_string :: =
    <arbitrary alphanumeric or  ' _ '  characters>

index_list :: =
    NULL  ¦
    ' [ '  <index_value>  ' ] '  <index_list>

index_value :: =
    <optional_any_spaces>  <index>  <optional_any_spaces>
```

FIG. 5f

```
index :: =
    <name_string> ¦
    <reference_string> ¦
    <command_string>
object_reference_string :: =
    <name_string> ¦
    <name_string> '{' <value_list> '}'
name_string :: =
    <any arbitrary string of alphanumerics, " / ", " _ ", or " . ">
literal_string :: =
    '\'' <any arbitrary string not including, '\' ' >  '\''
command_string :: =
    '‘' <command_string_list> '‘'
command_string_list :: =
    NULL ¦
    <command_string_token> <command_string_list>
command_string_token :: =
    <reference_string> ¦
    <any arbitrary string not including '‘'>
quoted_string :: =
    '"' <quoted_string_list> '"'
quoted_string_list :: =
    NULL ¦
    <quoted_string_token> <quoted_string_list>
quoted_string_token :: =
    <reference_string> ¦
    <command_string> ¦
    <any arbitrary string not including '"'>
optional_any_spaces :: =
    NULL ¦
    <any_space> <optional_any_spaces>
```

FIG. 5g

```
any_space :: =
    <space>  ¦
    <newline>

spaces :: =
    <space>  <optional_spaces>
optional_spaces :: =
    NULL   ¦
    <space>  <optional_spaces>

space :: =
    <any white space character except newline>
newline :: =
    '\n'
```

FIG. 5h

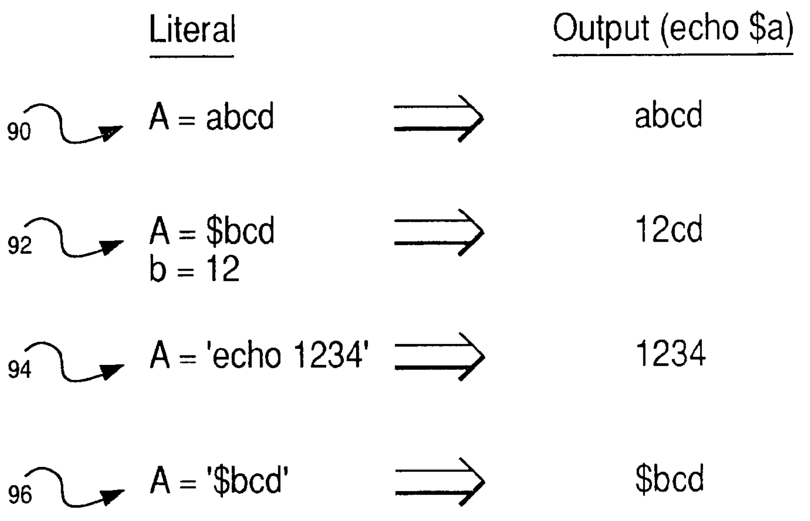


FIG. 6

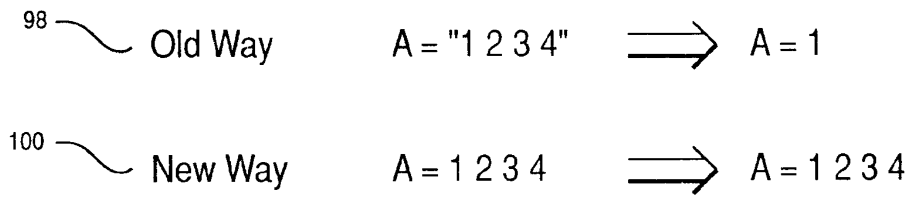


FIG. 7

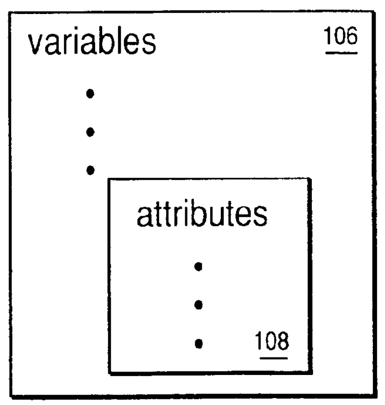
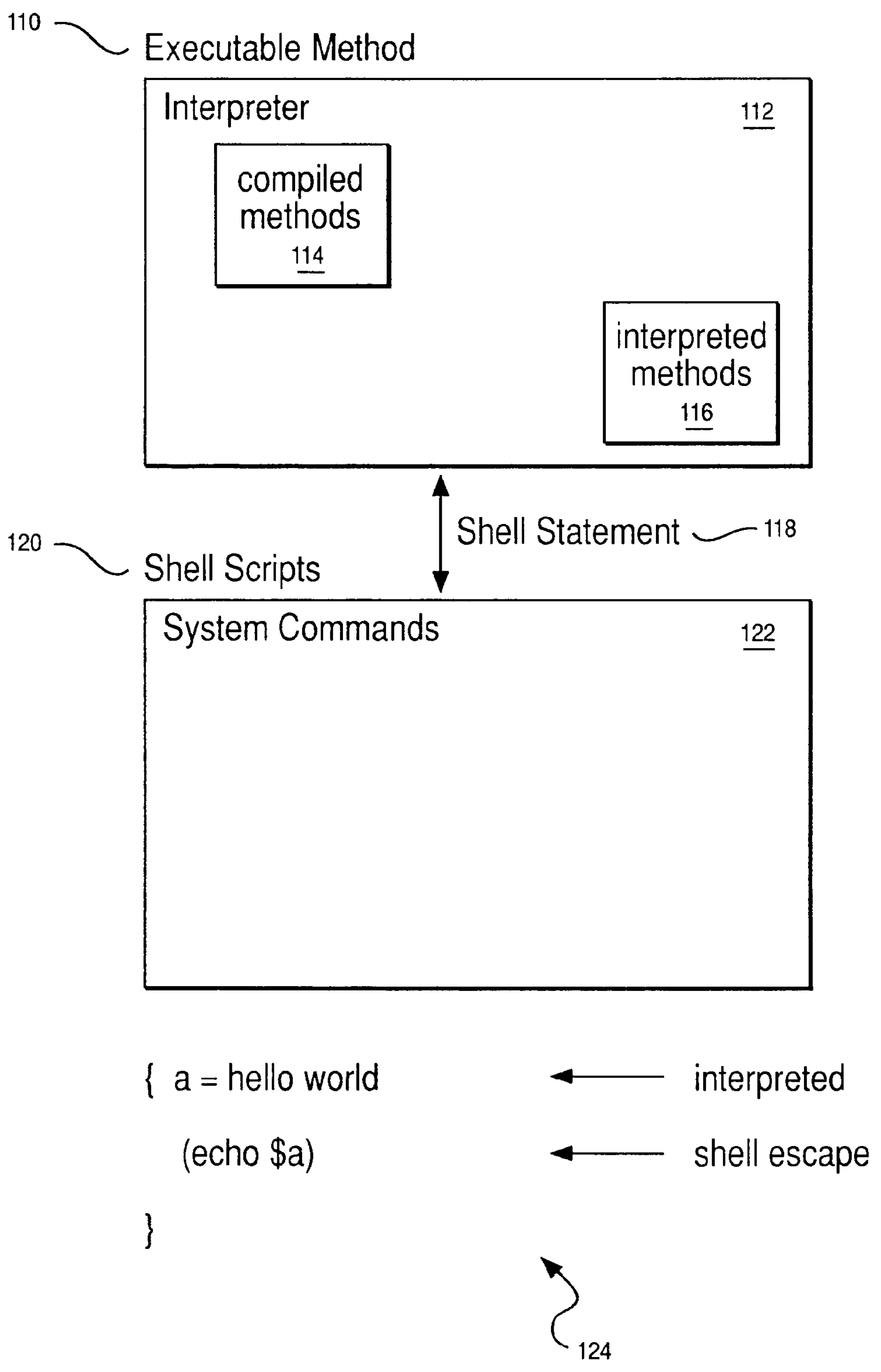
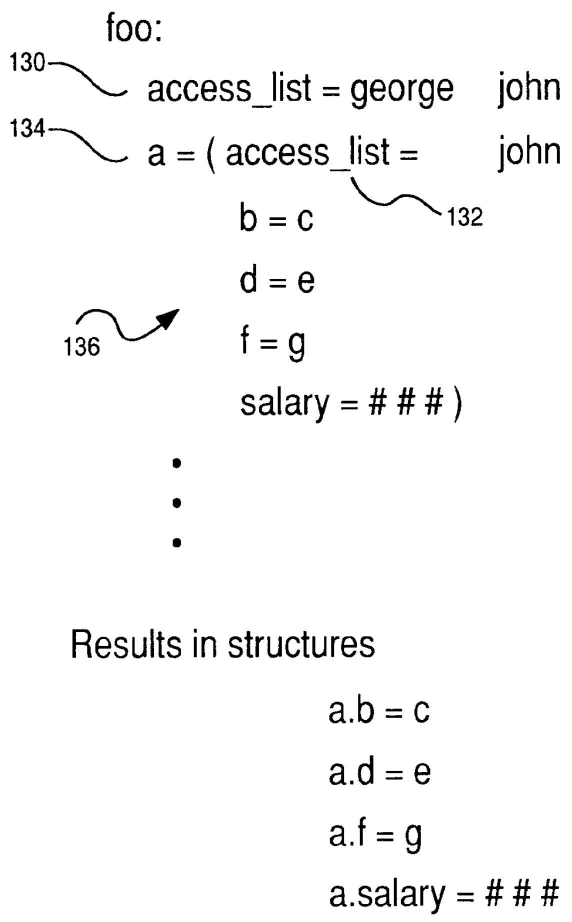


FIG. 8



110
Executable Method
Interpreter
112
compiled methods
114
interpreted methods
116
Shell Statement
118
120
Shell Scripts
System Commands
122
{ a = hello world
(echo $a)
}
interpreted
shell escape
124

FIG. 9



```
foo:
    access_list = george    john
    a = ( access_list =     john
          b = c
          d = e
          f = g
          salary = # # # )
      •
      •
      •

Results in structures
              a.b = c
              a.d = e
              a.f = g
              a.salary = # # #
```

FIG. 10

PCF EXAMPLE:

```
   controlled:
140   execute =
      {
         if [ $platoParms[ 0 ] in $$menu ]
         then
            if process.methods ( $platoParms[0] ).code
            then
               state=process.STD($state). transitions ( $*platoParms[0] ).next
            fi
         else
            (echo "Method $platoParms[0] is invalid in state $state for"\
                  "$platoObjName; see menu" >&2)
         fi
      }
142   menu =
      {
         for i in process.STD( $state ).transitions
         do
            if i.condition
            then
               echo $i.platoObjName
            fi
         done
      }
144 issue:
      type = controlled
      attributes = abstract owner due_date severity
      read_attributes = state history
      process =
         (states = opened answered closed
          initial = opened
          final = closed
          methods = (reroute:) (comment:) (answer:) (close:) (reopen:)
          STD =
            (opened:
             transitions =
```

FIG. 11a

```
            (answer:
              condition = { [ $owner != unknown ] }
              next = answered)
            (reroute: )
            (comment: ) )
        (answered:
          transitions =
            (reroute: next = opened)
            (comment: )
            (method = close
              next = closed) )
        (closed:
          transitions =
            (method = reopen
              next = opened) ) )

    issue4:
145 type = issue
    state = opened
    owner = unknown

    action:
146 "plato  -0   issue4  -C  "execute answer" "
    result:
       "Method answer is invalid in state opened
             for issue4; see menu"

    action:
148 "plato  -0   issue4  -C  menu "
    result:
       reroute
       comment
```

FIG. 11b

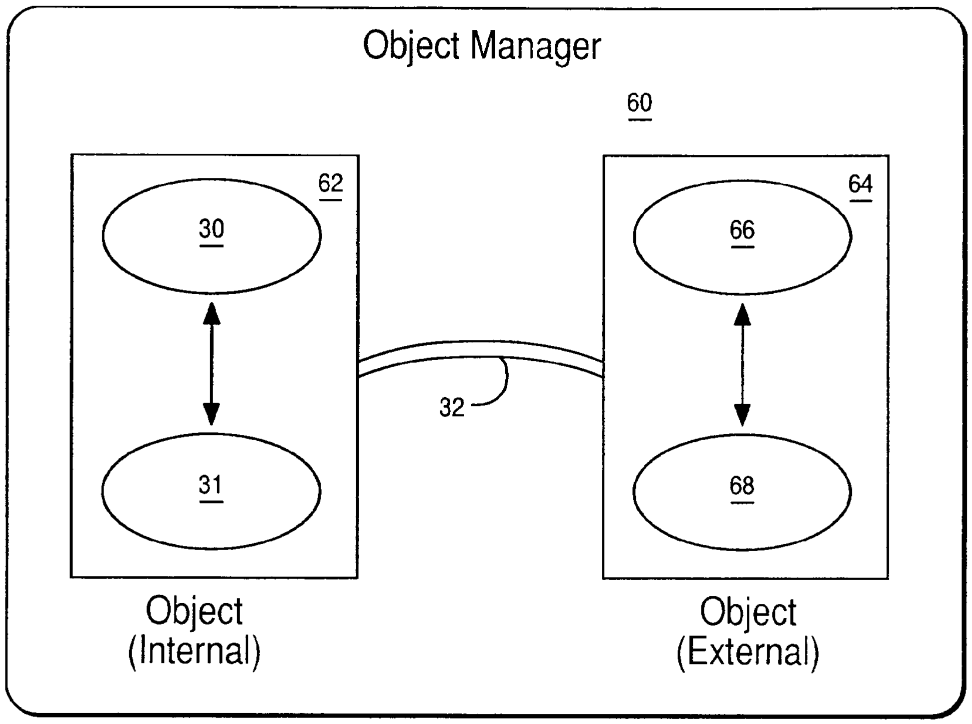

Object Manager
60
62
30
31
32
64
66
68
Object (Internal)
Object (External)

FIG. 12

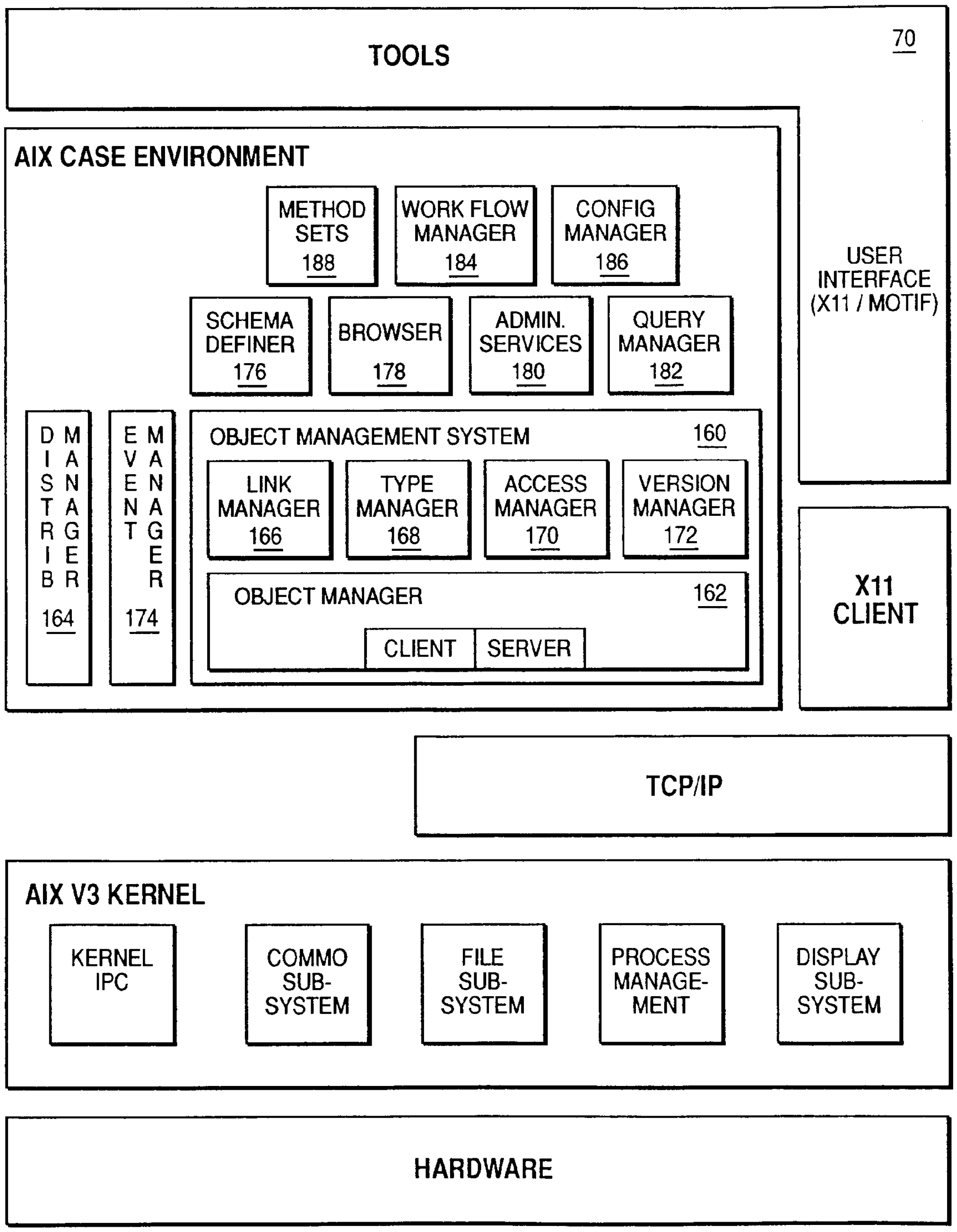

TOOLS
70
AIX CASE ENVIRONMENT
METHOD SETS 188
WORK FLOW MANAGER 184
CONFIG MANAGER 186
SCHEMA DEFINER 176
BROWSER 178
ADMIN. SERVICES 180
QUERY MANAGER 182
DISTRIB MANAGER 164
EVENT MANAGER 174
OBJECT MANAGEMENT SYSTEM 160
LINK MANAGER 166
TYPE MANAGER 168
ACCESS MANAGER 170
VERSION MANAGER 172
OBJECT MANAGER 162
CLIENT
SERVER
USER INTERFACE (X11 / MOTIF)
X11 CLIENT
TCP/IP
AIX V3 KERNEL
KERNEL IPC
COMMO SUB-SYSTEM
FILE SUB-SYSTEM
PROCESS MANAGE-MENT
DISPLAY SUB-SYSTEM
HARDWARE

FIG. 13

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 495 319 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **91312031.7**

(22) Date of filing : **24.12.91**

(51) Int. Cl.^{5} : **C07C 2/58,** C07C 9/16, B01J 8/20, B01J 27/12, C07C 2/60

The application is published incomplete as filed (Article 93 (2) EPC). The point in the description or the claim(s) at which the omission obviously occurs has been left blank (Page 40, claim 30).

A request for addition of the first line of claim 30 has been filed pursuant to Rule 88 EPC. A decision on the request will be taken during the proceedings before the Examining Division (Guidelines for Examination in the EPO, A-V, 2.2).

(30) Priority : **24.12.90 US 632478**

(43) Date of publication of application : **22.07.92 Bulletin 92/30**

(84) Designated Contracting States : **BE DE ES FR GB IT NL SE**

(71) Applicant : **CHEMICAL RESEARCH & LICENSING COMPANY**
10100 Bay Area Boulevard
Pasadena, Texas 77507 (US)

(72) Inventor : **Crossland, Clifford Stuart**
2323 Fairwind 869
Houston, Texas 77062 (US)
Inventor : **Pitt, Elliot George**
4017 Taffey Cresent
Mississaugh, Ontario L5L 2A6 (CA)
Inventor : **Johnson, Alan**
1496 Duncan Road
Oakville, Ontario L6J 2R3 (CA)
Inventor : **Woods, John**
8066 2nd Line North RR3
Campbellville, Ontario LOP 1BO (CA)

(74) Representative : **Cropp, John Anthony David et al**
MATHYS & SQUIRE 10 Fleet Street
London, EC4Y 1AY (GB)

(54) Process and apparatus for paraffin alkylation and catalyst for use therein.

(57) Paraffin alkylation using solid, particulate catalyst is carried out by preparing an alkane-catalyst mixture in a wash zone, passing the alkane-catalyst mixture to a plug flow reactor where a minor amount of olefin is introduced to contact the alkane-catalyst mixture and react to form alkylate and the alkane-catalyst-alkylate mixture is passed through the reactor with a minimum of back mixing to restrict the reaction of alkylate with olefin, thus substantially preventing polymerization. The alkane-catalyst-alkylate mixture, substantially free of olefin is passed to a disengaging zone where the liquid is removed and the solid particulate catalyst is recovered and returned to the wash zone for recycle. The alkane is present in the reactor in sufficient molar excess to react substantially all of the olefin. Any unreacted isoalkane is recycled to the reactor with make-up isoalkane added to maintain the molar excess. The preferred catalyst is an acid washed silica treated with antimony pentafluoride and more preferably treated with alkane at low temperature, e.g. -30 to -160°C.

Jouve, 18, rue Saint-Denis, 75001 PARIS

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to the alkylation of isoparaffins with olefins to yield hydrocarbons of enhanced octane number, the apparatus, catalyst and the method of preparing the catalyst.

Related Art

Isooctanes or trimethylpentanes (TMP) are among the most desirable components of motor alkylate gasoline and 2,2,4-trimethylpentane (isooctane) has long been the standard of measurement for the anti-knock properties of gasoline. The most common method of producing motor alkylate isooctane in commercial refineries is the alkylation of isobutane with butenes in the presence of a strong acid catalyst. Two acids currently used in alkylation plants are concentrated sulfuric acid and hydrofluoric acid. In these common processes the reactants are admixed with the acid in a contactor and the products separated from any unreacted reactants and the acid. The prior art in this area is well known. The drawbacks to the use of the sulfuric or hydrofluoric acid processes are readily apparent. Large quantities of the acids, which are highly corrosive, dangerous to handle and potentially a hazard to the environment, are required.

The search for safer particulate solid catalysts has been intense. Zeolites have been the most widely studied of the solid alkylation catalysts. For example, Kirsch, et al in U.S. Patents 3,665,813 and 3,706,814 disclose the use of such zeolites in "continuous" alkylation processes. European Patent 0174836 discloses sulfated zirconia as a solid superacid for paraffin isomerization and isoparaffin alkylation. US Pat. No.'s 4,056,578 and 4,180,695 disclose perfluoropolymersulfonic acid (PFPSA) as an alkylation catalyst. U.K. patent 1,389,237 discloses an antimony pentafluoride/acid on a carbon support as catalyst for alkylation. Other catalyst compositions which have been found to be initially active for alkylation include supported HF-antimony pentafluoride, (U.S. Patent 3,852,371); a Lewis Acid and Group VIII metal intercalated in graphite, (U.S. Patent 3,976,714); and a cation exchange resin complexed with BF_3 and HF, (U.S. Patent 3,879,489). U.S. Patent 4,918,255 describes a process for alkylating isoalkanes with olefins using a Lewis acid such as boron trifluoride, boron trichloride, antimony pentafluoride or aluminum trichloride deposited on inorganic oxide such as a wide pore zeolite, SiO_2 or Al_2O_3. Early work by Kirsch, et al, cited above using zeolites disclosed a catalyst life of about 10 grams of alkylate per gram of catalyst used. Further a method for increasing the life of zeolite catalysts using a regenerative process disclosed as in U.S. Patents 3,851,004 and 3,893,942 issued to Chang-Lee Yang, which disclose incorporating a Group VIII metal hydrogenation agent into the catalyst composition and regenerating the partially deactivated catalyst by periodic hydrogenation. A similar catalyst was used by Zabransky, et al, in a simulated moving bed reactor as disclosed in U.S. Patent 4,008,291.

Fenske et al. in U.S. Pat. No. 3,917,738 claims both oxidative and reductive regeneration techniques for zeolite catalysts. As described in this patent the olefins are adsorbed by the catalyst. A mixture of catalyst, isoalkane and olefin flows concurrently through an adsorption zone before the reactants and catalyst are introduced into the reaction zone. The controlled olefin adsorption was thought to prevent polymerization and improve catalyst life although this benefit was not quantified.

It is an advantage of the process of the present invention that the catalyst life is extended over that described in the art for solid paraffin alkylation catalysts. It is a feature of the present invention that the catalyst environment is controlled in a circulating bed reactor. It is a further feature of the present invention that the catalyst contact with olefin rich streams is minimized and contact with isoalkane is maximized. It is a further advantage of the present invention that back-mixing of the flow stream is limited. A further feature of the present invention is a catalyst which has the appropriate alkylation activity and fluidization properties for use in this process. These and other advantages and features will be seen in the following description.

SUMMARY OF THE INVENTION

This invention describes an alkylation process for producing high octane gasoline using a solid catalyst in a circulating solids reactor comprising a plugflow short contact time reaction zone, a solids disengaging zone, and a catalyst wash zone. An isoalkane-catalyst slurry is intimately mixed with an olefin rich stream at the inlet to the reaction zone then the mixture is rapidly moved through the reactor with a minimum of back mixing (to minimize secondary reactions and polymer formation). The reactor exits into a disengaging zone where the catalyst is separated from a major portion of the reactor effluent liquid. The catalyst and any associated residual alkylate then pass into a fluidized wash zone where the catalyst is washed with isoalkane to remove the residual alkylate. The isoalkane-catalyst slurry is then returned to the reactor. The reaction is preferably at least partially in the liquid phase. Suitable isoalkanes have 4-7 carbon atoms and suitable olefins include C_2 to C_5 olefins, e.g., ethylene, propylene, butenes, and pentenes or mixtures thereof.

The process may be used with any suitable solid catalyst having the appropriate alkylation activity and fluidization properties.

In addition, the rapid removal of catalyst from the reaction zone in a flowing liquid stream prevents excessive temperature excursions.

In another aspect the present invention relates to a catalyst for paraffin alkylations. Briefly the catalyst is an acid washed silica treated with antimony pentafluoride and preferably activated at low temperature with an alkane or isoalkane.

Another aspect of the present invention is the reactor in which the reaction is carried out.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a schematic representation of an upflow slurry reactor embodiment of the present invention.

Fig. 2 is a schematic representation of an alternative reactor embodiment of Fig. 1.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The scope of the present invention is not limited by any particular hypothetical mechanism. Isoalkanes comprise isobutane, isopentane, isohexanes and isoheptanes. The olefin comprises ethylene, propylene, butenes and pentenes. Mixtures of the various reactants within the ranges are not only contemplated, but are the usual condition in commercial streams.

In schematic Fig. 1 the three operational elements of the system as shown as reaction zone 36, disengaging zone 32 and wash zone 34. In practice these could be different zones of a single vessel with appropriate piping, screens and the like.

The liquid flow is depicted by the dashed lines 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 and 16 and the solid (catalyst) flow is depicted by the solid lines 20, 21, 22, 23, 24 and 25. The alkylation of isobutane with butenes is used to illustrate the invention.

The role of the isobutane wash is to prepare the catalyst for the alkylation reaction. In the present invention this preparation step may be carried out in the catalyst wash zone by purging the catalyst with isobutane prior to returning the catalyst to the reaction zone. In this wash zone the hydride transfer reaction is thought to occur:

$$R^+ + iC_4H_{10} \rightarrow t\text{-}C_4^+ + RH$$

Thus the catalyst wash zone serves a number of purposes including:

(a) increasing the proportion of t-butylcarbocations ($t\text{-}C_4^+$) at the catalyst active sites

(b) surrounding the active sites with isobutane rich fluid in the intraparticle and interparticle void space. To do this the reacted catalyst 23 is separated from the reactor effluent liquid 7 in the disengaging zone 32. As it is transported as stream 24 to the wash zone 34 it is washed in a countercurrent manner by the isobutane rich fluid stream 6 from the wash zone. The wash zone is typically operated as a fluid bed to provide efficient washing of the catalyst. When operating a fluid bed, the liquid superficial velocity can range from the minimum catalyst fluidization velocity to the catalyst free fall velocity. It is normally preferred to operate the wash zone above the minimum fluidization velocity and below the catalyst free fall velocity.

The residence time of the catalyst in the wash zone may be varied from about 5 seconds to about 1 hour but is preferably between 30 sec. and 5 minutes. It is desirable to minimize the wash time consistent with achieving the stated aims of the wash zone function.

The wash zone can be operated over a broad range of temperatures, for example, from -50°C to +100°C, preferably within the range -40°C to +50°C. The pressure in the wash zone may be extended over a wide range, for example, from atmospheric pressure to 1000 psig, but should always be sufficient to maintain the wash hydrocarbon as a liquid.

The wash fluid is typically a combination of the isobutane rich recycle stream 12 recovered from fractionation plus any make-up isobutane (stream 8) required to balance the consumption of isobutane in the reactor and any losses from the process.

Catalyst may be added (stream 20) and withdrawn (stream 25) from the wash zone both to control the catalyst inventory and the catalytic activity. The washed catalyst (stream 21) plus a portion of the associated wash fluid (stream 2) are withdrawn as a slurry from wash zone 34 and transferred to the reaction zone 36 where the slurry contacts and reacts with the olefin feed (stream 3). At the inlet to the reaction zone, the ratio of isobutane to olefin may be varied from about 2 to 1 to about 1000 to 1, preferably from 5 to 500 to 1. The desired ratio may be achieved by adding an isobutane rich stream to dilute the olefin feed stream 16 (either stream 13 or stream 14) prior to mixing with the catalyst slurry. The isobutane diluent for the olefin stream may be obtained directly from a portion of stream 5 (via stream 14) or stream 9 (via stream 13) or any mixture of these two streams. An external source of diluent (stream 15) may also be used either alone or in admixture with the above streams.

The catalyst slurry (streams 21/2) is uniformly dispersed into the feed stream 3 with the catalyst addition being controlled to provide sufficient active sites to react with all the olefin and maximize the production of trimethylpentanes. The amount of solid catalyst added will be dependent on the nature of the catalyst and reactor design but typically would be in the range from 1:100 to 1:1 volume of catalyst to volume of total liquid in the reactor and from 5:1 to 15:1 volume of catalyst to volume of olefin in the reactor. The reacting slurry (streams 2/4) is rapidly transported through the reaction zone with a minimum of back mixing to the disengaging zone 32.

Typical residence times in the reaction zone are from about 1 second to about 5 minutes or preferably from about 1 sec to 30 sec. The reactor can be operated over a broad range of temperatures, for example, from -50°C to 100°C, preferably within the range -40°C to +50°C. The pressure in the reaction vessel may be extended over a wide range, for example, from atmospheric pressure to 1000 psig, but should be sufficient to maintain at least a major portion of the hydrocarbon in the liquid phase. Within the scope of the invention a number of possible reactor configurations are envisaged, including an upflow reactor, a downflow reactor and a horizontal flow reactor. The movement of the reacting slurry through the reaction zone with the minimum of back mixing may be achieved by selecting the appropriate flow rates or by the use of mechanical devices such as an auger or a progressive cavity pump.

The disengaging zone 32 may be based on any device that rapidly separates the slurry (streams 23/7) into liquid stream 5 free of solids and a falling solid stream 24. Such devices include cyclones, or the like. The catalyst is immediately subjected to washing by the isobutane rich stream 6 as it is returned to the wash vessel. The reactor liquid effluent is fractionated in vessel 38 to yield alkylate products, stream 10, a sidedraw stream 11, an isobutane rich overhead stream 9. The temperature and pressure of the disengaging zone would typically be that of the reactor.

The process may be used with any suitable solid alkylation catalyst having the appropriate alkylation activity and fluidization properties. A number of catalysts and supports were tested and found useful for the present process and apparatus. A preferred catalyst comprises acid washed silica treated with antimony pentafluoride.

The silica is preferably a material having a surface area of about 5 m^2/g to about 250 m^2/g; pore volume of about 0.1 cc/g to about 4.0 cc/g; bulk density of 9-100 pounds/cu. ft. and particle size distribution in the range of 35-240 microns which has been acid washed, water washed and dried prior to treatment with antimony pentafluoride.

The acid wash preferably comprises a strong inorganic acid such as HCl, H_2SO_4 or H_3PO_4, however, relatively strong organic acids may be employed. The acid wash is conducted by contacting the support with an excess of acid from about 5 minutes to 16 hours or longer. After the acid is removed, the solid catalyst is washed with water to substantially remove the residual acid and dried for few minutes to several hours at 80 to 150°C then heated to between 160 and 650° C for several hours. The support may be prepared in an inert, reducing or oxidizing atmosphere.

Antimony pentafluoride as a liquid, a solution in an appropriate solvent, such as SO_2 or SO_2ClF, or as a vapor is contacted with the acid washed silica. The amount of antimony pentafluoride incorporated in the silica is from about 5 to 80% of the weight of the total of support and antimony pentafluoride.

This catalyst is preferably activated by treating it with an alkane (the term is used here to include both normal or isoalkanes) having 3 to 7 carbon atoms at a temperature in the range of -30°c to -160°C for a sufficient time to improve the activity of the catalyst over that of the untreated catalyst. A number of catalysts and supports were screened and found to be useful for the present process and apparatus, however it has been found that a silica support treated with SbF_5 produces an active catalyst with the required fluidization properties which produces alkylate at or in excess of present commercial sulphuric acid catalyst.

TYPICAL CATALYST PREPARATION

The following is a typical preparation of the preferred silica/SbF_5 catalyst.

Typical Silica Properties

United Catalyst Silica L3573

Surface Area	185 m^2/g
Bulk Density	16.3 lb/cu. ft.
pH	6.3

LOI (1000°C) 5 wt%

Preparation of Acid Washed Silica

250 g of silica were added to 1.5 L of 1N HCl with occasional stirring. The mixture was allowed to sit for 16 hours before filtering off the acid. The silica was washed with deionized water until the washings were neutral. The silica was heated in an oven at 90°C for 2 hours then at 120°C for 2 hours, and finally at 220°C for 2 hours.

The silica was then sieved and the 140-200 mesh (106-75 μ) material was stored in an oven at 220°C.

Preparation of Catalyst for Example 1

A sample of 140-200 mesh silica was removed from the oven and stored in a desiccator until cool. 0.61 g of silica was then transferred to a 30 cc Teflon vial. The silica was kept under dry nitrogen in a Glove Box as liquid SbF_5 (0.86 g) was added. The vial was capped and shaken for 20 minutes. The resulting free flowing catalyst was used in Examples 1 to 3.

The following three examples illustrate the benefits of the present invention over a fixed bed operation with and without recycle.

Example 1 (Comparative-Fixed Bed)

A solid silica (75-106μ) treated with antimony pentafluoride (as described above in the catalyst preparation) was packed into a 1/4" tubular reactor, which was cooled to -80°C then charged with isobutane. The temperature was increased to-10°C and a mixture of isobutane and butene-2 was charged to the reactor. The initial operating conditions are shown in Table I.

Table I

Fixed Bed Alkylation

Initial Operating Conditions

Catalyst, wt. g	1.39
SbF_5/SiO_2 Ratio, w/w	1.4
i-Butane Flow, ml/h	102
Butene-2 Flow, ml/h	3.5
Pressure, psig	150
Temperature, °C	-10
iC_4/olefin wt. ratio	30.2

The alkylation reaction was monitored by analyzing snap samples of the reactor effluent stream at 90 minute intervals using an on-line capillary gas chromatograph fitted with an automatic sample injection valve. After this the reactor effluent was partially vaporized across a back pressure regulator to produce an isobutane rich gas stream and an alkylate liquid. The liquid was collected in a receiver at ambient conditions. The receiver was periodically drained and the liquid weighed. The flow of the isobutane rich vapor was continuously measured using a wet test meter. The on-line analysis determines the concentration of all the major components in the reactor effluent from propane to 2,2,5-trimethylhexane as well as the remaining C_9+ components by carbon number. A summary of the analytical data along with data calculated from the analyses is presented in Table II. These include the research and motor octane numbers (RON and MON), the alkylate yield in terms of g of alkylate/g olefin charged, and the isobutane/olefin weight ratio. During the run the reactor temperature was raised to compensate for a decrease in catalyst activity. These changes together with the hours on-stream are also recorded in Table II. These data show that this catalyst is active for alkylation, although the quality of the alkylate is poor. The total weight of alkylate collected before catalyst deactivation was 18.9 g which corresponds to 23.3 g of alkylate/g SbF_5.

Table II

Fixed Bed Reactor Alkylation Results

Run No.: 347					
HRS ON-LINE	1.5	2.5	4.0	5.5	7.0
REACT. TEMP. C	-10	-10	-10	0	0
REACTOR EFFLUENT WT.% ANALYSIS					
C3	0.08	0.08	0.08	0.08	0.08
iC4	92.32	92.95	93.25	93.35	94.71
nC4	0.46	0.43	0.46	0.46	0.45
trans C4-	0.00	0.00	0.00	0.00	0.10
cis C4-	0.00	0.00	0.00	0.00	0.02
C5+ ALKYLATE	7.14	6.55	6.21	6.12	4.65
PRODUCT PROFILE	WT.%				
TMP	26.7	35.1	42.4	43.3	44.9
DMH	23.7	17.8	13.4	12.5	5.9
C5-C7	28.1	22.0	15.8	14.2	8.1
C9-C11	14.3	13.2	11.7	10.9	8.8
C12	4.3	7.4	9.7	10.9	17.2
C13+	2.9	4.5	7.0	8.2	15.2
RON	86.0	89.5	91.5	91.8	95.4
MON	85.6	88.5	90.3	90.6	93.3
ALKYLATE g/g	2.1	2.0	2.0	1.9	1.8

Example 2 (Comparative)

For this experiment, the fixed bed reactor of Example 1 was modified to return a portion of the effluent from the reactor outlet to the reactor inlet thus allowing operation as a recycle reactor. A further batch of alkylation catalyst of Example 1 was charged to the tubular reactor. Isobutane at -80°C was charged to the (cooled) reactor then the temperature was increased to-10C. An alkylation experiment was carried out with the initial conditions given in Table III.

Table III

Recycle Reactor Alkylation

Initial Operating Conditions

Catalyst, wt., g	1.37
SbF_5/SiO_2 Ratio, w/w	1.63
i-Butane Flow, ml/h	200
Butene-2 Flow, ml/h	5
Recycle Flow, ml/min	20
Pressure, psig	125
Temperature, °C	-10
iC_4/Olefin, wt. ratio	36.7

The experiment was monitored in a similar manner to Example 1 and the results for this experiment are summarized in Table IV.

Table IV

Recycle Reactor Alkylation Results

Run No.: 309

HRS ON-LINE	2	5	8	11	14	17
REACT. TEMP. C	-10	-10	-10	0	0	0
REACTOR EFFLUENT WT.% ANALYSIS						
C3	0.16	0.15	0.15	0.15	0.15	0.15
iC4	93.82	94.11	94.29	94.37	94.59	96.85
nC4	0.38	0.33	0.33	0.33	0.31	0.31
trans C4-	0.00	0.00	0.00	0.00	0.00	0.37
cis C4-	0.00	0.00	0.00	0.00	0.00	0.26
C5+ ALKYLATE						
PRODUCT PROFILE	WT.%					
TMP	43.7	57.5	62.5	63.8	68.6	50.9
DMH	21.3	11.2	6.5	5.7	4.9	3.7
C5-C7	21.9	14.9	10.4	10.5	10.9	9.3
C9-C11	9.0	7.9	7.1	6.8	5.8	12.1
C12	3.9	8.0	12.3	11.8	9.1	21.1
C13+	0.2	0.6	1.3	1.5	0.7	2.9
RON	89.0	93.9	96.4	96.7	97.4	96.6
MON	88.4	92.5	94.4	94.5	95.0	93.9
ALKYLATE g/g	2.1	2.1	2.0	2.0	2.0	1.9

As can be seen from the product profile there is an increase in the TMP concentration with on-stream time but there is also a proportionally greater increase in the C_{12}+ material concentration. To compensate for this the reactor temperature was increased after 11 hours on-stream. This caused a temporary decrease in the C_{12}+ concentration but eventually this heavy alkylate concentration started to increase and the catalyst was deactivated.

The amount of alkylate collected from this fixed bed recycle run was 78.5 g which corresponds to 92.4 g/g SbF_5. In comparing the results from Example 2 with that of Example 1 there is a significant improvement by using recycle.

Example 3

This example was carried out using a circulating bed reactor according to the present invention. Figure 2 shows the essentials of this upflow reactor unit. In this unit a portion of the reactor effluent is recycled to provide sufficient flow to transport the catalyst through the reactor. Initially the unit was filled with isobutane, then the catalyst as described in Example 1 after treatment with isobutane at -80°C was added via the disengaging zone 32 to the wash zone 34. The catalyst bed in the wash zone was fluidized with cooled isobutane wash fluid (stream 1) then the cooled recycle flow (stream 14) was adjusted to transport catalyst through the reactor to the disengaging zone thus establishing the appropriate catalyst circulation (streams 21, 22, 23 and 24) before the butene-2 feed, stream 3, was introduced to the unit via the recycle stream 14.

Table V gives the initial operating conditions while Table VI records the progress of the alkylation experiment.

Table V

Circulating Bed Alkylation

Initial Operating Conditions

Catalyst, wt. g	1.54
SbF_5/SiO_2 Ratio, w/w	1.52
i-Butane Wash Flow, ml/h	105
Butene-2 Flow, ml/h	3.3
Recycle Flow, ml/min	21.5
Pressure, psig	150
Reactor Inlet Temp., °C	-15
Disengager Outlet Temp., °C	4.8
iC_4/Olefin Wt. Ratio	29.2
Liq. Residence Time in Reactor, sec.	≈2.4
Cat. Residence Time in Reactor, sec.	≈3.2
Cat. Residence Time in Wash Zone, sec.	≈40

By comparing these results (Table VI) with those obtained from Example 2 (Table IV) it can be seen that the initial alkylate quality is much improved. There is a much higher proportion of TMP and much less C_{12} plus material, cracked products (C_5 to C_7 and C_9 to C_{11}) and isomerized products (DMH). This improvement is attributed to the nature of the circulating bed operation where the catalyst is continually rejuvenated by washing with isobutane then the isobutane rich catalyst is rapidly moved through the olefin reaction zone to allow TMP production but limit the production of other alkylate. As the alkylation reaction proceeds (Table VI) there is only a minor change in alkylate quality with on-stream time, with the C_{12}+ concentration being relatively constant for the initial 140 hours of operation.

Table VI

Circulating Bed Alkylation Results

Run No.: 2050-S

HRS ON-LINE	1	7	20	26	44	72	96	108	140	164	188	208
REACTOR Tin. C	-20		-15.0			-14.7	-15.0	-14.7	-14.9	-7.0	6.4	
REACTOR Tout. C			4.8			5.0	4.3	5.2	6.6	8.0	14.0	
REACTOR EFFLUENT WT.% ANALYSIS												
C3	0.15	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
iC4	95.03	93.46	93.29	93.55	93.25	93.52	93.39	93.34	93.62	93.74	92.99	97.04
nC4	0.36	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
trans C4-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.65
cis C4-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.38
C5+ ALKYLATE	4.46	6.53	6.71	6.45	6.75	6.48	6.61	6.66	6.38	6.26	6.99	1.93
PRODUCT PROFILE WT.%												
TMP	73.9	76.7	79.3	78.5	79.8	79.3	80.6	79.4	77.0	78.6	73.4	46.1
DMH	17.4	14.5	11.0	12.5	10.4	12.1	9.9	10.3	9.0	5.7	6.3	7.8
C5-C7	5.2	5.6	5.3	5.6	5.4	5.4	5.4	5.8	6.5	5.2	6.7	10.5
C9-C11	1.5	1.3	1.7	1.3	1.9	1.4	1.6	1.9	2.6	2.8	4.0	9.5
C12	1.9	1.9	2.6	2.1	2.5	1.7	2.4	2.6	4.8	7.6	9.4	21.3
C13+	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.2	4.8
RON	94.1	95.1	96.5	95.9	96.8	96.2	97.0	96.7	96.9	98.3	97.6	94.2
MON	93.1	93.9	94.9	94.5	95.1	94.7	95.4	95.1	95.2	96.1	95.3	92.0
ALKYLATE g/g	2.1	2.1	2.0	2.1	2.0	2.1	2.1	2.1	2.0	2.0	2.0	1.9

Product Quality Summary-Circulating Bed

		Average	Range
Calculated	RON	96.3	94.1-98.3
	MON	94.7	93.1-96.1

This experiment was terminated after 208 hours of operation due to catalyst deactivation. 506.3 g of liquid alkylate was collected which equates to 544.4 g alkylate/g SbF_5.

For this run, the butene-2 feed addition was accurately measured using a calibrated burette and this measurement was used to calculate a total alkylate production of 810 g or 871 g of alkylate/g SbF_5 (3.5 bbl/lb). The discrepancy between the alkylate produced and collected was shown to be caused by the loss of C_5+ material in the vaporized isobutane rich gas stream.

Thus it can be seen that the circulating bed reactor gives a much better catalyst life than either the fixed bed (Example 1) or the recycle reactor (Example 2). Furthermore, Tables II, IV and VI show that the alkylate quality obtained from the circulating bed reactor is far superior to either the fixed bed or recycle reactor.

Example 4

Downflow Operation

In one embodiment of the invention the reaction is carried out in the liquid phase with downflow through the reactor to limit back mixing. The solid particulate catalyst is slurried in the isobutane feed stream in the wash zone and the resultant slurry is pumped upward through a lift line. The slurry is then fed near the top of a downflow reactor into the feed zone. The butene containing stream is simultaneously fed to the reactor into or near the feed zone so as to contact the catalyst.

As the mixture passes through the reactor the solid catalyst is removed from further contact with the butene feed. The catalyst is then contacted with an excess of isobutane as it falls through the disengaging zone into the wash zone to facilitate alkylate removal from the catalyst surface and intra- and inter- particle voids.

The separated liquid product stream is fractionated to recover the alkylate product and the excess isobutane is recycled to the mixing zone. In the wash zone make-up isobutane may be added to replace that consumed by the reaction. Additionally fresh catalyst is added as necessary to replace any deactivated during the process or lost from the process.

To simulate the downflow mode of operation 11 grams of catalyst consisting of a carbon support coated with trifluoromethane sulfonic acid (CF_3SO_3H) and antimony pentafluoride (SbF_5) were loaded into a bench scale reactor. Isobutane and olefin feed (6.7% butene-2 in isobutane) flows were set at 120 and 60 ml/hr respectively and the catalyst was recycled about every 5 minutes. The isobutane was injected after the reactor to "wash" the alkylate product away from the catalyst. The liquid product was sampled and analyzed at intervals. these results are compared to a fixed bed experiment using another sample of the catalyst in Table VII, below.

TABLE VII.

	Downflow Alkylates		Fixed Bed Alkylates	
Component, wt%	Liquid 1	Liquid 2	Liquid 5	Liquid 8
TMP	50.78	50.18	29.35	31.72
C_{12}	14.80	16.56	19.33	18.35
Other Alkylate	27.59	24.02	19.41	28.94
C_{13}+	6.83	9.23	31.91	21.00

The downflow mode of operation favors TMP production when compared to the fixed bed.

Example 5

Upflow operation

In another embodiment the liquid phase reaction is carried out in upflow with the butene being injected into the lift line. With the bench scale "reactor" section acting as the disengagement segment and mixing zone, the lift line acted as the reactor. A four hour test was carried out operating in this mode using an identical catalyst to Example 4. The flow rates of olefin ($C_4^=$) and isobutane were set as in example 3 above. As olefin feed was injected, the C_6+ level, as recorded by the product on-stream analyzer, slowly increased to approximately 2%. Although there was no flow reading for the lift fluid, it is estimated that the olefin residence in the lift line was only seconds.

In this mode of operation on the bench scale unit the alkylate was circulated through the system with the isobutane, and the isobutane wash was not expected to be very effective in removing the alkylation product from the catalyst. Therefore a build up of dimethyl hexanes was expected. Four liquid samples were collected during the test run and all samples were good quality alkylate with the last three collected containing in excess of 60% TMP components which compares well with commercial units (see Table VIII below).

A second test of the upflow mode was made using a PFPSA (perfluoropolymersulfonic acid) support (catalyst 2, Table VIII) treated with SbF_5 as the circulating catalyst. The addition of the SbF_5 increased problems in circulating the catalyst and it was not possible to achieve smooth circulation with the treated catalyst. While the initial liquid was poor quality, two later samples contained over 60% TMP components with total C_{12}+ of only about 10%.

TABLE VIII
Analysis of Upflow Liquids

	Catalyst 1				Catalyst 2		
Component, wt%	Liq. 1	Liq. 2	Liq. 3	Liq. 4	Liq. 2	Liq. 5	Rec.
TMP	45.16	67.46	64.76	60.12	53.77	60.67	60.96
C_{12}	9.59	4.00	6.24	10.77	8.80	5.16	5.04
Other Alk.	21.79	22.27	25.35	24.82	21.78	28.68	28.70
$C_{13}+$	23.46	6.27	3.65	4.29	15.65	5.48	5.29
RON	87.9	91.6	91.3	91.2	88.9	87.9	89.3
MON	87.5	90.3	89.3	89.9	87.5	86.7	88.7

Example 6

Upflow With Mechanical Lift

A bench scale reactor was modified to provide a lifting screw conveyor or auger which substantially filled the inner diameter of the reactor. The auger was provided with a variable speed drive so that recirculation rates could be varied. A mixing or isobutane wash zone was provided along with a disengaging vessel so that product alkylate within the circulating liquid could be tested. Again, the alkylate product was not separated from the recirculating liquid, so a build up of dimethylhexanes could be expected if the isobutane wash was not effective.

The catalyst used was prepared by first depositing triflic acid (CF_3SO_3H) onto a carbon support by vapor deposition and then adding antimony pentafluoride (SbF_5) by vapor deposition. The final catalyst contained 12.88 g carbon, 0.69 g triflic acid and 5.73 g of antimony pentafluoride (SbF_5).

The reactor was purged with dry nitrogen during catalyst loading, and then pressured to 150 psig. The isobutane was introduced into the reactor to establish a liquid level with initial flow at 180 ml/hr which was reduced to a final flow of 60 ml/hr. The auger speed was set at 180 rpm and the pressure maintained at 150 psig. Olefin feed was set at 60 ml/hr. As the olefin was added the temperature increased from 21°C to about 29°C while the concentration of C_6+ and isopentane (iC_5) in the reactor effluent also increased.

The reactor effluent composition (see Liquid 1 in Table IX below) indicated significant cracking and isomerization. This was manifest in a low TMP/DMH ratio of 0.26 and significant C_5-C_7 components (11.8%). In addition there was a significant concentration of C_{12} and C_{13}+ components (12.1% and 9.8% respectively. This was attributed to inadequate contacting in the reactor or non-optimum active sites distribution on the catalyst surface.

At this point the auger speed was increased to approximately 400 rpm to increase the catalyst circulation. The C_6+, iC_5, nC_5, and nC_4 concentration in the reactor effluent and the reactor temperature continued to increase (35°C). Analysis of the resulting liquid (Liquid 2 in Table IX below) showed that isomerization and cracking activity had increased (TMP/DMH = 0.17, C_5-C_7 = 18%) while the production of C_{12} (5.4%) and C_{13}+ material (2.9%) had decreased.

A final test with the same catalyst loading was made to check the effect of high throughput. In addition, the reactor was placed in an ice bath to moderate cracking and isomerization reactions and improve temperature control.

The isobutane was started at 60 ml/hr and the unit flushed with isobutane. The olefin feed flow was then set at 240 ml/hr with the auger speed set at 180 rpm. The liquid recovered (Liquid 3, Table IX) contained a reasonable C_6+ to iC_5 concentration (12.2% and 0.4%) which suggested that the higher throughput and lower reactor temperature (0°C) reduced cracking and isomerization. The analysis of the liquid sample showed it to be commercial alkylate quality.

TABLE IX
Mechanical Lift Test Runs

Composition, wt %	Liq. 1	Liq. 2	Liq. 3
TMP	12.27	9.27	63.81
Trimethylhexane	1.36	2.23	1.89
C_5-C_7	11.80	18.02	5.95
DMH+MH	47.40	53.89	6.71
C_9-C_{11}	5.33	8.22	2.53
C_{12}	12.11	5.44	14.26
C_{13}+	9.75	2.93	4.85
RON (Calculation)	68.8	67.6	96.8
MON "	71.6	70.5	94.4

Example 7

A test run was made over a three day period to vary the reactor temperature and contact times. A new catalyst was prepared as above but had the following composition as loaded to the reactor: Carbon-11.17 g; triflic acid-0.75 g; and SbF_5-3.70.

Conditions and flow rates along with product analysis of the liquid products taken at various times in the run are reported in Table X, below. The conditions resulted in substantial improvement in the TMP/DMH ratios (as high as 7.15 in Liq. 10, Table X).

TABLE X

Mechanical Lift Test Runs

	Liq. 1	Liq. 2	Liq. 3	Liq. 4	Liq. 5	Liq. 6	Liq. 7	Liq. 8	Liq. 9	Liq. 10	Liq. 11
Temp.,°C	8.5	5.6	6.0	7.0	8.4	7.8	8.5	9.5	9.2	15.0	24.0
iC_4, ml/Min	60 --→										
Olefin, ml/Min	2 --→										
Auger Speed, RPM	180 ------------------------------------→									Slight Inc.	Reduced
Component, wt% Product Profile:											
TMP	35.55	24.41	29.12	53.16	62.32	65.07	66.01	62.16	62.64	61.67	57.51
TMH	2.49	3.87	4.62	4.68	3.06	2.49	2.23	1.63	3.13	2.30	1.24
C_5-C_7	6.85	14.40	6.80	6.78	4.98	5.34	3.69	3.26	2.92	2.68	3.08
DMH + MH	41.30	41.18	41.16	15.72	12.48	11.30	10.24	15.53	10.54	8.62	8.51
C_9-C_{11}	4.20	5.73	6.87	7.15	4.89	3.66	4.06	2.03	3.31	3.00	3.08
C_{12}	8.07	7.87	9.13	9.40	9.24	9.41	10.65	12.25	10.98	17.62	21.10
C_{13}^+	2.54	2.54	2.31	3.11	3.04	2.74	3.13	3.13	3.44	4.11	5.38
RON C_{12}^- portion	80.2	78.1	78.9	91.1	93.9	94.8	95.6	93.4	95.4	96.4	96.2
MON	81.3	79.5	80.2	90.0	92.5	93.3	93.9	92.2	93.5	94.1	93.4
TMP/DMH	.86	.59	.71	3.38	4.99	5.75	6.45	4.00	5.94	7.15	6.76

EXAMPLE 8

Using the upflow apparatus previously described the alkylation was carried out using a catalyst prepared as described in Example 1. The catalyst was transferred to the stainless steel hopper, cooled and isobutane added. The reactor was filled with isobutane and the catalyst was added from the hopper.

The catalyst was fluidized with isobutane wash and recycle and the bed height recorded for incipient circulation (see Table VII). The recycle rate was increased to give a bed dH (height differential) of 1 cm and butene-2 feed was charged to the unit. A summary of the reactor conditions are set out in TABLE XI and results in TABLE XII.

TABLE XI

Reaction Conditions: Summary

Catalyst	United Silica- 1.42 g	SbF_5 - 2.36 g
Feed	C.P. Grade Butene-2 (butadiene not detected)	

Initial Operating Conditions

i/o Ratio	40
Cooling Bath, temp., °C	-22
Reactor Inlet, temp., °C	-13
Reactor Outlet, temp., °C	11
Pressure, psi	120
iC_4 wash, ml/h	200
Olefin Feed, ml/h	5
Recycle Rate, ml/min	15
Initial Bed Ht., cm	14
Circulating Bed, Ht., cm	13

Product Yield	Total, g	g/g SbF_5	bbl/lb	mole/mole SbF_5
Alkylate Collected	870.9	369.0	1.5	702
Corrected For Vapor Loss	1320.8	559.6	2.28	1065

Product Quality	Average	Range
Calc. RON	95.2	88.5 - 99.0
Calc. MON	93.7	88.4 - 96.7

TABLE XII

TIME ON-STREAM, HRS.	156	180	204	210
COMPONENTS, WT.%				
C3	0.16	0.16	0.15	0.15
iC4	94.67	94.54	94.47	95.41
nC4	0.34	0.34	0.34	0.35
iC5	0.14	0.12	0.15	0.18
C6-C7	0.19	0.17	0.26	0.30
2,2,.4-TMP	2.37	2.39	2.08	1.22
2,2+2,4+2.5DMH	0.33	0.29	0.32	0.25
2,3,4-TMP	0.63	0.76	0.82	0.64
2.3.3-TMP	0.59	0.57	0.47	0.28
OTHER DMH	0.13	0.14	0.17	0.15
2,2,5-TMH	0.07	0.06	0.08	0.12
C9	0.05	0.05	0.07	0.13
C10-C11	0.06	0.06	0.11	0.17
C12	0.27	0.33	0.48	0.61
C13+	0.01	0.01	0.02	0.05
TOTAL (i-C5+)	4.83	4.97	5.03	4.09
PRODUCT PROFILE:				
TMP	77.79	78.78	70.34	54.85
DMH	5.81	4.99	6.25	7.18
C5-C7	6.85	5.84	8.15	11.58
C9-C11	3.74	3.43	5.26	10.34
C12	5.54	6.70	9.50	14.87
C13+	0.27	0.26	0.49	1.18
RON	97.99	98.47	97.20	95.28
MON	95.89	96.21	94.95	92.93
ALKYLATE g/g cat.	2.03	2.02	2.00	1.97
C8: OTHER ALKY.	5.1	5.2	3.3	1.6
DMH/TMP	0.07	0.06	0.09	0.13
OLEFIN ml/h	5.3	5.5	5.5	5.5
RECYCLE ml/min	30	27	30	30
BED ht cm	7.5	7.4	7	7.1
INLET T. deg. C	0	0 to 2	5 to 21	21
OUTLET T. deg. C	9	12	14 to 22	22
$C4^{=}$ CHARGED	490.4	570.9	651.4	671.6
$C4^{=}$/g SbF5	207.8	241.9	276.0	284.6

Example 9 (Catalyst Preparation)

As illustrated in Run (b) below acid washing of the silica significantly increases catalyst life compared to a non acid washed silica Run (a). Run (c) illustrates that the life of the acid washed silica/SbF_5 catalyst can be further greatly improved by initially contacting the catalyst with isobutane at a cool temperature typically between -160°C and -30°C.

Run (a) (without acid treatment/without low temperature alkane treatment)

Silica dried at 220°C was treated with SbF_5 (1 g SiO_2, 1.06 g SbF_5) by the general procedure then the mixture was packed into a 1/4" tubular reactor, which was charged with isobutane at 10°C. The alkylation activity of the catalyst was tested by charging a mixture of 6.74 wt% butene-2 in i-butane at 85 ml/h to the reactor. The equipment and procedure used to carry out the alkylation experiment is described in Example 2. No liquid alkylate was produced from this test.

Run (b) (Acid Treatment Without Low Temperature Alkane Treatment)

The silica was acid washed by contacting the silica with an excess of 1N aqueous HCl for 16 hours then washing the silica with deionized water until the wash effluent was neutral. The alkylation experiment as described in Run (a) was repeated using the acid washed silica dried at 220°C (1.02 g) treated with SbF_5 (1.16 g). The reactor was cooled to -22°C during the initial isobutane purge, then the temperature was raised to 10°C for the alkylation experiment. 40.85 g of alkylate was collected which corresponds to a catalyst life of 35.2 g alkylate/g SbF_5.

Run (c) (Acid Wash With Low Temperature Alkane Treatment)

The alkylation experiment described in Example (b) was repeated using acid washed silica dried at 220°C (1.04 g) treated with SbF_5 (1.05 g). The reactor was cooled to -78°C during the initial isobutane purge then the reactor temperature was raised to -10°C for the alkylation experiment. 209.7 g of alkylate was collected which corresponds to a catalyst life of 199.7 g alkylate/g SbF_5.

Claims

1. A process for the alkylation of alkane with olefin in the presence of a particulate solid acidic catalyst in a reactor, comprising first contacting said catalyst with alkane, feeding said catalyst-alkane mixture to a reactor, feeding an olefin to said reactor to contact said catalyst-alkane mixture to form an alkylate product and moving said catalyst-alkane-alkylate mixture through said reactor away from the olefine feed point to restrict additional contact of the catalyst-alkane-alkylate mixture with the olefin feed.

2. A process for the alkylation of alkane wherein the alkane is isoalkane, comprising:
(a) mixing a particulate solid acidic catalyst with C_4 to C_7 isoalkane feed stream in a mixing zone to form an isoalkane-catalyst mixture;
(b) feeding said isoalkane-catalyst mixture to a reactor in a feed zone;
(c) feeding C_2 to C_5 olefin to said reactor near said feed zone to allow said olefin to contact said isoalkane-catalyst mixture and reacting said olefin with said mixture to form alkylate in a reaction mixture;
(d) moving said reaction mixture through said reactor with a minimum of back mixing to restrict further contact of said alkylate-catalyst mixture with olefin;
(e) separating an alkylate stream from said catalyst in a disengaging zone; and
(f) recycling the catalyst to said mixing zone where make-up isoalkane is added.

3. The process according to claim 2 wherein the make-up is in excess of reacted isoalkane.

4. The process according to claim 2 or claim 3 wherein said reaction is carried out in at least partial liquid phase and said catalyst is slurried in said isoalkane in said mixing zone.

5. The process according to any preceding claim wherein the temperature in the reactor is in the range of -50°C to 100°C.

6. The process according to any preceding claim wherein the reaction temperature is in the range of -50°C to 80°C and the pressure within the reactor is sufficient to maintain said reaction mixture in the liquid phase.

7. The process according to any preceding claim wherein the alkane is selected from isobutane, isopentane or isohexane.

8. The process according to any preceding claim wherein said olefin is selected from butene, propylene, amylene or ethylene.

9. The process according to any preceding claim wherein said olefin comprises butene and said alkane com-

prises isobutane.

10. A process for the production of isooctane from the alkylation of isobutane with butene, comprising:
(a) mixing a particulate solid catalyst with a liquid isobutane feed stream in a mixing zone to form a slurry;
(b) feeding said slurry to a feed zone near the top of a vertical reactor;
(c) feeding liquid butene to said feed zone thereby contacting said butene with said isobutane-catalyst slurry and reacting said butene to form trimethylpentane in a reaction mixture;
(d) moving said reaction mixture downward through said reactor to prevent back mixing and further contact of said trimethylpentane-catalyst mixture with butene;
(e) separating said trimethylpentane from said catalyst in a disengaging zone; and
(f) recycling the catalyst to said mixing zone where make-up isobutane is added.

11. A process for the production of isooctane from the alkylation of butene with isobutane, comprising:
(a) mixing a particulate solid catalyst with a liquid isobutane feed stream in a mixing zone to form a slurry;
(b) feeding said slurry to a feed zone in the bottom of a reactor, said catalyst to be lifted upward in said reactor;
(c) feeding liquid butene to said feed zone thereby contacting said butene with said isobutane-catalyst slurry and reacting said butene with said isobutane to form trimethylpentane in a reaction mixture which is moving upward and away from said feed zone thus preventing further contact of said trimethylpentane-catalyst slurry with butene; and
(d) moving said reaction mixture upward through said reactor to prevent back mixing and further contact of said trimethylpentane-catalyst mixture with butene;
(e) separating said trimethylpentane from said catalyst in a disengaging zone; and
(f) recycling the catalyst and isobutane remaining in said reaction mixture to said mixing zone where make-up isobutane is added.

12. The process according to any one of Claims 1 to 11 wherein the alkane is isoalkane and the volume ratio of isoalkane to olefin is from about 2:1 to 1000:1.

13. The process according to any preceding claim wherein the volume ratio of catalyst to liquid in the reactor is in the range of about 1:100 to 1:1.

14. The process according to any preceding claim wherein the volume ratio of catalyst to olefin is in the range of about 5:1 to about 15:1.

15. An apparatus for conducting alkylation of alkanes with olefins comprising:
(a) a vertical reactor for contacting an alkylation catalyst slurried with the alkane for alkylation with an olefin,
(b) a wash/catalyst make-up vessel for washing the catalyst with fresh alkylation alkane to remove residual product and to slurry the catalyst, fluidly connected to the lower end of said vertical reactor, and having an entry for alkane wash and catalyst make-up,
(c) an olefin entry at the lower end of said vertical reactor to bring the alkylation olefin into contact with said alkane slurried catalyst,
(d) means to move said slurried catalyst through said vertical reactor,
(e) a disengaging vessel fluidly connected to said vertical reactor to receive the slurried catalyst in admixture wth alkane and alkylation product therefrom in fluid communication with said wash/catalyst make-up vessel to receive alkane wash therefrom and to remove slurried catalyst to said wash/catalyst make-up vessel, for contacting said slurried catalyst from said vertical reactor with the alkane wash and removing residual alkylation product from said alkane wash, and
(f) exit means to remove alkylation product from said disengaging vessel.

16. A solid particulate catalyst composition for paraffin alkylation comprising acid washed silica treated with antimony pentafluoride.

17. The catalyst composition according to Claim 16 wherein said composition is further treated with alkane having 3 to 7 carbon atoms.

18. A catalyst composition according to claim 16 or claim 17 wherein said silica has a surface area of about

5 m^2/g to about 250 m^2/g.

19. The catalyst composition according to any one of Claims 16 to 18 wherein said silica has a pore volume of about 0.1 cc/g to about 4.0 cc/g.

20. The catalyst composition according to any one of claims 16 to 19 wherein said silica has a particle size distribution in the range of 35-240 microns.

21. The catalyst composition according to any one of Claims 16 to 20 wherein said antimony pentafluoride comprises from 5 to 80% by weight of the composition.

22. A method of preparing a solid, particulate catalyst composition for paraffin alkylation comprising:
(a) washing particulate silica with a strong acid
(b) separating said particulate silica and acid
(c) washing said particulate silica with water
(d) heating said particulate silica and
(e) treating said particulate silica with antimony pentafluoride.

23. The method according to Claim 22 wherein said strong acid is inorganic.

24. The method according to claim 22 or claim 23 wherein said strong acid is HC1.

25. The method according to any one of claims 22 to 24 wherein said particulate silica is washed until neutral.

26. The method according to any one of Claims 22 to 25 wherein said heating is at a temperature in the range of 80 to 650°C.

27. The method according to any one of Claims 22 to 26 comprising
(f) contacting said antimony pentafluoride containing silica with C_3 to C_7 alkane at a temperature in the range of -30°C to -160°C.

28. The method according to any one of Claims 22 to 27 wherein said alkane comprising isoalkane.

29. The method according to Claim 28 wherein said isoalkane comprises isobutane.
heating comprises initially heating at a temperature in the range of 80 to 150°C for sufficient time to dry the particulate silica and thereafter heating at a temperature in the range of 160 to 650°C.

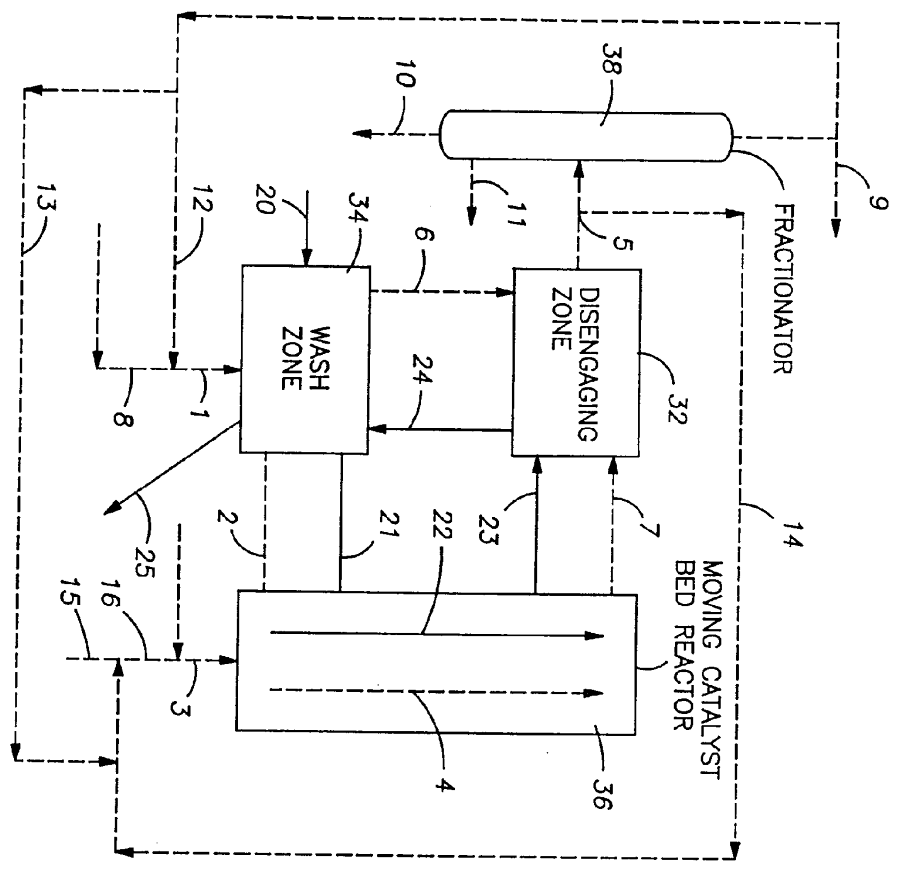

FRACTIONATOR
DISENGAGING ZONE
WASH ZONE
MOVING CATALYST BED REACTOR
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
20
21
22
23
24
25
32
34
36
38

FIG. 1

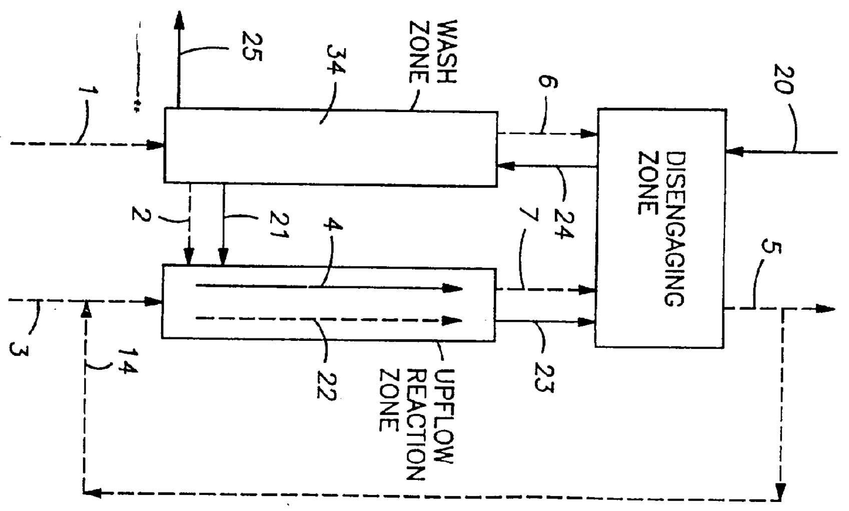

WASH ZONE
DISENGAGING ZONE
UPFLOW REACTION ZONE
1
2
3
4
5
6
7
14
20
21
22
23
24
25
34

FIG. 2

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 546 809 A2**

(12) **EUROPEAN PATENT APPLICATION**

(21) Application number : **92311207.2**

(22) Date of filing : **09.12.92**

(51) Int. Cl.^{5} : **G06F 9/44**

(30) Priority : **12.12.91 US 805779**

(43) Date of publication of application :
16.06.93 Bulletin 93/24

(84) Designated Contracting States :
CH DE FR GB IT LI NL SE

(71) Applicant : **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor : **Conner, Mike H.**
4416 Walhill Lane
Austin, Texas 78759 (US)
Inventor : **Martin, Andrew R.**
1070 Mearns Meadow No. 534
Austin, Texas 78758 (US)
Inventor : **Raper, Larry K.**
7860 Lakewood Drive
Austin, Texas 78750 (US)

(74) Representative : **Killgren, Neil Arthur**
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN (GB)

(54) **Computer with object oriented environment.**

(57) A method, system and program are provided for effectively managing class method names by collecting representations of all of the names and additional supporting information in a single data structure. Management is accomplished by the operation of an algorithm in the memory of a processor which employs two mechanisms. First, the class method procedure tables are initialized by class specific procedures. This allows applications to access the methods without requiring externalization of the method names. The information provided by the specific procedures is retained by the class object and is accessible via class methods whenever the information is required. Second, any additional supporting information for methods, in particular the offset in the method procedure table for each method, is recorded in a single externally named data structure. The combination of the two mechanisms eliminates the requirement of external names on a per method basis.

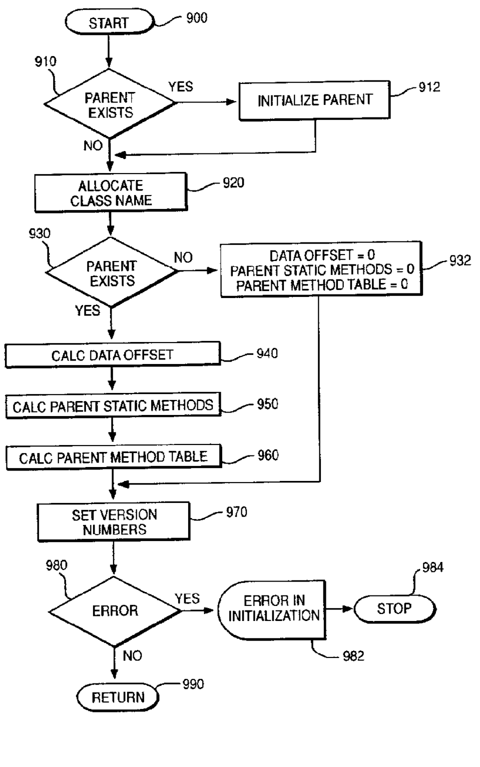


FIG. 9

EP 0 546 809 A2

Jouve, 18, rue Saint-Denis, 75001 PARIS

This present invention generally relates to improvements in object oriented computer systems.

Among developers of workstation software, object-oriented programming (or OOP) is increasingly recognized as an important new programming technology. It offers expanded opportunities for software reuse and extensibility, with improved programmer productivity when compared to conventional software development paradigms. Even so, object-oriented technology has not effectively penetrated major commercial software products to date. In particular, operating-systems have hesitated to embrace the new technology.

As with many new programming technologies, the early expressions of OOP concepts focused on the creation of new languages and toolkits, each designed to exploit some particular aspect. So-called **pure** object-oriented languages, such as Smalltalk, presume a complete run--time environment (sometimes known as a virtual machine) because their semantics represent a major departure from traditional procedurally oriented system architectures. Hybrid languages such as C++, on the other hand, require less run-time support but sometimes result in tight bindings between programs that provide objects and the client programs that use them. Tight binding between object-providing programs and their clients often require client programs to be recompiled whenever simple changes are made in the providing programs. Examples of such systems are found in US Patent 4,885,717; 4,953,080 and 4,989,132.

Because different languages and object-oriented toolkits emphasize different aspects of OOP, the utility of the resulting software is frequently limited in scope. A C++ programmer, for example, cannot easily use objects developed in Smalltalk, nor can a Smalltalk programmer make effective use of C++ objects. Objects and classes implemented in one language simply cannot be readily used from another. Unfortunately when this occurs one of the major benefits of OOP, the increased reuse of code, is severely curtailed. Object-oriented language and toolkit boundaries become, in effect, barriers to interoperability.

In OOP, as explained in more detail below, objects interact with external systems (eg other objects) in terms of methods. A method is a procedure or function that an object can perform - for example, if the object is a queue, one method would be to add a member to the queue, another is to delete a member from the queue. The external systems need only know the set of methods supported by a particular object, together with the required parameters - details of the actual implementation of the methods by the object are hidden within the object. Problems can arise however in the management of names of object methods, since in a system with many objects, the number of method names that must be externally available becomes very large.

Accordingly, the invention provides a computer having means for supporting an object oriented environment, including means for organizing a set of object classes having methods implemented by functions stored by the computer, comprising:

means for initializing a method procedure table for each class;
means for defining all functions statically for each class;
means for collecting offset variables into a data structure for each class; and
means for externalizing the data structure for each class in a computer memory.

Thus class method names are managed by collecting representations of all of the names and additional supporting information into a single data structure to improve manageability.

A preferred embodiment further comprises means for storing the method procedure table (eg on a disk drive or other appropriate storage medium), and means for initializing the offset variables at runtime. It is also preferred that the computer further comprises means for overriding the methods of the method procedure table, and means for uniquely differentiating between the methods of the method procedure table without requiring that the names of the methods be changed.

The invention also provides a method for organizing a set of object classes in a computer supporting an object oriented environment, the object classes having methods implemented by functions stored by the computer, comprising the steps of:

initializing a method procedure table for each class;
defining all functions statically for each class;
collecting offset variables into a data structure for each class; and
externalizing the data structure for each class in a computer memory.

Thus a method, system and program are provided for effectively managing class method names by collecting representations of all of the names and additional supporting information in a single data structure. Management is accomplished by the operation of an algorithm in the memory of a processor which employs two mechanisms. First, the class method procedure tables are initialized by class specific procedures. This allows applications to access the methods without requiring externalization of the method names. The information provided by the specific procedures is retained by the class object and is accessible via class methods whenever the information is required. Second, any additional supporting information for methods, in particular the offset in the method procedure table for each method, is recorded in a single externally named data structure. The combination of the two mechanisms eliminates the requirement of external names on a per method basis.

In other words an algorithm is used in the memory of a processor which employs two mechanisms. First, the class method procedure tables are initialized by class specific procedures produced by a System Object Model (SOM) compiler. These procedures are included in the file containing the method definitions. This allows the procedures to refer to the method names without requiring externalization of the method names The information provided by the specific procedures is retained by the class data structure and is accessible via class methods whenever the information is required. Second, any additional supporting information for methods, in particular the offset in the method procedure table for each method, is recorded in a single externally named data structure.

An embodiment of the invention will now be described by way of example with reference to the following drawings:

Figure **1** is a block diagram of a personal computer system;
Figure **2** is a drawing of a System Object Module (SOM) data structure;
Figure **3** is a drawing of a SOM class data structure;
Figure **4** is a flowchart depicting a language neutral object interface;
Figure **5** is a flowchart depicting a link, load and execution of an application using SOM objects;
Figure **6** is a flowchart depicting the creation of a new SOM class;
Figure **7** is a flowchart depicting the detailed construction of a new SOM class;
Figure **8** is a flowchart depicting the detailed construction of a new SOM generic class object;
Figure **9** is a flowchart depicting the detailed initialization of a new SOM class object;
Figure **10** is a flowchart depicting the detailed initialization of a SOM class data structure with offset values;
Figure **11** is a flowchart depicting the detailed parent class shadowing of a statically defined class hierarchies;
Figure **12** is a flow diagram depicting the redispatch method;
Figure **13** is a flowchart depicting the detailed initialization of the offset value in a SOM class data structure for a single public instance variable;
Figure **14** is a flowchart depicting the detailed control flow that occurs when a redispatch stub is employed to convert a static method call into a dynamic method call; and
Figure **15** is a flowchart depicting the detailed control flow that initialize a method procedure table for a class.

A representative system environment such as an IBM PS/2 computer and operating system is depicted in Figure **1**, which illustrates a typical hardware configuration of a workstation having a central processing unit **10**, such as a conventional microprocessor, and a number of other units interconnected via a system bus **12**. The workstation shown in Figure **1** includes a Random Access Memory (RAM) **14**, Read Only Memory (ROM) **16**, an I/O adapter **18** for connecting peripheral devices such as disk units **20** to the bus, a user interface adapter **22** for connecting a keyboard **24**, a mouse **26**, a speaker **28**, a microphone **32**, and/or other user interface devices such as a touch screen device (not shown) to the bus, a communication adapter **34** for connecting the workstation to a data processing network and a display adapter **36** for connecting the bus to a display device **38**. The workstation has resident thereon the OS/2 base operating system plus the computer software for implementing the functions described below which is included as a toolkit.

Object-Oriented Programming is quickly establishing itself as an important methodology in developing high quality, reusable code. The present invention has been implemented in a new system for developing class libraries and Object-Oriented programs called the System Object Model (SOM). A detailed description of object oriented programming, SOM, and a comparison to other object-oriented languages is provided to aid in understanding the invention.

INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

A new development in the software community is Object-Oriented Programming. Object-Oriented Programming Languages (OOPL) are being used throughout the industry, Object-Oriented Databases (OODB) are starting widespread interest even Object-Oriented Design and Analysis (OODA) tools are changing the way people design and model systems.

Object-Oriented Programming is best understood in contrast to its close cousin, Structured Programming. Both attempt to deal with the same basic issue, managing the complexity of ever more complex software systems. Structured Programming models a system as a layered set of functional modules. These modules are built up in a pyramid like fashion, each layer representing a higher level view of the system. Structured Programming models the system's behavior, but gives little guidance to modeling the system's information.

Object-Oriented Programming models a system as a set of cooperating objects. Like Structured Programming, it tries to manage the behavioral complexity of a system. Object-Oriented Programming, however, goes beyond Structured Programming in also trying to manage the informational complexity of a system.

Because Object-Oriented Programming models both the behavioral and informational complexity of a system, the system tends to be much better organized than if it was simply well "structured". Because Object-Oriented systems are better organized, they are easier to understand, debug, maintain, and evolve. Well organized systems also lend themselves to code reuse.

Object-Oriented Programming envisions the dual issues of managing informational and behavioral complexity as being closely related. Its basic unit of organization is the object. Objects have some associated data, which are referred to as an object's state, and a set of operations, which are referred to as an object's methods. A method is implemented by a subroutine. A class is a general description of an object, which defines the data representative of an object's state, and the methods for supporting the object.

OBJECT-ORIENTED PROGRAMMING IN C

Before examining SOM, consider Object-Oriented Programming in C; this will lead us naturally into the SOM philosophy. Consider a data structure definition containing information related to a generic stack. The data structure encompasses a series of functions designed to operate on a stack structure. Given a basic stack definition, multiple instances of this data structure may be declared within our program.

A generic stack definition, in C, appears below:

```
struct stackType {
        void *stackArray[STACK_SIZE];
        int stackTop;
};
typedef struct stackType Stack;
```

A definition of a generic stack function appears next:

```
Stack *create();        /* malloc and initialize a new stack. */
void *pop(        /* Pop element off stack. */
        Stack *thisStack);
void push(        /* Push onto stack. */
        Stack *thisStack,
        void *nextElement);
```

Most C programmers can imagine how such functions would be written. The <push()> function, for example, appears below.

```
void push(Stack *thisStack, void *nextElement)
{
        thisStack->stackArray[thisStack->stackTop] = nextElement;
        thisStack->stackTop++;
}
```

A client program might use this stack to, say, create a stack of words needing interpretation:

```
main()
{
        Stack *wordStack;
        char *subject = "Emily";
        char *verb = "eats";
        char *object = "ice cream";
        char *nextWord;
        wordStack = create();
        push(wordStack, object);
        push(wordStack, verb);
        push(wordStack, subject);
        /* ... */
        while (nextWord = pop(wordStack)) {
                printf("%s\n", nextWord);
                /* ... */
        }
}
```

This example can be used to review Object-Oriented Programming. A class is a definition of an object. The definition includes the data elements of the object and the methods it supports. A <stack> is an example of a class. A stack contains two data elements (<stackArray> and <stackTop>), and supports three methods, <create()>, <push()>, and <pop()>. A method is like a function, but is designed to operate on an object of a particular class. An object is a specific instance, or instantiation, of a class. The object <wordStack> is an object of class <Stach>, or <wordStack> is an instance of a stack.

Every method requires a specific object on which it is to operate. This object is called a target object, or sometimes a receiving object. Notice that each method (except <create()>) takes as its first parameter a pointer to the target object. This is because a program may have many objects of a given class, and each are potential targets for a class method.

There are three important advantages of this type of organization First, generic concepts are developed which can be reused in other situations in which similar concepts are appropriate. Second, self-contained code is developed, which can be fully tested before it is folded into our program. Third, encapsulated code is developed in which the internal details are hidden and of no interest to the client. A client <main()> program need know nothing about the <Stack> class other than its name, the methods it supports, and the interfaces to these methods.

COMPARISON TO C++

Another beneficial comparison is between SOM and the most widespread Object-Oriented programming language, C++. SOM has many similarities to C++. Both support class definitions, inheritance, and overridden methods (called virtual methods in C++). Both support the notion of encapsulation. But whereas C++ is designed to support stand-alone programming efforts, SOM is focused on the support of commercial quality class libraries. Most of the differences between SOM and C++ hinge on this issue. C++ class libraries are version dependent, while SOM class libraries are version independent. When a new C++ class library is released, client code has to be fully recompiled, even if the changes are unrelated to public interfaces.

C++ supports programming in only one language, C++. SOM is designed to support many languages. Rather than a language, SOM is a system for defining, manipulating, and releasing class libraries. SOM is used to define classes and methods, but it is left up to the implementor to choose a language for implementing methods without having to learn a new language syntax.

C++ provides minimal support for implementation hiding, or encapsulation. C++ class definitions, which must be released to clients, typically include declarations for the private data and methods. In SOM, the client never has to focus on these implementation details. The client need see only the <.sc> files, which contains only public information. C++ also provides a limited method resolution function. SOM offers several alternatives, such as offset method resolution, name lookup resolution, and dispatch resolution.

One other interesting difference between SOM and C++ is in its notion of class. In C++, the class declaration is very similar to a structure declaration. It is a compile-time package with no characteristics that have significance at runtime. In SOM, the class of an object is an object. The class object is itself an instantiation of another class, called the metaclass. The class object supports a host of useful methods which have no direct parallels in C++, such as <somGetName()>, <somGetParent()>, and <somFindMethod()>.

INTRODUCTION TO SOM

OS/2 2.0 includes a language-neutral Object-Oriented programming mechanism called SOM (for System Object Model). Although it is possible to write Object-Oriented programs in traditional languages, such as the stack example, SOM is specifically designed to support the new paradigm and to be used with both procedural (or non Object-Oriented) languages and Object-Oriented languages.

An important requirement of Object-Oriented programming is code reusability. Typically, code reusability is achieved through the use of class libraries. Today's library technology is limited in that class libraries are always language specific. A C++ library cannot be used by a Smalltalk programmer and a Smalltalk programmer cannot utilize a C++ library. Clearly it is necessary to create a language-neutral object model, which can be used to create class libraries usable from any programming language, procedural or Object-Oriented.

SOM introduces three important features lacking in most procedural languages. These are encapsulation, inheritance, and polymorphism (referred to here as "override resolution"). Inheritance refers to a technique of specifying the shape and behavior of a class (called a subclass) as incremental differences from another class (called the parent class or superclass).

Encapsulation refers to hiding implementation details from clients. This protects clients from making changes in an implementation which could adversely affect the system. For example, in the stack example there was no protection afforded to the C code. Although clients did not need to know the internal data structures of the stack, there was no way to prevent clients from looking at such implementation details. We could discourage, but not prevent, clients from writing code which used, and possibly corrupted, internal stack data elements.

Inheritance, or class derivation, is a specific technique for developing new classes from existing classes. This capability provides for the creation of new classes which are more specialized versions of existing classes. For example, we could create a <DebuggableStack>, which is like a <Stack> class, but supports further debugging methods, such as <peek()> for looking at the top value and <dump()> for printing a complete listing of the stack.

Inheritance also provides code consolidation. So, for example, a class defining <GraduateStudent> and <UnderGraduateStudent>, can be consolidated into a third class, <Student>. We then define <GraduateStudent> and <UnderGraduate> as more specialized classes, both derived from the common parent <Student>.

Inheritance introduces some additional semantics. A specialized class is said to be derived from a more generalized class. The general class is called the parent class, or sometimes, the base class. The specialized class is called the child class, or sometimes, the derived class. A child class is said to inherit the characteristics of its parent class, meaning that any methods defined for a parent are automatically defined for a child. Thus, because <GraduateStudent> and <UnderGraduateStudent> are both derived from <Student>, they both automatically acquire any methods declared in their common parent.

Override resolution refers to invoked methods being resolved based not only on the name of the method, but also on a class place within a class hierarchy. This allows us to redefine methods as we derive classes. We might define a <printStudentInfo()> method for <Student> and then override, or redefine, the method in both <UnderGraduateStudent>, and <GraduateStudent>. Override resolution resolves based on the type of the target object. If the target object type is a <Student>, the <Student> version of <printStudentInfo()> is invoked. If the target object type is a <GraduateStudent>, the <GraduateStudent> version of <printStudentInfo()> is invoked.

DEFINING CLASSES IN SOM

The process of creating class libraries in SOM is a three step process. The class designer defines the class interface, implements the class methods, and finally loads the resulting object code into a class library. Clients either use these classes directly, make modifications to suit their specific purposes, or add entirely new classes of their own.

In SOM, a class is defined by creating a class definition file. The class definition file is named with an extension of "csc". In its most basic form, the class definition file is divided into the following sections:

1. Include section
   This section declares files which need to be included, much like the C <#include> directive.
2. Class name and options
   This section defines the name of the class and declares various options.
3. Parent information
   This defines the parent, or base, class for this class. All classes must have a parent. If a class is not derived from any existing classes, then it's parent will be the SOM defined class <SOMObject>, the class information of which is in the file <somobj.sc>.
4. Data Section
   This section declares any data elements contained by objects of this class. By default, data can be accessed only by methods of the class.
5. Methods Section
   This section declares methods to which objects of this class can respond. By default, all methods declared in this section are available to any class client. The class definition file, <student.csc>, describes a non-derived <Student> class, and is set forth below.

Class Definition File: <student.csc>

```
include <somobj.sc>
class:
     Student;
parent:
     SOMObject;
data:
     char id[16];          /* student id*/
     char name[32];          /* student name */
methods:
     void setUpStudent(char *id, char *name);
```

```
        -- sets up a new student.
        void printStudentInfo();
        -- prints the student information.
        char *getStudentType();
        -- returns the student type.
        char *getStudentId();
        -- returns the student id.
```

How to Write a Method

Class methods are implemented in the class method implementation file. Each method defined in the method section of the class definition file needs to be implemented. They can be implemented in any language that offers SOM support. C is used for an exemplary language throughout the specification. However, one of ordinary skill in the art will realize that any programming language can be substituted. The student class method implementation file, <student.c>, is set forth below.

Class Method Implementation File: <student.c>

```
#define Student_Class_Source
#include "student.ih"
static void setUpStudent(
        Student *somSelf, char *id, char *name)
{
        StudentData *somThis = StudentGetData(somSelf);
        strcpy(_id, id);
        strcpy(_name, name);
}
static void printStudentInfo(Student *somSelf)
{
        StudentData *somThis = StudentGetData(somSelf);
        printf(" Id : %s \n", _id);
        printf(" Name : %s \n", _name);
        printf(" Type : %s \n", _getStudentType(somSelf));
}
static char *getStudentType(Student *somSelf)
{
        StudentData *somThis = StudentGetData(somSelf);
        static char *type = "student";
        return (type);
}
static char *getStudentId(Student *somSelf)
{
        StudentData *somThis = StudentGetData(somSelf);
        return (_id);
}
```

Notice that the method code appears similar to standard C. First, each method takes, as its first parameter, a pointer (<somSelf>) to the target object. This parameter is implicit in the class definition file, but is made explicit in the method implementation. Second, each method starts with a line setting an internal variable named <somThis>, which is used by macros defined within the SOM header file. Third, names of data elements of the target object are preceded by an underscore character "_" The underscored name represents a C language macro defined in the class header file. Fourth, methods are invoked by placing an underscore "_" in front of the method name. This underscored name represents a macro for message resolution and shields a programmer from having to understand the details of this process.

The first parameter of every method is always a pointer to the target object. This is illustrated below in the method <printStudentInfo()> which invokes the method <getStudentType()> on its target object.

SOM compiler generated <student.c>

```
#define Student_Class_Source
#include "student.ih"
static void setUpStudent(
        Student *somSelf, char *id, char *name)
```

```
{
      StudentData *somThis = StudentGetData(somSelf);
}
static void printStudentInfo(Student *somSelf)
{
      StudentData *somThis = StudentGetData(somSelf);
}
/* ...and so on for the other methods. */
```

MECHANICS OF USING SOM

There are a set of files involved with each class which are discussed below. The files have different extensions, but all have the same filename as the class definition file, <Student> in our example. These files are described below.

Student Class Files

<student.csc> - This is the class definition file, as described earlier.
<student.sc> - This is a subset of the class definition file. It includes all information from the **<.csc>** file which is public, including comments on public elements. For the student example, **<student.sc>** would include everything from **<student.csc>** except the data section. This file is created by the SOM compiler.
<student.h> - This is a valid C header file which contains macros necessary to invoke public methods and access public data elements of the class. This file will be included in any client of the class, and is created by the SOM compiler.
<student.ih> - Similar to **<student.h>**, but it contains additional information needed for implementing methods. This is the implementor's version of the **<.h>** file, and must be included in the class methods implementation file. This file is created by the SOM compiler and should not be edited
<student.c> - Contains the method implementations. Initially created by the SOM compiler and then updated by the class implementor.

BUILDING SOM CLASSES FROM OTHER CLASSES

There are two ways to use classes as building blocks for other classes. These are derivation (or inheritance) and construction.

DERIVATION

In this example, **<GraduateStudent>** is derived from **<Student>**, its base, or parent class. A derived class automatically picks up all of the characteristics of the base class. A derived class can add new functionality through the definition and implementation of new methods. A derived class can also redefine methods of its base class, a process called overriding. For example **<GraduateStudent>** adds **<setUpGranduateStudent()>** to those methods it inherits from **<Student>**. It overrides two other inherited methods, **<printStudentInfo()>** and **<getStudentType()>**. It inherits without change **<setUpStudent()>** and **<getStudentId()>** from the **<Student>** base class.
The class definition file for **<GraduateStudent>**, **<graduate.csc>**, is set forth below.
Class Definition File: <graduate.csc>

```
include <student.sc>
class:
      GraduateStudent;
parent:
      Student;
data:
      char thesis[128];        /* thesis title */
      char degree[16];         /* graduate degree type */
methods:
      override printStudentInfo;
      override getStudentType;
      void setUpGraduateStudent(
                char *id, char *name, char *thesis, char
        *degree);
```

The method implementation file, <graduate.c>, is shown below.

Class Method Implementation File: <graduate.c>

```
#define GraduateStudent_Class_Source
#include "graduate.ih"
static void printStudentInfo(GraduateStudent *somSelf)
{
        GraduateStudentData *somThis =
GraduateStudentGetData(somSelf);
        parent_printStudentInfo(somSelf);
        printf(" Thesis : %s \n", _thesis);
        printf(" Degree : %s \n", _degree);
}
static char *getStudentType(GraduateStudent *somSelf)
{
        static char *type = "Graduate";
        return (type);
}
static void setUpGraduateStudent(
        GraduateStudent *somSelf, char *id, char *name,
        char *thesis, char *degree)
{
        GraduateStudentData *somThis =
GraduateStudentGetData(somSelf);
        _setUpStudent(somSelf,id,name);
        strcpy(_thesis, thesis);
        strcpy(_degree, degree);
}
```

Often an overridden method will need to invoke the original method of its parent. For example, the <printStudentInfo()> for <GraduateStudent> first invokes the <Student> version of <printStudentInfo()> before printing out the <GraduateStudent> specific information. The syntax for this is "<parent_MethodName>", as can be seen in the <printStudentInfo()> method.

A given base class can be used for more than one derivation. The class, <UnderGraduateStudent>, is also derived from <Student>. The class definition file, <undgrad.csc>, is set forth below.

Class Definition File: <undgrad.csc>

```
include <student.sc>
class:
        UnderGraduateStudent;
parent:
        Student;
data:
        char date[16];          /* graduation date */
methods:
        override printStudentInfo;
        override getStudentType;
        void setUpUnderGraduateStudent(
                char *id, char *name, char *date);
```

The method implementation file, <undgrad.c>, is set forth below.

Class Method Implementation File: <undgrad.c>

```
#define UnderGraduateStudent_Class_Source
#include "undgrad.ih"
static void printStudentInfo(
        UnderGraduateStudent *somSelf)
{
        UnderGraduateStudentData *somThis =
                UnderGraduateStudentGetData(somSelf);
        parent_printStudentInfo(somSelf);
        printf(" Grad Date : %s \n", _date);
}
static char *getStudentType(UnderGraduateStudent *somSelf)
{
        static char *type = "UnderGraduate";
        return (type);
}
static void setUpUnderGraduateStudent(
        UnderGraduateStudent *somSelf,char *id, char *name, char *date)
{
        UnderGraduateStudentData *somThis =
                UnderGraduateStudentGetData(somSelf);
        _setUpStudent(somSelf,id,name);
        strcpy(_date, date);
}
```

The second technique for building classes is construction. Construction refers to a class using another class, but not through inheritance. A good example of construction is the class <Course> which includes an array of pointers to <Student>s. Each pointer contains the address of a particular student taking the course. <Course> is constructed from <Student>. The class definition file for <Course>, <course.csc>, is shown below.

Class Definition File: <course.csc>

```
include <somobj.sc>
class:
        Course;
parent:
        SOMObject;



  data:
    char     code[8];          /* course code               number */
    char     title[32];        /* course title */
    char     instructor[32];   /* instructor                teaching */
    int      credit;             /* number of credits
                                                   */
    int      capacity;           /* maximum number of              seats */
    Student  *studentList[20]; /* enrolled student            list */
    int      enrollment;         /* number of                enrolled students
                   */


methods:
        override somInit;
        void setUpCourse(char *code, char *title,
                char *instructor, int credit, int capacity);
        -- sets up a new course.
        int addStudent(Student *student);
        -- enrolls a student to the course.
        void dropStudent(char *studentId);
        -- drops the student from the course.
        void printCourseInfo();
        -- prints course information.
```

Often classes will want to take special steps to initialize their instance data. An instance of <Course> must at least initialize the <enrollment> data element, to ensure the array index starts in a valid state. The method <somInit()> is always called when a new object is created. This method is inherited from <SOMObject>, and can be overridden when object initialization is desired.

This example brings up an interesting characteristic of inheritance, the "is-a" relationship between derived and base classes. Any derived class can be considered as a base class. We say that a derived class "is-a" base class. In the previous example, any <GraduateStudent> "is-a" <Student>, and can be used anyplace we are expecting a <Student>. The converse is not true. A base class is not a derived class. A <Student> can not be treated unconditionally as a <GraduateStudent>. Thus, elements of the array <studentList> can point to either <Student>s, a <GraduateStudent>s, or a <UnderGraduateStudent>s.

The method implementation file for <Course>, <course.c>, is set forth below.

Class Method Implementation File: <course.c>

```
#define Course_Class_Source
#include <student.h>
#include "course.ih"
static void somInit(Course *somSelf)
{
        CourseData *somThis = CourseGetData(somSelf);
        parent_somInit(somSelf);
        _code[0] = _title[0] = _instructor[0] = 0;
        _credit = _capacity = _enrollment = 0;
}
static void setUpCourse(Course *somSelf, char *code,
        char *title, char *instructor, int credit, int capacity)
{
        CourseData *somThis = CourseGetData(somSelf);
        strcpy(_code, code);
        strcpy(_title, title);
        strcpy(_instructor, instructor);
        _credit = credit;
        _capacity = capacity;
}
static int addStudent(Course *somSelf, Student *student)
{
        CourseData *somThis = CourseGetData(somSelf);
        if(_enrollment >= _capacity) return(-1);
        _studentList[ _enrollment++] = student;
        return(0);
}
static void dropStudent(Course *somSelf, char *studentId)
{
        int i;
        CourseData *somThis = CourseGetData(somSelf);
        for(i=0; i<_enrollment; i++)
                if(!strcmp(studentId, _getStudentId(_studentList[i]))) {
                        _enrollment--;
                        for(i; i<_enrollment; i++)
                                _studentList[i] = _studentList[i+1];
                        return;
                }
}
static void printCourseInfo(Course *somSelf)
{
        int i;
        CourseData *somThis = CourseGetData(somSelf);
        printf(" %s %s \n", _code, _title);
        printf(" Instructor Name : %s \n", _instructor);
        printf(" Credit = %d, Capacity = %d, Enrollment = %d \n\n",
                        _credit, _capacity, _enrollment);
        printf(" STUDENT LIST: \n\n");
        for(i=0; i<_enrollment; i++) {
            _printStudentInfo(_studentList[i]);
            printf("\n");
        }
}
```

Notice in particular the method <printCourseInfo()>. This method goes through the array <studentList> invoking the method <printStudentInfo()> on each student. This method is defined for <Student>, and then overridden by both <GraduateStudent> and <UnderGraduateStudent>. Since the array element can point to any of these three classes, it is impossible at compile time to determine what the actual type of the target object is, only that the target object is either a <Student> or some type derived from <Student>. Since each of these classes defines a different <printStudentInfo()> method, it is impossible to determine which of these methods will be invoked with each pass of the loop. This is all under the control of override resolution.

THE SOM CLIENT

To understand how a client might make use of these four classes in a program, an example is presented below in the file <main.c.> The example illuminates object instantiation and creation in SOM, and how methods are invoked.

SOM client code: <main.c>

```
#include <student.h>
#include <course.h>
#include <graduate.h>
#include <undgrad.h>
main()
{
     Course *course = CourseNew();
     GraduateStudent *jane = GraduateStudentNew();
     UnderGraduateStudent *mark = UnderGraduateStudentNew();
     _setUpCourse(course, "303", "Compilers ",
          "Dr. David Johnson", 3, 15);
     _setUpGraduateStudent(jane,"423538","Jane Brown",
          "Code Optimization","Ph.D.");
     _setUpUnderGraduateStudent(mark,"399542",
          "Mark Smith", "12/17/92");
     _addStudent(course, jane);
     _addStudent(course, mark);
     _printCourseInfo(course);
}
```

A class is instantiated with the method <classNameNew()>, which is automatically defined by SOM for each recognized class. Methods are invoked by placing an underscore "_" in front of the method name. The first parameter is the target object. The remaining parameters illuminate additional information required by the method. When run, the client program gives the output shown below.

Client Program Output

```
     303 Compilers
     Instructor Name : Dr. David Johnson
     Credit = 3, Capacity = 15, Enrollment = 2

STUDENT LIST:

Id          : 423538
Name        : Jane Brown
Type        : Graduate
Thesis      : Code Optimization
Degree      : Ph.D.
Id          : 399542
Name        : Mark Smith
Type        : UnderGraduate
Grad Date   : 12/17/92
```

The client program output illustrates the override resolution at work in the different styles of displaying

<UnderGraduate>s and <GraduateStudent>s. A <Course> thinks of itself as containing an array of <Student>s, and knows that any <Student> responds to a <printStudentInfo()> method. But the <printStudentInfo()> method that an <UnderGraduate> responds to is different than the <printStudentInfo()> method that a <GraduateStudent> responds to, and the two methods give different outputs.

SOM Object Model

Figure **2** is a drawing of a basic SOM data structure. Label **210** is a state data structure for a particular object. The first full word at label **220** contains the address of the object's method procedure table label **240.** The rest of the state data structure set forth at label **230** contains additional information pertaining to the object. The method procedure table set forth at label **240** contains the address of the class object data structure **245** and addresses of various methods for the particular object **250** and **260.** The address at **245** points to the class object data structure **248.** All objects that are of the same class as this object also contain an address that points to this method procedure table diagrammed at label **240.** Any methods inherited by the objects will have their method procedure addresses at the same offset in memory as they appear in the method procedure table as set forth at label **240** of the ancestor class from which it is inherited.

Addresses of the blocks of computer memory containing the series of instructions for two of the method procedures are set forth at labels **250** and **260.** Labels **270** and **280** represent locations in a computer memory containing the series of instructions of particular method procedures pointed to by the addresses represented by labels **250** and **260.**

The SOM Base Classes

Much of the SOM Object Model is implemented by three classes that are part of the basic SOM support. Briefly these classes are:
SOMObject - This class is the root class of all SOM classes. Any class must be descended from SOMObject. Because all classes are descended from SOMObject they all inherit and therefore support the methods defined by SOMObject. The methods of SOMObject like the methods of any SOM class can be overridden by :he classes descended from SOMObject.
SOMClass - This class is the root meta class for all SOM meta classes. A meta class is a class whose instances are class objects. SOMClass provides the methods that allow new class objects to be created.
SOMClassMgr - This class is used to create the single object in a SOM based program that manages class objects.
The three SOM base classes are defined below.

SOMObject

This is the SOM root class, all SOM classes must be descended from <SOMObject>. <SOMObject> has no instance data so there is no per-instance cost to being descended from it.
SOMObject has the following methods:
Method: somInit
Parameters: somSelf
Returns: void
Description:

Initialize <self>. As instances of <SOMObject> do not have any instance data there is nothing to initialize and you need not call this method. It is provided to induce consistency among subclasses that require initialization.

<somInit> is called automatically as a side effect of object creation (ie, by <somNew>). If this effect is not desired, you can supply your own version of <somNew> (in a user-written metaclass) which does not invoke <somInit>.

When overriding this method you should always call the parent class version of this method BEFORE doing your own initialization.
Method: somUninit
Parameters: somSelf
Returns: void
Description:

(Un-initialize self) As instances of <SOMObject> do not have any instance data there is nothing to un-initialize and you need not call this method. It is provided to induce consistency among subclasses that require un-initialization.

Use this method to clean up anything necessary such as dynamically allocated storage. However this method does not release the actual storage assigned to the object instance. This method is provided as a complement to <somFree> which also releases the storage associated with a dynamically allocated object. Usually you would just call <somFree> which will always call <somUninit>. However, in cases where <somRenew> (see the definition of <SOMClass>) was used to create an object instance, <somFree> cannot be called and you must call <somUninit> explicitly.

When overriding this method you should always call the parentclass version of this method AFTER doing your own un-initialization.

Method: somFree
Parameters: somSelf
Returns: void
Description:

Releases the storage associated with <self>, assuming that <self> was created by <somNew> (or another class method that used <somNew>). No future references should be made to <self>. Will call <somUninit> on <self> before releasing the storage.

This method must only be called on objects created by <somNew> (see the definition of <somClass>) and never on objects created by <somRenew>.

It should not be necessary to override this method. (Override <somUninit> instead.)

Method: somGetClassName
Parameters: somSelf
Returns: Zstring
Description:

Returns a pointer to this object's class's name, as a NULL terminated string. It should not be necessary to override this method as it just invokes the class object's method (<somGetName>) to get the name.

Method: somGetClass
Parameters: somSelf
Returns: SOMClass *
Description:
Returns this object's class object.

Method: somGetSize
Parameters: somSelf
Returns: integer4
Description:
Returns the size of this instance in bytes.

Method: somRespondsTo
Parameters: somSelf, somId Mid
Returns: int
Description:

Returns 1 (true) if the indicated method is supported by this object's class and 0 (false) otherwise.

Method: somIsA
Parameters: somSelf, SOMClass *Aclassobj
Returns: int
Description:

Returns 1 (true) if <self>'s class is a descendent class of <Aclassobj> and 0 (false) otherwise. Note: a class object is considered to be descended from itself for the purposes of this method.

Method: somIsInstanceOf
Parameters: somSelf, SOMClass *Aclassobj
Returns: int
Description:

Returns 1 (true) if <self> is an instance of the specified <Aclassobj> and 0 (false) otherwise.

SOMObject methods that support dynamic object models. These methods make it easier for very dynamic domains to bind to the SOM object protocol boundary. These methods determine the appropriate method procedure and then call it with the arguments specified. The default implementation of these methods provided in this class simply lookup the method by name and call it. However, other classes may choose to implement any form of lookup they wish. For example, one could provide an implementation of these methods that used the CLOS form of method resolution. For domains that can do so it will generally be much faster to invoke their methods directly rather than going through a dispatch method. However, all methods are reachable through the dispatch methods. SOM provides a small set of external procedures that wrap these method calls so that the caller need never do method resolution.

These methods are declared to take a variable length argument list, but like all such methods the SOM object protocol boundary requires that the variable part of the argument list be assembled into the standard, platform-specific, data structure for variable argument lists before the method is actually invoked. This can be very useful in domains that need to construct the argument list at runtime. As they can invoke methods without being able to put the constructed arguments in the normal form for a call. This is helpful because such an operation is usually impossible in most high level languages and platform-specific assembler language routines would have to be used.

Note: Different methods are defined for different return value shapes. This avoids the memory management problems that would arise in some domains if an additional parameter was required to carry the return value. SOM does not support return values except for the four families shown below. Within a family (such as integer) SOM only supports the largest member.

Method: somDispatchV
Parameters: somSelf,
somId methodId,
somId descriptor,
Returns: void
Description:
Does not return a value.

Method: somDispatchL
Parameters: somSelf, somId methodId, somId descriptor
Returns: integer4
Description:
Returns a 4 byte quantity in the normal manner that integer data is returned. This 4 byte quantity can, of course, be something other than an integer.

Method: somDispatchA
Parameters: somSelf, somId methodId, somId descriptor
Returns: void *
Description:
Returns a data structure address in the normal manner that such data is returned.

Method: somDispatchD
Parameters: somSelf, somId methodId, somId descriptor
Returns: float8
Description:
Returns a 8 byte quantity in the normal manner that floating point data is returned.

SOMObject methods that support development

The methods in this group are provided to support program development. They have been defined in such a way that most development contexts will find them easy to exploit. However, some contexts may need to customize their I/O facilities. We have attempted to allow this customization in a very portable manner, however not all contexts will be able to perform the customization operations directly because they require passing function parameters. We chose this approach because it allows great platform-neutral flexibility and we felt that any provider of development support would find it reasonable to provide the customizations necessary for her/his specific development environment.

The chosen approach relies on a character output routine. An external variable, <SOMOutCharRoutine>, points to this routine. The SOM environment provides an implementation of this routine that should work in most development environments (it writes to the standard output stream). A development context can, however, assign a new value to <SOMOutCharRoutine> and thereby redefine the output process. SOM provides no special support for doing this assignment.

Method: somPrintSelf
Parameters: somSelf
Returns: SOMAny *
Description:
Uses <SOMOutCharRoutine> to write a brief string with identifying information about this object. The default implementation just gives the object's class name and its address in memory. <self> is returned.

Method: somDumpSelf

Parameters: somSelf, int level
Returns: void
Description:

Uses <SOMOutCharRoutine> to write a detailed description of this object and its current state. <level> indicates the nesting level for describing compound objects it must be greater than or equal to zero. All lines in the description will be preceded by <2*level> spaces.

This routine only actually writes the data that concerns the object as a whole, such as class, and uses <somDumpSelfInt> to describe the object's current state. This approach allows readable descriptions of compound objects to be constructed.

Generally it is not necessary to override this method, if it is overridden it generally must be completely replaced.

Method: somDumpSelfInt
Parameters: somSelf, int level
Returns: void
Description:

Uses <SOMOutCharRoutine> to write out the current state of this object. Generally this method will need to be overridden. When overriding it, begin by calling the parent class form of this method and then write out a description of your class's instance data. This will result in a description of all the object's instance data going from its root ancestor class to its specific class.

SOMClass

This is the SOM metaclass. That is, the instances of this class are class objects. When the SOM environment is created one instance of this class with the external name <SOMClassClassData.classObject> is created. This class object is unique because it is its own class object. That is, SOMClassClassData.classObject
==
_somGetClass(SOMClassClassData.classObject). This class introduces the somNew and somRenew methods that are used to create new instances of SOM objects. somNew applied to <SOMClassClassData.classObject> produces a new class object which can then be initialized to become a particular new class. SOMClass can be subclassed just like any SOM class. The subclasses of SOMClass are new metaclasses and can generate class objects with different implementations than those produced by
<SOMClassClassData.classObject>.
SOMClass is descended from SOMObject.
SOMClass defines the following methods.
Method: somNew
Parameters: somSelf
Returns: SOMAny *
Description:

Make an instance of this class. When applied to <SOMClassClassData.classObject>, or any other metaclass object, this will produce a new class object; when applied to a regular class object this will produce an instance of that class.

Method: somRenew
Parameters: somSelf, SOMAny *obj
Returns: SOMAny *
Description:

Make an instance of this class, but use the space pointed to by <obj> rather than allocating new space for the object. Note: no test is made to insury that <obj> points to enough space. <obj> is returned, but it is now a pointer to a valid, initialized, object.

Method: somInitClass
Parameters: somSelf, Zstring className, SOMAny *parentClass, integer4
instanceSize, int maxStaticMethods, integer4 majorVersion, integer4
minorVersion
Returns: void
Description:

Initialize <self>.

<parentClass> is the parent (or parent class) of this class, it may be NULL in which case it defaults to SOMObject (actually SOMObjectClassData.classObject the class object for SOMObject). If a parent class is specifed then it must have already been created as a pointer to its class object is required.

<instanceSize> should be just the space needed for this class, it is not necessary to consider the parent class's (if any) space requirements.

<maxStaticMethods> should be just the static methods defined by this class, it is not necessary to consider the parent class's methods (if any), even if they are overridden in this class.

<majorVersion> indicates the major version number for this implementation of the class definition, and <minorVersion> indicates the minor version number.

Method: somClassReady
Parameters: somSelf
Returns: void
Description:

This method is invoked when all of the static initialization for the class has been finished. The default implementation simply registers the newly constructed class with the SOMClassMgr. Metaclasses may override this method to augment the class construction sequence in any way that they wish.

Method: somGetName
Parameters: somSelf
Returns: Zstring
Description:

Returns this object's class name as a NULL terminated string.

Method: somGetParent
Parameters: somSelf
Returns: SOMClass *
Description:

Returns the parent class of self if one exists and NULL otherwise.

Method: somGetClassData
Parameters: somSelf
Returns: somClassDataStructure *
Description:

Returns a pointer to the static <className>ClassData structure.

Method: somSetClassData
Parameters: somSelf, somClassDataStructure *cds
Returns: void
Description:

Sets the class' pointer to the static <className>ClassData structure.

Method: somDescendedFrom
Parameters: somSelf, SOMClass *Aclassobj
Returns: int
Description:

Returns 1 (true) if <self> is a descendent class of <Aclassobj> and 0 (false) otherwise. Note: a class object is considered to be descended itself for the purposes of this method.

Method: somCheckVersion
Parameters: somSelf, integer4 majorVersion, integer4 minorVersion
Returns: int
Description:

Returns 1 (true) if the implementation of this class is compatible with the specified major and minor version number and false (0) otherwise. An implementation is compatible with the specified version numbers if it has the same major version number and a minor version number that is equal to or greater than <minorVersion>. The major, minor version number pair (0,0) is considered to match any version. This method is usually called immediately after creating the class object to verify that a dynamically loaded class definition is compatible with a using application.

Method: somFindMethod
Parameters: somSelf, somId methodId, somMethodProc **m
Returns: int
Description:

Finds the method procedure associated with <methodId> for this class and sets <m> to it. 1 (true) is returned when the method procedure is directly callable and 0 (false) is returned when the method procedure is a dispatch function.

If the class does not support the specified method then <m> is set to NULL and the return value is meaningless.

Returning a dispatch function does not guarantee that a class supports the specified method; the dispatch may fail.

Method: somFindMethodOk
Parameters: somSelf, somId methodId, SomMethodProc **m
Returns: int
Description:
Just like <somFindMethod> except that if the method is not supported then an error is raised and execution is halted.

Method: somFindSMethod
Parameters: somSelf, somId methodId
Returns: somMethodProc *
Description:
Finds the indicated method, which must be a static method defined for this class, and returns a pointer to its method procedure. If the method is not defined (as a static method or at all) for this class then a NULL pointer is returned.

Method: somFindSMethodOk
Parameters: somSelf, somId methodId
Returns: somMethodProc *
Description:
Just like <somFindSMethod> except that an error is raised if the method is not defined for this class.

Method: somSupportsMethod
Parameters: somSelf, somId Mid
Returns: int
Description:
Returns 1 (true) if the indicated method is supported by this class and 0 (false) otherwise.

Method: somGetNumMethods
Parameters: somSelf
Returns: int
Description:
Returns the number of methods currently supported by this class, including inherited methods (both static and dynamic).

Method: somGetInstanceSize
Parameters: somSelf
Returns: integer4
Description:
Returns the total size of an instance of <self>. All instances of <self> have the same size.

Method: somGetInstanceOffset
Parameters: somSelf
Returns: integer4
Description:
Return the offset in the body part of this object for the instance data belonging to this class.

Methods somGetInstancePartSize
Parameters: somSelf
Returns: integer4
Description:
Returns the size in bytes of the instance data required for this class. This does not include the instance data space required for this class' ancestor or descendent classes.

Method: somGetNumStaticMethods
Parameters: somSelf
Returns: int
Description:
Returns the number of static methods that this class has. This is used by a child class in initializing its method table.

Method: somGetPClsMtab
Parameters: somSelf
Returns: somMethodTab *
Description:
Returns a pointer to the method table of this class's parent class. If this class is a root class (SOMObject)

then NULL is returned.
Method: somGetClassMtab
Parameters: somSelf
Returns: somMethodTab *
Description:
Returns a pointer to the method table of this class.
Method: somAddStaticMethod
Parameters: somSelf, somId methodId, somId methodDescriptor, somMethodProc *method, somMethodProc *redispatchStub, somMethodProc *applyStub
Returns: somMOffset
Description:
Adds/overrides the indicated method, returns the value that should be used to set the offset value in the class data structure for this method name.
<methodDescriptor> is a somId for a string describing the calling sequence to this method as described in <somcGetNthMethodInfo> defined in the SOMObject class definition.
<method> is the actual method procedure for this method
<redispatchStub> is a procedure with the same calling sequence as <method> that re-dispatches the method to one of this class's dispatch functions.
<applyStub> is a procedure that takes a standard variable argument list data structure applies it to its target object by calling <method> with arguments derived from the data structure. Its calling sequence is the same as the calling sequence of the dispatch methods defined in SOMObject. This stub is used in the support of the dispatch methods used in some classes. In classes where the dispatch functions do not need such a function this parameter may be null.
Method: somOverrideSMethod
Parameters: somSelf, somId methodId, somMethodProc *method
Returns: void
Description:
This method can be used instead of <somAddStaticMethod> or <somAddDynamicMethod> when it is known that the class' parent class already supports this method. This call does not require the method descriptor and stub methods that the others do.
Method: somGetMethodOffset
Parameters: somSelf, somId methodId
Returns: integer4
Description:
Returns the specified method's offset in the method procedure table assuming this is a static method, returns 0 if it was not. This value is used to set the offset value in this class data structure. It should only be necessary to use this method when a class used to define a method that it now inherits.
Method: somGetApplyStub
Parameters: somSelf, somId methodId
Returns: somMethodProc *
Description:
Returns the apply stub associated with the specified method. NULL is returned if the method is not supported by this class. An apply stub is a procedure that is called with a fixed calling sequence, namely (SOMAny *self, somId methodId, somId descriptor, ap_list ap) where <ap> is a varargs data structure that contains the actual argument list to be passed to the method. The apply stub forwards the call to its associated method and then returns any result produced by the method.

SOMClassMgr

SOMClassMgr is descended from SOMObject.
SOMObject defines the following methods:
Method: somFindClsInFile
Parameters: somSelf, somId classId, int majorVersion, int minorVersion, Zstring file
Returns: SOMClass *
Description:
Returns the class object for the specified class. This may result in dynamic loading. If the class already

exists <file> is ignored, otherwise it is used to locate and dynamically load the class. Values of 0 for major and minor version numbers bypass version checking.
Method: somFindClass
Parameters: somSelf, somId classId, int majorVersion, int minorVersion
Returns: SOMClass *
Description:
Returns the class object for the specified class. This may result in dynamic loading. Uses somLocateClassFile to obtain the name of the file where the class' code resides, then uses somFindClsInFile.
Method: somClassFromId
Parameters: somSelf, somId classId
Returns: SOMClass *
Description:
Finds the class object, given its Id, if it already exists. Does not load the class. Returns NULL if the class object does not yet exist.
Method: somRegisterClass
Parameters: somSelf, SOMClass *classObj
Returns: void
Description:
Lets the class manager know that the specified class is installed and tells it where the class object is.
Method: somUnregisterClass
Parameters: somSelf, SOMClass *classObj
Returns: int
Description:
Unloads the class file and removes the class from the SOM registry
Method: somLocateClassFile
Parameters: somSelf, somId classId, int majorVersion, int minorVersion
Returns: Zstring
Description:
Real implementation supplied by subclasses. Default implementation returns the class name as the file name. Subclasses may use version number info to assist in deriving the file name.
Method: somLoadClassFile
Parameters: somSelf, somId classId, int majorVersion, int minorVersion,
Zstring file
Returns: SOMClass *
Description:
Loads the class' code and initialize the class object.
Method: somUnloadClassFile
Parameters: somSelf, SOMClass *classObj
Returns: int
Description:
Releases the class' code and destroys the class object
Method: somGetInitFunction
Parameters: somSelf
Returns: Zstring
Description:
Supplies the name of the initialization function in the class' code file. Default implementation returns (*SOMClassInitFuncName)().
Method: somMergeInto
Parameters: somSelf, SOMObject *targetObj
Returns: void
Description:
Merges the SOMClassMgr registry information from the receiver to <targetObj>. <targetObj> is required to be an instance of SOMClassMgr or one of its subclasses. At the completion of this operation, the <targetObj> should be able to function as a replacement for the receiver. At the end of the operation the receiver object (which is then in a newly uninitialized state) is freed. Subclasses that override this method should similarly transfer their sections of the object and pass this method to their parent as the final step. If the receiving object is the distinguished instance pointed to from the global variable SOMClassMgrObject, SOMCLassMgrObject is then reassigned to point to <targetObj>.

Managing Object Names

To improve upon past object oriented techniques of requiring unique external names for every method for a class the class method table is initialised at runtime via a special procedure associated with each class implementation and by collecting the set of method offsets into a single externally named class data structure. This improvement reduces the complexities of managing a large list of external variables, reduces the problem of creating unique names (referred to as name mangling), reduces the memory requirements and reduces the load time of the generated execution module.

Figure **3** is a SOM class data structure that implements such a strategy. Label **310** represents a pointer to the class object data structure set forth in Figure **2** at **248**. Label **320** represents an offset into the method procedure table set forth in Figure **2** at label **240** or into the object's state data structure set forth in Figure 2 at label 230. Similarly, labels **330** and **340** represent additional offsets into the method procedure table or into its state data structure. For additional methods that are first defined in this class or methods that are mentioned in the class release order section but defined by one of the class' ancestor classes, or public instance variables defined by this class, there are similar entries in the class data structure representing offsets associated with this class as signified by the elipses and "N + 1" at label **350**. The additional entry is necessary because of the first entry represents a pointer to the class object data structure **248** in Figure 2.

The order of the values in the class data structure is determined by the order of the corresponding method or public instance variable name in the release order section of the class OIDL file. Methods or public data members defined in the class but not mentioned in the release order section are ordered after those mentioned in the release order section and in the order in which they appear in the class OIDL file.

Object Interface Definition Language (OIDL)

Language dependent object definitions are redefined as a neutral set of information from which object support for any language is provided. The neutral set of information is referred to as an Object Interface Definition Language (OIDL) definition in SOM. SOM OIDL provides the basis for generating binding files that enable programming languages to use and provide SOM objects and their definitions (referred to as classes). Each OIDL file defines the complete interface to a class of SOM objects.

OIDL files come in different forms for different languages. The different forms enable a class implementer to specify additional language-specific information that allows the SOM Compiler to provide support for constructing the class. Each of these different forms share a common core language that specifies the exact information that a user must know to use a class. One of the facilities of the SOM Compiler is the extraction of the common core part of a class definition. Thus, the class implementer can maintain a language-specific OIDL file for a class, and use the SOM Compiler to produce a language-neutral core definition as needed.

This section describes OIDL with the extensions to support C-language programming. As indicated above, OIDL files are compiled by the SOM Compiler to produce a set of language-specific or use-specific binding files.

The SOM Compiler produces seven different files for the C language.

o A public header file for programs that use a class. Use of a class includes creating instance objects of the class, calling methods on instance objects, and subclassing the class to produce new classes.
o A private header file, which provides usage bindings to any private methods the class might have.
o An implementation header file, which provides macros and other material to support the implementation of the class.
o An implementation template, which provides an outline of the class' implementation that the class provider can then edit.
o A language-neutral core definition
o A private language-neutral core file, which contains private parts of the class interface.
o An OS/2 .DEF file that can be used to package the class in the form of an OS/2 DLL.

OIDL files can contain the following sections:

o Include section;
o Class section;
o Release Order section;
o Metaclass section;
o Parent Class section;
o Passthru section;
o Data section; and
o Methods section.

Include section

This required section contains an include statement that is a directive to the OInL preprocessor telling the compiler where to find the class interface definition for this class' parent class, the class' metaclass if the class specifies one, and the private interface files for any ancestor class for which this class overrides one or more of its private methods.

Class Section

This required section introduces the class, giving its name, attributes and optionally a description of the class as a whole.

Release Order Section

This optional section contains a **release order** statement that forces the compiler to build certain critical data structures with their items arranged in the order specified. This allows the class interface and implementation to be evolved without requiring programs that use this class be recompiled.

Release order applies to all method names and public data items. If the release order of some method or public data item is not specified, it will default to an implementation-specific order based on its occurrence in the OIDL file. The introduction of new public data items or methods might cause the default ordering of other public data items or methods to change; programs using the class would then need to be recompiled.

Metaclass section

This optional section specifies the class' metaclass, giving its name and, optionally, a description of the reason for the metaclass, or other comments about its role in this class' interface. If a metaclass is specified, its definition must be included in the **include** section. If no metaclass is specified, the metaclass of this class' parent class will be used.

A class' metaclass can also be implicitly defined through the combined use of the **class** attribute in the **data** section and the class attribute in the method section. If either of these attributes are used, then the **metaclass** section must be bypassed. In this case, the implied metaclass will be a subclass of the **metaclass** of the parent class.

Parent Class Section

This required section specifies the class' parent class by indicating the name and optionally a description of the role of the parent class in this class' interface.

Passthru Section

This optional section provides blocks of code to be passed by the compiler into various binding files. The contents of the passed information are ignored by the compiler. Even comments contained in **passthru** lines are processed without modification.

Data Section

This optional section lists the instance variables for this class. This section i s generally present only in the language specific version of the class interface definition (a .CSC file). However, it must be present in the public form of the class interface definition if the class contains public instance variables. ANSI C syntax is used to describe these variables.

Methods Section

This optional section lists the methods supported by this class. ANSI C function-prototype syntax is used to define the calling sequence to each method.

SOM Compiler

The SOM Compiler translates the OIDL source definition of a SOM class into a set of bindings appropriate for a particular programming language. The SOM Compiler supplied with the OS/2.0 toolkit produces a complete set of bindings for the C programming language.

The compiler operates in two phases - a precompile phase and an emission phase. In the first phase a precompiler reads and analyzes a user-supplied class definition and produces intermediate output files containing binary class information, comments and passthru lines. In the second phase, one or more emitter programs run to produce the appropriate language binding files. Two additional programs serve as preprocessors for the SOM precompiler phase. The sequencing and execution of all of these programs is directed by the SOM Compiler.

The output from the emitters, plus user-supplied logic for the class' methods, are subsequently compiled by the C compiler and linked by the OS/2 linker to create a loadable module. Loadable modules can be packaged in self-contained files or placed in a DLL so the class can be used from many programs.

Referring to Figure **4**, control commences at terminal **400** and flows directly into function block **404** where a SOM language neutral object interface definition (OIDL) **402** is input to the SOM OIDL compiler **404**. The SOM OIDL compiler parses the object definitions in OIDL into a canonical form **406** to simplify the code generation process as input to the target language emitter **410**. The language emitter **410** generates language bindings **414** which include the class data structure depicted in Figure **3**. Control flows to the language compiler shown in function block **420** which receives additional inputs from the language applications **416** and the SOM bindings **412**. The language compiler could be a C, Fortran, Cobol or other compiler depending on user preference. Output from the language compiler is an object file **422** which can be link edited with the SOM runtime library for subsequent execution.

Figure **5** is a flowchart depicting a link, load and execution of an application using SOM objects. Processing commences at terminal **500** and immediately flows into function block **530** for a dynamic link and load of the SOM objects **510** created in Figure **4** at label **422** and the SOM run time library **520**. Then, at function block **540**, the application is started, which invokes the creation of necessary classes and objects as set forth in function block **550** and detailed in Figures **6**, **7**, **8**, **9** and **10**. Finally, the application is executed as shown in function block **560** and control is terminated at terminal block **570**.

Version Independence For Object Oriented Programs

This feature generally relates to improvements in object oriented applications and more particularly solving problems arising from the independent evolution of object definition libraries and the computer applications that use them.

The version independence processing isolates the executable binary form of computer applications that use object definition libraries (also called object class libraries) from certain changes in the implementations or specification of the object definitions that naturally arise during the lifecycle of the libraries. Specifically, the following changes can be made to an object definition without compromising its use by the unmodified executable binary form of a computer application which dynamically loads the object definition each time the application is executed:

1) add new methods to an object definition;
2) move the point of definition for a method from a child class to its parent class;
3) add to, delete from, or otherwise change the private instance data associated with an object definition; and
4) insert a new class definition into a class hierarchy.

This processing is accomplished by the operation of an algorithm in the memory of a processor employing several techniques as follows. Method and instance offset are removed from application binary images. In static object models, such as the one defined in C++, an offset (an integer number) into a method procedure table is used to select a method procedure for each particular method name. The offset depends on the number and order of the methods of the class the method is defined in and the number of methods defined by its ancestors.

This approach has the benefit of being a very fast form of method resolution. However, in the prior art object models have placed these offsets in the binary images of the applications that used a particular object class, resulting in the requirement to recompile the application whenever the offsets required a change.

In SOM, the offsets associated with methods are collected into a single memory data structure for each class, called the class data structure, detailed in the discussion of Figure 3. This data structure is given an external name and its contents are referred to in applications. Each class data structure is initialized to contain the appropriate offset values when a class object is initialized as detailed in Figure **10**. Thus each time an application is executed all the offset values are recalculated based on the current definitions of the classes used by the application.

Note that any references in an application's binary images to the values stored in the class data structure contain offsets. However, these offsets can remain constant across the four kinds of changes enumerated above. This is because the class data structure only contains offsets for the methods defined in a particular class, not for offsets of methods inherited by the class. Thus, new methods added to a class can have their offsets added at the end of the class data structure without disturbing the positions of the offset values for methods that were already defined in the class.

The SOM Object Interface Definition Language (OIDL) contains a Release Order Section, discussed in the section titled "SOM Object Model" above. The release order section of OIDL allows the class implementor to insure that new method offset values are added after the method offset values for methods already defined in a class. The release order section in an OIDL file also causes an entry to be retained in a class data structure if one of the methods defined in the class is moved to a parent class as highlighted in Figure **3**. This entry is then initialized from the parent offset value by a simple assignment statement that the OIDL compiler adds the logic initializing the class data structure as described in Figure **10**.

A similar problem arises with public instance data. An application that accesses a public instance variable contained in one of the application's object's state data structure must do so via a offset into the object's state data structure. In the prior art, this offset was contained in application's binary image. If this technique is employed, then the application's binary image must be regenerated (via recompilation) any time the offset changes due to a change in the size of one or more of the object's ancestor classes' instance data requirements or due to changes in the object's own instance data layout.

In SOM this problem is solved by putting the offset for each public data variable in the class data structure detailed in Figure **3** and the ensuing discussion. Each class data structure is initialized to contain the appropriate offset values when the class object is initialized as detailed in Figures **7** and **13**. Thus, each time an application is executed all the offset values are recalculated based on the current definitions of the classes used by the application.

Remove object state data structure sizes from applications' binary images

When new instances of objects are created, a correct amount of computer memory must be allocated to hold the object's state data structure. In the prior art, the size of this block of memory was contained in an application's binary image. If this technique is employed, then the application's binary image must be regenerated (via recompilation) any time the size of the object's state data structure changes. In SOM, this value is available via a call to the object's class object and therefore need not be contained in an application's binary image.

The techniques described above allow each of the four changes previously highlighted to occur with respect to class definitions used by an application without requiring that the application's binary image to be regenerated.

Figure **6** is a flowchart depicting the creation of a new SOM class. Control commences at terminal **600** which flows immediately into a test for a correct version number at decision block **610** where a check is performed to verify the correctness of the version number. If an incorrect version number is detected, then a message is displayed in output block **612** and control is terminated at terminal block **614**. If a correct version number is detected, then another test is performed at decision block **620** to determine if the SOM class exists. If the SOM class exists, then processing is returned at terminal block **622**.

If the SOM class does not exist at decision block **620**, then a test is performed at decision block **630** to determine if the SOM runtime environment is active. If it is not active, then the SOM runtime environment is invoked at function block **632**. Whether the SOM environment was initially present or not, control then flows to decision block **640** to check for an error in the SOM environmemt at decision block **640**. If an error is detected, then an appropriate message is presented at output block **642** and processing is terminated at terminal block **644**. If an error is not detected, then control passes to function block **650** where a default metaclass is prepared. Next, a class is constructed in function block **652** as detailed in Figure **7**. Finally, processing is returned at terminal block **660**.

Figure **7** is a flowchart depicting the detailed construction of a new SOM class. Control commences at terminal **700** and flows immediately into function block **710** where a generic class object is created as detailed in Figure **8**. Next, the new generic class is initialized to default values at function block **720** and detailed in Figure **9**. Then, at function block **730**, the instance data offset is initialized for the particular new class. Control flows to function block **740** where the class data structure (Figure **3**) for the new class is initialized by assigning values representing each static method for the new class as detailed in Figure **10**.

At function block **750**, **760** and **770** the parent class is set, the class data is initialized and the class is registered. These steps involve updating the new class data structure as detailed in the discussion of Figures **2**, **10** and **13**. Finally, control is returned at terminal **780**.

Figure **8** is a flowchart depicting the detailed construction of a new SOM generic class object. Control commences at terminal **800** and immediately flows into function block **810** where memory is allocated for the object. Then, a test is performed at decision block **820** to determine whether the memory was allocated. If an error is detected, then an appropriate error message is displayed at output block **830** and processing is terminated at terminal block **840**. If no error is detected, then the default values of the object are set at function block **850** and control is returned at terminal block **860**.

Figure **9** is a flowchart depicting the detailed initialization of a new SOM class object. Control commences at terminal **900** and immediately enters a decision block **910** and a test is performed to detect if the parent class of the new SOM class object exists. If a parent class exists, then the parent class is initialized in function block **912**. Once the parent class is initialized, then memory for the class name is allocated at function block **920**. Next, a test is performed again to detect if the parent class of the new SOM class object exists at decision block **930**.

If a parent class does not exist, then initial variables are set to zero as shown in function block **932** and control passes to function block **970**. If a parent class exists, then the initial variables are updated based upon the values from the parent class in function blocks **940**, **950**, and **960**. Then, in function block **970**, the version number for the class is set and error processing is performed in decision block **980**. If an error is detected, then an appropriate message is displayed at output block **982** and processing terminates at terminal block **984**. If no error is detected, then control is returned at terminal block **990**.

Figure **10** is a flowchart depicting the detailed initialization of a SOM class data structure with offset values. Control commences at terminal block **1000** and immediately flows into function block **1010** where a loop commences with the acquisition of the next static method. In function block **1020**, the new method id is registered with the SOM runtime environment. Then, a test is performed to determine if the method has already been registered in a parent class in decision block **1030**. If the method has been registered, then the method offset is overridden at function block **1032** and control passes to decision block **1070**.

If the method has not been registered with any parent class, then a test is performed to determine if the method has been defined in the current class at decision block **1040**. If the method has been defined, then the existing offsets are employed at function block **1042** and control is passed to decision block **1070**. If the method has not been defined, then memory is allocated and values are initialized in function blocks **1050** and **1060**. In function block **1060** the offset is calculated by adding the number of inherited static methods to the number of inherited static methods processed to date by the class. Error processing is performed in decision block **1070**, and if an error is detected, then an appropriate message is displayed at output block **1072** and processing terminates at terminal block **1074**. After error processing is completed, another test is performed at decision block **1080** to determine if any additional methods require processing. If there are additional methods, then control passes to function block **1010** for the next iteration of the loop. Otherwise, control flows to terminal **1090** where control returns.

Parent Class Shadowing

Logic for providing a dynamic insertion of a replacement parent class, referred to in object programming as a parent class shadow, is detailed in this section. This processing allows the statically compiled definition of what parent class is linked to a particular class at runtime to be dynamically altered during execution. The ability to insert a new parent class into a statically compiled class hierarchy offers more flexibility to maintain and enhance existing code after it has appeared in binary form. It also offers a new degree of freedom for customizing code without access to source materials since this result can be achieved without recompilation.

Prior art systems have inherent limitations associated with statically linking derived classes and their parent classes. These limitations include, computation of the size of the derived object state data structure, initialization of the derived method procedure table, and the inability to provide access to a parent class' methods from within the derived class' methods (called parent class resolution).

The SOM object model removes these static references by having all the parent class information available at runtime through the parent class object. Thus, when the derived class implementation needs information about the size of the parent class' state data structure, the addresses of the parent class' method procedures, or access to the parent class' method procedure table (to support parent class resolution) an appropriate call is placed to acquire the information from the parent class object. The detailed processing to obtain this information are given in Figures **7**, **8**, **9**, and **10**.

SOM introduces a class manager for every SOM process. The class manager is responsible for keeping a registry of classes. The class construction code generated by the SOM compiler works with the class manager to establish the relationship between a class and its parent class whenever a child class object is created. The SOM class manager is an instance of a class which can be subclassed like any other SOM class.

Derived classes establish a connection to their parent class object by making calls on the SOM Class Manager object. An application designer wanting to substitute an alternate class implementation for the original class implementation follows the following steps:

1) Subclass SOMClassMgr providing a new set of application specific rules for determining a class object from a class name (i.e., changing the implementations of somClassFromId, somFindClass, and somFindClsInFile)

A simple and useful way to do this is to add a method to register a shadow class object under an existing class name and then return the shadow class object to the calling application in any subsequent calls to somClassFromId, somFindClass, or somFindClsInFile where the shadowed name is specified.

2) Before creating any derived class objects that are to have a shadowed parent class object, create an instance of the new class manager class (as described in step 1 above), initialize it from the existing SOMClassMgr instance (via the somMergeInto method), and then replace the existing SOMClassMgr instance with the new class manager instance by overriding the address of the existing SOMClassMgr instance in the SOM runtime.

3) Still before creating any derived class objects that are to have a shadowed parent class object, use the facilities of the application specified class manager object to register the shadow class objects.

After the above three steps have been completed, derived class objects can be created. They will be linked to the appropriate parent shadow class objects. This will work because of the specific logic used to initialize a class object and link to its parent class object as depicted in Figure 11. This logic consists of two basic steps:

1) First, a call is made to insure that the statically known parent class object has been created. This serves two important purposes:

(a) It creates a static reference to the binary image of the statically known parent class definition, thus insuring that the parent class implementation will be linked into the binary image of the application.

(b) It insures that the at least the statically known parent class object has been registered with the SOM class manager object before the next step occurs.

If the statically known parent class object has already been created (say by an application following the shadowing steps discussed above) then a second attempt at this time is ignored.

2) Second, a call is made to the SOM class manager object to retrieve the address of the appropriate class object based on the name of the derived class' parent class. If the parent class has been shadowed then this call will return the shadow class object.

The combination of the techniques and mechanisms described above effectively isolate a derived class' binary image from any dependency on the exact class of the class object that the derived class uses to extract parent class data from.

Two restrictions must be observed when inserting a new class between a child class and its parent class. First, the insertion must be accomplished before any instances of the child class have been created. Second, the inserted class must also be an immediate child of the original parent class. Because the SOM class manager is used as an intermediary when establishing the relationships between classes at run time, even a statically linked class can be shadowed in this manner.

Figure **11** is a flowchart depicting the detailed parent class shadowing of a statically defined class hierarchies. Control commences at terminal block **1100** and immediately flows into function block **1110** where the statically defined parent class object is created. Next, the shadow parent class is created and used to override the statically defined parent class at function block **1120**. Then, the child class is created as shown in function block **1130** and the child class interrogates the SOM class manager to ascertain its current, rather than statically defined, parent class. Control returns at terminal block **1140**.

Redispatch Method Stubs

A central aspect of object oriented programming is referred to as method resolution. This processing selects a particular method given an object, the method's id and the arguments passed to the method invocation. In many object models, such as the one used in C++, method resolution consists of determining an offset into an object specific table of procedure entry points based on an analysis of the program's source code. This type of resolution is referred to in object models as static. In other object models such as the one used in Smalltalk, a more dynamic model is used that consists of using the name of the object to determine a specific method at runtime. In object models this is referred to as dynamic.

A programming mechanism is provided based on redispatch stubs to ameliorate the difference between static and dynamic models. A redispatch stub is a small procedure with an entry point that can be placed into a table of procedure entry points. The table of procedure entry points are used in a static object model as a substitute for the actual method entry point that is expected. The redispatch stub is generated automatically based on the requirements of the dynamic object model. The redispatch stub converts the call generated in the static object model into the form necessary in the dynamic object model and supplies any missing information in the process. Thus, if an object is accessed from a static object model that is provided by a dynamic object model, it can be represented to the static object model via a table of entry points which each indicate a particular redispatch stub.

Figure **12** is a flow diagram depicting the redispatch method. Label **1200** is a state data structure for a particular object. The first full word at label **1210** contains the address of the object's method procedure table label **1240**. The rest of the state data structure is set forth at label **1230** contains additional information pertaining to the object. The method procedure table set forth at label **1240** containing the addresses of various methods for the particular object. All objects that are of the same class as this object also contain an address that points to this method procedure table diagrammed at label **1240**. Any methods inherited by the objects will have their method procedure addresses at the same offset in memory as they appear in the method procedure table as set forth at label **1240** of the ancestor class from which it is inherited.

In the figure, label **1250** contains a pointer to a redispatch stub **1270**. A redispatch stub is a sequence of instructions that appear as a method to a client program. However, the instructions merely convert the method call into a call to an object's appropriate dispatch function as illustrated at label **1260**. The address at label **1260** is a pointer to the object's dispatch function **1280**. All SOM objects have a dispatch function. The dispatch function **1280** implements an algorithm to select a particular method based on the parameters passed by the redispatch stub. These parameters include the method's identifier, a string describing a set of arguments passed to the identified method, and a data structure containing the set of arguments.

Offset Values

Figure **13** is a flowchart depicting the detailed initialization of the offset value in a SOM class data structure for a single public instance variable. This logic sequence is repeated for each public instance variable defined in a particular class (see the discussion of the OIDL Data Section above). Control commences at the terminal block **1300** and immediately flows into the function block **1310** where the offset of the instance variable is calculated by adding the instance variable's offset within this class' object state data to the offset of the beginning of this class' object state data within the object state data structure set forth in Figure **2** at label **230**.

The beginning of the class' object state data is determined by adding up the sizes of each of this class' ancestor classes' object state data. Control then passes to function block **1320** when the calculated offset is stored into the position in the class data structure as determined by the position of the public instance variable's name in the OIDL files Release Order Section (see the OIDL Release Order section above and Figure **3** above). Control then flows to the terminal block **1330** and the process is complete.

Redispatch Stubs

Figure **14** is a flowchart depicting the detailed control flow that occurs when a redispatch stub is employed to convert a static method call into a dynamic method call. Control commences at the terminal block **1400** and immediately flows into the function block **1410** where the address of the redispatch stub is determined in the normal static method resolution manner by getting the address stored in the object's method procedure table at an offset contained in the appropriate class data structure at position determined when the class was defined.

Control then passes to function block **1420** where the redispatch stub is called exactly like it was the real static method procedure. Function block **1430** depicts how the redispatch stub calls the object's dispatch method (using normal method resolution as described above). The redispatch stub adds the method's identifier and descriptor to the call as required by the object's dispatch method. These values are incorporated into the redispatch function definition when it is generated by the SOM OIDL compiler. (Note: as detailed in the definition of the SOMObject class above, all classes must support dispatch methods). The object's dispatch method procedure determines which actual method procedure should be called using an algorithm specific to the object's class as shown in function block **1440**.

SOM provides a default implementation of such an algorithm that looks the method's identifier up in a table contained in the object's class object to determine the address of a method procedure. Other object models might use other algorithms. Control then passes to function block **1450** where the method procedure determined in block **1440** is called. When the method procedure returns its return value if any is returned to the original caller of the redispatch stub at terminal block **1460.** The redispatch stub allows the original static method call to be converted to one of arbitrary dynamics without requiring any changes to the application program that is manipulating the object.

Method Procedure Table Initialization

Figure **15** is a flowchart depicting the detailed control flow that will properly initialize a method procedure table for a class that may change the association of method procedures to method during the execution of an application using the class. Control. commences at terminal block **1500** and immediately flows into function block **1510** where space is allocated for the method procedure table. Enough space is allocated to contain an entry for the address of the class' object and each of the method inherited or defined by the class in accordance with Figure **7**. Control then passes to function block **1520** where each method entry in the method procedure table is replaced by its redispatch stub. Redispatch stubs for inherited are determined by requesting them from the class' parent class. Redispatch stubs for the class are generated by the SOM compiler and supplied to the class initialization procedure in the calls to register each of the class' static method. Control then passes to function block **1530** where the method procedure table entries for the class' dispatch function are replaced by the actual address of the class' dispatch function (it is never correct to have a redispatch stub address in a dispatch function slot as this would result in a infinite loop). Finally control passes to the terminal block **1540** and processing is complete.

Claims

1. A computer having means for supporting an object oriented environment, including means for organizing a set of object classes having methods implemented by functions stored by the computer, comprising:
means for initializing a method procedure table for each class;
means for defining all functions statically for each class;
means for collecting offset variables into a data structure for each class; and
means for externalizing the data structure for each class in a computer memory.

2. A computer as recited in claim 1, further comprising means for storing the method procedure table.

3. A computer as recited in claim 1 or 2, further comprising means for initializing the offset variables at runtime.

4. A computer as recited in any preceding claim, further comprising means for overriding the methods of the method procedure table.

5. A computer as recited in any preceding claim, further comprising means for uniquely differentiating between the methods of the method procedure table without requiring that the names of the methods be changed.

6. A method for organizing a set of object classes in a computer supporting an object oriented environment, the object classes having methods implemented by functions stored by the computer, comprising the steps of:
initializing a method procedure table for each class;
defining all functions statically for each class;
collecting offset variables into a data structure for each class; and
externalizing the data structure for each class in a computer memory.

7. A method as recited in claim 6, wherein the step of initialising includes the step of storing the method procedure table in the computer.

8. A method as recited in claim 6 or 7, including the step of initializing the offset variables at runtime.

9. A method as recited in any of claims 6 to 8, including the step of overriding the methods of the method procedure table.

10. A method as recited in any of claims 6 to 9, including the step of uniquely differentiating between the methods of the method procedure table without requiring that the names of the methods be changed.

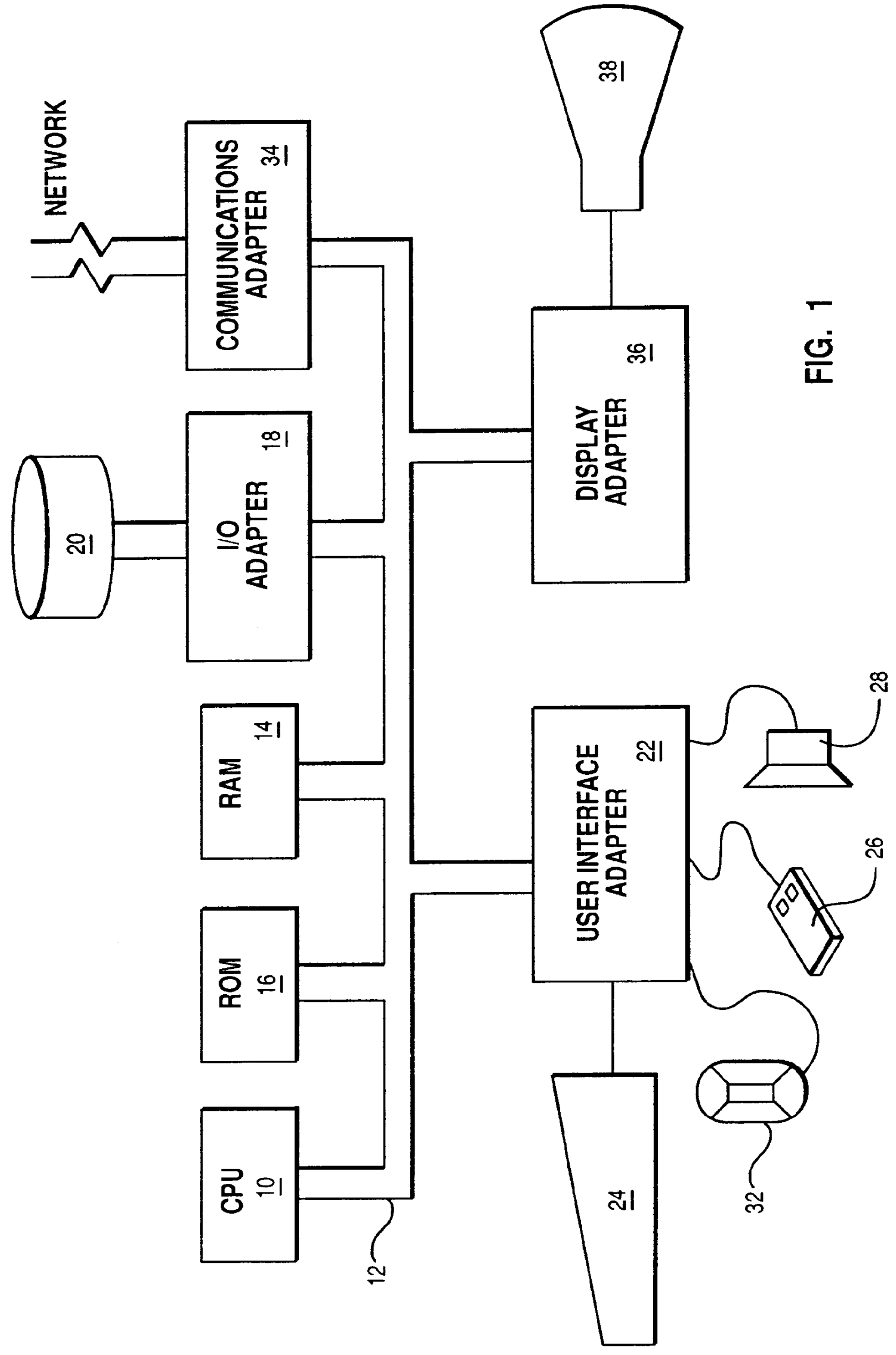
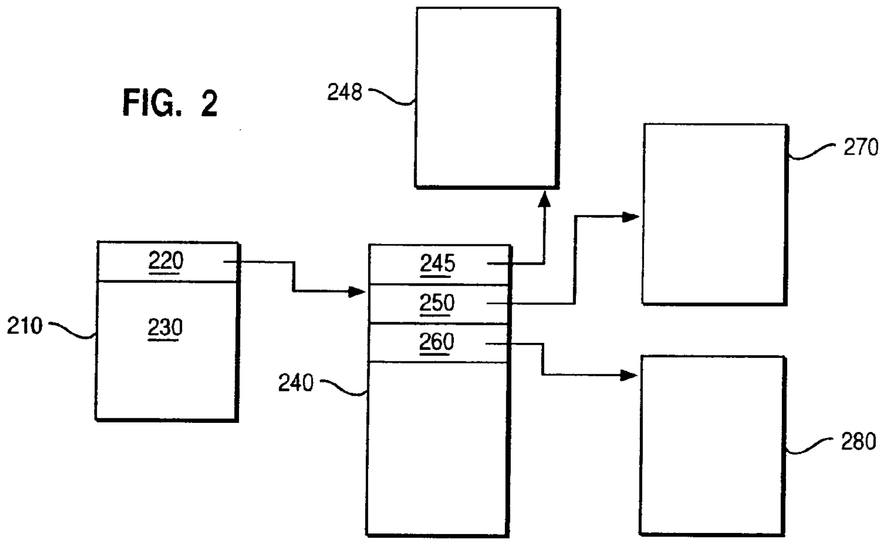

NETWORK
COMMUNICATIONS ADAPTER 34
I/O ADAPTER 18
20
RAM 14
ROM 16
CPU 10
12
DISPLAY ADAPTER 36
38
USER INTERFACE ADAPTER 22
24
26
28
32

FIG. 1

FIG. 2



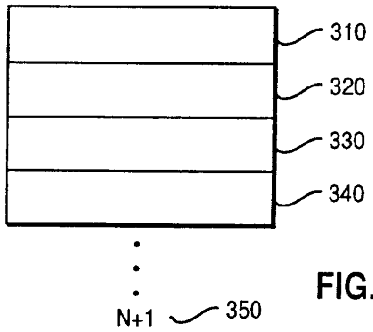


FIG. 3

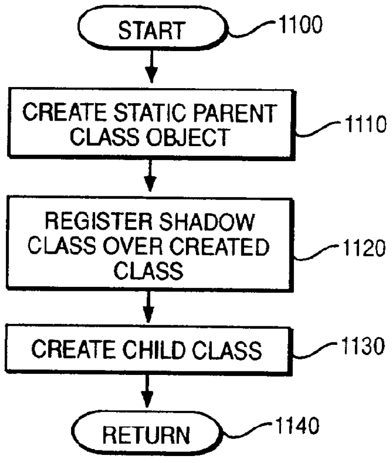

START
1100
CREATE STATIC PARENT CLASS OBJECT
1110
REGISTER SHADOW CLASS OVER CREATED CLASS
1120
CREATE CHILD CLASS
1130
RETURN
1140

FIG. 11

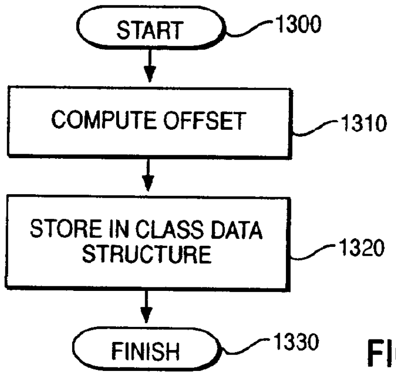


FIG. 13

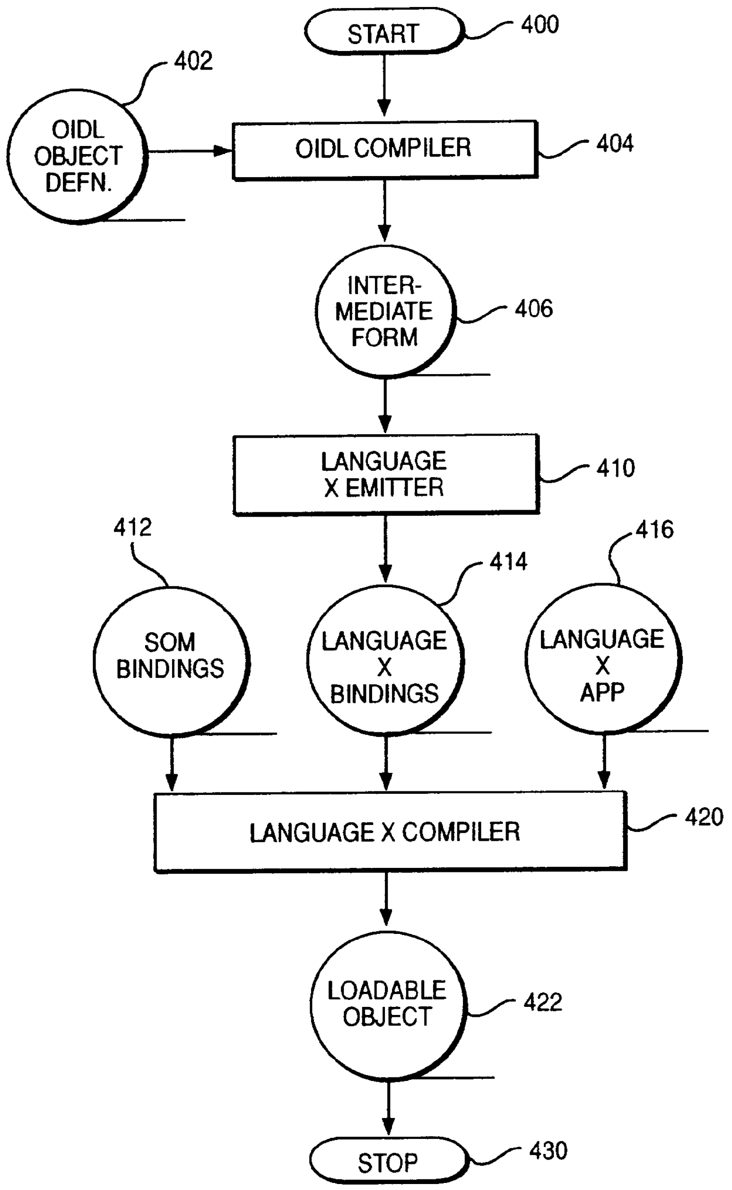

START
400
402
OIDL OBJECT DEFN.
OIDL COMPILER
404
INTER-MEDIATE FORM
406
LANGUAGE X EMITTER
410
412
414
416
SOM BINDINGS
LANGUAGE X BINDINGS
LANGUAGE X APP
LANGUAGE X COMPILER
420
LOADABLE OBJECT
422
STOP
430

FIG. 4

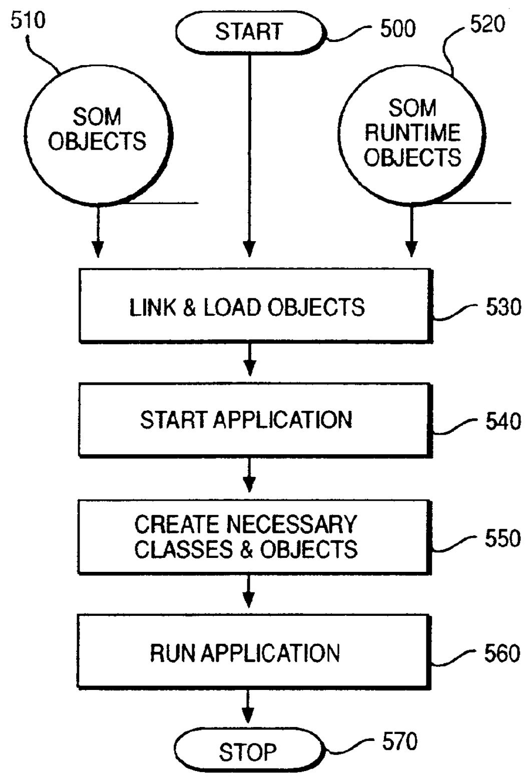


FIG. 5

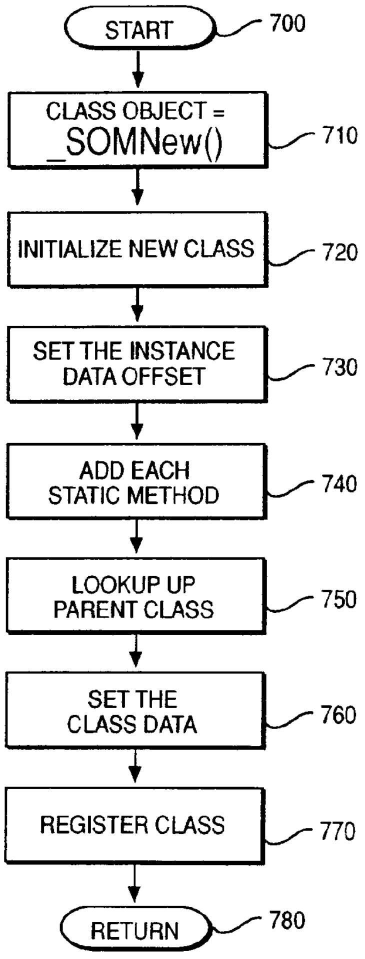


FIG. 7

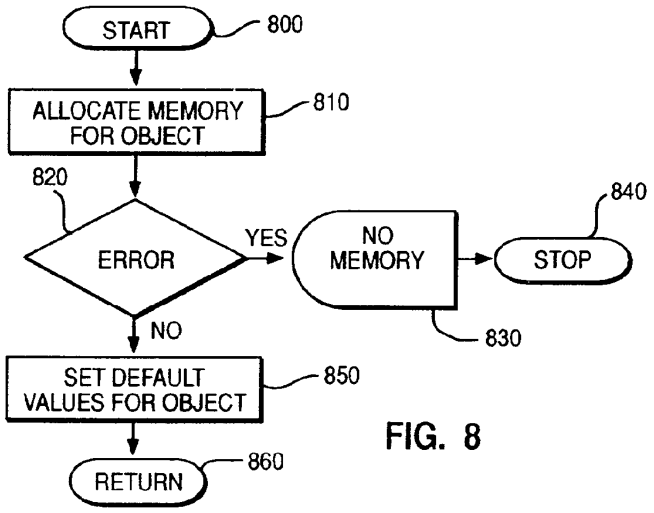


FIG. 8

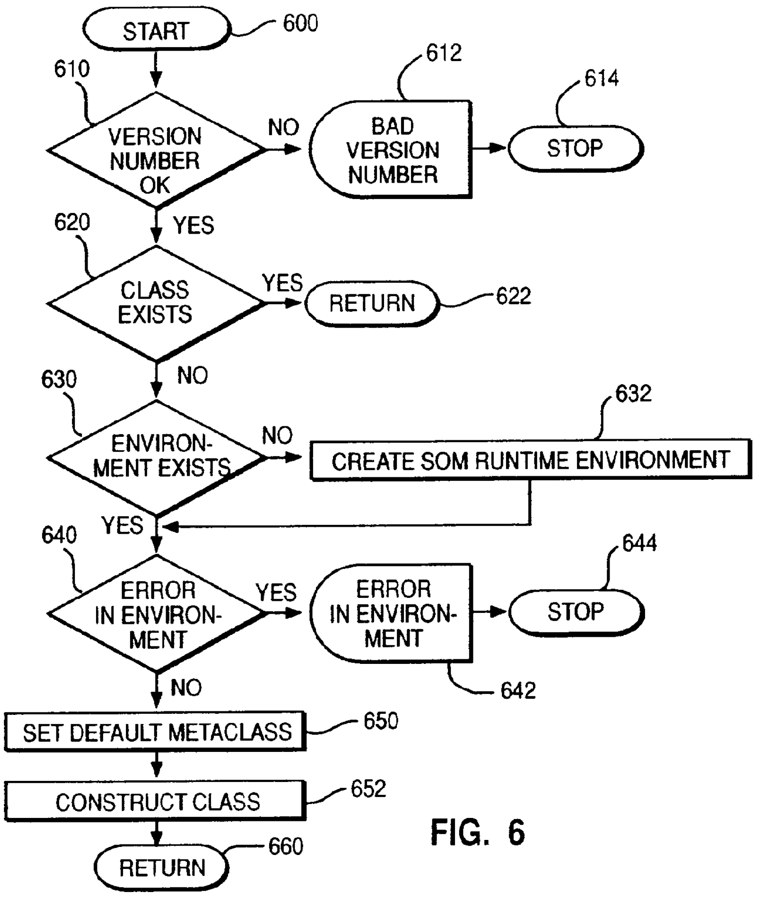

START
600
610
VERSION NUMBER OK
NO
612
BAD VERSION NUMBER
614
STOP
YES
620
CLASS EXISTS
YES
RETURN
622
NO
630
ENVIRON-MENT EXISTS
NO
632
CREATE SOM RUNTIME ENVIRONMENT
YES
640
ERROR IN ENVIRON-MENT
YES
ERROR IN ENVIRON-MENT
644
STOP
642
NO
SET DEFAULT METACLASS
650
CONSTRUCT CLASS
652
RETURN
660

FIG. 6

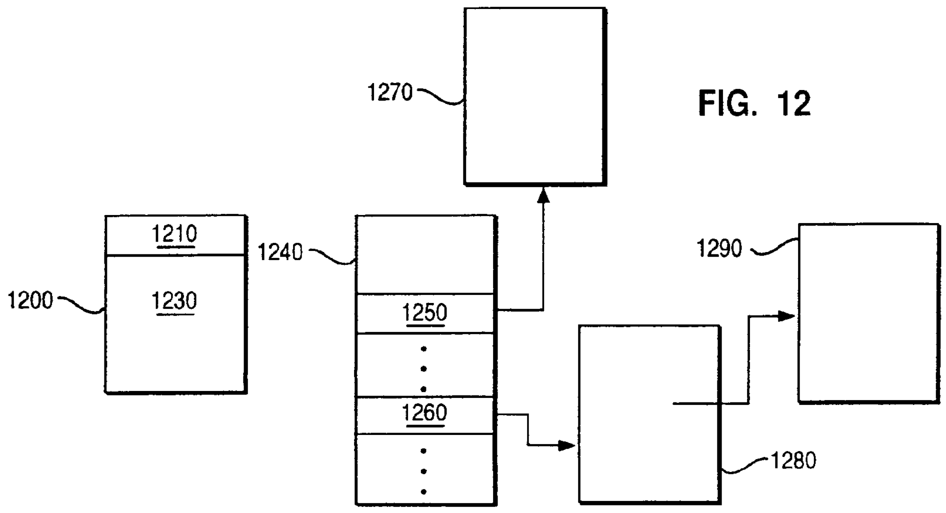

1270
1210
1230
1200
1240
1250
1260
1290
1280

FIG. 12

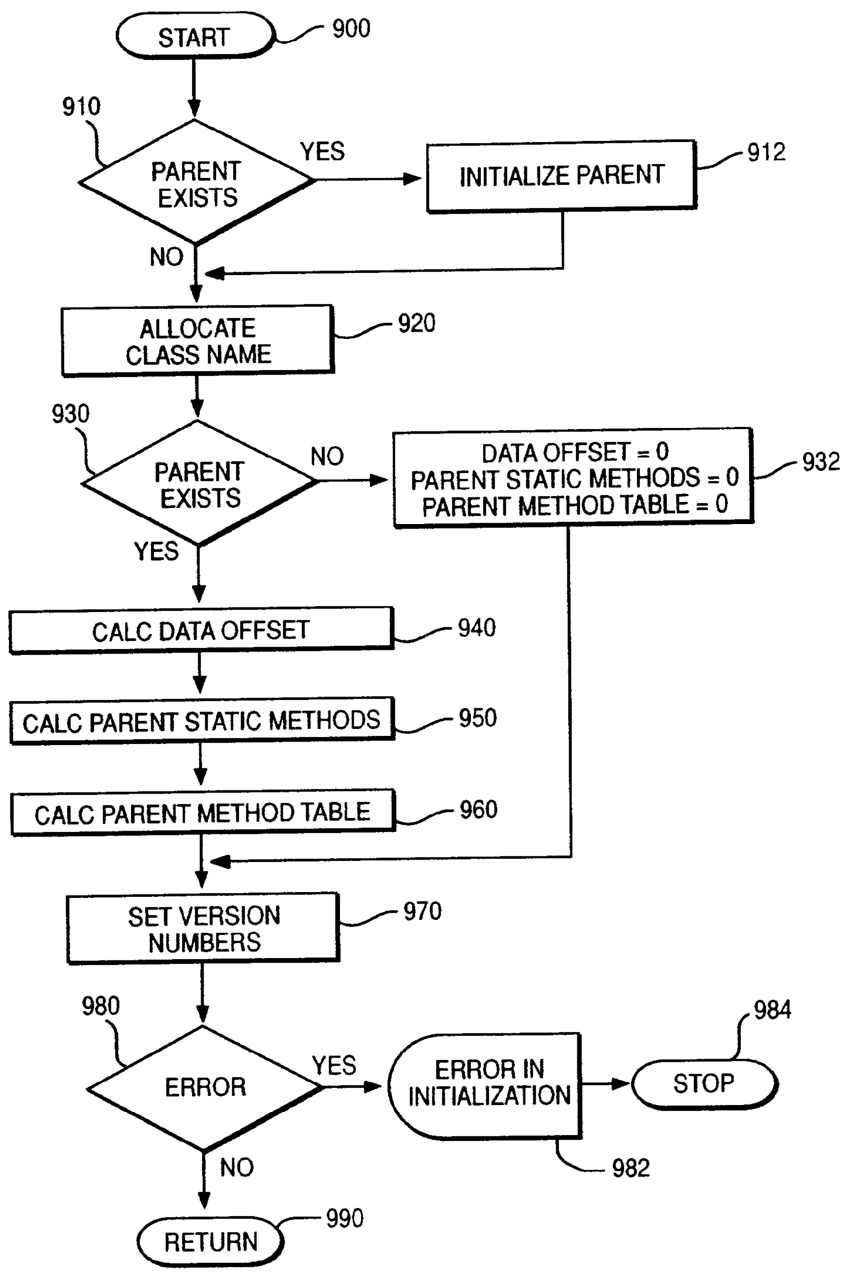

START
900
910
PARENT EXISTS
YES
INITIALIZE PARENT
912
NO
ALLOCATE CLASS NAME
920
930
PARENT EXISTS
NO
DATA OFFSET = 0
PARENT STATIC METHODS = 0
PARENT METHOD TABLE = 0
932
YES
CALC DATA OFFSET
940
CALC PARENT STATIC METHODS
950
CALC PARENT METHOD TABLE
960
SET VERSION NUMBERS
970
980
ERROR
YES
ERROR IN INITIALIZATION
STOP
984
982
NO
RETURN
990

FIG. 9

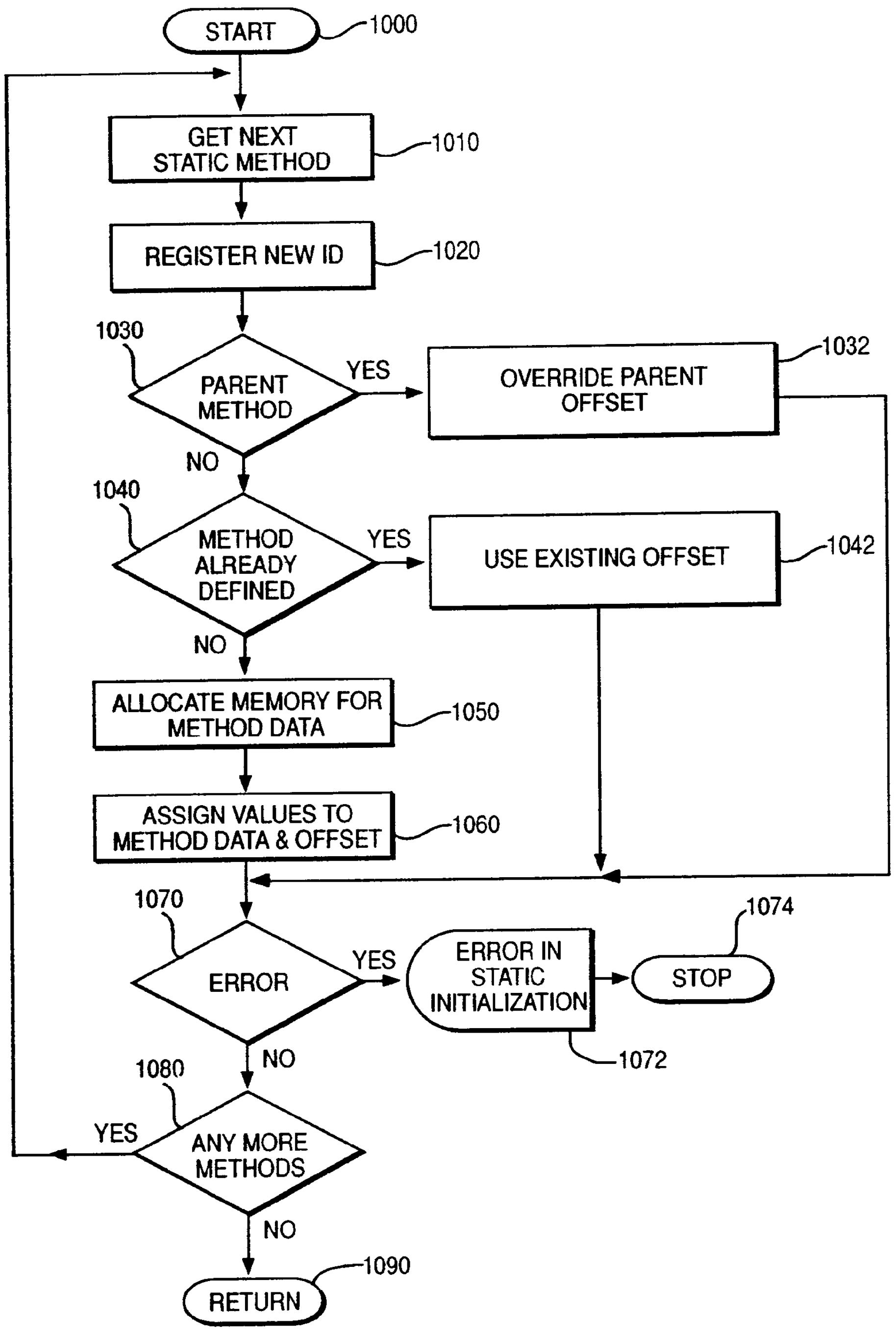

START
1000
GET NEXT STATIC METHOD
1010
REGISTER NEW ID
1020
1030
PARENT METHOD
YES
NO
OVERRIDE PARENT OFFSET
1032
1040
METHOD ALREADY DEFINED
YES
NO
USE EXISTING OFFSET
1042
ALLOCATE MEMORY FOR METHOD DATA
1050
ASSIGN VALUES TO METHOD DATA & OFFSET
1060
1070
ERROR
YES
NO
ERROR IN STATIC INITIALIZATION
1072
STOP
1074
1080
ANY MORE METHODS
YES
NO
RETURN
1090

FIG. 10

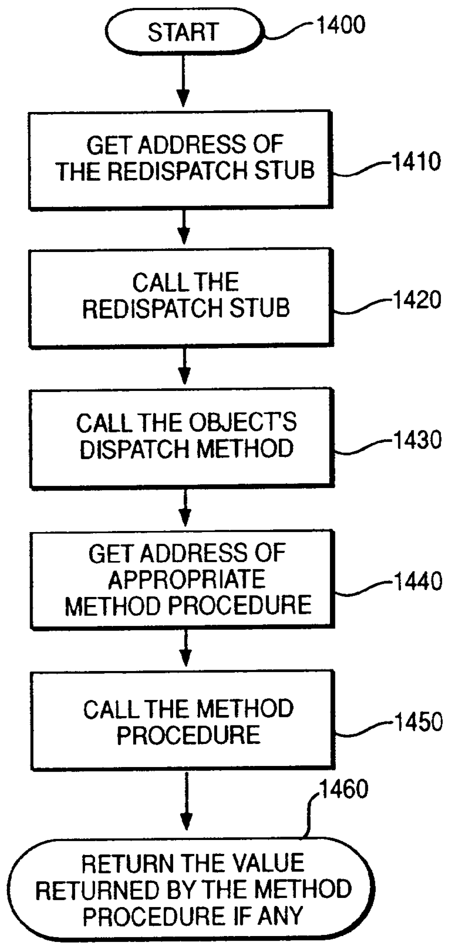

START
1400
GET ADDRESS OF THE REDISPATCH STUB
1410
CALL THE REDISPATCH STUB
1420
CALL THE OBJECT'S DISPATCH METHOD
1430
GET ADDRESS OF APPROPRIATE METHOD PROCEDURE
1440
CALL THE METHOD PROCEDURE
1450
1460
RETURN THE VALUE RETURNED BY THE METHOD PROCEDURE IF ANY

FIG. 14

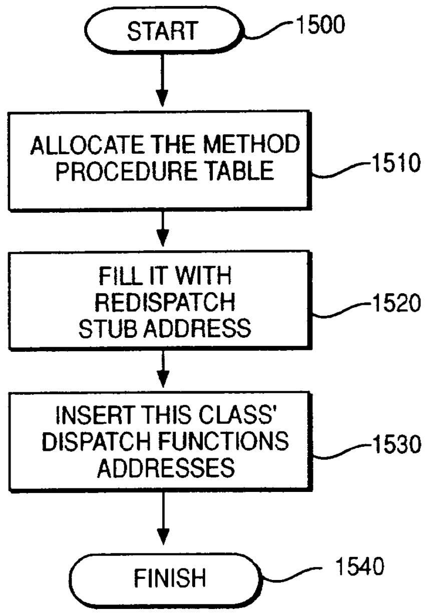

START
1500
ALLOCATE THE METHOD PROCEDURE TABLE
1510
FILL IT WITH REDISPATCH STUB ADDRESS
1520
INSERT THIS CLASS' DISPATCH FUNCTIONS ADDRESSES
1530
FINISH
1540

FIG. 15

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 697 780 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
21.02.1996 Bulletin 1996/08

(51) Int Cl.6: **H04M 3/50**

(21) Application number: **95305595.1**

(22) Date of filing: **10.08.1995**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **19.08.1994 GB 9416823**

(71) Applicant: **International Business Machines Corporation**
Armonk, N.Y. 10504 (US)

(72) Inventors:
- **Martin, David G.**
  Eastleigh, Hampshire, S053 4SR (GB)
- **Porter, Larry L.**
  Lyndhurst, Hampshire, S043 7EA (GB)

(74) Representative: **Davies, Simon Robert**
Winchester, Hampshire SO21 2JN (GB)

(54) Voice response system

(57) The present invention relates to system for varying the voice menus and segments presented to the user of a voice response system according to the competence of the user. The response time of a user to voice prompts is measured and an average response time is determined. It is assumed that the lower the average response time, the greater the competence of the user. The average response time is used as an index to a table of ranges of response times. Each range has respective voice segments associated therewith. The voice segments comprise oral instructions or queries for the user and vary according to the anticipated competence of the user. If the average response time changes such that the voice segments indexed are different to the current voice segments then a data base containing information relating to user competence is updated to reflect such a change. Accordingly, when the user next interacts with the voice response system a new set of voice segments more appropriate to the user's competence with be played.

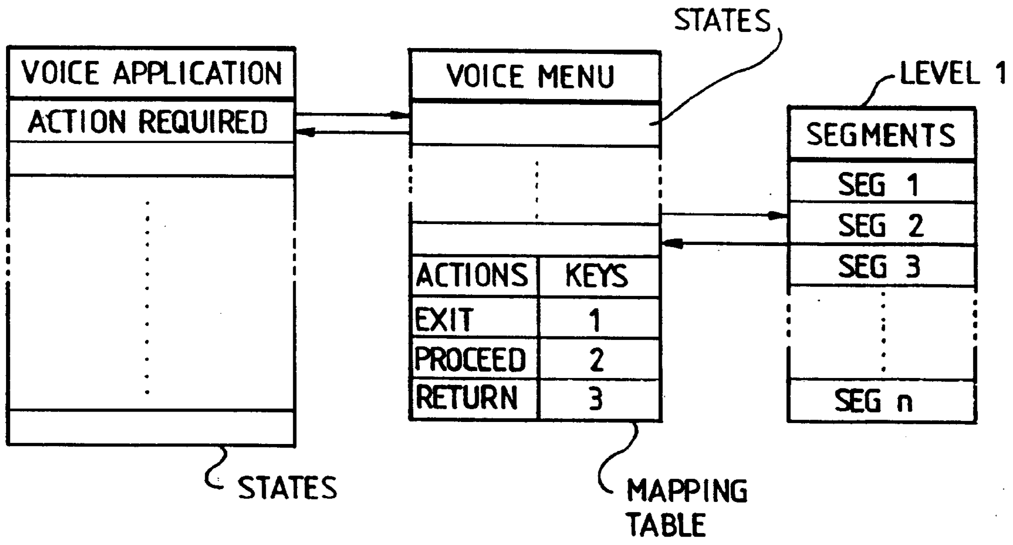


FIG. 2

Printed by Jouve (FR), 18, rue Saint-Denis, 75001 PARIS

Description

The present invention relates to a voice response system having dynamic voice menus.

Voice response systems enable users thereof to access information using a conventional telephone. The interaction between the users and the system comprises various voice prompts output by the system and responses thereto input, via the telephone keypad, by the user. Voice response systems are used by service providers, such as banks, to fully or partially automate telephone call answering or responding to queries. Typically a voice response system provides the capability to play voice prompts comprising recorded voice segments or speech synthesised from text and to receive responses thereto. The prompts are generally organised in the form of voice menus invoked by state tables. A state table can access and play a voice segment or synthesise speech from given text. The prompts are usually part of a voice application which is designed to, for example, allow a customer to query information associated with their various banks accounts.

An example of such a voice response system is the IBM CallPath DirectTalk/6000 product as described in "IBM CallPath DirectTalk/6000 General Information and Planning" and "IBM CallPath DirectTalk/6000 Voice Application Development" (IBM, DirectTalk, DirectTalk/6000 and CallPath are trade marks of International Business Machines Corporation).

The IBM DirectTalk/600 product provides voice mail capabilities. Voice mail provides features such as those found in a telephone answering machine together with the capability to manipulate any stored messages. For example, if a subscriber wishes to listen to the messages stored, the voice mail will use a voice response system to indicate how many messages have been received, at what time and, possibly, from whom. The list of messages are manipulated using various voice menus or prompts presented to the subscriber by the voice response system. The voice response system typically asks the subscriber whether or not the messages are to be stored or forwarded to another subscriber.

Facsimile mail systems also use voice response systems in a similar manner to voice mail systems. Subscribers of facsimile system can manipulate stored facsimiles or have selected documents faxed to a specified facsimile number. Again, the voice response system present the subscriber with various options or menus which are used to manipulated the facsimiles. Actions are selected from the voice menus using the DTMF tones generated by conventional DTMF telephones. US patent number 4, 918, 722 and US patent number 4,974,254 disclose methods and system for retrieving facsimile data using voice response systems.

As the users of such system may not be familiar with the use thereof, it is necessary to ensure that the instructions or voice prompts are sufficiently comprehensive to allow an novice user to successfully interact with the system.

However, the more competent users are in using a particular voice response system the more they begin to anticipate the various voice prompts and it becomes increasingly tedious for them to have to listen to such comprehensive instructions when more succinct instructions would suffice.

Accordingly, the present invention provides a voice response system comprising
means for conducting a telephone call with a user,
means for storing a plurality of sets of voice prompts, each voice prompt comprising at least one voice segment which is capable of being played to said user,
means for selecting one from said plurality of sets of voice prompts for use during the telephone call,
means for outputting a voice prompt from said selected set of voice prompts to said user and for receiving a response thereto from the user, said response indicating to the system the user's requirements.

Therefore, when a particular user instigates, for example, a telephone call the system selects a set from the plurality of sets of voice prompts appropriate to the competence of the user. The sets of voice prompts are varied according to the anticipated or actual user competence. The use of more succinct voice prompts increases the speed with which a competent user can interact with the system.

An embodiment provides a system further comprising means for determining the competence with which said caller interacts with said system, and wherein said means for selecting selects a voice prompt according to the determined competence of the caller.

Therefore, the competence with which a user interacts with the system can be measured and the voice prompts selected for use during the current interaction or future interactions can be varied according to said measurements.

A further embodiment provides a system wherein the means for determining the competence comprises
means for determining the response time between playing a voice prompt to the user and receiving said response thereto, and
wherein said means for selecting is responsive to said means for determining a response time to select a voice menu according to the response time.

Therefore, one way in which the competence of the user can be gauged is, for example, by measuring the response time of a user to a voice prompt and assuming that a short response time is indicative of great familiarity with the voice response system. It is reasonable to assume that the faster the response the greater the competence of the user.

A still further embodiment provides a system wherein said means for receiving a telephone call comprises a plurality of telephone interfaces each capable of receiving a telephone call from said caller, and said system further comprises
means for identifying upon which telephone interface a

telephone call made by said caller was received, and wherein
said means for selecting selects a set of voice prompts according to which telephone interface received the telephone call from said caller.

If several users have the same telephone, each user can be given a telephone number with which they can access the system and the voice prompt output to the user depends upon which number they used to access the system. Each such given number has associated therewith respective voice data, said voice data reflecting the anticipated competence of the users.

Alternatively, the selection of the set of voice prompts can be matched to individual callers. The users can be identified in many different ways. For example, if different user's have unique respective telephone numbers, the call identification code of a telephone call can be used as an index to data stored in a user data base comprising information relating to the competence of a user.

Alternatively, the user can be asked to enter a password before further access is allowed to the system. The password can then serve as an index to the stored data associated with the user. The stored data identifies which set of voice data is appropriate for use during an interaction with said user.

Alternatively, determining the number of times per day which a user accesses the system or the length of time which a user has subscribed to such a system may also be indicative of their competence.

A further embodiment provides means for determining how many times a user has accessed said system, and
means for accessing and amending said stored data according to said determination.

An embodiment of the present invention will now be described, by way of example only, with reference to the accompanying drawings in which:

figure 1 shows schematically a voice response system,

figure 2 illustrates schematically data structures used in a voice response system: namely a voice application, voice menu and voice segment tables,

figure 3 shows schematically the voice application of figure 2,

figure 4 illustrates schematically the voice menu of figure 2 comprising voice states for playing and receiving responses to voice prompts and determining the speed of the response thereto,

figure 5 shows the voice segment table comprising a plurality of voice segments,

figure 6 illustrates a table comprising a plurality of average response times together with respective voice segments,

figure 7 shows a flow diagram illustrating the operation of the voice menu in the voice response system.

Referring to figure 1 there is schematically shown a voice response system VRS, implemented using DirectTalk/6000, comprising an telephony interface TI, such as T1D4-Mode 3 interface for T1 or a CCITT G.703 interface for E1, for receiving from or transmitting to a telephone T via a communication network CN voice and signalling data. The voice response system VRS further comprises a voice menu data base VMDB in which voice menus Voice_menu$_1$ to Voice_menu$_n$ are stored in the form of voice tables together with a plurality of sets of voice prompts, a processor for executing voice applications, controlling access to the voice menu data base and, in conjunction with a timer, determining, via a voice menu, the average speed of response of a user to the voice prompts. The timer is provided via AIX system services. Each set of voice prompts comprises voice data which are played to the user during an interaction with the system. Each set of voice prompts is also designed for different levels of user competence. The system also comprises a user data base UDB which contains information from which a decision relating to the level of competence of the users can be made. The information allows appropriate voice prompts to be selected, and used in a voice menu, for user interaction with the system. The user data base also contains an identification of the current voice segments which are or will be used for any such interaction. A voice prompt for a reasonably competent user may be as follows:

For local maps and street plans, press 1,
For country maps and atlases, press 2,
For other countries, press 3,
For other types of maps, press 4.

It can be seen from the above example that the voice prompt comprises all four options in a single voice segment. An alternative way of viewing the above voice prompts is to consider it as a voice menu comprising a number of options to which only one response is required. The above example can also be constructed from a plurality of separate voice segments which are played contiguously to the caller.

By contrast a set of voice prompts which may be suitable for a less competent user may be as follows:

For local maps and street plans, press 1, for next item press 3
(await response)
For country maps and atlases, press 1, to go back press 2, for next item press 3,
(await response)
For other countries 1, to go back press 2, for next item press 3,
(await response)

For other types of map, press 1, to go back press 2, for next item press 3.

(obtain response)

It can be seen in the above example that the voice prompt comprises a plurality of voice segments and the user has an opportunity to respond to each segment before proceeding to the next segment. An alternative way of viewing the above voice prompt is to consider it to be a voice menu comprising a plurality of voice segments each of which requires a response from the caller.

Referring to figure 2, there are schematically shown data structures used in the voice response system to facilitate interaction with the user. The voice application controls the overall commands of the voice response system. The voice menu presents to a user at least one voice prompt comprising a set of voice segments and determines the speed of response thereto. The voice menu also updates the stored data concerning the level of the user's competence when appropriate.

Figure 3 shows schematically the voice application. Voice applications handle information requests and perform the data processing within a voice response system VRS. A voice application typically comprises at least one state table having a sequence of states or instructions which accomplish the aim of the voice application. Each state of the voice application performs a given function such as update account or obtain a command from a caller so that a decision as to how to continue can be made. Whenever a command is required by the voice application it calls a voice menu. The voice menu outputs to the user appropriate voice prompts or segments and receives responses thereto.

Figure 4 illustrates schematically the voice menu. The voice menu comprises at least the following states which are executed in the order presented: "Reset Timer" which resets the timer of the voice response system, "Play Prompt" which plays at least one identified voice segment or set of voice segments, "Receive Input" which receives the response to a prompt or voice segment on an interrupt driven basis, "Stop Timer" which stops the timer and records the time indicated, "Calculate" calculates the user's average response time to the voice segments (described in further detail below), "Update" updates, when necessary, the information relating to the user's competence which is stored in the user data base, "Determine Command" determines which one of a plurality of commands should be returned to the voice application, and "Return Command" which returns said one of a plurality of commands to the voice application. The voice menu also has a look-up table which contains the plurality of commands and corresponding keys. When the user selects a particular key in response to a prompt, that key is used to index the corresponding command in the table and the command so indexed is returned to the voice application.

The execution of the voice application is determined according to the commands returned by the voice menus. The commands are determined by the responses to the voice prompts presented to the user. Each application having instigated the playing of a voice prompt expects to execute one of a plurality of possible future actions. For example, if the following voice menu was played to the user:

To exit the system, press 1;
To continue with the query, press 2;
To return to the previous menu, press 3.

The voice application will expect to receive from the voice menu one of three possible commands. The commands are represented symbolically as, for example, EXIT, CONTINUE and RETURN. It can be seen that the voice application only receives information relating to any future command to be taken and is not concerned with how a returned command was obtained. For example, if the voice prompt was changed to:

To exit the system, press 7;
To continue with the query, press 8;
To return to the previous menu, press 9.

The application will still receive one of the three symbolic commands EXIT, CONTINUE or RETURN even though different keys were depressed to indicate to the voice menu the desired command or course of action.

The advantage of having the main voice application guided by symbolic commands becomes evident when one modifies either voice application or the voice prompts. Modifications can be effected to either the voice application or voice menu independently of the other. Therefore, as illustrated above, a change to the keys which must be depressed in order to guide the voice application does not necessitate a corresponding modification to the voice application to accommodate such a change. The voice menu is configured to expect different key depression but maps the different keys to the symbolic commands accordingly. In this way the operation and maintenance of the main voice application is isolated from the user interface or voice menus.

The sets of voice segments comprise a collection of voice segments which can be used to form a voice menu and to interact with the user. A schematic example of a voice segment table is shown in figure 5. For example, segment 1, plays the welcome message heard by users when they initially access the system.

Referring again to figure 1, the timer T is used to record the elapsed time between the beginning of a voice segment and the response thereto. Immediately prior to the voice menu playing, for example, voice segments embodying a list of options for which a response is required, the state, "Reset Timer", which resets and invokes the timer is executed. The user responds to the voice segments by pushing one of the keys of their telephone pad. The system detects the response, on an interrupt driven basis, by listening for an appropriate DTMF frequency in the conventional manner. When a response to the voice segments is detected the timer is stopped and the elapsed time between the beginning of the voice segments and the response thereto is determined.

The voice response system operates as follows.

Upon initialisation the voice response system awaits an incoming telephone call. The voice response system uses the calling identification to retrieve information from the user data base which is suitable for use in the voice menus. That is, a set of voice prompts are identified which is suitable for the forth coming interaction. Accordingly, the name of the suitable set of voice segments is loaded into the voice menu under the heading of "Voice Segment Table" as shown in figure 4.

When the system answers the incoming call it plays a prompt welcoming the user as is conventional and then presents the user with a number of options. As the voice application at this stage requires an command in order to continue processing, a voice menu requesting an input from the user is invoked. The voice menu resets the timer T to zero and then commences playing the voice segments 1 to n identified by the "Voice Segment Table" field of the voice menu. The user will be expected to respond to the voice segments by pressing one of the keys on their telephone key pad. When the user presses a key, the voice segments currently being played are interrupted and the key so pressed is noted by the "Receive Input" state. Alternatively, if all of the voice segments have finished the system awaits a response for a predetermined time. After the predetermined time has elapsed the system assumes that the user has hung up and returns to a state awaiting the next incoming call. Assuming the user has responded, the timer is stopped via state "Stop Timer" and the time is recorded. The call identification of the incoming call is used, by the state "Calculation", to locate and retrieve information relating to the particular user. The information is stored in the user data base UDB. The information comprises the following: calling number identification number, average response time and number of determined response times. The "Calculation" state updates the average response time information as follows:
new average response time = (average response time X number of recorded times + current response time)/ (number of determined response times + 1).

The information relating to the number of determined response times is also incremented by one and stored. The "Update" state updates the stored data in the user data base in a manner described below. The "Determine Command" state maps the users response to one of the plurality of possible commands expected by the voice application. The "Return Command" state returns the determined command to the voice application.

Having received a command the voice application continues processing in the conventional manner until another command is required via another corresponding input from the user.

A suitable state table implementing the above is schematically shown in figure 4.

Once a new average response time has been determined a check is made by the "Update" state to ascertain whether or not the current set of voice segments is appropriate to the level of competence of the user. Referring to figure 6, there is shown a table comprising ranges of response times and respective sets of voice segments. The range into which the new average response time falls is determined and the corresponding set of voice segments identified. If the identified voice segments are different to current voice segments then the voice menu is updated to contain a reference to the identified set of voice segments. For example, if the current voice segments are voice_segs_1 and the new average response time fell within the range 10-20, the "Voice Segment Table" field of figure 4 would be changed from "Voice_segs_1" to "Voice_segs_2". The data stored in the user data base would also be updated to reflect the change in voice segment. Accordingly, when the user next accesses the system or when the voice menu is next invoked by the voice application the voice segments, voice_segs_2, contained within the newly identified set of voice segments will be used.

The sets of voice segments of figure 6 are tailored to reflect different levels of user competence. It is assumed that the lower response times are a reflection of greater user competence. The average response times shown in figure 6 are in arbitrary units. Voice_segs_1 may comprise the following voice segments which are all played contiguously

For local maps and street plans, press 1,
For country maps and atlases, press 2,
For other countries, press 3,
For other types of maps, press 4.

It can be seen that when using the above the user must remember all of the options or be very familiar therewith as there is no opportunity to listen to each individual voice segment again.

A suitable set of voice segments for a less competent user might be voice_segs_5 in which the voice segments are played separately as follow:

For local maps and street plans, press 1, for next item press 3
(await response)
For country maps and atlases, press 1, to go back press 2, for next item press 3,
(await response)
For other countries 1, to go back press 2, for next item press 3,
(await response)
For other types of map, press 1, to go back press 2, for next item press 3.
(obtain response)

It can be seen that voice_segs_5 allows the user more time to consider each voice segment or prompt separately and obviates the burden of remembering all of the possible options presented. Voice_segs 2 to 4 would represent sets of graduated voice segments which fall between voice_segs_1 and voice_segs_5.

Figure 7 shows a flow diagram schematically illustrating the operation of the voice menu in the voice response system. As a consequence of the voice application requiring an command in order to be able to proceed the voice menu is invoked. Step 700 resets the timer T. Step 705 commences playing the voice segments currently identified in by "Voice Segment Table" in the voice menu. The system awaits an input or is interrupted with such an input from the user via their telephone at step 710. At step 715 the timer is stopped and the time recorded. At step 720 the voice menu uses the calling identification to identify and retrieve stored data indicative of the level of competence of the user from the user data base. In an embodiment the retrieval of said information can be insitgated before an incoming telephone call is answered thereby playing a welcome message which is tailored to the callers competance. The data comprises an identification of voice segments appropriate to the level of competence of the user. The response time to the voice segments played to the user is determined step 720. Step 720 also calculates and updates the average response time to the voice segments. Next a determination is made as to whether or not the data stored in the user data base identifying the set of voice segments appropriate to a user should be updated to reflect the familiarity of that user with the voice response system. The determination is made by comparing the average response time with a series a ranges of response times at step 725. The range into which the average response time falls is determined together with the associated or new voice segments. If the new voice segments are different to the current voice segments, the data stored in the user data base is amended to store a reference to the new voice segments thereby reflecting a change in competence of the user at steps 730 and 735. If the new set of voice segments is the same as the current set of voice segments there is no need to amend the data stored in the user data base. Having amended the stored data or determined that there is no need to amend the data, the command to be returned to the voice application by the voice menu is determined at step 740 as follows. The response to the voice segments is used as index to the commands table. The command corresponding to the index or response is then returned to the voice application at step 745. Execution of the voice menu is terminated at step 750 and execution of the voice application recommences in accordance with the returned command.

Although the above embodiment gauges user competence by measuring the response time to voice prompt, the present invention is not limited thereto. User competence can equally well be determined by, for example, keeping track of the total number of time which a user has accessed the system and varying the voice data according to said number.

Further, a voice response system can be realised comprising many or all of the above techniques. For example, having gained initial access to the voice response system via one of a plurality of possible telephone interfaces and accordingly retrieved voice prompt information associated with the telephone interface, the competence of the user can be monitored and the various voice prompts presented can be varied accordingly.

Similarly, the embodiment requiring a pass word to be enter before allowing full system access can also include the means for measuring the speed of response of the user and varying the stored data and voice prompts accordingly.

The present invention can be used to vary either the voice menus used by the voice application or the voice segments accessed by the voice menus.

Although embodiments have been described in which the voice menus are dynamically varied during a call the present invention is not limited thereto. An embodiment can equally well be realised in which an average response time for a complete interaction with the user is determined and the stored data is accessed and amended only after the conclusion of the interaction. Such an embodiment would reduce the amount of time the system needed to update data associated with the user.

Further, even though the above embodiments describe systems in terms of an incoming call, the present invention can equally well be used to tailor the voice prompts for outgoing calls. The system, prior to instigating a call to a particular user, would identify from the user data base, using, for example, the user's telephone number, stored data indicative of the level of competance of the user. The identified stored data would then be used to determine suitable sets of voice prompts to be used for the interaction with the caller user.

Claims

1. A voice response system comprising
means for conducting a telephone call with a user,
means for storing a plurality of sets of voice prompts, each voice prompt comprising at least one voice segment which is capable of being played to said user,
means for selecting one from said plurality of sets of voice prompts for use during the telephone call,
means for outputting a voice prompt from said selected set of voice prompts to said user and for receiving a response thereto from the user, said response indicating to the system the user's requirements.

2. A system as claimed in claim 1, further comprising means for determining the competence with which said user interacts with said system, and wherein said means for selecting selects a voice prompt according to the determined competence of the user.

3. A system as claimed in claim 2, wherein said means for determining the competence comprises
means for determining the response time between playing a voice prompt and receiving said response

thereto, and
wherein said means for selecting is responsive to said means for determining a response time to select a set of voice prompts according to the response time.

4. A system as claimed in either of claims 2 or 3, further comprising
means, responsive to said means for determining the competence with which said user interacts with said system, for accessing and amending stored data reflecting said competence.

5. A system as claimed in claim 1, wherein said means for receiving a telephone call comprises a plurality of telephone interfaces each capable of receiving a telephone call from said user, and said system further comprises
means for identifying upon which telephone interface a telephone call made by said user was received, and wherein
said means for selecting selects a set of voice prompts according to which telephone interface received the telephone call from said user.

6. A system as claimed in any preceding claim, further comprising
means for identifying the user, and
means, responsive to said identification, for accessing stored data associated with the user, the stored data being indicative of the competence of the user, and wherein
said means for selecting is responsive to said stored data.

7. A system as claimed in claim 6, further comprising
means for recording how many times a user has interacted with the system, and
means for accessing and amending said stored data according to said determination.

8. A system as claimed in any preceding claim, further comprising voice menus and means for modifying said sets of voice segments independently of said voice menus, said means for modifying comprising memory for storing a table comprising a plurality of indices, each index corresponding to a possible user response, and respective data corresponding to respective commands, wherein said voice menus are responsive to said commands,
means for indexing using a user response one of said plurality of commands, and
means for guiding the operation of said system according to said one of said plurality of commands.

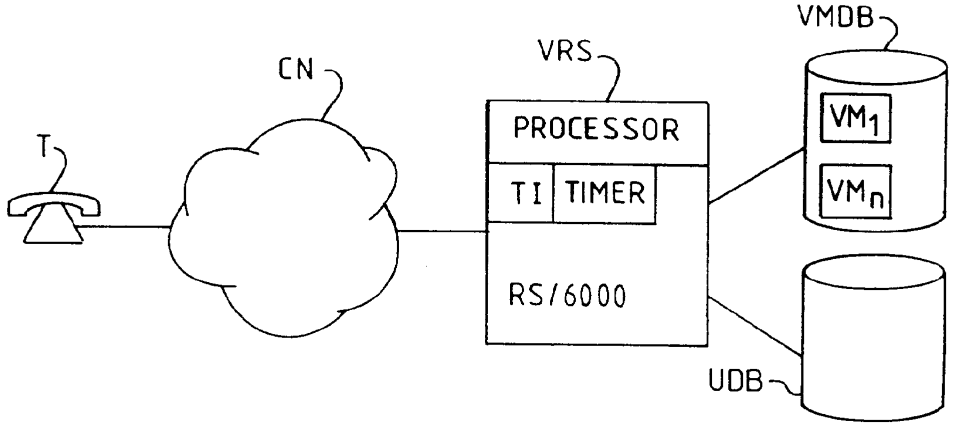

VMDB
VRS
CN
T
PROCESSOR
TI
TIMER
RS/6000
VM1
VMn
UDB

FIG 1

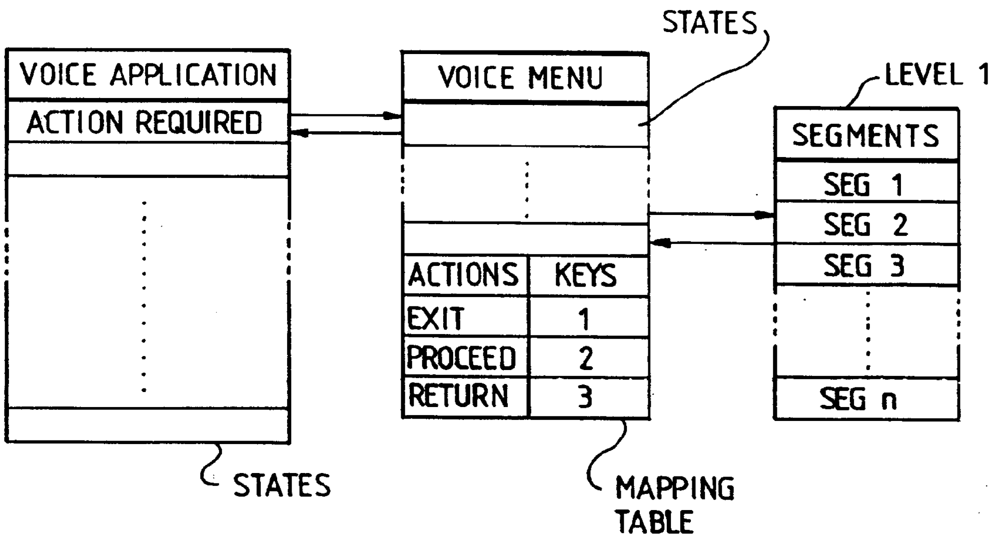

STATES
LEVEL 1
VOICE APPLICATION
ACTION REQUIRED
VOICE MENU
SEGMENTS
SEG 1
SEG 2
SEG 3
SEG n
ACTIONS
KEYS
EXIT
1
PROCEED
2
RETURN
3
STATES
MAPPING TABLE

FIG. 2

STATE TABLE NAME : ACCOUNT BALANCE			
STATE LABEL	ACTION	POSSIBLE RESULTS	NEXT STATE
START	ANSWER CALL	SUCCEED NOT SUCCEED	WELCOME, EXIT
WELCOME / RETURN	OBTAIN COMMAND	SUCCEED, FAIL	PROCEED, EXIT RETURN
PROCEED			
⋮	⋮	⋮	⋮
EXIT	CLOSE EVERYTHING		

FIG. 3

VOICE MENU VOICE SEGMENT TABLE : voice_segs_1		
STATE	NEXT STATE	PARAMETERS
RESET TIME		
PLAY PROMPT	RECEIVE INPUT	SEG 1 TO n
RECEIVE INPUT	STOP TIMER	BUFFER 1
STOP TIMER	CALCULATION	NONE
CALCULATION	UPDATE	RESPONSE TIME
UPDATE	DETERMINE COMMAND	VOICE RANGE TABLE, DATA BASE ACCESS
DETERMINE COMMAND	RETURN COMMAND	COMMAND TABLE
RETURN COMMAND		DETERMINED COMMAND

COMMAND	KEY (INDEX)
EXIT	0
PROCEED	1
RETURN	2

FIG. 4

VOICE SEGMENT NAME : voice_segs_1		
SEGMENT ID	VOICE SEGMENT	DURATION
1	WELCOME TO IBM VOICE RESPONSE SYSTEM	3s
⋮	⋮	⋮
n	KEY PRESSED IS INVALID	2s

FIG. 5

RANGE OF AVERAGE RESPONSE TIMES	VOICE SEGMENTS
0 – 10	voice_segs_1
11 – 20	voice_segs_2
21 – 30	voice_segs_3
31 – 40	voice_segs_4
41 – 50	voice_segs_5

FIG. 6

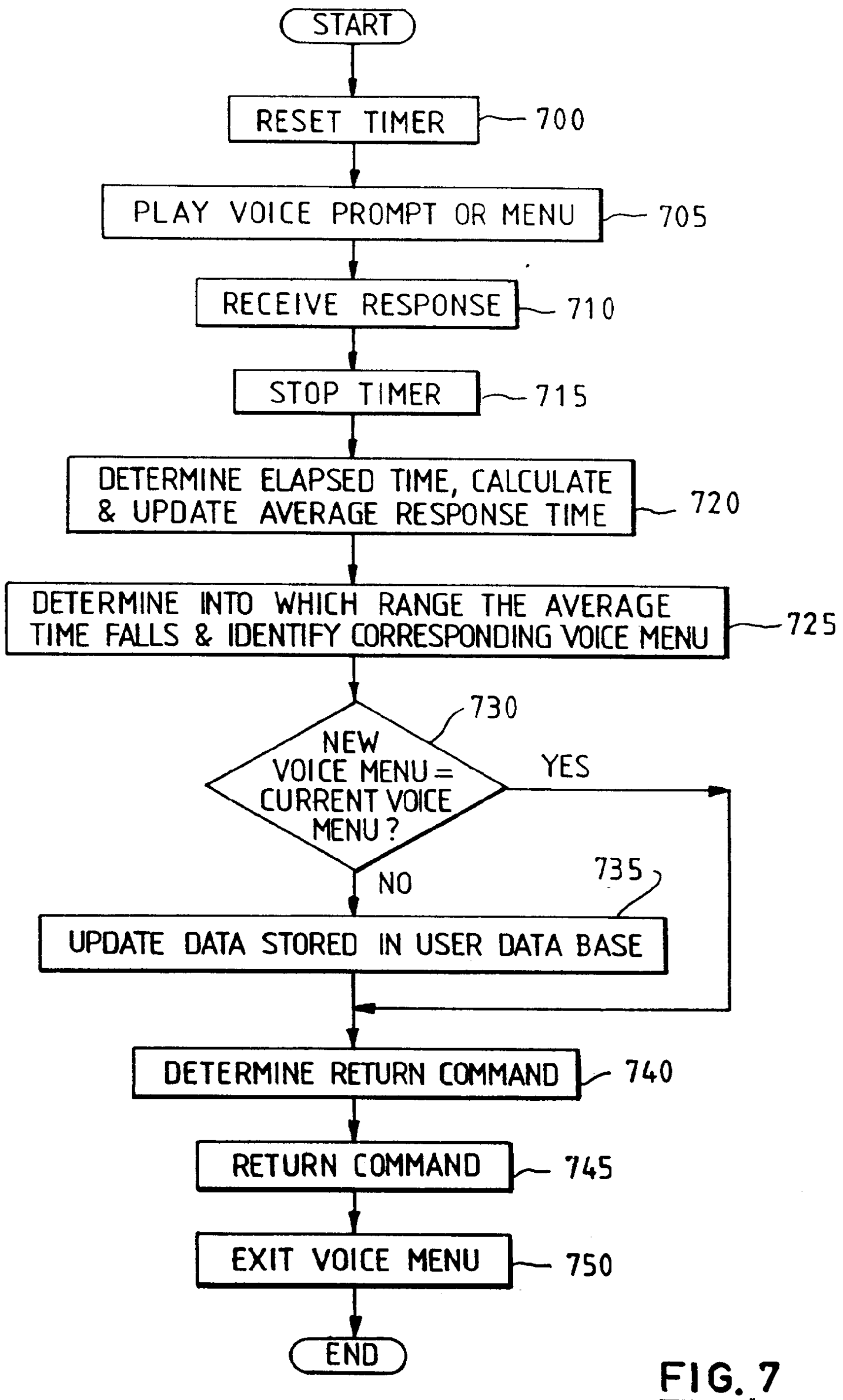

START
RESET TIMER
700
PLAY VOICE PROMPT OR MENU
705
RECEIVE RESPONSE
710
STOP TIMER
715
DETERMINE ELAPSED TIME, CALCULATE & UPDATE AVERAGE RESPONSE TIME
720
DETERMINE INTO WHICH RANGE THE AVERAGE TIME FALLS & IDENTIFY CORRESPONDING VOICE MENU
725
730
NEW VOICE MENU = CURRENT VOICE MENU ?
YES
NO
735
UPDATE DATA STORED IN USER DATA BASE
DETERMINE RETURN COMMAND
740
RETURN COMMAND
745
EXIT VOICE MENU
750
END

FIG. 7

Electronic Acknowledgement Receipt

EFS ID:	2814146
Application Number:	10995159
International Application Number:	
Confirmation Number:	5640
Title of Invention:	Network system extensible by users
First Named Inventor/Applicant Name:	Danny Lange
Customer Number:	26111
Filer:	Lori Ann Gordon/Michael Wain
Filer Authorized By:	Lori Ann Gordon
Attorney Docket Number:	2222.0300002
Receipt Date:	05-FEB-2008
Filing Date:	24-NOV-2004
Time Stamp:	17:31:08
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes) /Message Digest	Multi Part /.zip	Pages (if appl.)
1		2222_0300002_IDS_filing2.pdf	743407 bf5cac2f65b23f6d0233a31802e7a91b4850b3b4	yes	15

	Multipart Description/PDF files in .zip description		
	Document Description	**Start**	**End**
	Information Disclosure Statement Letter	1	7
	Information Disclosure Statement (IDS) Filed	8	15

Warnings:

Information:

2	Foreign Reference	FP_1.pdf	1317018 ae477397ea7665a7b9c34e4a49559cd77ae07825	no	33

Warnings:

Information:

3	Foreign Reference	FP_2.PDF	5069178 6c41d2716a963a65f7df31261bcc7503e5f2bcdb	no	142

Warnings:

Information:

4	Foreign Reference	FP_3.pdf	1096210 d612d4c887aaaa5430d9c484f21cc20f1eac9d42	no	29

Warnings:

Information:

5	Foreign Reference	FP_4.pdf	2286186 c71907dd3c173bfcd941fc07900231457ba49a77	no	39

Warnings:

Information:

6	Foreign Reference	FP_5.pdf	884965 a0f7e51c15ef297e9700198fad4eed8d69113251	no	21

Warnings:

Information:

7	Foreign Reference	FP_6.pdf	1883437 a8e5d03362cbe204e63dfecaa3776b996396cdac	no	37

Warnings:

Information:

8	Foreign Reference	FP_7.pdf	621876 3de3ab080f540b3eecbd8712d8c46f91a7382e52	no	11

Warnings:

Information:

9	NPL Documents	NPL_1.PDF	2823346 86ff1246c71dd0aa630bb3bb068441c30136c426	no	14
Warnings:					
Information:					
10	NPL Documents	NPL_2.PDF	1737887 a19a02273d3534642c64e58eee2c2afa1b008724	no	9
Warnings:					
Information:					
11	NPL Documents	NPL_3.PDF	1753467 3d469b9d054a098e8854c2852f8cb32d18e2078e	no	9
Warnings:					
Information:					
12	NPL Documents	NPL_4.PDF	1153188 d98f97f6a876bea7a6efd56d12b7c13f76948340	no	8
Warnings:					
Information:					
13	NPL Documents	NPL_5.PDF	2993087 6027da138eef24ff38cb7ab13b742b091403b175	no	22
Warnings:					
Information:					
14	NPL Documents	NPL_6.PDF	1007036 4af637961410830d7918e37b19f47a174f984980	no	18
Warnings:					
Information:					
15	NPL Documents	NPL_7.PDF	1110946 c145fad6a2ccb1b359b1b7fc15f0beb342fdf8fb	no	7
Warnings:					
Information:					
16	NPL Documents	NPL_8.pdf	466898 a288b5dd9a2d14d11c7aed35c14467e48ac8ce97	no	9
Warnings:					
Information:					
17	NPL Documents	NPL_9.PDF	2311831 88db0517f6eb9c9ae26154bf33fc38d78d8ebc04	no	9
Warnings:					
Information:					

18	NPL Documents	NPL_10.PDF	1993995 babae6ad4df2cb530b309eabf7986d58 6c5bbde3	no	11
Warnings:					
Information:					
19	NPL Documents	NPL_11.PDF	2374249 5608288a24c2bdc8f886af9cb1189100c 6d90cfa	no	11
Warnings:					
Information:					
20	NPL Documents	NPL_12.PDF	2452436 2616885ccc8b8b950862146cf593f8474 ba15a04	no	31
Warnings:					
Information:					
			Total Files Size (in bytes):	36080643	

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/995,159	11/24/2004	Danny Lange	2222.0300002	5640

26111 7590 12/15/2008
STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.
1100 NEW YORK AVENUE, N.W.
WASHINGTON, DC 20005

EXAMINER	
BLAIR, DOUGLAS B	

ART UNIT	PAPER NUMBER
2442	

MAIL DATE	DELIVERY MODE
12/15/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

PTOL-90A (Rev. 04/07)

Office Action Summary	Application No. 10/995,159	Applicant(s) LANGE ET AL.	
	Examiner DOUGLAS B. BLAIR	Art Unit 2442	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *1* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *24 November 2004*.
2a)☐ This action is **FINAL**. 2b)☐ This action is non-final.
3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *77-97* is/are pending in the application.
4a) Of the above claim(s) _____ is/are withdrawn from consideration.
5)☐ Claim(s) _____ is/are allowed.
6)☐ Claim(s) _____ is/are rejected.
7)☐ Claim(s) _____ is/are objected to.
8)☒ Claim(s) *77-97* are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.
10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
a)☐ All b)☐ Some * c)☐ None of:
1.☐ Certified copies of the priority documents have been received.
2.☐ Certified copies of the priority documents have been received in Application No. _____.
3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

Election/Restrictions

Restriction to one of the following inventions is required under 35 U.S.C. 121:

I. Claims 77-85, drawn to a system architecture comprising an agent and a agent server, and service wrapper (same invention claimed in parent cases, now patents 6,839,733 and 6163794) , classified in class 709, subclass 202.

II. Claims 86-97, drawn to methods and a system for allowing a user to create and invoke and agent for using a service and service resource to perform an operation, classified in class 709, subclass 227.

The inventions are distinct, each from the other because of the following reasons:

Inventions I and II are related as combination and subcombination. Inventions in this relationship are distinct if it can be shown that (1) the combination as claimed does not require the particulars of the subcombination as claimed for patentability, and (2) that the subcombination has utility by itself or in other combinations (MPEP § 806.05(c)). In the instant case, the combination as claimed does not require the particulars of the subcombination as claimed because the combination does not require the process of Invention II in order to be functional. The subcombination has separate utility such as being a process that is not claimed as being tied in any way to the architecture of Invention I.

The examiner has required restriction between combination and subcombination inventions. Where applicant elects a subcombination, and claims thereto are subsequently found allowable, any claim(s) depending from or otherwise requiring all the limitations of the allowable subcombination will be examined for patentability in accordance with 37 CFR 1.104. See MPEP § 821.04(a). Applicant is advised that if any claim presented in a continuation or divisional application is anticipated by, or includes all the limitations of, a claim that is allowable in the present application, such claim may be subject to provisional statutory and/or nonstatutory double patenting rejections over the claims of the instant application.

Restriction for examination purposes as indicated is proper because all these inventions listed in this action are independent or distinct for the reasons given above and there would be a serious search and examination burden if restriction were not required because one or more of the following reasons apply:

(a) the inventions have acquired a separate status in the art in view of their different classification;

(b) the inventions have acquired a separate status in the art due to their recognized divergent subject matter;

(c) the inventions require a different field of search (for example, searching different classes/subclasses or electronic resources, or employing different search queries);

(d) the prior art applicable to one invention would not likely be applicable to another invention;

(e) the inventions are likely to raise different non-prior art issues under 35 U.S.C. 101 and/or 35 U.S.C. 112, first paragraph.

Applicant is advised that the reply to this requirement to be complete must include (i) an election of a invention to be examined even though the requirement may be traversed (37 CFR 1.143) **and (ii) identification of the claims encompassing the elected invention**.

The election of an invention may be made with or without traverse. To reserve a right to petition, the election must be made with traverse. If the reply does not distinctly and specifically point out supposed errors in the restriction requirement, the election shall be treated as an election without traverse. Traversal must be presented at the time of election in order to be considered timely. Failure to timely traverse the requirement will result in the loss of right to petition under 37 CFR 1.144. If claims are added after the election, applicant must indicate which of these claims are readable on the elected invention.

If claims are added after the election, applicant must indicate which of these claims are readable upon the elected invention.

Should applicant traverse on the ground that the inventions are not patentably distinct, applicant should submit evidence or identify such evidence now of record showing the inventions to be obvious variants or clearly admit on the record that this is the case. In either instance, if the examiner finds one of the inventions unpatentable over the prior art, the evidence or admission may be used in a rejection under 35 U.S.C. 103(a) of the other invention.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DOUGLAS B. BLAIR whose telephone number is (571)272-3893. The examiner can normally be reached on 9:00am-5:30pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Andrew Caldwell can be reached on (571) 272-3868. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Douglas B Blair/
Primary Examiner, Art Unit 2442

Index of Claims	**Application/Control No.** 10995159	**Applicant(s)/Patent Under Reexamination** LANGE ET AL.
	Examiner DOUGLAS B BLAIR	**Art Unit** 2442

✓	**Rejected**	-	**Cancelled**	N	**Non-Elected**	A	**Appeal**
=	**Allowed**	÷	**Restricted**	I	**Interference**	O	**Objected**

☐ **Claims renumbered in the same order as presented by applicant** ☐ **CPA** ☐ **T.D.** ☐ **R.1.47**

CLAIM		**DATE**								
Final	Original	12/11/2008								
	77	÷								
	78	÷								
	79	÷								
	80	÷								
	81	÷								
	82	÷								
	83	÷								
	84	÷								
	85	÷								
	86	÷								
	87	÷								
	88	÷								
	89	÷								
	90	÷								
	91	÷								
	92	÷								
	93	÷								
	94	÷								
	95	÷								
	96	÷								
	97	÷								

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Confirmation No.: 5640
LANGE *et al.*	Art Unit: 2442
Appl. No.: 10/995,159	Examiner: Douglas B. Blair
Filed: November 24, 2004	Atty. Docket: 2222.0300002
For: **Network System Extensible By Users**	

Response to Restriction Requirement and Preliminary Amendment Under 37 C.F.R. § 1.115

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Sir:

In advance of examination and in response to the restriction requirement dated **December 15, 2008**, Applicants submit the following amendments and remarks.

The Claims are reflected in the listing of claims which begins on page 2 of this paper.

Remarks and the Response to the Restriction Requirement begin on page 9 of this paper.

It is not believed that extensions of time are required, beyond those that may otherwise be provided for in accompanying documents. However, if additional extensions of time are necessary to prevent abandonment of this application, then such extensions of time are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required therefor are hereby authorized to be charged to our Deposit Account No. 19-0036.

Amendments to the Claims

The listing of claims will replace all prior versions, and listings of claims in the application.

1-85. (Canceled)

86. (Currently Amended) A system for performing user customized network-based operations, comprising:

means for allowing a user to create a network-based agent associated with the user, wherein the network-based agent is configured to perform an operation on behalf of the user;

means for invoking the execution of the network-based agent on the occurrence of an event;

means, including the network-based agent, for using a service and a service resource when performing the operation on behalf of the user; and

means for communicating the result of the operation to the user over a network communications link.

87. (Previously Presented) The system of claim 86, wherein the network communications link is a communications link in a public-switched communications network.

88. (Previously Presented) The system of claim 87, further comprising:

means for mediating the interaction between the means for using the service and the service.

89. (Previously Presented) The system of claim 88, wherein the means for mediating comprises:

means for monitoring the amount of the service resource used by the network-based agent.

90. (Previously Presented) The system of claim 89, wherein the means for mediating further comprises:

means for converting between a first messaging protocol used by the network-based agent and a second messaging protocol used by the service.

91. (Previously Presented) The system of claim 86, further comprising:

means for allowing the user to modify the network-based agent associated with the user.

92. (Currently Amended) A tangible computer-readable medium ~~program product comprising a computer useable medium~~ having stored thereon computer-executable instructions ~~program logic recorded thereon for enabling a~~ that, if executed by a computing device, ~~processor in a computer system~~ cause the computing device to perform ~~user customized network-based operations,~~ a method comprising:

~~means for~~ enabling the processor to allow a user to create a network-based agent associated with the user, wherein the network-based agent is configured to perform an operation on behalf of the user;

~~means for~~ enabling the processor to invoke the execution of the network-based agent on the occurrence of an event;

~~means for~~ enabling the processor to use a service and a service resource, using the network-based agent, when performing the operation on behalf of the user; and

~~means for~~ enabling the processor to communicate the result of the operation to the user over a network communication link.

93. (Currently Amended) The ~~system~~ computer-readable medium of claim 92, further comprising:

~~means for~~ enabling the processor to allow the user to modify the network-based agent associated with the user.

94. (Currently Amended) A method for performing user customized computer network-based operations, comprising:

~~(a)~~ using a computing device, receiving data for creating an agent customized to perform a task for a user upon the occurrence of an event;

~~(b)~~ using the computing device, creating the agent, wherein the agent has a plurality of executable instructions for performing the task;

~~(c)~~ using the computing device, executing the agent instructions upon the occurrence of the event, ~~wherein step (c) includes~~ including:

~~(i)~~ providing instructions to a service to define the operations supported by the service required to perform the task,

~~(ii)~~ receiving a response from the service including parameters required by the agent to complete task, and

~~(iii)~~ providing an output associated with the task to the user over a network communications link.

95. (Previously Presented) The method of claim 94, wherein the response received from the service includes data.

96. (Previously Presented) The method of claim 94, wherein the instructions include a request to access a service resource.

97. (Previously Presented) The method of claim 94, wherein the network communications link is a communications link in a public-switched communications network.

98. (New) The system of claim 86, wherein the means for invoking the execution of the network-based agent comprises an agent server coupled to the agent and coupled to the user via a network communications link.

99. (New) The system of claim 86, further comprising a service wrapper associated with the service, wherein the service wrapper is configured to mediate the interaction between the service and the agent.

100. (New) The system of claim 98, wherein the agent server comprises:

an engine configured to control the operation of the agent server;

a scheduler coupled to the engine, wherein the scheduler is configured to trigger the execution of the agent upon occurrence of one or more events; and

an agent object coupled to the agent, wherein the agent object includes data and executable instructions associated with the agent.

101. (New) The system of claim 100, wherein the agent object comprises:

permission means associated with the agent; and

event handler means, including data and executable instructions for directing the operation of the engine upon the occurrence of the one or more events.

102. (New) The system of claim 101, wherein the permission means comprises a computational permission means defining one or more computational resources that the agent is permitted to use.

103. (New) The system of claim 102, wherein the computational permission means further defines the extent to which the agent is permitted to use the one or more computational resources.

104. (New) The system of claim 101, wherein the permission means comprises service permission means defining one or more services that the agent is permitted to use.

105. (New) The system of claim 104, wherein the service permission means further defines the extent to which the agent is permitted to use the one or more services.

106. (New) A tangible computer-readable medium having stored thereon computer-executable instructions that, if executed by a computing device, cause the computing device to perform a method comprising:

receiving data for creating an agent customized to perform a task for a user upon the occurrence of an event;

creating the agent, wherein the agent has a plurality of executable instructions for performing the task;

executing the agent instructions upon the occurrence of the event, including:

providing instructions to a service to define the operations supported by the service required to perform the task,

receiving a response from the service including parameters required by the agent to complete task, and

providing an output associated with the task to the user over a network communications link.

Remarks

Upon entry of the foregoing amendment, claims 86-106 are pending in the application, with claims 86, 92, 94, and 106 being the independent claims. Claims 86 and 92-94 are sought to be amended. Claims 1-76 were previously canceled without prejudice to or disclaimer of the subject matter therein. Claims 77-85 are sought to be canceled without prejudice or disclaimer of the subject matter therein. New claims 98-106 are sought to be added. Applicants reserve the right to prosecute similar or broader claims, with respect to the canceled and amended claims, in the future. These changes are believed to introduce no new matter, and their entry is respectfully requested.

Response to Restriction Requirement

In reply to the Office Action dated December 15, 2008, requesting an election of one invention to prosecute in the above-referenced patent application, Applicants hereby elect to prosecute the invention of Group II, represented by claims 86-106. This election is made without prejudice to or disclaimer of the other claims or inventions disclosed. This election is made without traverse. Reconsideration and allowance of all pending claims are respectfully requested.

Conclusion

Prompt and favorable consideration of this Preliminary Amendment is respectfully requested. Applicant believes the present application is in condition for allowance. If the

Examiner believes, for any reason, that personal communication will expedite prosecution of this application, the Examiner is invited to telephone the undersigned at the number provided.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Edward J. Kessler
Attorney for Applicant
Registration No. 25,688

Date: March 16, 2009

1100 New York Avenue, N.W.
Washington, D.C. 20005-3934
(202) 371-2600
928475_1.DOC

Electronic Patent Application Fee Transmittal

Application Number:	10995159
Filing Date:	24-Nov-2004
Title of Invention:	Network system extensible by users
First Named Inventor/Applicant Name:	Danny Lange
Filer:	Edward J. Kessler/Vanessa Burgess-Allen
Attorney Docket Number:	2222.0300002

Filed as Large Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				
Extension - 2 months with $0 paid	1252	1	490	490

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Miscellaneous:				
		Total in USD ($)		**490**

Electronic Acknowledgement Receipt	
EFS ID:	4973567
Application Number:	10995159
International Application Number:	
Confirmation Number:	5640
Title of Invention:	Network system extensible by users
First Named Inventor/Applicant Name:	Danny Lange
Customer Number:	26111
Filer:	Edward J. Kessler/Vanessa Burgess-Allen
Filer Authorized By:	Edward J. Kessler
Attorney Docket Number:	2222.0300002
Receipt Date:	16-MAR-2009
Filing Date:	24-NOV-2004
Time Stamp:	16:41:23
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Credit Card
Payment was successfully received in RAM	$490
RAM confirmation Number	2330
Deposit Account	
Authorized User	

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)

1		2222_0300002_RestrictionRequirement.pdf	348621 2000edd26edbb550fa2664098a2c34ab4f67e451	yes	10

Multipart Description/PDF files in .zip description

Document Description	Start	End
Miscellaneous Incoming Letter	1	1
Extension of Time	2	2
Response to Election / Restriction Filed	3	3
Claims	4	8
Applicant Arguments/Remarks Made in an Amendment	9	10

Warnings:

Information:

2	Fee Worksheet (PTO-06)	fee-info.pdf	29896 518e58d8f48cd543ffb3197ab0bafdfcaa92b3b1	no	2

Warnings:

Information:

Total Files Size (in bytes):	378517

Sterne Kessler Goldstein Fox
ATTORNEYS AT LAW

March 16, 2009

WRITER'S DIRECT NUMBER:
(202) 772-8550
INTERNET ADDRESS:
EKESSLER@SKGF.COM

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Art Unit 2442

Mail Stop Amendment

Re: U.S. Utility Patent Application
Application No. 10/995,159; Filed: November 24, 2004
For: **Network System Extensible By Users**
Inventors: LANGE *et al.*
Our Ref: 2222.0300002

Sir:

Transmitted herewith for appropriate action are the following documents:

1. Online Credit Card Payment Authorization in the amount of $**490.00** to cover two-month extension of time fee;
2. Petition for Extension of Time Under 37 CFR 1.136(a); and
3. Response to Restriction Requirement and Preliminary Amendment Under 37 C.F.R. § 1.115.

The above-listed documents are filed electronically through EFS-Web.

In the event that extensions of time are necessary to prevent abandonment of this patent application, then such extensions of time are hereby petitioned.

Fee payment is provided through online credit card payment. The U.S. Patent and Trademark Office is hereby authorized to charge any fee deficiency, or credit any overpayment, to our Deposit Account No. 19-0036.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Edward J. Kessler
Attorney for Applicants
Registration No. 25,688

EJK/WPL:vba
Attachments
952136_1.DOC

PETITION FOR EXTENSION OF TIME UNDER 37 CFR 1.136(a) **FY 2009** (*Fees pursuant to the Consolidated Appropriations Act, 2005 (H.R. 4818).*)	Docket Number (Optional) 2222.0300002
Application Number 10/995,159	Filed November 24, 2004
For Network System Extensible By Users	
Art Unit 2442	Examiner Douglas B. Blair

This is a request under the provisions of 37 CFR 1.136(a) to extend the period for filing a reply in the above identified application.

The requested extension and fee are as follows (check time period desired and enter the appropriate fee below):

	Fee	Small Entity Fee	
☐ One month (37 CFR 1.17(a)(1))	$130	$65	$
☒ Two months (37 CFR 1.17(a)(2))	$490	$245	$ 490.00
☐ Three months (37 CFR 1.17(a)(3))	$1110	$555	$
☐ Four months (37 CFR 1.17(a)(4))	$1730	$865	$
☐ Five months (37 CFR 1.17(a)(5))	$2350	$1175	$

☐ Applicant claims small entity status. See 37 CFR 1.27.

☐ A check in the amount of the fee is enclosed.

☒ Payment by credit card. ~~Form PTO-2038 is attached~~

☐ The Director has already been authorized to charge fees in this application to a Deposit Account.

☒ The Director is hereby authorized to charge any fees which may be required, or credit any overpayment, to Deposit Account Number 19-0036.

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

I am the ☐ applicant/inventor.

☐ assignee of record of the entire interest. See 37 CFR 3.71.
Statement under 37 CFR 3.73(b) is enclosed (Form PTO/SB/96).

☒ attorney or agent of record. Registration Number 25,688

☐ attorney or agent under 37 CFR 1.34.
Registration number if acting under 37 CFR 1.34 ____________

[illegible]
Signature

March 16, 2009
Date

Edward J. Kessler
Typed or printed name

(202) 371-2600
Telephone Number

NOTE: Signatures of all the inventors or assignees of record of the entire interest or their representative(s) are required. Submit multiple forms if more than one signature is required, see below.

☒ Total of One (1) forms are submitted.

952,111