A publish/subscribe CORBA Persistent State Service Prototype

C. Liebig, M. Cilia[†], M. Betz, A. Buchmann

Database Research Group - Department of Computer Science
Darmstadt University of Technology - Darmstadt, Germany
{chris,cilia,betz,buchmann}@dvs1.informatik.tu-darmstadt.de

Abstract. An important class of information dissemination applications requires 1:n communication and access to persistent datastores. CORBA's new Persistent State Service combined with messaging capabilities offer the possibility of efficiently realizing information brokers between data sources and CORBA clients. In this paper we present a prototype implementation of the PSS that exploits the reliable multicast capabilities of an existing middleware platform. This publish/subscribe architecture makes it possible to implement an efficient update propagation mechanism and snooping caches as a generic service for information dissemination applications. The implementation is presented in some detail and implications of the design are discussed. We illustrate the use of a publish/subscribe PSS by applying it to an auction scenario.

1 Introduction

The deployment of large scale information dissemination systems like Intranet and Extranet information systems, e-commerce applications, and workflow management and groupware systems, is key to the success of companies competing in a global marketplace and operating in a networked world. Applications like warehouse monitoring, auctions, reservation systems, traffic information systems, flight status tracking, logistics systems, etc. consist of a potentially large number of clients spread all over the world demanding timely information delivery. Many of these applications span organizational boundaries and are centered around a variety of data sources, like relational databases or legacy systems that maintain business data. The business logic may be spread over separate modules and the entire system is expected to undergo continuous extension and adaptation to provide new functionality.
Common approaches in terms of systems architecture can be classified into traditional 2-tier client/server, 3-tier TP-heavy using TP monitors and n-tier Object-Web systems.
In 2-tier client/server the client part implements the presentation logic together with application logic and data access. This approach depends primarily on RPC-like communication and scales well only if client and server are close together in terms of

[†] Also ISISTAN, Faculty of Sciences, UNICEN, Tandil, Argentina.

J. Sventek and G. Coulson (Eds.): Middleware 2000, LNCS 1795, pp. 231-255, 2000.

network bandwidth and access latency. However, it does not scale in the face of wide-area distribution. Moreover, the fat-client approach renders the client software dependent on the data model and API of the backend.

In a 3-tier architecture a middle-tier – typically based on a TP monitor - is introduced to encapsulate the business logic and to hide the data source specifics. TP monitors provide scalability in terms of resource management, i.e. pooling of connections, allocating processes/threads to services and load balancing. The communication mechanisms used in 3-tier architectures range from peer-to-peer messaging and transactional queues to RPC and RMI. TP monitor based approaches assume that the middle-tier has a performant connection to the backend data sources, because database access protocols for relational systems are request/response and based on "query shipping". In order to reduce access latency and to keep the load of the data source reasonably low, the application programmers are urged to implement their own caching functionality in the middle-tier. A well known example of such an architecture is the SAP system [21].

In n-tier Object-Web systems the clear distinction between clients and servers gets blurred. The monolithic middle-tier is split up into a set of objects. Middleware technology, such as CORBA, provides the glue for constructing applications in distributed and heterogeneous environments in a component-oriented manner. CORBA leverages a set of standard services [22] like Naming Service, Event and Notification Service, Security Service, Object Transaction Service, and Concurrency Control Service. CORBA has not been able to live up to expectations of scalability, particularly in the information dissemination domain, because of a limiting (synchronous) 1:1 communication structure and the lack of a proper persistence service. The new CORBA Messaging standard [23] will provide true asynchronous communication including time independent invocations. We argue, that the recently proposed Persistent State Service [14], which replaces the ill-fated Persistent Object Service, will not only play a key role as integration mechanism but also provides the opportunity to introduce efficient data distribution and caching mechanisms.

A straightforward implementation of the PSS relying on relational database technology is based on query shipping. The PSS must open a datastore connection to the server, then ships a query that is executed at the server side and the result set is returned in response. Such a PSS implementation realizes storage objects as stateless incarnations on the CORBA side, that act as proxies to the persistent object instance in the datastore. Operations that manipulate the state of objects managed by the PSS are described in datastore terms. This approach generates a potential bottleneck at the datastore side, because each operation request on an instance will result in a SQL query. Furthermore, for information dissemination systems, where the user wants to continuously monitor the data of interest, polling must be introduced which results in a high load at the backend, wasting resources and possibly delivering low quality of data freshness.

For information dissemination systems an alternate approach based on server-initiated communication is more desirable. Techniques ranging from cache consistency mechanisms in (OO)DBMSs [33,5] and triggers/active database rules [10] to broadcast disks [1] can be used to push data of interest to clients. In the context of the PSS a new publish/subscribe session is needed. A publish/subscribe session represents the scope of the objects an application is interested in, i.e. subscribes to. For those

objects in a publish/subscribe session the cache is loaded and updated automatically. Additionally, this session provides notifications about insert, modify and delete events to the application. While publish/subscribe sessions currently are not part of the PSS specification they are definitely not precluded by it and would represent a useful extension to the spec.

In this paper we present an implementation of a PSS prototype that provides an intelligent caching mechanism and active functionality in conjunction with message oriented middleware (MOM) that is capable of 1:n communication. By removing two crucial bottlenecks from the CORBA platform we claim that highly scalable Object-Web systems become feasible.

In our PSS prototype[1] we take advantage of commercial publish/subscribe middleware that provides the paradigm of subject based addressing and 1-to-many reliable multicast message delivery. We show how a snoopy cache can be implemented for multi-node PSS deployment. We make use of a prototype of a database adapter for object-relational databases (Informix IUS, in particular) that was partially developed and extended in the scope of this project. The database adapter allows to use publish/subscribe functionality in the database and to push data to the PSS caches when update transactions are issued against the data base backend or when new data objects are created.

This paper concentrates on the basic infrastructure needed to provide scalability with respect to dissemination of information from multiple data sources. We explicitly exclude from the scope of this paper federated database and schema integration issues.

The remainder of this paper is organized as follows: Section 2 briefly introduces key concepts of the PSS specification and the multicast-enabled message oriented middleware; Section 3 provides an overview of the architecture of our prototype implementation of the PSS and identifies the main advantages of integrating the reliable multicast functionality of the TIBCO platform; Section 4 describes the implementation; Section 5 introduces auctions as a typical scenario for middleware-based Web-applications and Section 6 presents conclusions and identifies areas of ongoing research.

2 CORBA PSS and Messaging Middleware

2.1 CORBA Persistent State Service

The need for a persistence service for CORBA was recognized early on. In 1995, the Persistent Object Service was accepted but failed because of major flaws: the specification was not precise, persistence was exposed to CORBA clients, transactional access to persistent data was not covered, and the service lacks integration with other CORBA services. Recently, the Persistent State Service (PSS) was proposed to overcome those flaws. The goals of the PSS specification [14] are to

[1] The work of this project is partially funded by TIBCO Software Inc., Palo Alto.

make the state of the servant persistent, to be datastore neutral and implementable with any datastore, to be CORBA friendly, consistent with other OMG specifications (Transactions, POA, Components, etc.) and also with other standards like SQL3 [18] and ODMG [7].

The PSS provides a single interface for storing objects' state persistently on a variety of datastores like OO-, OR-, R-DBMS, and simple files. The PSS provides a service to programmers who develop object implementations, to save and restore the state of their objects and is totally transparent to the client. Persistence is an implementation concern, and a client should not be aware of the persistence mechanisms. Therefore, the PSS specification does not deal with the external interface (provided by a CORBA server) but with an internal interface between the CORBA-domain and the datastore-domain.

Due to numerous problems with IDL valuetypes - used in previous proposals as requirement imposed by the RFP - the IDL was extended with new constructs to define storage objects and storage home objects. The extended IDL is known as Persistent State Definition Language (PSDL). Storage objects are stored in storage homes, which are themselves stored in datastores. In order to manipulate a storage object, the programmer uses a representative programming-language entity, called storage object *instance*. A storage object instance may be connected to a storage object in the datastore, providing direct access to the state of this storage object. Such a connected instance is called storage object *incarnation*. To access a storage object, a logical connection between the process and the datastore is needed. Such a connection is known as *session*.

There is also a distinction between abstract storage type specification and concrete storage type implementation. The abstract storage type spec defines everything a servant programmer needs to know about a storage object, while an implementation construct defines what a code generator needs to know in order to generate code for it. A given abstract specification can have more than one implementation and it is possible to update an implementation without affecting the storage objects' clients. So, the implementation of storage types and storage homes lies mainly in the responsibility of the PSS. An overview of these concepts is depicted in Figure 1.

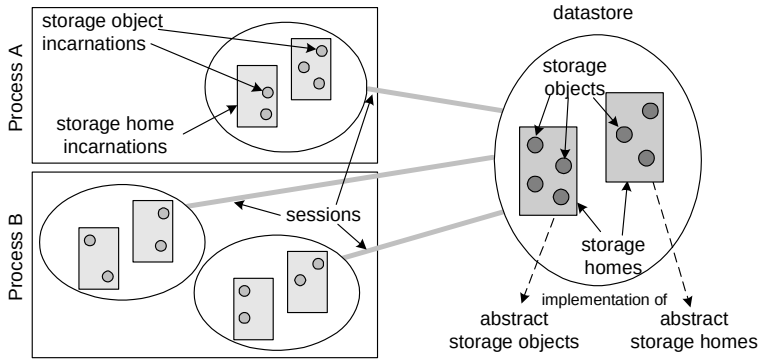

Fig. 1. PSS concepts [14].

A storage object can have both state and behavior, defined by the storage type : its state is described by attributes (also called state members) and its behavior is described by operations. State members are manipulated through equally named pairs of accessor functions. Operations on storage objects are specified in the same manner as with IDL. In addition to IDL parameter types, storage types defined in PSDL may be used as parameters. In contrast to CORBA objects, operations on storage objects are locally implemented and not remotely accessible.
A *storage home* does not have its own state, but it can have behavior, which is described by operations in the abstract storage home. A storage home can ensure that a list of attributes of its storage type forms a unique identifier for the storage objects it manages. Such a list is called a *key*. A storage home can define any number of keys. Each key declaration implicitly declares associated finder operations in the language mapping. To create or locate a storage object, a CORBA server implementor calls `create(`<parameters>`)` or `find_by_`<some key>`(`<parameters>`)` operations on the storage home of the storage type and in return will receive the according storage object instance.
The inheritance rules for storage objects are similar to the rules for interface inheritance in IDL. Storage homes also support multiple inheritance. However, it is not possible to inherit two operations with the same name; as well as to inherit two keys with the same name.
In the PSS spec the mapping of PSDL constructs to several programming languages is also specified. A compliant PSS tool must generate a default implementation for storage homes and storage types based on the given PSDL definition.
For the case that the underlying datastore is a database system, the PSS introduces a *transactional session* orchestrated by OTS through the use of the X/Open XA interface [34] of the datastore. Access to storage objects within a transactional session produces executions that comply with the selected isolation level i.e. *read uncommited*, *read commited*. Note that stronger isolation levels like *repeatable read* and *serializable* are not specified.

2.2 Multicast-enabled MOM

We use COTS MOM [31] to build the PSS prototype, namely TIB/Rendezvous and TIB/ObjectBus products. TIB/Rendezvous is based upon the notion of the *Information Bus* [26] (interchangeable with the wording “message bus” in the following) and realizes the concept of *subject based addressing*, which is related to the idea of a *tuple space*, first introduced in LINDA [6]. Instead of addressing a sender or recipient for a message by its identifier, which in the end comes down to a network address, messages are published under a subject name on the *message bus*. The subject name is supposed to characterize the contents - i.e. the type - of a message. If a participant, who is connected to the *message bus*, is interested in some specific message types, she will subscribe for the subjects of interest and in turn be notified of messages published under the selected subject names. The subject name space is hierarchical and subscribers may use subject name patterns to denote a set of types to which they want to subscribe.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.