MAKING A STATEMENT WITH CORBA

M. Böge, J. Chrin, Paul Scherrer Institut, 5232 Villigen PSI, Switzerland

Abstract

An interface to the Oracle relational database management system (RDBMS), implemented using the Common Object Request Broker Architecture (CORBA), provides clients with a set of methods that execute Structured Query Language (SQL) statements and return a "result set" within the context of a CORBA object. The database client-server framework is presented, including a description of the CORBA object to relational database mapping scheme.

1 INTRODUCTION

Beam dynamics applications at the Swiss Light Source (SLS) use the Common Object Request Broker Architecture (CORBA) as the middleware layer and access point to essential software packages, one of which is the Oracle relational database management system (RDBMS). The use of a RDBMS in an object oriented environment is in itself a pseudo-science, presenting the developer with the challenge of reconciling the object-oriented paradigm with that of the relational database. Nevertheless, through a simple object-relational mapping scheme, the so-called "impedence mismatch" between the two orthogonal technologies can be largely overcome, at least for the purpose of our intent. A CORBA interface to the Oracle database provides functions that translate object constructs to relational constructs that communicate with the database through the Structured Query Language (SQL). In this way, clients may perform database operations in an object-oriented manner and without the need to acquire knowledge of SQL syntax or the vendor's Application Programme Interface (API). Conversely, methods that permit generic database retrieval and modification operations have also been provided for clients that require the increased flexibility attained through direct execution of SQL commands. The methodology behind the database application objects is described, along with critical CORBA subsystem components that serve to provide for the scalability of the CORBA database server.

2 CORBA FUNDAMENTALS

An extensive overview of the CORBA framework appears in previous work [1-2]. Nevertheless, the most fundamental subsystems are reiterated here to enable continuity. In particular, the capability of the POA and the distinction between CORBA objects and servants is emphasized since their understanding is pertinent to grasping the concept upon which database application objects are implemented.

The most fundamental component of CORBA is the Object Request Broker (ORB) whose task is to facilitate communication between objects. Given an Interoperable Object Reference (IOR), typically obtained from a Naming Service wherein self-describing names are mapped to object references, the ORB is able to locate target objects and transmit data, both to and fro, through remote method invocations (RMIs). The interface to a CORBA object is specified using the CORBA Interface Definition Language (IDL). An IDL compiler translates the interface from IDL into an application programming language (e.g. C++, Java, Tcl) generating IDL stubs and skeletons that respectively provide the framework for client-side and server-side proxy code. Compilation of applications incorporating IDL stubs provides a strongly-typed Static Invocation Interface (SII). Requests and responses between objects are delivered in a standard format defined by the Internet Inter-ORB Protocol (IIOP). Requests are marshalled in a platform independent format by the client stub and unmarshalled on the server-side into a platform specific format by the IDL skeleton and the object adapter, which serves as a mediator between an object's implementation, the servant, and its ORB, thereby decoupling user code from ORB processing. In its mandatory version, the Portable Object Adapter (POA) provides CORBA objects with a common set of methods for accessing ORB functions, ranging from user authentication to object activation and object persistence. It's most basic task, however, is to create object references and to dispatch ORB requests aimed at target objects to their respective servants. The characteristics of the POA are defined at creation time by a set of POA policies. A server can host any number of POAs, each with its own set of policies to govern the processing of requests. Among the more advanced features of the POA is the servant manager which assumes the role of (re-)activating server objects (i.e. servants) as they are required. Requests for the activation of servants can, alternatively, be delegated to a single default servant which provides implementations for many objects, thereby increasing the scalability for CORBA servers. Indeed, this is the pattern adopted for the database application objects. The database server is able to produce an arbitrary number of object references while keeping the number of *active* objects constant!

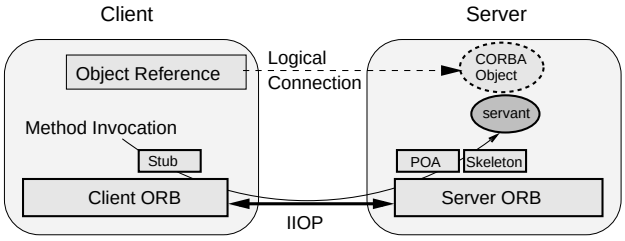

Figure 1: The CORBA client-server architecture

Fig. 1 illustrates the principal ORB subsystems involved in dispatching a request from client to server. The distinction between CORBA object and servant is deliberate. A CORBA object in essence remains a virtual entity until incarnated by a servant.

3 DATABASE APPLICATION OBJECTS

In constructing database application objects a number of aspects need first be considered, including:

- the database API,
- mapping of native database types to IDL datatypes,
- object-relational mapping schemes,
- performance versus ease of use.

3.1 The Database API

The API chosen to access the Oracle8i database management system is the Oracle Template Library (OTL) [3], implemented using the the Oracle Call Interface (OCI). OTL provides a set of C++ classes that handle database connections and transaction management, execution of SQL statements and stored procedures, exception handling and operations with Large Objects (LOBs). The OTL templates are expanded into direct database API function calls and, as such, provide high performance and reliability. Fig. 2 illustrates database retrieval times as a function of data volume, both with and without the CORBA transport layer. A comparison is made with the Java-based JDBC API. Values quoted are exemplary values obtained from our system. The difference in performance between OTL and JDBC is striking but not unexpected. It is interesting to note that data retrieval through CORBA and OTL is twice as fast as that obtained through JDBC alone.

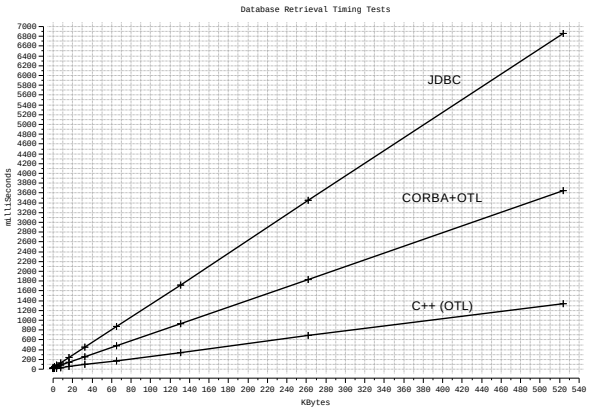

Figure 2: Database retrieval timing tests

OTL defines a number of datatypes which map to those of Oracle 8, hence a mapping of CORBA IDL to OTL datatypes (and vice versa) is what is implemented. Column data is retrieved using the native OTL datatype and converted to the recommended IDL type according to the mapping scheme. Methods that provide database metadata facilitate clients requiring knowledge of the relevent datatypes. For convenience, methods that permit clients to retrieve an OTL datatype in a form other than the recommended IDL datatype, have also been provided. Data arrays are mediated through OTL datatypes that map onto Oracle Binary Large OBjects (BLOBs). Conversely, BLOBs are decomposed into sequences of the recommended IDL datatype.

3.2 Object to Relational Database Mapping

An important database application object to develop at the SLS is one in which the many "holy" tables, containing data pertinent to the numerous hardware components that constitute the SLS accelerator complex, are made available. Here, the problem of how best to map a relational database to a CORBA object presents itself. Perhaps the easiest and most straightforward of object-relational mapping schemes is to treat each single table as a class and each row (or line) of the table as an instance of that class. A column and its value is then regarded, respectively, as an attribute of that class and its instance. Each row of a table can thus effectively be mapped to a CORBA object, whose reference once exported to the Naming Service, can be readily accessed by any CORBA aware client. To help implement and manage the inevitably vast number of object references, the naming graph feature of the Naming Service is used. A naming graph is a hierarchy of contexts and bindings. A name binding is the term given to a name-to-reference association, while a naming context refers to an object that stores name bindings.

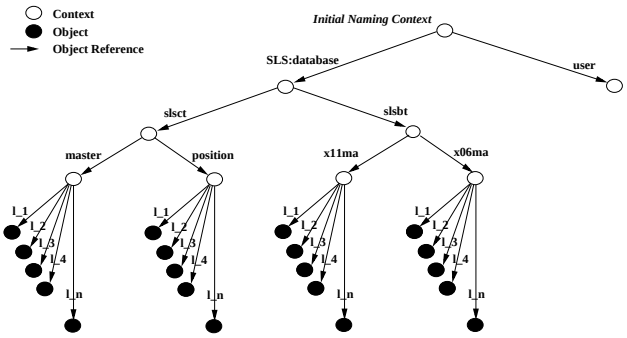

Figure 3: A naming graph for implementing database object references

Fig. 3 shows the naming graph in use for implementing database object references. Each context object (hollow node) implements a table that maps names to object references that point to either an application object (solid node) or to another context object in the Naming Service. Given a starting context, one can navigate to a target node by traversing a path from the starting context (e.g. SLS:database) through other contexts (e.g. database instance: slsbt; table name: position) to the target node (line number: l_n). The sequence of bindings forms a pathname that uniquely identifies the target object.

The SQL required to establish a database connection, to execute a SQL statement and retrieve the data, is the domain of the server application object. Each table row is indexed, using the primary key, to expediate data retrieval. The client is thus able to perform database operations without imparting any SQL syntax, as is illustrated with the Tcl scripting language in Table 1.

Table 1: Database table/row retrieval operation from Tcl

```
#### $ObjectRef Operation Output
$ObjRefToTable($LineNo) GetRow rowData
#### Procedure to unfold Tcl list (rowData)
$DisplayData $rowData
```

3.3 Performance Considerations

In the above mapping scheme, a separate CORBA object represents each database entry, allowing clients to perform database retrieval operations using familiar object oriented techniques. Despite the onset of numerous tables with countless number of rows, and with each entry mapped to a persistent CORBA object, the cost in terms of memory consumption at the server is inexpensive, since each object is incarnated by the same default servant.[1] Such a technique offers a general purpose solution to the problem of mapping CORBA objects to relational databases while simultaneously achieving server scalability. There is, however, a trade off in terms of performance. Access to each single table row requires a RMI. The factors that limit the speed of remote invocations are call latency, i.e. message overhead, and marshalling rate, i.e. rate at which an ORB is able to transmit and receive data. The call latency of our chosen ORB, MICO [4], is $\sim$ 1 ms, while the marshalling rate, typically $\sim$ 500 kBytes/s depending on the data type, becomes significant only for data transfers larger than about a kByte. The use of a default servant itself trades off the time required to uncover the target object identity from the POA against the space required for using multiple servants. Locating and reading the data from the database disk also has its cost in time. Consequently, a table holding 3300 rows of three columns, each of datatype "union", requires 18 seconds to be read (examplary values.) Such throughput is within the requirements of clients performing database operations offline and for the configuration of applications. Nevertheless, for clients requiring to act on a complete table, improvements in performance can be achieved by invoking a method that returns the entire content of a table. In such cases an object reference to a table is provided and its content returned with one RMI. In this way, the table of the above example can now be retrieved in full in less than 2 seconds. Data is presented to the client as a sequence of the user-defined "rowData" type, which itself comprises a sequence of the user-defined "columnData" type.

[1] In practice, the database server hosts one POA per table

3.4 Dynamic SQL statements through CORBA

Developing further the notion that it is more efficient to send more data with each RMI, it becomes evident that it would be equally constructive to provide a method that allows the client to execute its own query statement. The parameters returned are of the same format (or type) as in the previous case. However, rather than a reference to a table, a reference to the database instance is obtained and a method that directly executes the client's SQL statement is invoked. In the same vain of granting clients the capability of executing SQL statements directly, methods that allow insert and modification operations have been appended. These functions are presently in use by applications for the acquistion and analysis of data online.

Finally, methods that perform specific database query operations and package the retrieved data in the context of a specific, informative and self-describing CORBA object (as opposed to the generic "rowData" object) have also been developed for clients with particular needs.

4 PRESENT DIRECTION

CORBA objects adhere to the so called "Pass by Reference" semantics. A recent addition to the CORBA standard is the "Objects by Value" specification. Here "value types" that declare both state members and operations/attributes are used to create "value objects" that can be passed by value when transmitted as a parameter or return value of an operation. Its advent offers the tools to develop interfaces that provide more scope. One possibility would be to develop a CORBA interface that returns a "result set" class similar to that offered by JDBC. Here operations that help navigate through the data returned are also provided to the client.

5 CONCLUSION

A strategy for developing CORBA based database application objects has been presented. The CORBA API translates object constructs to relational constructs that ultimately communicate with the database through SQL using the C++ OTL library for optimum performance. Default servants provide the ability to support an arbitrary number of objects in a fixed amount of memory, demonstrating the scalability of the database server. The CORBA database API has been usefully employed during the first year of SLS operation for configuring applications and for the acquisition and analysis of data online.

6 REFERENCES

[1] M. Böge, J. Chrin, PCaPAC 2000 (ID:054), http://desyntwww.desy.de/pcapac/Proceedings/

[2] M. Böge, J. Chrin, ICALEPCS'01 (ID:THAT002), http://www-project.slac.stanford.edu/icalepcs/

[3] OTL, http://otl.sourceforge.net/

[4] MICO, http://www.mico.org/