Gilbert Babin, Waiman Cheung, Lester Yee, and Sofienne Bahri. "An Object Oriented Shell for Distributed Processing." in Distributed Computing on the Web Workshop (DCW'99). Rostock, Germany. June 1999. pp. 85-93.

AN OBJECT ORIENTED SHELL FOR DISTRIBUTED PROCESSING

GILBERT BABIN[1], WAIMAN CHEUNG[2], LESTER YEE[3], SOFIENNE BAHRI[1]

[1] Département d'informatique,
Université Laval
Ste-Foy, Québec, Canada, G1K 7P4
+1 (418) 656-3395 babin@ift.ulaval.ca

[2] Department of Decision Sciences and Managerial Economics,
Chinese University of Hong Kong
Shatin, New Territories, Hong Kong
+852 2609 7816 wcheung@cuhk.edu.hk

[3] Babson College
231 Forest St.
Babson Park MA USA 02457
+1 (781) 235-1200 yee@babson.edu

Abstract

The advancement in computer network domain and the globalization of economy have forced enterprises to adopt a distributed structure which, in turn, implies a distribution of resources of those enterprises, particularly their information systems. Therefore, it is important to provide enterprises with integration tools to consolidate the information available throughout the distributed information systems and databases. One original approach to the integration problem that assures flexibility and scalability is the use of the metadata concept. This concept has led to the development of a Metadatabase system, that is, a knowledge base containing both logical and physical data characteristics of local systems. The Metadatabase may be seen as a specialized information warehouse which stores information about information systems, the data structures they use, and the mechanisms used to actually process and store the information. Based on the Metadatabase, a software infrastructure, the Rule-Oriented Programming Environment (ROPE), was developed to achieve the integration of distributed information systems. This paper presents a ROPE shell which makes better use of object oriented technologies.

Keywords: *Interoperatility, Distributed Systems, Reactive Agents*

Gilbert Babin, Waiman Cheung, Lester Yee, and Sofienne Bahri. "An Object Oriented Shell for Distributed Processing." in Distributed Computing on the Web Workshop (DCW'99). Rostock, Germany. June 1999. pp. 85-93.

1. INTRODUCTION

The advancement in computer network domain and the globalization of economy have forced enterprises to adopt a distributed structure which, in turn, implies a distribution of resources of those enterprises, particularly their information systems. Therefore, it is important to provide enterprises with integration tools to consolidate the information available throughout the distributed information systems and databases. The integration concept should support an interoperable process which means the ability for two or more distributed information systems to mutually exchange information and share functionalities independent of their constraints of distribution and heterogeneity, as well as work together to execute well-defined and delimited tasks jointly. Notice that the traditionally proposed integration approaches assume the interoperability at the application level (i.e., between local systems) via an integrated schema or a common manipulation language [7,15]. On one hand, designing an integrated schema implies resolving conflicts and imposing standards to local systems, which is generally difficult to achieve and maintain. On the other hand, finding a common language is not easy because each system has its own standards and needs. One original approach to the integration problem that assures flexibility and scalability is the use of the *metadata* concept. This concept has led to the development of a Metadatabase system, that is, a knowledge base containing both logical and physical data characteristics of local systems. The Metadatabase work [2,4,5,8,9,10] has focused on creating an integration environment and defining its principal components to produce a Metadatabase-supported, Rule-Oriented concurrent systems solution for the enterprise information integration and management problem. The Metadatabase may be seen as a specialized information warehouse which stores information about information systems, the data structures they use, and the mechanisms used to actually process and store the information.

A software infrastructure, the Rule-Oriented Programming Environment (ROPE), was developed to achieve the integration of distributed information systems, using a Metadatabase as a driving force. ROPE uses (identical) shells to (1) interface with the local systems and (2) establish communication channels across the local systems. The Metadatabase and the shells provide a powerful environment for interoperability, provided that the Metadatabase contains enough knowledge about the systems to be integrated.

Different versions of the shell have been designed, each version taking advantage of different available tools :

- The ROPE shell-α is a generic and portable shell, and provides the basic functionality of ROPE [1].
- The ROPE shell-β takes advantage of the distributed query management tools embedded in SQL3 [14].
- The ROPE shell-γ was designed to handle real-time systems [13].

This paper presents a fourth instance of the ROPE shell, the shell-ω. This shell makes better use of object oriented technologies [3], in particular CORBA (Common Object Request Broker Architecture) [11,12], which supports direct interoperability among various object-oriented systems and some gateway services for connecting relational or other traditional systems into an object environment.

The paper is organized as the following. Section 2 describes in more details the Metadatabase approach and the internal functions of ROPE shells. The rationale justifying the use of an object-oriented approach is also presented. In Section 3, we present a generic object model of the ROPE shell. Section 4 describes the design of the ROPE shell-ω, which uses CORBA as a communication middleware. We conclude the paper in Section 5.

2. THE METADATABASE APPROACH

Rensselaer's Metadatabase approach was developed to integrate information systems, more specifically manufacturing systems. Manufacturing systems are heterogeneous and distributed by nature. To produce finished goods, many systems must cooperate. These systems might include an Order Processing System, used to record customers' orders, a Process Planning System, determining the steps to follow to obtain finished goods, and a Shop Floor Control System, dealing with process planning, job assignment, production status, etc. These systems are highly specialized, in the sense that only certain systems can perform specific tasks; for instance, only certain machines can drill a hole into a sheet of metal, for instance. These systems are also autonomous; each one manages its own database, which makes data consistency an issue. The problem then is how to coordinate the tasks performed by each of these systems and integrate the information stored in their local databases.

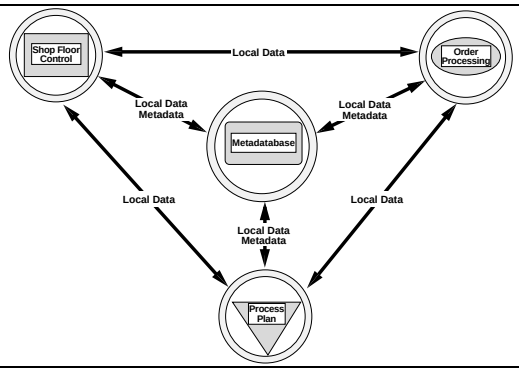

Figure 1 : The Metadatabase Concurrent Architecture

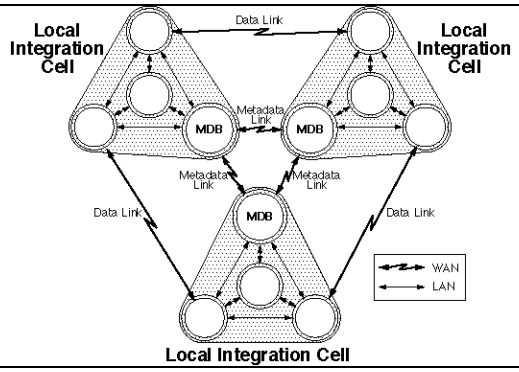

Figure 2 : Distributed Metadatabase Architecture

2.1 The Concurrent Architecture

The Metadatabase approach uses a concurrent architecture (Fig. 1) which contains: (1) a central knowledge base and (2) distributed rule processors. The central knowledge base, called a Metadatabase, contains a description of the different (manufacturing) systems and the knowledge describing how they are integrated and semantically interrelated. This knowledge includes (1) database integrity rules, enforcing consistency across the distributed databases, and (2) business rules, automating information flows across these systems.

For each of the (manufacturing) systems, we define a rule processor. The rule processor's role is twofold. First, it encapsulates the system, ensuring that the system remains autonomous and independent, while making the system's capabilities available to other systems. Hence, although the shells are identical in structure, they differ in their capabilities, since each one has access to the special capabilities of the system it encapsulates. Second, it enables integration by providing knowledge processing capabilities to the system it encapsulates.

The Metadatabase and the rule processors form an integrated cell. The Metadatabase manages the knowledge, providing a global coherent view of the whole system, and distributes it to the local rule processors, hence achieving operational integration. In a Wide-Area Network (WAN), we can rely on a number of such integrated cells to provide a more robust framework (Fig. 2).

We should point out that the central shell (Fig. 1) used by the Metadatabase has the same structure as the other shells. However, it provides specialized services. In particular, it manages metadata about the other systems participating in its local integration cell. Therefore, these shells are not very different from WOS nodes, since the Metadatabase may be viewed as a specialized resource.

2.2 The Rule-Oriented Programming Environment

The ROPE (Rule-Oriented Programming Environment) was developed as one implementation of the Metadatabase concurrent architecture. It creates a distributed rule processing environment, where fact bases and inference engines are distributed. The collaboration of rule processors is minimized by utilizing most of the information contained in the Metadatabase.

The ROPE approach defines the shell technology needed for the concurrent architecture evolution. Specifically, it defines: (1) how rules are stored in the local shells, (2) how rules are processed, distributed, and managed, (3) how the shell interacts with its corresponding system, (4) how the different shells interact with each other, and finally (5) how the shells are structured. The main advantage of the ROPE approach is that shells are invisible to the users. They are able to control the external behaviors of the local systems, without user intervention.

The ROPE shells have the following functions (See [1] for more details):

- Global Query Processing: a global query is a data retrieval query that needs the participation of one or many local systems from the integration environment.
- Global Update Processing: a global update processing represents a set of transactions (insertion, deletion or modification) that act on the data of the local systems. As a result, every local behavior change that has an impact (as updating data that pertain to another local system) on the behavior of the integration environment, has to be immediately transmitted to the rest of the appropriate systems.
- Adaptability and Flexibility: the adaptability of the concurrent architecture refers to the ability of shells to change their behavior. This change is based on the knowledge stored in the Metadatabase.

Figure 3 shows the components of a shell.

The Rule Segment. A key element of the shell is the possession of rules. Logically, the Rule Segment implements the intersection of the global rulebase model and the local application logic. All the rules are originated from and globally managed by the Metadatabase. Whenever an operating rule is changed, the change is propagated to all local systems that will be affected by it. Also, when changes occur in the global data model of the enterprise, new data management rules are generated to replace the existing rules in the Rule Segment of the different shells. The Rule Segment is separate from the source code, which allows for easy implementation on different platforms.

Network Monitor. The Network Monitor is in charge of the interface between the shell and the network. It receives incoming messages and passes them to the Message/Rule Processor (see below).

Application Monitor. The Application Monitor “spies” on the behavior of the application and reports any globally significant event to the Message/Rule Processor, such as changes to the local database, or a call to a specific procedure, that would result in the execution of a global behavior rule. The Application Monitor functionality can be specified in general terms, allowing implementations on very different platforms. Furthermore, the use of the Global Query System (GQS) facilities [5] actually removes the need to tailor the Application Monitor to the local database. Current implementations of the Application Monitor focus only on the detection of local database activities.

Timer. The Timer manages a list of time events. When a time event is due for processing, the Timer activates the command associated to that time event.

Database Interface. This object converts the data item values to a neutral format used by the shells. This is to remove the burden of processing conversion rules every time we need to use a data value.

Message/Rule Processor. This object is the inference engine of the shell. It is generic and neutral to particular implementations. On the basis of the events it receives from the Network Monitor, the Application Monitor, the Timer, and the local application, it will trigger the

appropriate rule(s) stored in the Rule Segment, or execute the appropriate functions of the shell. In addition to rule execution, the Message/Rule Processor performs four other functions: (1) it serves as a dispatcher, transmitting local query requests to the Database Interface; (2) it transmits any structural change notification it receives from MDBMS to the Application Monitor; (3) it updates the Rule Segment and notifies the Application Monitor and Timer, when a rule deletion or insertion notification is received; and (4) it updates the frequency at which events are processed. From an implementation standpoint, the Message/Rule Processor consist of two distinct modules : the Message Processor and the Rule Processor. The Message Processor generates the Rule Processor, which requires to be linked to local functions.

Result Integrator. The Result Integrator is used to assemble the results of local queries requested by the local system. In addition, it performs operations on these queries, like producing a sum of the values on the different rows. The functions this object performs are based on MQL and GQS. The processing of the Result Integrator is straightforward. It receives a set of messages as input, one of which contains a script describing (1) the local queries, (2) the integration operations to be performed, and (3) the output file to be used. The Result Integrator simply executes that script.

2.3 The Need for an Object-Oriented Approach

The growing popularity of object orientation has brought us to investigate how object oriented approaches and technologies could benefit ROPE. By design, the ROPE shell is highly modular. Furthermore, the ROPE shell-α, although developed in C, was designed with the notion of encapsulation. Specifically, for every data structure defined in the shell environment, a single file contains the structure definition and all the routines used to manipulate that structure; as described above, the shell is the agregation of independant modules, each providing specialized services. However, we feel that designing using object orientation is not quite sufficient. The use of C makes maintenance of the shells a problem. There exists many versions of C and any change made must guaranty that the shell is still portable. In addition, the development of CORBA as an interoperability infrastructure for objects also made us think of the benefits of using an object oriented approach for the shell, therefore simplifying the communication problem between shells.

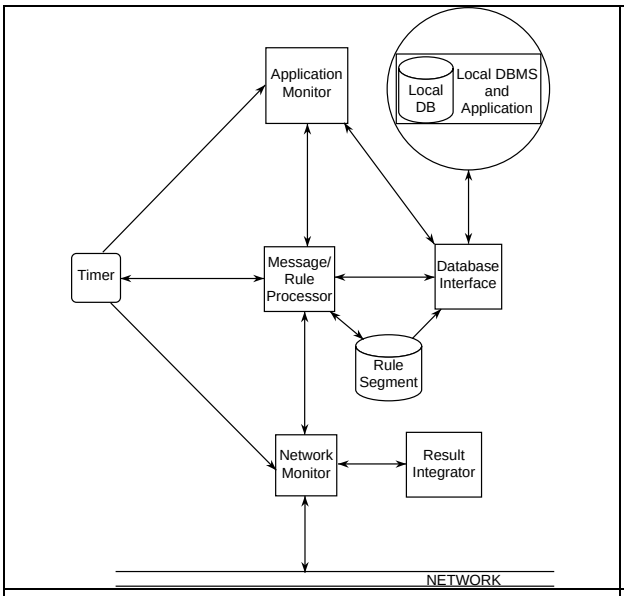

Figure 3 : The Generic Structure of a ROPE Shell

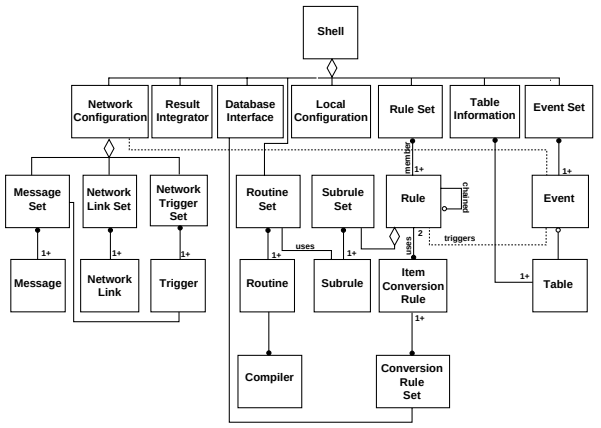

Figure 4 : A Generic Object Model of ROPE Shells

3. A GENERIC OBJECT MODEL OF ROPE SHELLS

A class diagram of the shell, using the FUSION methodology notation [6], is shown in Figure 4. We will now describe the major classes composing this model. The different classes correspond to the information required by the shell to perform its tasks :

The Rule Segment. The Rule Segment is described by classes Event Set, Table Information, Rule Set, Conversion Rule Set, Subrule Set, Routine Set, Compiler. The shell stores the Rule Segment in files. For each of these classes, methods are defined (1) to load the Rule Segment from files, (2) to update the Rule Segment, or (3) to search for a particular information within the Rule Segment.

Network Monitor. The communication links are established using class Network Configuration, which is the agregation of classes Message Set (the messages received ot to be sent by the shell), Network Link Set (communication links with other shells), and Network Trigger Set (used when many messages are required before an action is taken). Specifically, a call to method `process_new_messages()` will trigger the processing of messages received or to be sent.

Application Monitor. This module is implemented using class Table Information. Before a request to monitor a table can be made, the object describing that table must be found.

Timer. The Timer is an active module. It triggers the appropriate shell modules based on a list of events, which are stored in class Event Set.

Database Interface. The functions of the Database Interface are performed by class Database Interface. Every database query run against the local database is filtered to convert the values to/from the global format, as described in instances of class Item Conversion Rule.
Message Processor. This module includes all the functionalities to process messages, as described in class Message. Depending on the type of the message, it will launch the appropriate module.
Rule Processor. The Rule Processor uses classes Subrule Set, Routine Set, and related classes. This module must be regenerated every time the Rulebase Segment is modified to properly bind the locally defined routines to the shell.
Result Integrator. Class Result Integrator fully specifies the functionalities of this module.

4. THE ROPE SHELL-ω

4.1 The Common Object Request Broker Architecture (CORBA)

The shell-ω takes advantage of object interoperability services provided by CORBA. CORBA was developed to achieve the interoperability of object oriented distributed systems. We give here a short description of CORBA's main features :
Handling heterogeneity. Objects developed in different languages usually have a hard time communicating with one another. CORBA solves this heterogeneity problem by providing a uniform way to describe interfaces to available services, the Interface Description Language (IDL).
A Common Object Model. The CORBA infrastructure proposes an object model used for the interactions across the different platforms.
Dealing with Legacy Systems. The use of IDL simplifies the development of interfaces to access legacy application systems.
CORBA defines the communication architecture between the client and server applications. The main component of this architecture is the Object Request Broker (ORB). The client requests the use of a service to the ORB, using CORBA API (Fig. 5). The ORB is responsible for locating the service on the network (which was previously registered to the ORB). The service is then accessed via the Object Adaptor.

4.2 Using CORBA for ROPE

CORBA may be used to support the interactions between the ROPE shells. The question really is how to use CORBA services efficiently, while preserving the shells structure as much

as possible. In the original shell, the Nerwork Monitor handles all network related tasks, namely, sending and receiving messages. These messages always correspond to some service request made to the shell. Thes requests are always dispatched to the appropriate module by the Message Processor, with an exception that global queries are sent directly by the Network Monitor to the Result Integrator. From this, it can be argued that the Network Monitor can act as the CORBA server and provide the entry point to all requests sent to a shell. It can also be argued that all service requests are made by the Message Processor, which controls all the processing of messages, therefore the logical interactions with other nodes. The Message Processor will therefore act as a CORBA client (Fig. 6).

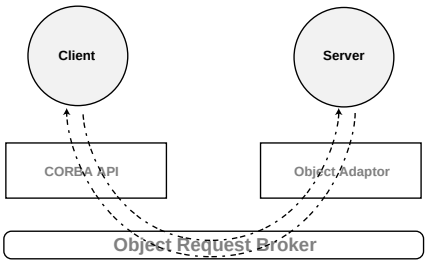

Figure 5 : Processing a Service Request in CORBA

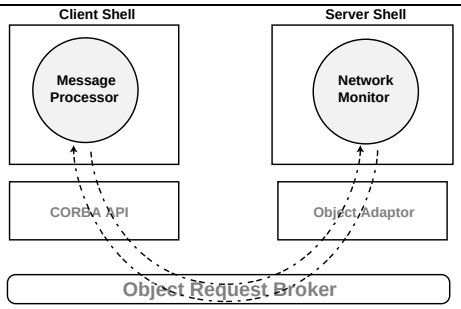

Figure 6 : Processing a Service Request in ROPE using CORBA

All the shells provide the same services and therefore have the same IDL interface description. What distinguishes a shell from other shell is its identifier. Therefore, the binding of a service request is dynamic, since the shell must select the right server (shell) to perform the required service.

5. CONCLUSION

We developed a small scale prototype to evaluate the use of CORBA to handle the communication between the shells. The results showed that multiple shells may share the same IDL interface description, provided that they are uniquely identified. In ROPE, this problem is readily solved, since each shell has a unique identifier, which can be used to dynamically select the appropriate interface (therefore, the appropriate shell). From this result, we conclude that a single communication protocol should be used to simplify the implementation of the shell. We are currently working on a prototype ROPE shell using Java where the communication layer will be handled by TCP/IP, independantly from CORBA.
The main limitations of the approach reside in two aspects~:

- Prior knowledge is required about the different systems (functions supported, databases used, database structures, shared data elements). This knowledge, however, represents all the information an enterprise should have about its information systems in order to use and manage them properly. This knowledge may be changed over time. The shell will still be able to perform its tasks properly, since the structure of the shell is static. The shell's adaptiveness is achieved by the use of a rule segment, which is in fact a subset of the Metadatabase.
- Interfaces between the shell and the local system must be developed on an *ad hoc* basis. This limitation is not as difficult to overcome as it seems. In the original shell, the local configuration of the shell included a series of commands that may not be supported by the local C compiler and the local operating system. The use of Java to develop the shells solves most of these problems. It does not, however, solve the access to the local database problem, which depends on the database management system (DBMS) used. It may be argued that the number of DBMS vendors is not that large and that specialized modules may be developed for each of these vendors. Furthermore, the use of JDBC (Java Database Connectivity) methods may be seen as a transparent solution to this last problem.

Therefore, the key to integration is and remains knowledge about the systems to integrate. If information systems are willing to share there metadata, which should not compromise their data, at least not directly, an approach similar to the Metadatabase could be developed to share processes and data. It would seem interesting to investigate how the Metadatabase and ROPE concepts may be implemented in a WOS environment.

REFERENCES

[1] G. Babin. Adaptiveness in Information Systems Integration. PhD thesis, Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, Troy, N.Y., August 1993.

[2] G. Babin and C. Hsu. Decomposition of knowledge for concurrent processing. IEEE Transactions on Knowledge and Data Engineering, 8(5):758–772, 1996.

[3] Bahri, S. La base de métadonnées et l'architecture concurrente : des «shells» orientés objet. Masters thesis, Département d'informatique, Université Laval, January 1998.

[4] M. Bouziane. Metadata Modeling and Management. PhD thesis, Computer Sciences, Rensselaer Polytechnic Institute, Troy, N.Y., June 1991.

[5] W. Cheung. The Model-Assisted Global Query System. PhD thesis, Computer Sciences, Rensselaer Polytechnic Institute, Troy, N.Y., November 1991.

[6] W. Coleman et al. The FUSION method. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1994.

[7] D.M. Dilts and W. Hua. Using knowledge-based technology to integrate cim databases. IEEE Expert, 3(2):237–245, 1991.

[8] C. Hsu. Enterprise Integration and Modeling — the Metadatabase Approach. Kluwer Academic Publisher, Boston, Mass., USA, 1996.

[9] C. Hsu, G. Babin, M. Bouziane, W. Cheung, L. Rattner, and L. Yee. Metadatabase modeling for enterprise information integration. Journal of Systems Integration, 2(1):5–39, January 1992.

[10] C. Hsu, M. Bouziane, L. Rattner, and L. Yee. Information resources management in heterogeneous, distributed environments: A metadatabase approach. IEEE Transactions on Software Engineering, 17(6):604–625, June 1991.

[11] Mowbray, T.J and R. Zahavi. The Essential CORBA. John Wiley, 1995.

[12] Object Management Group. The Common Object Request Broker: Architecture and Specification. ACM Press, 1995.

[13] Shaefer, O. and C. Hsu. Distributed Rulebases for Real Time Process Control on the Shop Floor: The Metadatabase Approach. ASME Material Handling, 2:49–59, November 1994 .

[14] Tao, Y. Differential Control on Distributed Database Updates Using Concurrent Rulebase Shells. PhD Thesis, Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, Troy, N.Y., June 1997.

[15] W. Wu and D. M. Dilts. Integrating diverse cim data bases: The role of natural language interface. IEEE Trans. on Syst., Man and Cyb., 22(6):1331–1347, 1992.