Integration of Real-time Object-Oriented Database and Real-time CORBA into Legacy Software

BY
QINGLI JIANG

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

1998

Abstract

The Joint Force Level Execution Aid (JFLEX) is military software that collects, dispatches, processes and saves data from different organizations. All the current JFLEX client and server software works upon the CORBA middleware, which limits the portability of the system. Although time constraints of tasks and data are important to JFLEX, JFLEX lacks the ability to enforce timing constraints.

In this thesis, we present a solution to enhance the portability and real-time features of JFLEX by integrating CORBA and the real-time object-oriented paradigm into JFLEX. The new JFLEX uses IIOP to work across different ORBs to increase its portability. A CORBA interface has been developed to isolate the JFLEX server from the database. The interface hides all the implementation details from the JFLEX and provides the possibility to connect to different types of database management systems (DBMS). In order to meet JFLEX's real-time requirements, we used URI's Real-time Open Object-oriented Database (Open OODB). In this project we integrated Open OODB by creating a "wrapper" for it that facilitates its replacement.

Acknowledgement

I have always known that there are so many people out there who have helped me in many ways during my struggle with the thesis. First, I would like to thank my major advisor Dr Victor Fay-Wolfe. Without his guidance and wisdom, I would not have had a chance to get through this interesting thesis research.

Also I would like to thank Mike Squadrito who has given me great even after he left URI and Dr Lisa Dipippo who is so kind to help me understand her semantic locking mechanism theory.

I wouldn’t forget to thank my teammate Yuruo Chen who has made the implementation much easier and more enjoyable than it would have been.

Finally, I must thank my parents for their support and love throughout all these years. I also thank my husband Jie for all his love and for comforting me whenever I was too nervous about my problems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Statement of Problem

Joint Force Level Execution Aid (JFLEX) is military software that provides a user-friendly method for monitoring the progress of a plan's execution. One of JFLEX's functions is to gather objectives, activities and status information of plans from different responsible organizations and save them into database for later processing.

The Common Object Request Broker Architecture (CORBA) is a specification of client/server architecture provided by Object Management Group (OMG). It is an interface that allows an application to make request of objects (servers) in a transparent, independent manner, regardless of platform, operating system or locale considerations. Clients interact with servers through the Object Request Broker (ORB), the central piece of middleware in CORBA.

JFLEX was originally implemented by using CORBUS, which is a CORBA compliant software system from BBN Inc. The General Inter-ORB Protocol (GIOP) is the OMG protocol for ORB interoperability. It allows different ORB implementations to communicate without restricting ORB implementation flexibility, which means a client

of an ORB can invoke operations on an object in a different ORB via GIOP. Internet Inter-ORB Protocol (IIOP) is the mapping of GIOP message transfers to TCP/IP connections. CORBUS did not provide IIOP compliance, which means that a JFLEX client and a JFLEX server have to work on the same CORBUS ORB.

The Database management system (DBMS) used by JFLEX is Mini SQL (MSQL), which is a lightweight relational DBMS. Even though this database has been designed to provide rapid access to data sets with as little system overhead as possible, it does not suited very well to a real-time application like JFLEX due to its lack of support for managing objects.

Object-oriented database models (OODM) are better suited for a real-time system than relational data models. In object-oriented databases the only way to access or update the database entities is through the methods defined on an object. So, the use of object-oriented databases allows the database concurrency control mechanism more flexibility in granting locks based on the semantics of the individual methods of the object.

In order to improve the real-time support of JFLEX, this thesis project first converted JFLEX from CORBUS to Orbix, which is another commercial CORBA implementation. Since MSQL does not provide the same level of concurrency control as the real-time Object Oriented Database (OODB), the project provided a CORBA interface that allows replacement of MSQL by a real-time OODB that will allow JFLEX to update and process the data in a much better way.

1.2 MOTIVATION

After a system is installed it has a certain lifetime and as time goes by it may not be able to meet more and more user's requirements. So people need to find ways to leverage their legacy assets when designing and implementing new architectures and next generation systems. The goal is to reuse most of the legacy system. How to get the most benefit from it becomes a challenge.

Generally, there are two possible solutions: legacy extension and legacy integration [1]. Legacy extension modifies legacy systems without determining how the system fits into, and contributes to, the new requirements and information architecture. Legacy extension focuses on correcting system deficiencies and enhancing the system while addressing the short-term requirements. On the other hand, legacy integration tries to reuse the legacy system to implement a new architecture without propagating the weaknesses of past design and developments methods. Integration hides legacy systems behind consistent interfaces that hide implementation details and allows for changing or replacing implementations without affecting other systems later on.

In this thesis, we will demonstrate that legacy integration is a good solution by applying it to the integration of JFLEX with CORBA technology and a real-time OODB. Since transactions are executed in a first-in-first-out manner in the old JFLEX, the transaction with shorter deadline and higher importance that follows the transaction with longer

deadline and lower importance may have a chance to miss the deadline. Therefore, the new JFLEX requires the ability to process both transactions and data with time constraints. This introduces the need of real-time database system. We also used CORBA to enhance the distributed ability of JFLEX and encapsulate the existing implementation.

1.3 Objectives

In this study, our primary research objective is to develop a model to integrate real-time CORBA and real-time OODB into the existing JFLEX software to meet distributed and real-time requirements. We will demonstrate the steps of evaluating legacy code and new requirements and integration. Finally, we will apply it to the development of the new JFLEX to show that legacy integration is a practical solution of reusing existing systems.

1.4 Thesis Outline

Chapter 2 describes work that is the background or related to the integration, including CORBA, real-time OODB and related team work that is going on at the same time as this research. Chapter 3 presents the system design of the integration. Chapter 4 describes the implementation and integration details of JFLEX with CORBA and real-time OODB. Chapter 5 demonstrates the performance evaluations of the new JFLEX. Chapter 6 concludes the thesis with a summary and discussion of the contributions, limitations, and future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we will describe the background and work related to the design and implementation of the integration of JFLEX. First, we will introduce CORBA. Next we will explain the real-time OODB used for JFLEX, and show the advantages of it over traditional related Database Management System (DBMS). We also analyze the semantic content, usage patterns and the structure of the system. Finally, we will introduce another related part of the JFLEX project, which is concerned with integrating real-time scheduling into JFLEX.

2.1 Common Object Request Broker Architecture(CORBA)

OMG is an international organization supported by over 500 members, including information system vendors, software developers and users. It creates standards for distributed object computing that are realistic, commercially available and usable. OMG realizes the standards through its Object Management Architecture (OMA) at the heart of which is the CORBA. CORBA specifies the ORB technology that allows applications to communicate with one another, no matter where they are located on a net. It defines a framework for different ORB implementations to provide common ORB services and interfaces to support portable clients and implementations of object [2]. CORBA also presents new opportunities for the integration of legacy systems.

2.1.1 The Object Model

CORBA is based on a object model of computation. In this model, services are provided in the context of a set of objects and requests that can be made relative to those objects. A provider of services, a server, “contains” a set of objects and performs operations on those objects upon requests received from clients. The server and clients are isolated by a well-defined encapsulating interface. The definition of such interfaces is an important part of CORBA and is the purpose of CORBA’s Interface Definition Language (IDL).

2.1.2 Interface Definition Language (IDL)

A key element of CORBA is IDL. IDL is used to describe the interfaces of service provided by an object. The semantics of the interfaces are defined by the implementation of the object and are not relevant to CORBA implementations. The role of a CORBA implementation is to provide an IDL compiler and support the data transfer protocols required to carryout object requests sent by clients.

IDL is purely a declarative language. It includes interfaces, types, operations and modules. IDL has been defined as an independent language in order to support multiple programming languages. Clients may invoke the functions provided by IDL without knowing where the object is and how it is implemented. This provides an opportunity for reusage since a CORBA interface (wrapper) can be programmed for an existing application, to make its functionality directly accessible from other programs. For more details about how to use IDL please read [3].

2.1.3 Object Request Broker (ORB)

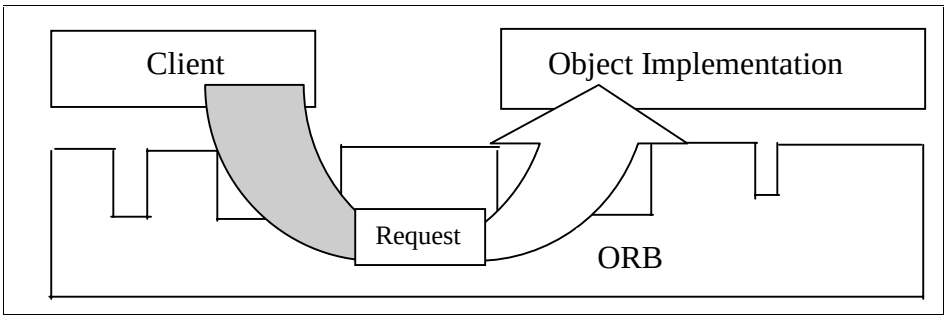


Figure 2.1.3 ORB Structure

The Object Request Broker (ORB) provides the mechanisms by which objects transparently make requests and receive responses. ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems. In the figure above, the client is an entity that wishes to perform an operation defined in the IDL of the object and the object implementation is the code and data that actually implement the object. The ORB is responsible for all of the mechanisms required to find the object implementation for the request, to prepare the object implementation to receive the request, and to communicate the data making up the request. The IDL the client sees is completely independent of where the object is located, what programming language it is implemented in, or any other aspect which is no reflected in the object's interface [2].

2.1.4 Internet Inter-ORB Protocol (IIOP)

The goal of the ORB Interoperability is to allow communications between independent implementations of the CORBA standard, which means that it allows a client of one ORB to invoke operations on an object in a different ORB via an agreed protocol. Thus, invocations between client and server objects are independent of whether they are on the same or different ORB.

The OMG protocol for ORB interoperability is called the General Inter-ORB Protocol (GIOP). GIOP defines the on-the-wire data representation and message formats [2]; it also makes general assumptions about the nature of the transport layer – in particular, that it is connection-oriented. The OMG defines a specialization of GIOP that uses TCP/IP as the transport layer. This specialization is called the Internet Inter-ORB Protocol (IIOP). CORBA and IIOP assume the client server model of computing in which a client program always makes requests and a server program waits to receive requests from clients. All these communications are done through the Internet's transport layer using the Transmission Control Protocol (TCP).

An object accessible via IIOP is identified by an interoperable object reference (IOR). An IOR is managed internally by the interoperating ORBs. It is not necessary for an application programmer to know the structure of an IOR.

2.1.5 CORBA Implementations

In this project, we involved the following two CORBA Implementations:

• ***Corbus*** **-** BBN's Corbus is a CORBA-compliant, distributed, object-oriented system that facilitated the development of distributed applications in both local and wide area environments. Developed by BBN under the sponsorship of the United States Air Force's Rome Laboratory and the Naval Command, Control and Ocean Surveillance Center (NCCOSC) Research, Development, Test and Evaluation Division (NRaD), Corbus specifically addressed the U.S. Military's requirements to build and operate mission-critical information systems over globally dispersed wide area networks. But it didn't support IIOP in CORBA 2.0.

•***Orbix*** **–** Orbix is the market leading CORBA 2.0 compliant Object Broker Request developed by IONA Technologies PLC. Orbix supports the native Orbix protocol and IIOP as alternative protocols. In order for clients to know about an IOR, a server must publish that IOR. One way of publishing an IOR is to make the name of an object known via the Naming Service. If the Naming Service is not available, servers must use some ad hoc ways of publishing an IOR. One way is to convert the interoperable object reference to a string and write it to a file. A client receives an IOR published by a server. For example, by reading the string form from a file and transforms it to an Orbix reference of the desired type and then makes invocation in a normal way.

2.2 Real-time Object-oriented Database (RTOODB)

Real-time applications such as JFLEX require time constraints for both the transactions and the data that reflect the state of the application's environment. The time constraints for transactions mean the computations must complete before their deadline. The time constraints for data require updating the data within a specified time interval so as to accurately reflect the environment it represents. A traditional database system only provides part of the functionality required by these applications, such as consistent access to shared data. A real-time database system combines the features from both real-time systems and database systems. The concurrency control for a real-time database must be capable of maintaining the logical and temporal consistency of the data as well as the transactions. In the following sections we will introduce the feature of a real-time database, OODB (for a survey of object-oriented database research see [4]) and a real-time OODB research model developed in University of Rhode Island (URI).

2.2.1 Real-time Database

A real-time database provides data management services for applications that require typical logical consistency as well as temporal consistency for both transaction and data.

Transaction temporal consistency constrains when each transaction must execute. Correctness can range from requiring no support to requiring full capability of enforcing timing constraints such as start times, deadlines, and periods on transactions.

Transaction logical consistency expresses how a transaction must be scheduled with respect to accessing data objects. Transaction-based semantic correctness criteria allow the designer to express logical correctness of a transaction schedule based on knowledge of the specific application and of the actions performed by each transaction. These criteria are enforced through techniques such as those that employ user-defined compatibility sets of transactions [5]. Epsilon-serializability provides increased concurrency by allowing each transaction to define its own limits on the amount of inconsistency that it may view and that it may write [6].

Data temporal consistency correctness criteria can range from requiring no temporal constraints to absolute temporal constraints to requiring absolute temporal consistency and / or relative temporal consistency of data.

Data logical consistency is a type of consistency (data integrity) that is maintained in a traditional database.

Unfortunately, it is difficult to satisfy all four requirements at the same time because of the incompatibilities among them [7]. For instance, preserving temporal consistency of a data item may require preempting a transaction that is using the data item in favor of an update transaction. However, this preemption may violate the logical consistency of the data item and/or the logical consistency of the preempted transaction. As a result, most real-time database systems only guarantee soft transaction deadlines.

2.2.2 Object-oriented Database(OODB)

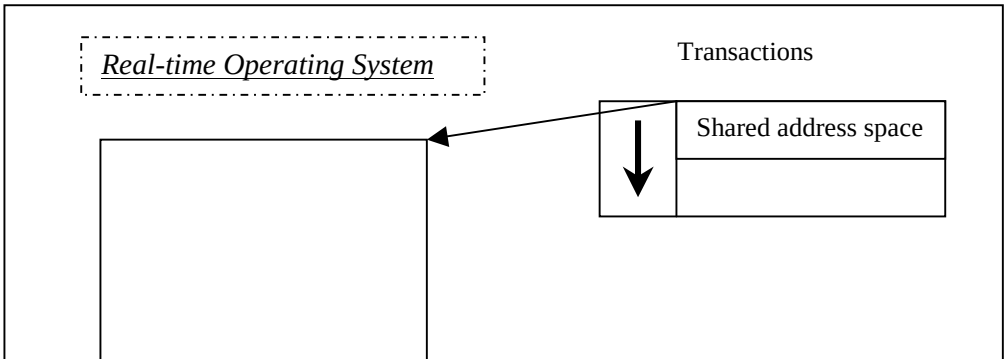
An OODB manages objects instead of tables of data as in a relational database. An OODB is more suitable for supporting logical and temporal consistency in transactions and data than a relational database because of the following reasons:

1) The encapsulation mechanisms of an OODB allow concurrency control specific to a data object to be enforced within the object;

2) The capability of including user-defined operations(methods) on data objects can improve real-time concurrency by allowing a wide range of operation granularities for semantic real-time concurrency control;

3) An OODB potentially makes it easier to integrate constraint expression and checking as compared to relational models.

The OODB we are going to use in this project is Open OODB. Open OODB is a researching product from Texas Instruments that it uses Exodus to store data in files on disks [9]. The research group at URI has converted it into a shared-memory OODB to provide better performance. Since the data are in the main memory, the longest lifetime of data can only be the lifetime of the operating system kernel. If the power fails, the data will be lost. The following diagram shows the structure of this shared-memory OODB:



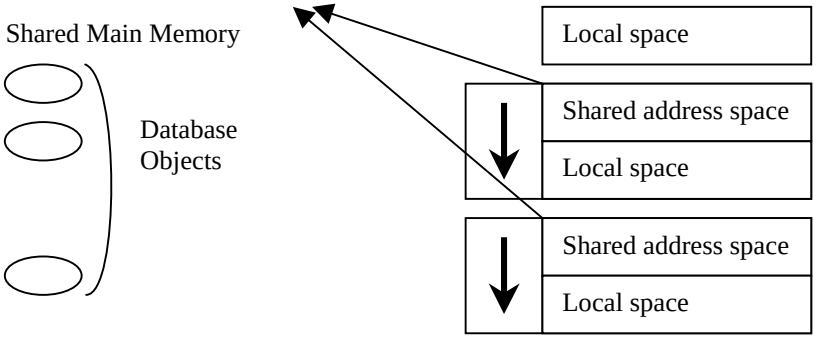


Figure 2.2.2 The Architecture of OODB

Data are saved in the shared main memory as objects. Transactions can access them as if they were in their local address space. Any change that is made by one transaction can be seen by other transactions.

2.2.3 The RTSORAC Model

The semantic locking concurrency control technique is based on a URI research model of a real-time object-oriented database called RTSORAC (Real-Time Semantic Objects, Relationships and Constraints)[9]. It extends object-oriented database models by incorporating time into objects and transactions. Incorporating time allows for explicit specification of data temporal consistency and transaction temporal consistency. RTSORAC consists of a database manager, a set of object types, a set relationship type and a set of transactions. The database manager performs typical database management operations including scheduling of all executions on a processor, but not necessarily including concurrency control. The database object types describe the structure of

database objects. The relationships specify associations between objects and are used to define inter-object constraints. The transactions are executable entities that access the objects and relationships in a database.

2.2.4 The Semantic Locking Technique

Typical database management systems provide concurrency control techniques that preserve logical consistency of data items (e.g. read/write locking) and logical consistency of transactions (e.g. two-phase locking). Although these two requirements still exist for concurrency control techniques in real-time databases, two additional requirements are needed to be supported: the real-time scheduling algorithm in enforcing transaction timing constraints, and preservation of data temporal consistency constraints. The semantic locking technique integrates support for the four real-time concurrency control (RTCC) requirements: transaction temporal consistency, transaction logical consistency, data temporal consistency, and data logical consistency.

Designing a concurrency control technique that meets all four forms of Real-time Concurrency Control requirements is difficult because of the reason mentioned above. Therefore, a concurrency control technique must be capable of expressing the tradeoffs in sacrificing one requirement for another. These tradeoffs are application-specific. In the semantic real-time concurrency control technique that we are going to use for JFLEX, the Concurrency Control mechanism of each object uses semantic locking to enforce the allowable concurrency expressed by the compatibility function of the object. A

transaction must acquire a semantic lock for a method invocation before the method is allowed to execute.

2.3 Joint Force Level Execution Aid (JFLEX)

JFLEX is distributed software developed by the U.S. Navy Research Lab in San Diego, CA (NRaD). It depicts the status of a plan by graphically organizing the plan's objectives, chains-of-command, activities and status information into an easy-to-read display. A plan is composed of responsible organizations and subplans (sequences of actions). Each responsible organization is assigned a subplan for execution. JFLEX distributes the status information for a plan to each workstation running a JFLEX client over a wide area TCP/IP network. It also allows clients to change and update the status of plan activities. This allows managers to get the information of how well a plan is succeeding. JFLEX estimates plan viability by monitoring the continued validity of the underlying assumptions of the plan.

JFLEX consists of three main operational modules: Plan Editor, Plan Monitor, and Plan Server. The Plan Server is a central server process. It manages a central database containing plan-related information and provides IDL to support requests from client processes through CORBA. The plan Editor serves as a client process that allows users to create a plan representation consisting of plan objectives, subplans, responsible organizations, conditions and states. The client Plan Monitor processes display the plan information and support the user interface for monitoring plans.

- **JFLEX Architecture**

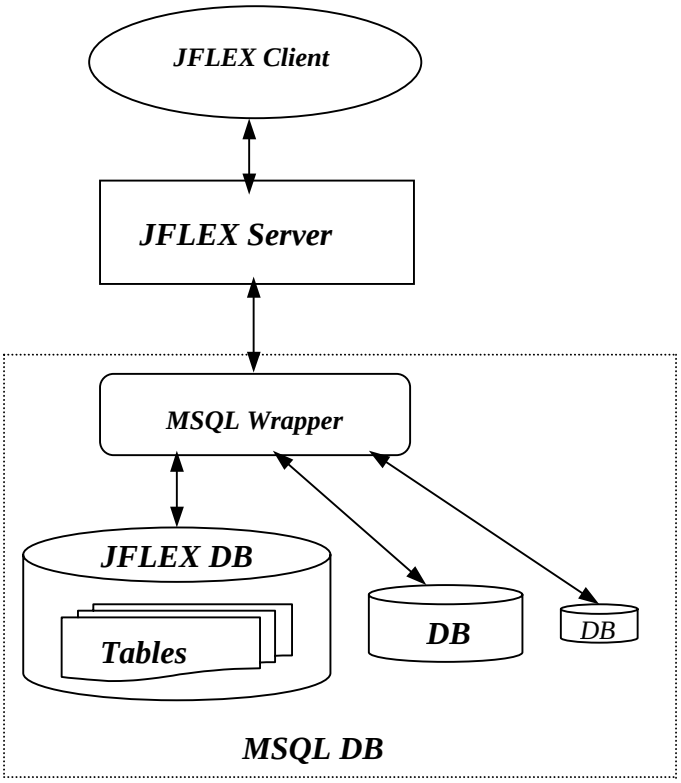


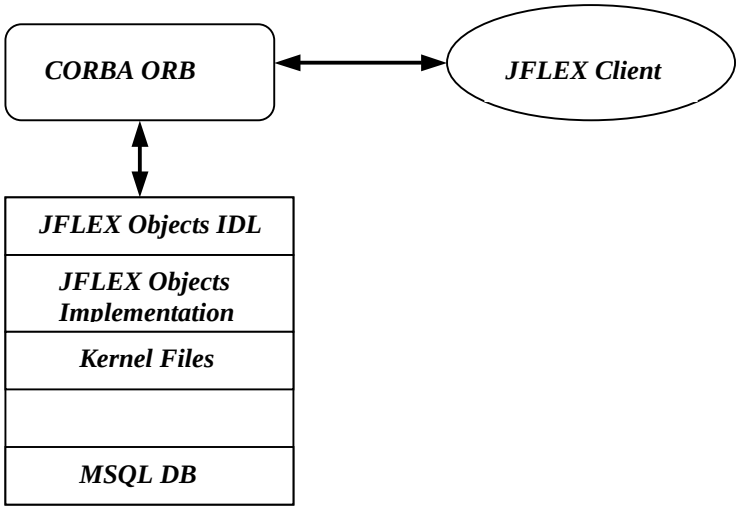
Figure 2.3.1 The JFLEX Architecture

Figure 2.3.1 shows the architecture of JFLEX. A JFLEX client includes a Plan Editor and a Plan Monitor. JFLEX Server is a Plan Server. All the data managed by JFLEX are saved as tables in a MSQL database. The MSQL Wrapper is a C++ interface provided by MSQL database management system. The JFLEX server manipulates the data in the JFLEX database through the MSQL wrapper.

- **Internal Structure of JFLEX Server**

The JFLEX server contains objects that provide CORBA IDL to the JFLEX clients. These IDL interfaces hide all the implementation details from the clients and offer great flexibility to reuse and update the old code. The clients make a call to objects without knowing their locations, which means remote calls appear similar to local calls. The CORBA ORB processes requests and returns results to the client transparently. In the original JFLEX, the CORBA ORB is CORBUS. The JFLEX objects implementation create and use objects in several kernel files that communicate with the MSQL wrapper.

In the Figure 3.1.2, a JFLEX client makes a request to the JFLEX server. The format of the request is the same to the client no matter whether it is a local server or a remote one. The underlying CORBA ORB is responsible for selecting of a JFLEX server that can process the request and for handling the communication between the client and the server. The kernel files layer contains objects that provide member functions to communicate with the MSQL wrapper. The object implementation creates objects in the kernel files layer to manipulates and maintains the data in MSQL database according to the request from the client. Finally, the server sends the reply back to the client through CORBA ORB.



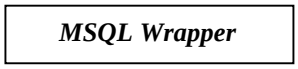


Figure 2.3.2 The Internal Structure of JFELX

2.4 Related Team Work

CORBA is currently inadequate to support real-time requirements. For example, IDL describes the interface to the functional behavior distributed components, but does not explicitly describe timing constraints for their behaviors. Furthermore, system services provided by distributed environments offer little support for end-to-end real-time scheduling across the environments. The real-time research group at URI has extended ORBIX to a subset of a real-time CORBA system, which provides additional services such as real-time event service, priority server for scheduling, and extends the concurrency control service [10].

Because of the real-time features of JFLEX, the related teamwork done by Yuruo Chen focuses on changing it into a real-time distributed application using URI's RT CORBA. The new JFLEX uses a data structure (Real-time Environment) to keep the necessary information to process a CORBA request in real-time. The RT_Manager and Pserver developed by the RT research group at URI [10] are used to handle the real-time environment and schedule according to the environment variable.

CHAPTER 3

SYSTEM DESIGN

The goal of this thesis research is to develop a practical model to integrate real-time CORBA and real-time OODB into the legacy system JFLEX. In the system design phase, we evaluated the old structure with new requirements. Finally, we created a high level model that not only makes the maximum use of the old code and makes least change out of it but also meets the new requirements.

3.1 The System Requirements

The system requirements of JFLEX fall into four categories: portability, real-time, extensibility and isolation. We now discuss each of these requirements.

- **Portability**

Since CORBUS does not support IIOP, the JFLEX server and clients have to work under the same ORB. However, a new client that uses Orbix cannot communicate with JFLEX objects. The new JFLEX must work across multiple ORBs, therefore we use IIOP to hide the underlying ORB.

- **Real-time Features**

In order to meet the deadlines of transactions and keep the consistency of data in the database, JFLEX needs real-time scheduling to make a better effort to avoid missing deadlines of transactions by arranging them in an more efficient way [see the related team work section]. JFLEX also needs a real-time OODB to maintain both logical and temporal consistency of transactions and data in the manipulation of the database.

- **Extendibility and Isolation**

JFLEX should be able to use multiple database management systems instead of just a specified one. This requires the database interface to be well designed so that it can be easily adapted to various database management systems. Isolation requires that users can access data through the interface without the knowledge of which database they belong to. The implementation of the interface must process the request from the user and select an appropriate database to do all the operations.

3.2 The System Analysis

To meet the above requirements, we first need to transport JFLEX from CORBUS to Orbix. In the transportation, the architecture of JFLEX will remain unchanged. The kernel files layer of JFLEX and the MSQL wrapper layer have nothing to do with the CORBA ORB, so we can leave them unchanged.

To integrate a real-time a OODB into JFLEX, we seek to minimize changes to legacy codes. The kernel files layer can only use the objects with similar interface as the MSQL wrapper. We chose to implement the real-time OODB with a wrapper with a similar interface to MSQL. We also ensured that the structure of the result returned from the database is the same in both the MSQL wrapper and real-time OODB wrapper. In order to satisfy extendibility and isolation requirements, we can use a CORBA server to deal with the MSQL wrapper and OODB wrapper and provide a unique interface to users. At this point, the kernel files layer becomes a client of this server that does not know which database it reaches.

3.3 The System Design

Based on the analysis in the last section, we can follow the below steps to finish the design of the new architecture of JFLEX.

3.3.1 Integration Techniques

Using a object wrappers is one method for bridging the new application architecture's abstract interface to the legacy software identified by the technical architecture. On one side of the bridge, the wrapper presents systems with a clean, abstract interface. On the other side, the wrapper communicates with legacy systems using their existing facilities or other new systems. Object wrappers provide a natural way to integrate legacy systems with new systems. Once wrapped, legacy systems can participate in the new architecture

with least change and have a chance to extend its system structure to meet more requirements. At the same time, wrappers provide constant interfaces to other systems as they do with the legacy systems. This keeps systems from being aware of implementation changes and eliminates the need for modifying these systems.

The key benefit of using object wrappers is that they provide consistent services to both legacy and new systems. The following are the approaches to object wrapping.

- **Layer –** Layering, the most basic type of wrapping, maps one form of interface onto another. The functions provided by layering depend on the sophistication and availability of the legacy system's existing interface. If a legacy system has no or minimal interface, little can be done using layering techniques. Layering can be used to aggregate multiple legacy systems. Conversely, layered object wrappers can partition a complex legacy system into multiple objects.

- **Data Migration –** Data migration maps the existing data model onto a different model. Wrapping involves adding a layer that bridges the new data model and the legacy database. Data migration wrappers change database schemes and the databases themselves without affecting the objects that use these wrappers.

- **Middleware –** Middleware is system integration software for distributed processing and for database and user interfaces. CORBA is one kind of the middleware that can be used to do the integration. Database middleware provides common access mechanisms

for using a variety of database systems and file structures. It allows a system to issue a single information request and to access, in turn several data sources that can each be using a different database management system.

- **Encapsulation –** Encapsulation, the most general form of object wrapping, separates interfaces from implementations. All accesses, including direct and indirect accesses, are performed through interface methods. Using interface allows implementation details to be changed without requiring other changes. Using CORBA and its IDL is a good solution to encapsulate systems that hide differences in programming languages, systems locations, operating systems, algorithms and data structures. In the context of system architecture, wrappers translate between the architecture and legacy systems. Wrappers should implement the architecture in all of its details, including providing metadata and data conversions.

3.3.2 The New Architecture of JFLEX

In the new architecture of JFLEX shown in Figure 3.3.2, we selected CORBA as database middleware for the distributed and encapsulation purpose. Thus, the CORBA query server provides a common interface for querying data in the database, which can be used by other systems besides JFLEX. Because of the distributed feature of CORBA, the JFLEX objects and database do not need to reside on the same host. Since the real-time OODB is a shared memory OODB, it has to reside on the same host as the CORBA query server so as to retrieve the data stored in the local main memory. IDL encapsulates the

implementation details. So the objects above it are unaware of how the server decides which database management system to be used. It is also convenient to extend to other database management system independently of others.

Above the CORBA query server in Figure 3.3.2, we mapped the MSQL wrapper onto one layer, the C++ query client, which makes use of CORBA IDL to manipulate data in the database. We need this layer because the legacy kernel files must instruct the MSQL wrapper. This layer serves as a CORBA client and must convert the query result sent back by the CORBA query server into the data structure that MSQL wrapper has, otherwise the kernel files cannot process them.

We also need to encapsulate real-time OODB since it cannot process query statements (SQL). We added an OODB wrapper that provides basic ability to process query statements and bridges the MSQL data structure and the OODB data. So the wrapper has the similar interface as MSQL wrapper and it converts the query result into the data structure of MSQL wrapper. Thus, the data structure of query result got by CORBA query server is just the same no matter which database management system it comes from.

Our design uses several object wrapper techniques to isolate and encapsulate the database part of JFLEX yet to keep other parts unchanged.

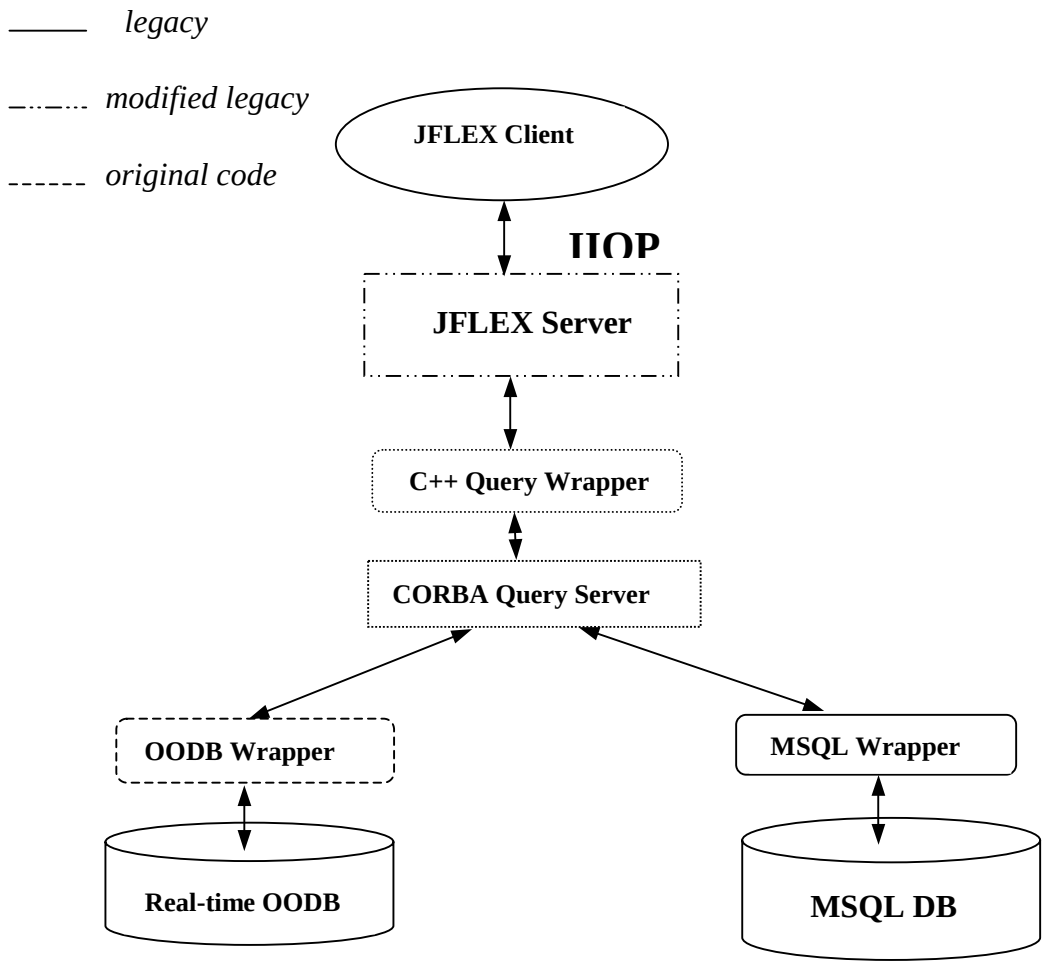


Figure 3.3.2 RT JFLEX Architecture

CHAPTER 4

THE IMPLEMENTATION

In this chapter we will describe the implementation of the new JFLEX architecture based on the design presented in Chapter 3. In the following sections, we will explain the implementation of each part of the new architecture in full details.

4.1 MSQL Wrapper

The MSQL Wrapper consists of several C++ objects to provide functionality to access and retrieve data in MSQL [see figure 4.1]. The MsqlDB is an important part of the wrapper. It opens the connection from a user to a MSQL database daemon, selects a database and sends queries to daemon. The result will be sent back in a instance of MsqlResult which is a set of rows (tuples).

Class MsqlDB

{ public:

Boolean connect (string host_or_ip_addr);

//provide the name or IP address of the host running the Msql daemon. Returns true if it connects

//successfully.

void disconnect( );

//disable the connection

Boolean isConnected( );

Boolean selectDB (string db_name);

//choose a database by name

int query(string query);

int query(string query, MsqlResult &);

//use the first query if there is no result back from the database, like creating a table

//use the second query if you want an MsqlResult to be set, like select queries

//the input string should be a valid SQL statement

//returns the number of rows affected by the query or –1 unsuccessful

};

The result is returned as an instance of MsqlResult. From an MsqlResult object you can generate objects of class MsqlRowIter, MsqlRow, MsqlFieldIter, MsqlField and finally get raw data of the result by using MsqlField::getRawData( ).

```
class MsqlResult
{    public:
     void set(m_result *);
     Boolean isGood ();
     MsqlRowIter getRowIter ();
     int numRows( );
     int numFields( );
};
```

If the instance of MsqlResult is valid (can be checked by using MsqlResult::isGood( )) , MsqlRowIter can be gotten from MsqlResult and users can use it to retrieve all rows.

```
Class MsqlRowIter
{    public:
//derefernce the iterator
     MsqlRow & operator*();
//pre-increment the iterator
     MsqlRowIter & operator++();
     Boolean isGood();
};
```

From the instance of MsqlRowIter, users can get a instance of MsqlRow that represents a tuple.

Class MsqlRow

```
{   public:
    MsqlFieldIter getFieldIter();
    Int numField( );
};
```

From the instance of MsqlRow, users can get a MsqlFieldIter instance that can be used to retrieve the fields in the tuple.

Class MsqlFieldIter

```
{   public:
    MsqlField & operator *();
    MsqlField & operator++();
};
```

By using the instance of MsqlFieldIter, users can get a instance of MsqlField that can get the raw data of the field. The raw data are in the format of string and users need to convert them to the corresponding data type.

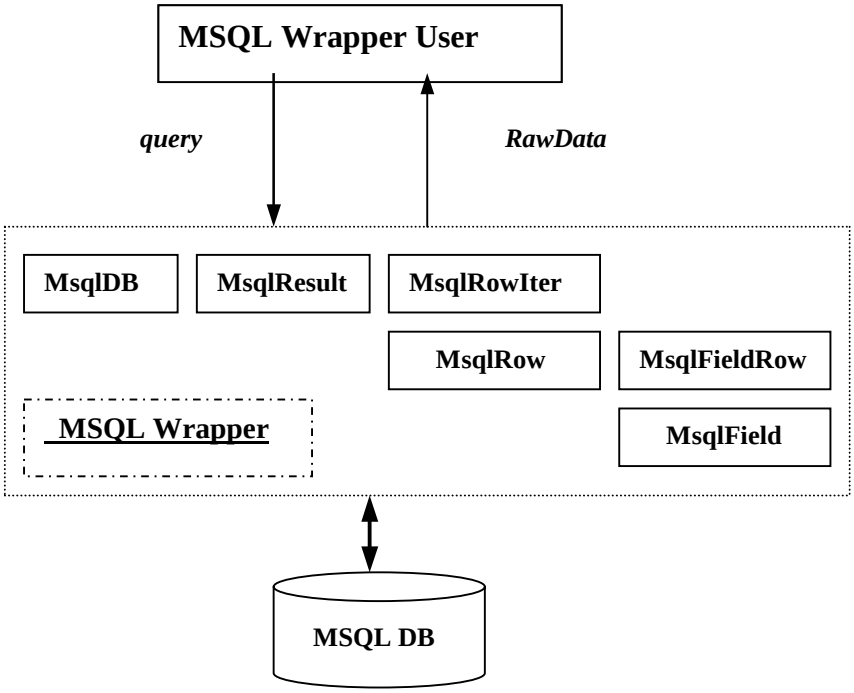


Figure 4.1 MSQL Wrapper Classes Architecture

```
Class MslqField
{    public:
     string getRawData( );
};
```

The basic data structures that the MSQL Wrapper deals with are shown below. They save all the information about the field and the value of the result.

```
string m_row;     //the raw data of one row
struct m_data   //a link list of rows
{    m_row data;
     m_data* next;
};
struct m_field    //field information
{    string  field_name;
     string  table_name;
     int field_type;
     int field_length;
};
struct m_fdata   //a link list of fields
{    m_field field;
     m_fdata* next;
};
struct m_result   //keeps the information about the rows and fields
{    m_data * query_data;
     m_fdata * field_data;
     int numRow;
     int numFields;
};
```

Note that MSQL data structures and classes can represent the result and manipulate it in a clean and efficient way. We decided to make the real-time OODB wrapper and CORBA server IDL a similar interface as MsqlDB and store the result in the MSQL data

structures. By doing so, we can use the implementations of MsqlResult, MsqlRowIter and other classes directly that allows us to access the raw data in the data structures without rewriting the codes again. We only need to link the library of these classes into the object file. This certainly makes the best use of the legacy asset of the MSQL wrapper and makes our work easier.

4.2 The Real-time OODB Wrapper

The real-time OODB that we are using in this study is a shared memory database developed at URI[11]. It provides different concurrency control options we described before and allows users to specify the semantics for compatibility function [7] to meet different application requirements.

This real-time shared memory OODB has very good performance compared to the original Open OODB with concurrency control disabled [11]. It only provides basic functionality of manipulating data stored as objects in shared memory:

- *Insert one object into a list*
- *Delete one object from a list*
- *Fetch a list of objects that can be retrieved one by one.*

So we need a wrapper that can accept a query statement, parse it, do the corresponding operations on the objects and converts the result into MSQL data structures. The

following is the interface of OODB wrapper. It is quite similar to the interface of the MsqlDB class.

```
Class OpenDB
{   public:
    int connect( );  //create an OODB object and be ready to do basic operations
    int isConnect( );
    long query( string query);
    long query(string query, MsqlResult & );
    . . .
};
```

Besides the wrapper, we also need to convert the relational tables into objects in OODB. Basically, we made one object out of one table for convenience and consistency.

The fields in a table became the private members of a object and users can access and update the information of them through the public member functions. In the OpenDB class, we have a private pointer point to the list of objects and we can maintain the objects in the list through it. If we have different classes of objects, we need different pointers. Each one points to one kind of lists of objects.

4.3 CORBA Query Server and Query Client

The CORBA query server provides IDL for general operations on different database management systems. The Query client is used to substitute the MSQL wrapper used by the

kernel files layer. It offers the similar C++ interface as the MsqlDB to the kernel files layer. In the following sections, we will demonstrate the steps of implementing this CORBA part.

4.3.1 The Query Server IDL

After we mapped the IDL of Query Server onto MsqlDB, sending the query result back to query client remained a problem. In the query server, we sent the query to the corresponding database and got the result as an object of the MsqlResult class (see at 4.1). We could not send this object directly back to clients since it hadn't been defined in IDL and ORB could not bind the clients' call of the object to its implementation. Because we don't have the source code of MsqlResult, and MsqlResult has relationships with other classes, it is too complicated and unnecessary to define all of them in IDL and rewrite the codes for them. Therefore, we decided to define a structure in IDL containing all the information about the result for the clients to reassemble a MsqlResult object on its own side. The following is the IDL:

```
typedef sequence<string> q_row;
struct q_field
{    string name,table;
     long type,length;
};
struct q_data
{    long width;
     q_row data;
```

```
};
typedef sequence<q_data>q_datas;
typedef sequence<q_field> q_fdata;
struct q_result
{    q_datas query_data;
     q_fdata field_data;
          long numRows, numFields;
};
interface QueryServer
{    boolean connect( in string host);
     void disconnect ( );
     boolean isConnected( );
     boolean selectDB( in string db_name);
     long query(in string query);
     long selectquery( in string query, out q_result mresult);
};
```

IDL doesn't support pointers. Instead, we map the MSQL data structures onto IDL by using IDL sequences. The server then has to get the information out the object of MsqlResult and assemble it into sequences and clients need to reassemble it to create a MsqlResult object for the kernel files layer.

4.3.2 Implementation of the IDL Interface

Each IDL type will be implemented by a corresponding C++ type. For instance, interfaces map to classes. The implementation class must implement each of the functions that

correspond to IDL operations and attributes. Instances of such a class are Orbix objects – they are accessible from any code in the distributed system.

Clients are not concerned with the implementation class. They are only aware of IDL interfaces. When a client makes a function call, the call will be sent to an instance of the corresponding implementation class through ORB.

The implementation class of query server contains two private static pointers, one points to an instance of MsqlDB, and the other points to an instance of OpenDB. The query server is responsible to create these two instances and set the pointers. These two instances will create connections to the corresponding database management system. When a query statement comes in, the server will decide which database it should be sent to and get the result back.

```
class QueryServer_i
{    private:
     static MsqlDB* sDB;
     static OpenDB* oDB;
     . . .
};
```

4.3.3 Providing a Server

To keep clients and servers in different process address space, we need to provide a server program in which a query server will run.

This server program creates a query server object and indicates to Orbix that the server's initialization has completed.

```
int main()
{    Query Server_ptr tie_query =
                    new TIE_QueryServer(QueryServer_i)(new QueryServer_i);
        //create a query server object
         CORBA:: Orbix.impl_is_ready
         ("QueryServer", CORBA::Orbix.INFINITE_TIMEOUT);
        //initiate the server
        . . .
}
```

4.3.4 The Query Client

A query client passes query statements from the kernel files layer to the CORBA query server. It needs to convert the data in sequence back into the MSQL data structure.

```
class MsqlDB
{    public:
     Boolean connect(string host_or_ip_addr);
     void disconnect( );
     Boolean isConnected( );
     Boolean selectDB (string db_name);
     int query(string query);
     int query(string query, MsqlResult& )
     . . .
};
```

We make the name of the query client just the same as MsqlDB because we only want to substitute the implementation of the class with a new one that can make a connection to CORBA query server. The kernel files layer can keep unchanged since the interface and the name of the class is the same as the old ones. With the object wrapper techniques, we isolate the database part from the JFLEX and make it a relative independent part that can provide service to other users.

CHAPTER 5

THE EVALUATION

In this chapter we present three groups of tests, one for Open OODB wrapper, one for CORBA query server and one for query client to ensure the runtime correctness of the implementations. We had planned to bind CORBA query server to both MSQL database and Open OODB but failed the Open OODB part. We will explain the reason in this section.

5.1 Testing the Open OODB Wrapper

Before testing the Open OODB wrapper, we need an initial program that first initializes the shared memory and then loads test data of objects into the shared memory. The data are “persistent” in the shared memory as long as the kernel is running.

Next, we wrote a test program to create an instance of Open OODB wrapper and test the member functions in it. We could bind the wrapper to the CORBA query server only when it passes this test.

In the table 5.1, we list out the functions need to be tested:

int connect();
int isConnected();
long query(char * query, MsqlResult& mresult);
long query(char * query);

Table 5.1.1 Test Functions

The Connect () function is responsible for fetching several lists of objects. It must be run before the query functions. The IsConnected () indicates if the fetching has been done or not. The query(char * query, MsqlResult& mresult) works for select statements that require the result to be returned back to the caller. The query (char * query) is for other query statements, like update, insert, etc.

The initial program loads several objects in the shared memory: *State, GeographicArea, StateIndicator and StateCounterIndicator*. As we mentioned in the implementation section, we coded functions for the queries used in the JFLEX for these objects. We selected typical queries of different operations on the data related to the object *State* to be the test suite.

SELECT * FROM State WHERE StateID = number
SELECT * FROM State
UPDATE State SET Name =‘string’, Description=‘string’ WHERE StateID = number
INSERT INTO State VALUES (long, float, float, float, float)
DELETE FROM State WHERE StateID = number

Table 5.1.2 Query Test Suite

The test results of the above queries were correct.

5.2 Testing the CORBA Query Server and Query Client

In the new system, the CORBA Query Server provides a common interface of different DBMSs to isolate the JFLEX from the database. To change the code of JFLEX as little as possible, we added a query client whose interface is similar to that of MSQL wrapper to invoke the functions in the query server. The test was done incrementally.

5.2.1 Testing the CORBA Query Server

First, we created a test program that binds to the query server and then tests several query statements that process different data in the different database management systems. The CORBA query server is expected to specify the database management system to which the query belongs, send the query to the DBMS, get the result back and send it to the test program.

When we tried to substitute the test program of Open OODB with the CORBA query server so that the query server can deal with both MSQL database and Open OODB, we failed to do so. All the source code related to the Open OODB must be compiled by a preprocessor from Texas Instrument. The preprocessor, called ppCC, preprocesses class definitions and extends them with member functions for use by the Open OODB[12]. The ppCC is a driver program that invokes several other programs, including the C++ preprocessor (cpp), the Open OODB C++ preprocessor (ccpp) and the C++ compiler. Because of cpp and ccpp, the ppCC is more rigorous than the Solaris C++ compiler that we use. The ppCC can detect errors in a program that can be compiled by the C++ compiler [12]. In our case, the ppCC failed to preprocess the source code of the CORBA query server because ccpp complains about one of the Orbix system header files. So we only tested the CORBA query server with the MSQL database.

We selected different types of query statements to process the data stored in MSQL database through the CORBA query server. The test results were all correct.

5.2.2 Testing the CORBA Query Client

After the test above, we replaced the test program with the CORBA query client, expecting that the JFLEX server can manipulate the data in MSQL database through the CORBA query client and the query server. The execution steps were as following:

1. Run MSQL daemon;

2. Run Orbix daemon;

3. Register the JFLEX server and the CORBA query server to Orbix;

4. Execute a JFLEX client as we run the original JFLEX.

The result was the same as we expected - the new JFLEX system had the same functionality as the old one and the JFLEX server was not aware of the existence of the CORBA query client and query server layer we added.

CHAPTER 6

CONCLUSION

In this thesis research we have explored several object wrapper techniques to integrate new functions into legacy software. Our approaches have been demonstrated in the

design and implementation of integrating the real-time OODB and CORBA IDL interface into the JFLEX. In this chapter, we will summarize what we have achieved in this thesis project, describe our research contributions, and project our future work.

6.1 Thesis Summary

In Chapter 3, we listed a set of system requirements (Table 3.1) for the new JFLEX system. Our design and implementation presented in Chapter 3 and 4 have demonstrated that these requirements have been mostly fulfilled:

Portability: We used the IIOP protocol for ORB interoperability. So the JFLEX client and server could be developed upon different ORBs independently and communicate freely.

Real-time Features: The related team work mentioned in Chapter 2 added real-time scheduling to help transactions of the JFLEX meet deadlines better. Furthermore, we introduced a real-time OODB to maintain the real-time consistency of transactions and data of database, such as data temporal consistency. We were not able to make the Open OODB work for the JFLEX because the CORBA system file could not pass the compiling of the preprocessor of Open OODB. However, we designed wrapper functionality to allow the integration of Open OODB and tested it independently.

Extendibility and Isolation: The CORBA interface of the query server we developed isolates users from different database management systems. The CORBA query server can be extended to work for different types of DBMSs, such as relational database and OODB, and users can access data without the knowledge of the DBMS.

6.2 Contributions

Our research has the following contributions:

1. It explored the technology of integrating real-time OODB and real-time CORBA into legacy software and reusing the existing source code as much as possible.

2. It developed a model to add a CORBA server to legacy software to meet more system requirements.

3. It provides a common CORBA query interface that hides all the implementation details of dealing with different types of DBMSs from users, which allows more flexibility to expand the system.

4. It provides a methodology to develop a C++ wrapper for Open OODB, which also works for other DBMSs.

5. It shows a way to convert information from relational database to OODB.

6. It shows how to convert the application from one ORB to another and make it IIOP compliant.

6.3 Future Work

As described in our implementation and tests sections, the Open OODB wrapper can process simple format of query statements from the JFLEX. Obviously, this is not enough. We need a wrapper that can process queries as general as possible (the format of query is in SQL set). One possible improvement is to write parser for the wrapper using LEX and YACC that also can check the syntax of queries.

During our implementation the CORBA query server didn't connect to the Open OODB wrapper as designed due to the problem come from the preprocessor of the Open OODB. One possible way to solve it is to study the parser in the cpp written by YACC so that we can relax the syntax.

Another interesting future project is adding real-time scheduling to the CORBA query server and client as we already did with the JFLEX server and client [see the related team work]. This will allow the complete scheduling for transactions from JFLEX clients.

LIST OF REFERENCES

[1] Donald E. Rimel Jr. and Ronald C. Aronica, "Leveraging Legacy Assets", in Legacy Integration, Dec. 1996.

[2] The Common Object Request Broker: Architecture and Specification, OMG Document Number 93.xx.yy, Revision 1.2, Draft 29 December 1993.

[3] Orbix 2 Reference Guide, IONA Technologies PLC, March 1997.

[4] S. Zdonik and D. Maier, Reading in Object Oriented Database Systems. San Mateom CA: Morgan Kauffman, 1990.

[5] H. Garcia-Molina, "Using semantic knowledge for transaction processing in a distributed database system," ACM Transactions on Database Systems, vol.8,pp. 186-213, June 1983.

[6] K. Ramamritham and C. Pu, "A formal characterization of epsilon serializability," Transactions on Knowledge and Data Engineering.

[7] L. Dipippo and V. F. Wolfe, "Object-based semantic real-time concurrency control", in Proc. of IEEE Real-Time System Symposium, December 1993.

[8] Open OODB 1.0 Executive Summary, Open OODB Project Texas Instruments Incorporated, 1989-1995

[9] J. Prichard, L. C. DiPippo, J. Peckham and V. F. Wolfe, "RTSORAC: A real-time object-oriented database model," in The 5th international Conference on Database and Expert Systems Applications, Sept.1994.

[10] Victor Fay Wolfe, Lisa Cingiser Dipippo, Roman Ginis, Michael Squadrito, Steven Wholever, and Igor Zykh, “OrbixAwards Submission: Expressing and Enforcing Timing Constraints in Orbix”.

[11] Yong Yuan, "SMOS: A Memory-resident Object Store", master thesis, University of Rhode Island, Kingston, RI, 1997.

[12] Open OODB 1.0 C++ API User Manual, OpenOODB Project Texas Instruments Incorporated, 1989-1995

BIBLIOGRAPHY

Coulouris, G., Dollimore, J. *Distributed Systems. Concepts and Design.* Second Edition. Addison-Wesley Publishing Company, Reading, MA, 1994.

Dipippo, L.C., R. Ginis, M.Squadrito, S. Wohlever, V. F. Wolfe, I. Zykh. *Expressing and Enforcing Timing Constraints in a Real-time CORBA System*, University of Rhode Island Technical Report TR 97-252, Kingston, RI, February 1997.

H. M. Deitel, P. J. Deitel, *How to Program C++*. Prentice-Hall, Inc, New Jercy, 1998.

IONA Technologies Ltd. *Orbix 2 Programming Guide.* IONA Technologies Ltd., Dublin, Ireland, 1995.

Object Management Group. *CORBA services: Common Object Services Specification.* Object Management Group, Inc., Framingham, MA, 1996.

Sun Microsystems, Inc. *Solaris 2.5 Introduction*, Sun Microsystems, Inc., Mountain View, CA, 1995.