2001

A tool for building distributed transactions system with XML.

Jenane Hassib. Abouzeki
University of Windsor

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A Tool for building Distributed Transactions System with XML

by

Jenane Hassib Abouzeki

A Thesis
Submitted to the Faculty of Graduate Studies and Research
Through the School of Computer Science in Partial
fulfillment of the requirements for the
Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
2000

ABSTRACT

The evolution of distributed transaction system has taken the dramatic but positive path towards secure atomic transactions. CORBA services plays an effective role in building distributed systems, especial service is the transactional service. CORBA Transactional service was implemented by lead software companies, companies who are striving to provide solutions to programmers, by providing developers with helping tools. Tools like Jbuilder and C++Builder ease the programmer's job with implementation by reading the IDL and generating the foundation or so called the skeleton code for making it ready for further implementation by the programmer. Providing such a foundation allows the programmer to only focus on the logic. In this thesis we investigate an innovative way to use XML, as specifications language, for configuring a distributed transactions system based on existing/reusable components. We implement a tool as an interpreter for XML specifications of the transactions system which will generate smart program code;i.e. client/server programs. The generated code includes implementation code which otherwise would be the programmer's responsibility to implement manually. We then test our tool in two application areas; the Bank Transfer and the Point of Sale.

To my parents, with all the respect and admiration.

To my sisters Iman, Aida, and Intisar,
and to my brothers Mouine, Ayman, Salim and Firas,
with love.

ACKNOWLEDGEMENTS

Many thanks to my supervisor Dr. Indra Tjandra for his guidance and support. My deepest gratitude to my committee members Dr. Richard Frost and Dr. Sudhir Paul. My heart felt thanks to my parents and to my brother Sal for their endless support and motivation.

TABLE OF CONTENTS

ABSTRACT iv

DEDICATION v

ACKNOWLEDGEMENTS vi

LIST OF FIGURES viii

CHAPTER 1 INTRODUCTION 1

1.1 THE THESIS STATEMENT AND TOPICS TO BE INVESTIGATED 3

1.2 THESIS OVERVIEW 4

CHAPTER 2 REVIEW OF LITERATURE 5

2.1 DISTRIBUTED TRANSACTION SYSTEM 5

2.2 WHAT IS A TRANSACTION? 6

2.3 TYPES OF TRANSACTIONS 6

2.4 TWO-PHASE COMMIT PROTOCOL 7

2.5 INTEROPERABLE TRANSACTIONS 8

2.6 OMG CORBA 10

2.7 CORBA SERVICES 12

2.8 WHY OTS? 13

2.9 OBJECT TRANSACTION SERVICE (OTS) 13

2.10 CORBA OTS SPECIFICATIONS SCENARIO 14

2.11 COMPONENTS OF THE OTS 15

2.12 OTS FUNCTIONALITY 16

2.13 INTEROPERABILITY IN OTS SYSTEMS 16

2.14 COSTRANSACTIONS MODULE 16

CHAPTER 3 XML TECHNOLOGY 17

3.1 WHAT IS XML? 17

3.2 USES OF XML 17

3.3 XML & MODELING TOOLS 18

3.4 XML & CORBA 19

3.5 XML VS JAVA CUP 21

CHAPTER 4 OUR APPROACH 24

4.1 OVERVIEW 24

4.2 CHOICE OF CORBA TRANSACTION SERVICE IMPLEMENTATION 25

4.3 SPECIFICATIONS LANGUAGE 26

4.4 XML AS SPECIFICATIONS LANGUAGE MEDIA 28

4.5 XML DTD 28

4.6 THE INTERPRETER 30

4.7 XML PARSER 30

4.8 THE SPECS TRANSLATOR / EVENTS HANDLER 31

4.9 SEVERWRITER CLASS AND CLIENTWRITER CLASS 37

4.9.1 SEVERWRITER CLASS 38

4.9.2 CLIENTWRITER CLASS 39

CHAPTER 5 PROTOTYPE 40

5.1 CASE STUDY: BANK TRANSFER EXAMPLE 40

5.2 CASE STUDY: POINT OF SALE EXAMPLE 44

CHAPTER 6 TOOL EVALUATION 46

6.1 ADVANTAGES OF OUR TOOL 46

6.2 LIMITATIONS OF OUR TOOL 47

6.3 FUTURE WORK 48

6.4 CONCLUSION 49

APPENDIX A LISTING OF PROGRAM CODE

TRANSLATOR 51

XMLPARSER 52

SERVERWRITER 58

CLIENTWRITER 60

APPENDIX B LISTING OF PROGRAM GENERATED CODE FOR THE BANK APPLICATION

CLIENT.JAVA 64

SERVER.JAVA 66

APPENDIX C LISTING OF PROGRAM GENERATED CODE FOR POINT OF SALE APPLICATION

CLIENT.JAVA 67

SERVER.JAVA 69

APPENDIX D LISTING OF COSTRANSACTIONS.IDL INTERFACE 70

BIBLIOGRAPHY 74

VITA AUCTORIS 85

List of figures

Figure 1 steps for building distributed transactions system 2

Figure 2 Flat Transaction 7

Figure 3 Nested Trasnaction 7

Figure 4 Request passing from the client to the Object through ORB 11

Figure 5 ITS in action 14

Figure 6 XML-ITS tool, thebig picture 25

Figure 7 Rational Rose class diagram of the system/Logical view 38

Chapter 1 Introduction

Owing to the rapid increase in the demand for reliable distributed transaction systems, OMG CORBA Transaction Service specifications came to light for efficient and reliable construction of transaction systems that preserve the ACID[1] properties of its transactions.

Different implementations of OMG CORBA specifications have been presented to the public. The most popular implementations are IONA ORBIX package, Inprise VBroker, and HITACHI TPBroker. As they compete on how to provide complete OMG compliant CORBA services, these companies are trying to provide the best tools that automate some of the work thereby lessening the work for develpers.

Using the Interface definition language (IDL) and CORBA Transaction Service developers find it more convenient to build and monitor transactions of a complicated system. Borland provides C++Builder and Jbuilder for Inprise Visigenic as tools to ease the job of the programmer by automatically generating skeletonized code according to the specified IDL. Such tools are helpful if the developer is intending to implement the entire system by him or herself. The development of such a system would include the following steps:

- Generate the IDL specifications written in the Interface Definition Language specific to the CORBA implementation.
- Compile the IDL to generate Stubs and Skeletons for Objects and classes we're dealing with.
- Implement classes for each interface as necessary.

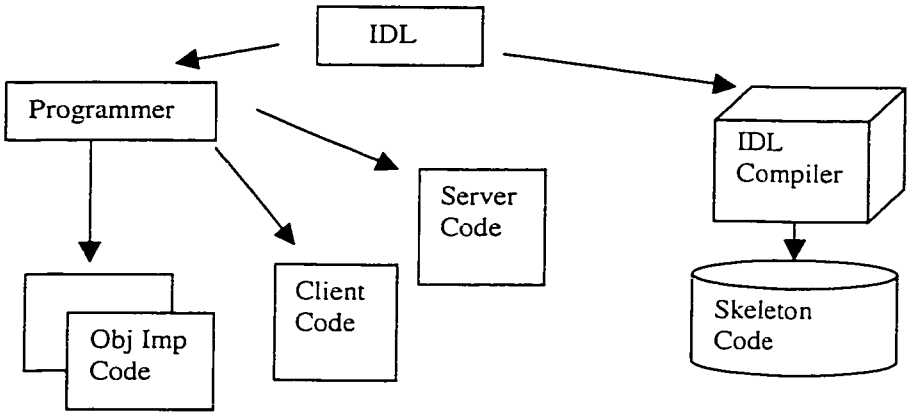
---

[1] ACID properites defined in section 2.2 of Chapter 2

- Implement server classes to initiate, and register Objects and listen for request.
- Implement a client class that normally would locate Objects on the ORB and invoke requests on those Objects.
- Compile all implemented classes for a full system ready to be run.

With the increase use and advantages of the Internet Orfali [Orfali98] anticipated that system components pre-implemented and made CORBA compliant would be made available to public where one could download the component and integrate it with one's existing system. Knowing that we are presenting this thesis to provide easy integration of CORBA compliant components with transactions systems.

Figure 1 steps for building distributed transactions system



Tools like Jbuilder and C++Builder ease the programmer's job with implementation by reading the IDL and generating the foundation or so-called skeleton code for the interfaces' implementations. Providing such a foundation allows the programmer to only focus on the logic. Figure –3 illustrates sample generated code by JBuilder. Although the tools are very convenient, it still requires the programmer to do a lot of coding him/herself, which could be eliminated with our tool. The aim of this thesis is to improve

on the above tools -specifically JBuilder, as we will be using Java and Visigenic CORBA implementations.

1.1 The Thesis Statement and topics to be investigated

The major motivation of this study concerns an innovative way using XML to configure a distributed transactions system based on available/reusable components. A new system specifications-language is presented in this thesis. Our approach employs XML technology as a means of representing data in a well-formed structure, and Java language to interpret that structure and automatically generate components necessary for building a transaction system. The objectives of our study are the following: to develop partial high-level specification language from core elements of a transactions system. Also to show whether XML usefulness can be extended as a specifications-language in terms of its flexibility, availability and ease of use.

In order to establish that our approach is beneficial we have attempted to study the following questions:

- What are the basic elements that are the essential building blocks in every transactions system?
- How XML is useful in our solution?

To answer this question we have implemented a prototype of a tool that reads XML document and uses a Java Servlet to perform action based on the interpretation of XML data structure provided.

1.2 Thesis Overview

In chapter two the basic concepts of distributed transaction systems are discussed. We also review CORBA technology and specifically CORBA Transaction Service and its functionality. Chapter three discusses the XML technology and its advantages and current uses in the World Wide Web. Chapter four discusses our XML -ITS tool as well as its design and implementation details. Chapter five presents two case studies of transactional systems, and chapter six evaluates the tool and gives the advantages, limitations, and potential future improvement of the tool. Chapter seven holds the conclusion.

Chapter 2 Review of literature

2.1 Distributed Transaction System

Before the evolution of the Internet, transactions were known as operations for commercial applications such as in banking systems. A client accesses an ATM machine to withdraw and deposit some amount of money. The bank transaction of withdraw consists of operations such as subtracting the withdrawn amount from the account balance and displaying the new balance to the client.

Recently the term 'transaction' has been extended to suit the widely used and continuously developing distributed systems. Clients are able to access their bank accounts from anywhere in the world in a matter of seconds. More than one server is being involved. The client can access accounts at more than one bank branch at the same time. All this increased complexity introduced new requirements of transactions; transactions that should be reliable, preserve data consistency and should account for failure. Moreover, transactions should either fully commit the changes or effects of operations on saved data or should rollback and undo any changes. Such transactions are called ***Atomic***. Atomic transactions provide a mean of ensuring the consistency of data in the presence of concurrency and failure. The ACID properties [section 2.2] add the atomicity property to distributed systems. In addition, transactions can be nested which extend the transaction paradigm by providing for independent failure of subtransactions and supporting the modularity of applications.

Transactions are considered vital for the reliability of distributed applications. In conjunction with the advantages of object-orientation mechanisms, transactions are

thought of as a significant aspect of distributed applications' productivity and quality of performance.

One of the systems that achieved success in introducing a reliable distributed application to the world is described in Arjuna [Shrivistava95]. The concern of recent studies is concentrated on object-oriented transaction systems that support nested transactions to deal with the rapid increase in distributed applications over heterogeneous distributed environments.

2.2 What is a Transaction?

A transaction as defined by Gray [Gray81] and Mullendar [Mullendar85] a unit of work that has the following ACID properties:

- *Atomic*: either a transaction commits as a whole or not at all (aborts) to preserve consistency of data.
- *Consistent*: the result of a transaction should preserve the consistency of the affected resources.
- *Isolation*: a transaction is expected to have no effects on other transactions.
- *Durable*: effects of a transaction are to be permanent if it completes.

2.3 Types of Transactions

[Coulouris et al. 96] describes two types of distributed transactions; i.e.; a flat transaction, also called a traditional transaction, that accesses one or more servers directly to complete it's purpose. The second type is the nested transaction that accesses one or more servers that need to access other servers via subtransaction(s) to complete its job. A subtransaction is defined in [NGU93] as:

"... a logical unit of work of the interoperable transaction. Each subtransction consists of one or more logical unit of operations. An operation is a logical unit of work (atomic) at a particular autonomous agent. "

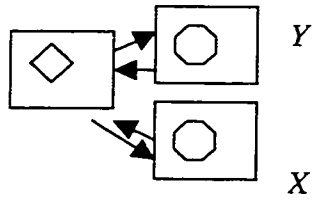


Figure 2 Flat Transaction

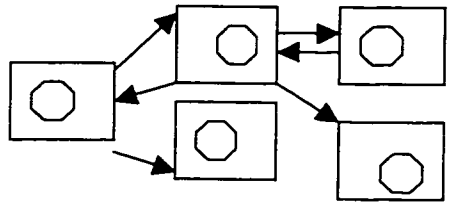


Figure 3 Nested Trasnaction

The coordination between transactions can be modeled as a communication process that has been defined in a two-phase commit protocol [Gray92].

2.4 Two-phase commit protocol

The two-phase commit protocol has been the most commonly used in applications. [Coulouris et al. 96] describes how it works and this description is summarized as follows:

The first server contacted by a client's request is called the coordinator. The coordinator is responsible for committing or rolling back the transaction. It also has the privileges to add and delete servers for the process as needed, and call them workers (referred to as slaves in other papers like [Zhou92]).

The coordinator server keeps track of the involved workers using a Transaction Identifier (TID) besides a server identifier. In a typical transaction-processing scheme the coordinator adds the servers that are interested in processing the transaction. Then it informs those workers that the transaction has begun while asking each of them whether it can commit or not. Each worker then sends back its vote to the coordinator with "yes"

vote if it can commit or a "No" vote if it is unable to commit it's part. This is called the voting phase of the two-phase commit protocol and it ends when the coordinator collects the votes from all the workers involved.

In the second phase, called the commitment phase, the coordinator decides on an action to be taken. That is, if the all workers send a vote in favor to commit, the coordinator sends them an order to commit the transaction. However, if at least one of the workers has voted against the commitment of the transaction the coordinator has to send an order to all workers to abort.

The two-phase commit protocol is effective and fault tolerant under the worst cases where it might face a successive number of failures and still guarantee transaction completion.

2.5 Interoperable Transactions

A transaction could span more than one server accessing multi database systems [Ngu93]. Such a transaction should be an interoperable transaction also called a distributed transaction.

A distributed transaction on heterogeneous distributed systems needs to preserve its ACID properties and specifically its atomicity.

Portability and interoperability among heterogeneous distributed systems are of recent concern in industry, hence application-independent semantics are being subject for standardization to enhance application development and productivity [Wong97].

Tarr and Sutton addressed the problem of interoperability in transaction models, and the role of the programming languages in providing the necessary features that facilitates

transaction processing interoperability. They introduce two approaches for interoperability; the centralized and the decentralized approaches.

The centralized approach tends to rely on either standardized representations or shared central interfaces or services, i.e.; the use of a central transaction manager facilitating interoperability. This approach has many drawbacks. An observation made by Tarr and Sutton is:

"none of the centralized approaches are themselves sufficient to support atomicity of heterogeneous transactions..... Standardizing on a single transaction model for individual languages and/or heterogeneous transactions limits the extent to which heterogeneity can be accommodated, both among the individual and the heterogeneous transaction model. Accommodating existing languages and applications also becomes difficult."

For the above disadvantages, the authors redirected their research to the decentralized approach. The decentralized approach eliminates the limitations of the centralized approach since it requires transaction models to provide flexible control over the ACID properties that they support. It therefore provides the flexibility required to define multiple heterogeneous transactions.

Three main communication styles exist to facilitate distributed transactions; dialogue, RPC calls and messages. The most popular and the most commonly used is the RPC calls on which most of distributed applications and transaction services are based.

2.6 OMG CORBA

Object Management Group which is a consortium of about 600 computing-companies has suggested the Common Object request Broker (CORBA) architecture and specifications in an attempt to solve the rapidly growing networks and the Internet. CORBA's main purpose is to provide interoperability among different software in a distributed environment. Objects from different worlds speaking different languages would be able to communicate in a common language or media, which is the Object Request Broker ORB. Using Interface Definition Language (IDL), the object can advertise its services in a contract of an interface to the entire system it is participating in. Each object would present an interface, which is isolated from the object's implementation. Basic Object Adapter (BOA) resides on top of the ORB's communication services instantiating server objects/classes, passing request to them and assigning Object Ids – called object references. The BOA also registers the classes and their run-time instances with an Interface Repository IR in the ORB exposing them to all components of the system, (which is an ideal solution for distributed systems). In other words, Inter-ORB communication is the key feature that gives CORBA its unparalleled flexibility. A client and object implementation may:

- Reside on different vendor's ORBs;
- on different platforms;
- on different operating system;
- and be written in different programming languages by programmers who never met.

And yet they still interoperate perfectly as long as client and object use the same IDL syntax and underlying semantics.

All the programmers need is a copy of the IDL file, the description of each operation and an object reference. Then integrating the third party software components with locally implement objects, programmer's job becomes easier if they provide IDL interfaces.

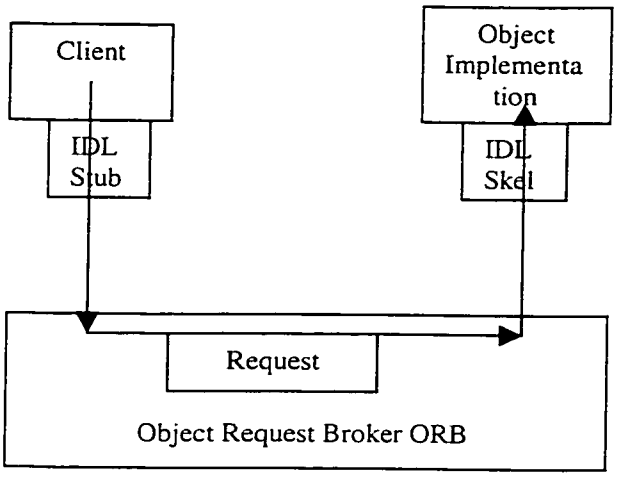


Figure 4 Request passing from the client to the Object through ORB

OMG's CORBA services (Trading, Naming, Lifecycle, Persistence, Event, Timing, and Transaction Service) provide system-level services needed by almost any object-based system discussed in section (2.7).

The major CORBA vendors are Inprise, Hitachi, Iona, and BEA. Each of these vendors were given the freedom to provide their independent implementation of CORBA services as long as it is compliant with OMG CORBA services specifications. The following section describes CORBA services, while section (2.8) discusses the Object Transaction Service OTS of CORBA services on which this thesis is focused.

2.7 CORBA Services

CORBA is a communication infrastructure between distributed objects, however, it is available from over 20 vendors, and it also supports Microsoft operating systems. CORBA provides a set of services such as:

- Concurrency Control that protects the integrity of data.
- Event Service that supports the notification of interested parties when program-defined events occur.
- Naming service that associates objects with references.
- Transaction service that protects the ACID[2] property of transactions; atomicity, consistency, isolation and durability.
- Persistence
- Trading
- Timing

OrbixOTS and OrbixOTM, HitachiTPBroker, BEA M3 are Vendor products that support CORBA services and specifications.

OMG CORBA specification [CORBAOTS 97] describes a communication infrastructure for distributed applications in an object-oriented manner. The client/server structure of this architecture is nearly transparent to the applications. The call for an object is passed through an environment that facilitates the communication between different remote systems. Currently companies are looking for a reliable intelligent pipe that provides communication between their distributed different business objects,

[2] Refer to section 1.1 of this paper for more on ACID properties of a transaction

especially when remote transactions are involved and are very critical to the success of the company. The most promising in the current days is the OMG CORBA OTS.

2.8 Why OTS?

CORBA separates the Transaction Service from other services. The main goal for separating the Transaction service from other services as in [OMGCORBA 97]:

"

- *To allow transactions to include multiple separately defined ACID objects.*
- *To allow objects from the object world communicating with objects from a non-object world.* "

The following is a brief on the transaction service called Object Transaction Service OTS.

2.9 Object Transaction Service (OTS)

The purpose of the CORBA OTS is to support the synchronization between distributed applications such as Client/Server applications. It defines interfaces that allow distributed objects to communicate and preserve atomicity. Allowing multiple object to be involved in multiple requests. OTS functionality also involves operations to control the context and the duration of a transaction, combine internal changes of object states within a transaction as well as operations for the coordination of the two-phase protocol at the end of a transaction.

2.10 CORBA OTS Specifications scenario

A client registers with the OTS if it desires to begin a transaction. The OTS returns a Transaction Context. This Transaction Context is associated with each client's request and shared by other client requests associated with the same transaction, it is also propagated to all transaction objects participating in the transaction.

The transaction can be completed in either one of the following ways:

- The transaction originator –(here this is the same as the client), issues a commit command and the OTS takes care of committing the entire transaction if all participants agree; otherwise, the transaction is rolled back,
- Any component in the application can commit the transaction, if it has the authority to do so.
- The transaction times out.

OTS supports both flat and nested transactions. Figure 5 shows the role of OTS in a transaction between a client, a server and a Database. The following section describes the ITS components in detail.

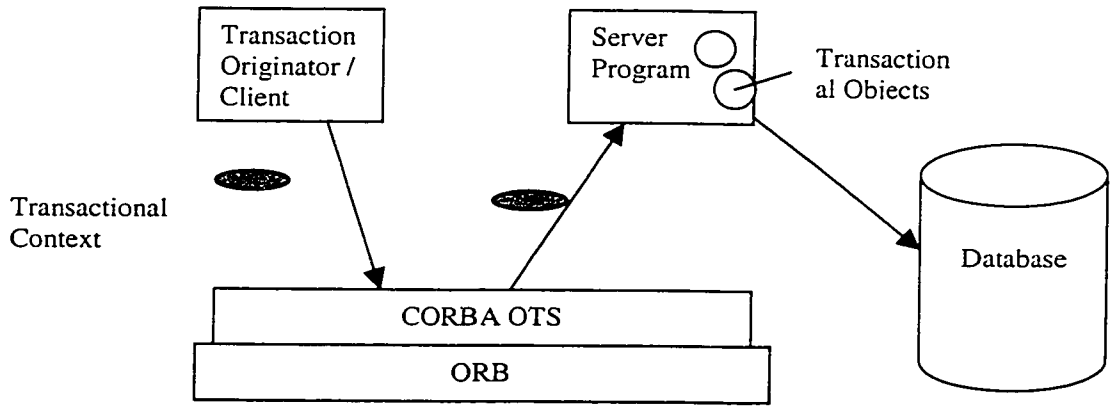


Figure 5 ITS in action

2.11 Components of the OTS

[OMGCORBA 97] classifies OTS components as objects, protocols and servers:

Transaction Client: a program that can invoke operations of many transactional objects in a single transaction.

Transactional Object: an object whose behavior, invoked within the scope of its state, is affected by the invoked transaction, it is also called persistent data. A non-transactional object; however, is one whose state is not affected by any invoked transaction.

Since the object has the choice to support transactional behavior, to implement such behavior the object must participate in certain protocols defined by OTS. These protocols are necessary to ensure that all participants in a transaction speak the same language and thus, agree on the same outcome; i.e. to commit or abort and to recover from failures. An object whose data is affected by either decision is called a *Recoverable Object*. Object participating in a transaction must register as resources in the transaction service.

Transaction Servers: a collection of one or more objects whose behavior is affected by a transactions but have no recoverable state of their own. Such objects implement the transaction changes using other recoverable objects. The transaction servers do not participate in the commit request however they do in the abort request.

Recoverable Servers: a collection of objects of which at least one is a recoverable object, they participate in the protocols by registering resource objects in the ORB.

2.12 OTS functionality

[OMGCORBA 97] outlines OTS functionality as providing operations to:

"

- *Control the scope and duration of a transaction*
- *Allow multiple objects to be involved in a single, atomic transaction*
- *Allow objects to associate changes in their internal state with a transaction*
- *Coordinate the completion of transactions.* "

2.13 Interoperability in OTS Systems

The interoperability of the OTS is influenced by the interoperability of the ORB systems. ORB domains cooperate via protocols such as Interoperable Internet ORB protocol IIOP, or Environment Specific Inter-ORB Protocol ESIOP. A transaction manager in one of these domains can control the resource objects beyond its ORB domain [Kunkelmann et al. 97] [Grasso97] [Siegel 96].

2.14 CosTransactions Module

CosTransactions Module defines OMG specifications for CORBA Transaction Service functionality. It was implemented by CORBA third party such as Inprise Visigenic who added extra methods to the Interface to make even more flexible and usable to the programmer. Appendix D is a list of CosTransactions Module and its interfaces.

Chapter 3 XML Technology

3.1 What is XML?

Extensible Markup Language XML was developed by W^3C the (World Wide Web Consortium) is an ideal data format for storing structured and semi-structured text intended for dissemination and ultimate publication on a variety of media. It was developed to overcome the limitations of HTML and its strict tags [Bradely 2000].
With XML developer can create his/her own tags, structure documents and data, using user named tags, in other words, self-describing tags. Such tags can be located, extracted, manipulated as desired.

An XML document has a logical structure in which information is divided into named units and sub-units called Elements. The document syntax can be validated by an XML parser. Also, the document structure can be validated, against a Document Type Definition DTD. The DTD defines elements that are allowed in the document. It defines elements, their type, sub-elements, and how the elements should be logically structured in the document. A well-formed XML document must conform to the specifications set in the DTD without syntax or logical errors [Floyd 98].

3.2 Uses of XML

XML has the potential to be a suitable format for exchanging data between two programs, such as Electronic Data Interchange EDI and general meta-data applications. Netscape Meta-Content Framework MCF is a proposal for an XML-based standard to describe information about information (meta-data). Microsoft XML-data is a proposed XML schema for defining and documenting object classes. Resource Description

Framework by W3C for better search engines capabilities and cataloguing the content of a web site.

XML can be used to exchange data with relational database systems. Data can be collected from different database tables and presented in any desired structure. XML tags may be a representation of columns, records and fields of tables in the database.

XML is also used to identifying complex data structures that may never be viewed or printed. Such task for XML is utilized by SMIL the Synchronized Multimedia Integration Language. SMIL employs XML to identify and mange presentation of files containing text, images, sound and video, to create multi-media presentation [Brad2000].

XML can is used to mark up semi-structured documents, such as refrences works, training guides, technical manuals, catalogues, journals, and reports. Documents can include hypertext links with XLink[3] and XPointer[4], also formatting output with XSL and XSLT or CSS. Two popular programming APIs: SAX and DOM.

3.3 XML & Modeling tools

Modeling tools like Rational Rose has added to its suite of products the XML technology. Rational Rose provides visualization, modeling and tools for XML documents that use document type definition (DTD). To support mapping XML to UML, Rose extends UML with stereotypes for XML elements, attribute lists, entities and notations. An example follows:

Rose <<DTDElement>> maps a <!ELEMENT> from the DTD.

[3] An adjunct standard to XML that defines specifications to hypertext linking.
[4] A standard that complements XLink to allow links to objects with significant contextual location.

Forward-Engineering can generate XML DTD from a Rose 'logic view' design model after assigning a stereotype to the class and setting its type and multiplicity property.

The visual model of the XML DTD can help the developer view the structure of the XML document as a class diagram in Rose XML DTD. Element definition can be added, or removed. The XML DTD model created can also be checked for syntax errors by Rose syntax checker. The tool supports reverse-engineering as well. A DTD document can be reversed into a Rose class diagram via reverse-engineering.

3.4 XML & CORBA

OMG started incorporating XML into several proposed CORBA specification. XML is the core for another standard XMI or XML Metadata Interchange which is a way of interchanging meta data between modeling tools and meta data repositories based on Meta Object Facility MOF standard [OMG 2000].

The two technologies XML and CORBA were developed separately to serve distributed systems however, they can work together as described in [Simha & Russell 99] summarized as follows:

Simha and Russell introduce three approaches to make XML serve CORBA. First they mention a simple approach which is passing an XML document to a CORBA Object which involves converting the whole XML document into a string. This approach was found to be inefficient in use of space and time and of being not type safe.

The second approach is mapping XML to IDL using value types. This approach seems to be a better one as it aims at using CORBA to implement value types that represent as much types of the DOM API as possible.

The third approach aims at mapping XML to CORBA IDL. This approach takes advantage of the IDL types existing. It involves mapping XML document into CORBA types before any operation is invoked, and convert CORBA types back to XML document when received by the receiving object. With this approach the semantics of the XML document are preserved as they are translated into CORBA structures. In brief , an XML element is mapped to CORBA struct, while an attribute is mapped to a string as member of the structure. Simha and Russel [Simha & Russell 99] provide examples of which is the following: Mapping XML to IDL

```
<element color = 'green'>  // Start tag with an attribute color
</element>                 // End tag
```

The above would be mapped into CORBA code as follows:

```
struct element
{
        string color;
}
```

An object that takes an XML document as input is defined in IDL as follows:

```
Interface MyObject
 {
        void invoke(Any xmldocument)
}
```

Mapping IDL to XML DTDwould be as in the following example:

```
Struct Quantity
{
        float _value;
};
```

Mapped to a DTD as:

```
<!ELEMENT Quantity>
```

And an XML document as:

<Quantity>1000</Quantity>

This approach is type-safe and network efficient.

The latter approach inspired our attempt to integrate XML with CORBA beyond the above mentioned approaches to assist in building distributed systems and specifically distributed transaction systems. Our approach aims at using XML as a specifications language for transaction systems translated to generate CORBA code. The following Chapter explains.

3.5 XML Parser vs JAVA CUP Parser

Since we are working with JAVA and system specifications formed as grammar we came across JAVA CUP and its ability to generate JAVA parsers for a BNF grammar. JAVA CUP is Java Based Constructor of Useful Parsers. CUP is a system for generating LALR(1)[5] parsers from simple specifications. It serves the same role as the widely used program YACC[6] and in fact offers most of the features of YACC, however, CUP is written in Java. It uses specifications including embedded Java code, and produces parsers, which are implemented in Java.

Using CUP involves creating a simple specification based on the grammar for which a parser is needed, along with construction of a scanner capable of breaking characters up into meaningful tokens (such as keywords, numbers, and special symbols).

[5] Look Ahead Left Recursive, the (1) indicates that this approach is limited to a single token.
[6] A program that takes a concise description of a grammar and produces a C routine that can parse the grammar, a parser.

For our use a parser generated using JAVA CUP would be for our specific grammar (system specifications grammar). However, XML was found to have its own parsers.

Using XML gives the opportunity to reuse its pre-implemented parser(s), and manipulate its language and tags to form a grammar. XML parsers have been implemented by multiple vendors, each providing a rich library for the parser, it gives us the flexibility to implement our tool using any XML Parser desired. Some of the available well-known parsers are listed below:

- Xerces the Apache XML Project,
- XML4J the IBM's XML Parser for Java ,
- Java Project X Sun's JavaSoft's XML Parser,
- Oracle XML Parser,
- XMLBooster by XMLBooster,
- SXP the Silfide XML Parser
- MSXML by Microsoft
- DXP DataChannel XML Parser (DXP) by DataChannel and Microsoft ebmedded in Explorer5.0

Also, Writing XML grammar doesn't require special expertise with BNF grammar like Java Cup grammar would. The use of tag based language such as XML was found sufficient to express the core elements of transactions system in a partial system specifications. XML introduces wild cards like "*", "+" and "?" which adds to its power compared to a BNF grammar. Wild cards eliminate the need for the grammar writer to

think of each and every scenario and try avoid ambiguity, providing also a more concise grammar.

XML's interoperability was also a motivation to use XML for our thesis. As mentioned in section (3.3 - XML & Modeling tools) Rational Rose is one of the applications that support XML. An XML DTD created for our tool can be reverse-engineered with Rational Rose to view the classes/objects visually. As well as, a forward engineering of a class diagram can generate an XML DTD document for our tool. Overall XML language is easy to read and to construct, and many tools are available for that purpose; such as, full suites like Visual XML by Bluestone software, includes a parser, and a graphical editor [xml.com]. Other XML editors such as Microsoft XML Notepad, which offers an intuitive and simple user interface that graphically represents the tree structure of XML data [msdn.microsoft.com].

Chapter 4 Our approach

In our approach we take the development of a transactions system to a higher level by introducing a high-level specifications for the core elements of the system. The aim is to automatically generate standard and common code to any transactions system in addition to some of the implementation code knowing partial specifications of the system. The high level specifications would be automatically translated into CORBA-JAVA code, generating the Server and the Client, assuming the implementation is ready made by a vendor. Having in mind that the vendor had tailored his/her products to be CORBA compliant.

4.1 Overview

Providing a high-level specification-language that would allow the designer to define the under developed system's requirements. The specifications would be stated in an XML document. Each Object would be tagged as an <Element>. The XML would then be parsed and validated against an XML schema or Data Definition Document containing production rules. Production rules are laws for the parser to follow when validating specifications in XML documents.

In case the validation was successful a tree of the structure of the specifications would be generated and displayed to the designer. A Java servlet using SUN's XML parser is used to access the parsed tree nodes determining each element's name and value. The Java servlet would then locate the Listener Objects so they can be registered with the ORB in a Server Program. Likewise, the Java servlet would identify the Transactions to be invoked from the Client Program. At the end of the process we will have generated

Client/Server Java programs which would require minimum modifications before they can be compiled and run. See figure 6 below.

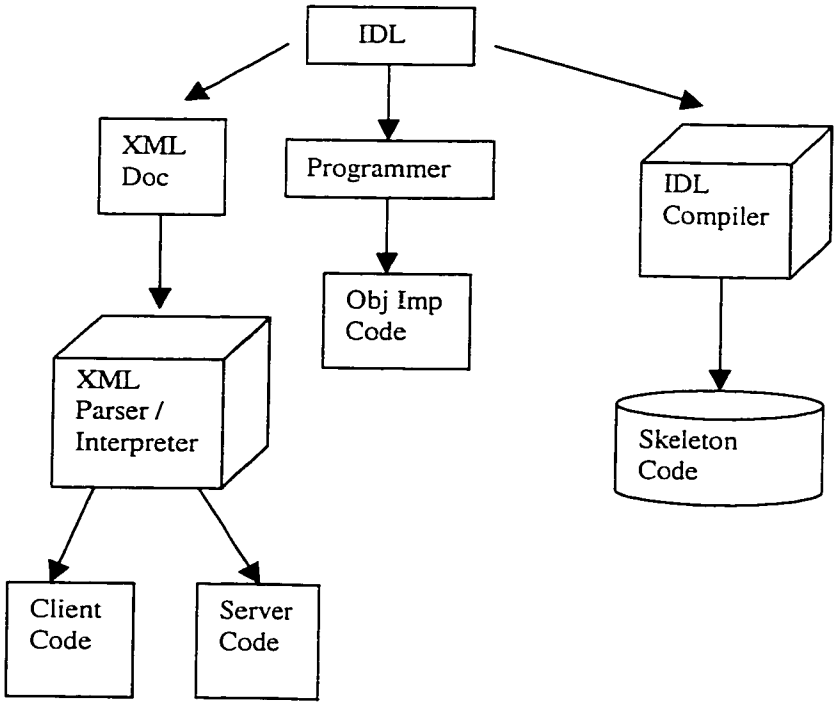


Figure 6 XML tool, the big picture

4.2 Choice of CORBA Transaction Service Implementation

This thesis uses Visigenic ITS first, for its availability for the experiment and second, for it is a reputable product for its ease of use and its reliability.

ITS is an implementation of the OMG CORBA Transactions Service. ITS version 3.0 unfortunately does not include the implementation for handling nested transactions, which limits our research to flat transactions.

4.3 Specifications Language

We chose to define partial specifications of a transactional system capturing only its core elements and structure. Partial specifications were sufficient to generate the standard and common code for a transactions system. On the other hand, the complexity of defining a full specification language contradicts the goal and purpose of this thesis. Defining full specifications for a transactions system is cumbersome to designers as well as to developers. It is more adequate instead to build the system manually and avoid such complexity.

Defining the specifications of a transactions system is the first step to building such a system. With the specifications language we define objects, such as listener objects, the relationships between objects and transactions. According to Backus-Normal Form BNF grammar, we define a set of terminals, a set of nonterminals and a series of production rules. A nonterminal is defined in a production rule, while a terminal is a symbol in the language being defined. In a production rule, a nonterminal on the left-hand side, known as the left part, is defined in terms of sequence of nonterminals, and terminals on the right-hand side known as the right part.

In the following set of production rules in BNF-form, Specs is a non-Terminal while Method is a Terminal.

Terminal:= ObjectName, Method, ReturnType

Specs: PackageName, Propagation, CurrentRef, ListenerObject, ClientObject, Commit |
PackageName, ListenerObject |
PackageName, ListnerObject, Tx |
ListenerObject, Commit |
ListenerObject, Tx, Commit |

ListenerObject

Propagation: Implicit | Explicit | ExplicitToImplicit | ImplicitToExplicit

CurrentRef: VITS | OMG

ListenerObject: ObjectName, Tx, BindwithObject |

ObjectName, Tx |

ObjectName, BindwithObject

BindwithObject: ObjectName, Tx |

ObjectName

ClientObject: ObjectName, Tx

Tx: ReturnType, Method |

Method | String

ReturnType: float | int | String | Object | void | any | boolean

Method: String

ObjectName: string

|: 'OR' for optional forms of specifications.

Method: indicates the Object's method to be invoked by the client on the Object.

The same specification could be presented in the following form:

Specs: PackageName$^{0-1}$, Propagation^{1}, CurrentRef$^{0-1}$, ListenerObject $^{1-*}$, Object0-*, Commit$^{0-1}$

ListenerObject: ObjectName^{1-*}, (Tx^{0-*} | BindwithObject^{0-*})

BindwithObject: ObjectName^{1-*}, Tx $^{0-*}$

Object: ObjectName^{1-*}, Tx^{0-*}

ObjectName: Tx

Tx: ReturnType, Method^{1-*} | String

ReturnType: float | int | string | ObjectName | void

Method: string

0-* : indicates that it is allowed to have zero or more instances of an element.

1-* : indicates that it is allowed to have at least one or more instances of an element.

| : indicates 'OR' for acceptable alternative forms of specifications.

Method: indicates the Object's method to be invoked by the client on the Object.

Other specifications can be included, such as:

System property such as; ORBServices: OMG | Visigenic

Propagation: Using Current interface | Coordinatore / Terminator.

4.4 XML as specifications language media

We use XML as specifications language media for its ability to parse data structure against production rules specified in a DTD document.

4.5 XML DTD

XML Document Type Definition DTD sets the production rules to be followed by an XML parser. The BNF production rules, in other words, grammar presented in the earlier section can be stated in a DTD document as follows starting with Specs as the root element:

```
<!DOCTYPE Specs [
<!ELEMENT Specs  (package, Propagation, CurrentRef?, ListenerObject*, Object?, Commit*)>
<!ENTITY Implicit "Implicit">
<!ENTITY Explicit "Explicit">
<!ENTITY OMG "OMG">
<!ENTITY VITS "VITS">
<!ELEMENT Propagation (#PCDATA)>
<!ATTLIST Propagation type (Implicit | Explicit) "Implicit" >
<!ELEMENT CurrentRef (#PCDATA)>
<!ATTLIST CurrentRef type (OMG | VITS) "OMG">
<!ELEMENT ListenerObject (#PCDATA | Tx | BindWithObject)*>
```

```
<!ATTLIST ListenerObject name CDATA #IMPLIED>
<!ELEMENT BindWithObject (#PCDATA | Tx)*>
<!ATTLIST BindWithObject name CDATA #IMPLIED>
<!ELEMENT Object (#PCDATA | Tx)*>
<!ATTLIST Object name CDATA #IMPLIED>
<!ELEMENT Tx (ReturnType?, Method)>
<!ELEMENT ReturnType (#PCDATA)>
<!ATTLIST ReturnType type CDATA #IMPLIED>
<!ELEMENT Method (#PCDATA)>
<!ELEMENT package (#PCDATA)>
<!ELEMENT CommitRoutine (#PCDATA)>
<!ELEMENT Commit (#PCDATA)>
<!ELEMENT Rollback (#PCDATA)>
]>
```

? : indicates that it is allowed to have zero or more instances of an element.

* : indicates that it is allowed to have at least one or more instances of an element.

#PCDATA: indicates a string or text.

The following shows a tree structure of an XML document, for a point of sale application (POS), which adheres to the above DTD rules:

```
<Specs>
  <package>POS</package>
        <Propagation type = "Implicit"></Propagation>
        <CurrentRef type = "OMG"></CurrentRef>
  <ListenerObject name = "Store">store</ListenerObject>
  <ListenerObject name ="StoreAccess ">storeaccess
        <Tx>
        <Method>FindPrice()
        </Method>
        </Tx>
```

```
        <Tx>
                <ReturnType type = "float">STotals</ReturnType>
                <Method>GetStoreTotals()</Method>
        </Tx>
    </ListenerObject>

    <ListenerObject name = "Tax">taxObj
        <Tx>
                <ReturnType type ="float">tax</ReturnType>
                <Method>calculateTax(Amt)</Method>
        </Tx>
    </ListenerObject>
<Object>Visa
        <Tx>
                <Method>GetBalance()</Method>
        </Tx>
</Object>
<Commit>yes</Commit>
</Specs>
```

4.6 The Interpreter

The Interpreter is a Java servlet which consist of a Translator, an XMLParser, ServerWriter class, and a ClientWriter class. The following sections will explain the functionality of each.

4.7 XML Parser

A parser is required to validate the specifications and identify data elements syntactically and semantically. Syntactically, by ensuring the XML tags are accurately stated. And semantically, by parsing the XML document tags against the DTD document. Making sure the specifications are well formed.

4.8 The Specs Translator / Events Handler

The Translator is the lexical scanner that reads the XML document and initiates an instance of the XMLParser class. The XMLParser's main functionality is to parse the XML tags one by one and handle events accordingly. Each tag in the XML document represents an important element of the system's specification and should be handled carefully. An instance of a ServerWriter is initiated besides a ClientWriter Object instance.

Unlike BNF parsers like YACC the lack of look-ahead mechanism with XML was faced in the implementation of the parser of the prototype. Look-ahead parsers are used to evaluate an expressions while parsing. With our XML grammar evaluation of expressions was unecessary since the grammer is used to to overcome the limitation and to determine the element ahead of time we had to use a two-pass technique to parse the specifications. In the first pass we locate system property tags and listener objects to register them with the orb. In the second pass we locate the object's methods to be invoked and on which object to be invoked.

XML tags are translated into appropriate corba java code and printed to either client.java or server.java files. Details of such translation is as follows:

The first tag encountered in the specifications would be <Specs>. It indicates the beginning of the user specifications and the root element of the tree structure.

<Specs>

As stated in section (4.3) consists of several other elements (Package, Propagation, CurrentRef, ListenerObject, Commit) each explained later in this section.

<Package>

A CORBA application usually has its source code located in a common directory as a package. In other words, the Package is where the skeletons generated from the IDL reside, as well as where the Objects' implementations can be found. Therefore, <Package> tag triggers the generation of code:

Import PackageName.*; into both the server and the client programs.

<Propagation>

ITS manages transactions by propagating transaction context to objects participating in the transactions. Usually the transaction originator is provided a transaction context immediately after transaction initiation on an object. The ITS provides the context and associates it with the working thread. With <Propagation> tag the user is given the freedom in our specifications to choose between implicit propagation and explicit propagation of transaction context. It was challenging to decide on how to present the avaiable choice to the user. Therefore, we chose to declare those choices as !ENTITY. An XML entity can hold a constant and can be referenced in the XML document/user specifications with "&entityname;".

<Implicit> propagation; Indirectly pass the transaction context via ITS to the participating objects in the transaction using Current Object and its methods. As we will see later in tag <CurrentRef>.

<Explicit> propagation; directly pass the transaction context as a parameter in the method of the participating object using Coordinator / Terminator Objects and their methods to control the transaction and its context. Basically it would look like this;

```
Control_Var control;
Terminator_var newTranTerminator;
Coordinator_var newTranCoordinator;
NewTranTerminator = control.get_terminator();
NewTranCoordinator = control.get_coordinator();
```

And control would be passed as transaction context in the method parameter of the object that allows it as in:

```
Bank.get_account(accountname, control);
```

<ImplicitToExplicit> and <ExplicitToImplict>

The designer may choose to switch between Implicit and explicit propagation, therefore, a new tag was provided for that purpose; <ImplicitToExplicit> and <ExplicitToImplict> tags. Implicit to Explicit obtains a control object reference by suspending current and obtaining a control object with current.get_control() function. To change from explicit propagation to implicit current.resume() would do the trick.

The reason sometimes the user would like to switch between the two types of propagating the transaction context depends on the object's implementation, some objects were built to be passed transaction context as a parameter of their methods.

<CurrentRef>

In case user's choice was to use implicit propagation to gain access to an ITS-managed transaction we have to obtain an object reference to the Current Object. The following describes the steps to obtain a Current Object reference:

1. Call the ORB resolve_initial_reference("TransactionCurrent");
2. Narrow the returned object to either;

Again to give the user options to choose from, !ENTITY type was used. OMG and VITS are declared as entities in DTD and to be referenced in the user specifications as desired.

<OMG> A tag used to indicate user's choice of using CosTransations::Current specifying the intention to use the original set of methods of CosTransactions Module which is OMG compliant.

```
org.omg.CORBA.Object
obj = orb.resolve_initial_reference("TransactionCurrent");
        org.omg.CosTransactions.Current
current = org.omg.CosTransactions.CurrentHelper.narrow(obj);
```

<VITS> VISTransactions::Current specifying the intention to use the additional set of methods Visigenic provides over the original CosTransctions module.

```
org.omg.CORBA.Object
obj = orb.resolve_initial_reference("TransactionCurrent");
org.visigenic.VISTransactoins.Current
current = org.visigenic.VISTransactoins.CurrentHelper.narrow(obj);
```

Therefore, we introduced an XML tag or element <CurrentRef> to allow designer flexibility in implementation as mentioned earlier.

<ListenerObject>

If a ListenerObject tag is encountered we write it out to the server program in the correct logic and syntax needed to register this Object in the ORB. To give the user the choice of variable names and instance references an attribute list !ATTLIST was used to associate an attribute name to listener object indicating its class type, while the value set between the start tag and end tag indicates the instance name/reference holder.

Create an instance of the object and register it in the ORB as follows:

```
Obj_A  instance_LO_a = new Obj_A();
Boa.obj_is_ready(instance_LO_a);
```

<BindWithObject>

When a 'BindWithObject' is encountered as a child of 'ListenereObject' we register the 'BindWithObject' with the ORB with a reference to the 'ListenerObject' to be associated with it as follows:

```
Obj_A instance_LO_a = Obj_Ahelper.bind(orb, b_ref );
Obj_B i instance_LO _b = new Obj_BImpl(b_ref);
Boa.is_ready(instance_b);
```

Where b_ref is the object name specified by the user.

<Tx>

A Tx could be a method to be invoked on the Listener Object instance and associated with its parent ListenerObject tag, or it could be a child of the BindWithObject tag. Either case the generated code should use the correct instance of either objects.

<Tx> may consists of the following tags:

<ReturnType> determines if the method/function has a return type if the tag exists, otherwise the method doesn't return any value.

<Method> a string of the operation/method to be invoked.

<Object> If the parent of <Tx> is the tag <Specs> it indicates that the transactions to be invoked are not directly on a listening objects but probably on a delegated object or an object in the instantiated in the client program as we will see in the Bank prototype section (5.1).

<ReturnType>

An operation could return a value of a certain type, a string, float, boolean, the return value could also be a reference to an instance of an object.

ReturnType has an attribute type in which to specify the type of the return value. It also holds a value, which is a variable to declare the data type with.

Ex: <ReturnType type = "String">str </ReturnType>

<Method>

Method a tag to represent the object method to be invoked as a string. The interpreter writes the method out as a string, to the client program as specified in the <Tx> subtag <Method> as follows:

```
ReturnType var = instance_LO _a.Tx();
```

The declaration of a method was a challenge as it would open a whole new door on the implementation, it tends to shift the focus from the ITS specifications into java language interpretation and variable declaration which was decided to be left for future work.

<Object>

Some objects methods can be invoked from the client program they either need not be registered with the orb on the server, or they belong to other objects who are responsible to register them with the orb. The difficulty here lies in the question of how to specify these non-listening objects in the specification and handle it in the interpreter. We use <Object> to associate the method with its object and avoid the interpreter from confusing it with the listener object.

```
<Tx>
    <Object>acc1</Object>
    <Method>Debit(Amt)</Method>
</Tx>
```

4.9 SeverWriter Class and ClientWriter Class

The ServerWriter and the ClientWriter classes are special classes that write out code for each event or element in the XML document. An instance of each of the classes is created in the XMLParser to be able to access their functions and methods. Each code segment that should be included in the Server or Client programs is represented by a method of a class as listed in the following subsections. Figure 7 is a logical view created by Rational Rose modeling tool.

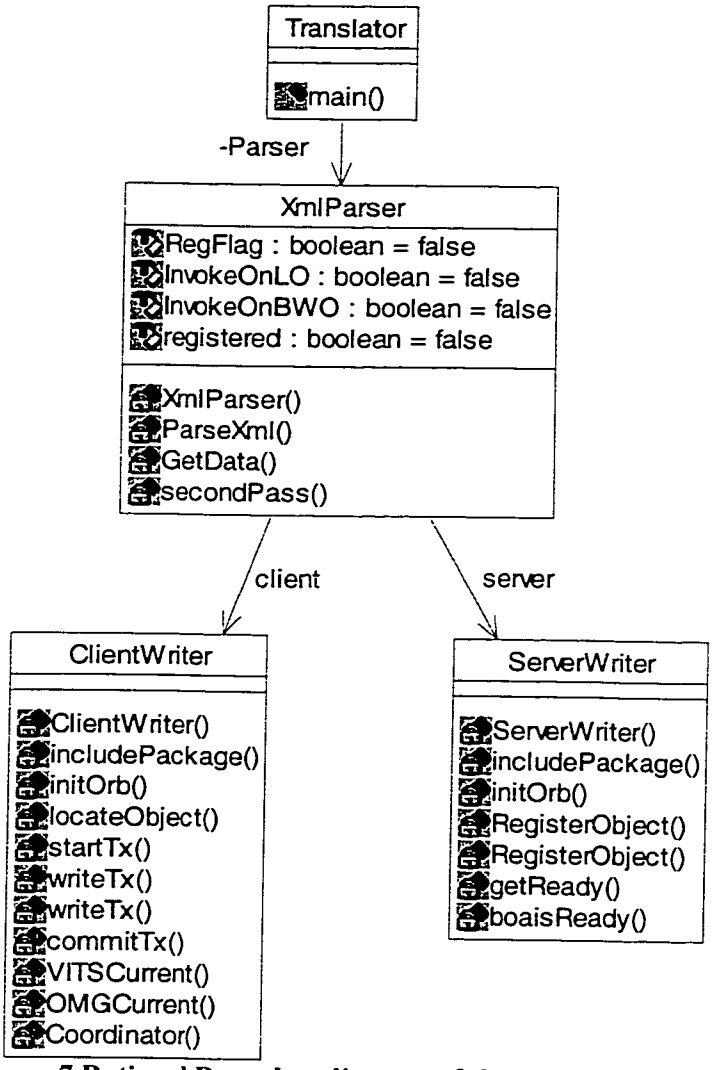


Figure 7 Rational Rose class diagram of the system/Logical view

4.9.1 SeverWriter Class

InitOrb() generates code necessary to:

- set ORBServices to include CosTransactions service, and
- initialize the ORB, and
- activates the BOA[7].

RegisterObject(ObjectName) generates code necessary to:

[7] Basic Object Adapter BOA is defined in section 2.6

- Create an object instance, and
- register an Object's instance with the ORB

4.9.2 ClientWriter Class

InitOrb generates code necessary to:

- Set ORBServices to include CosTransactions service, and
- initialize the ORB, and
- Activate the BOA.

LocateObject(ObjectName) generates code necessary to:

- Locate the Object that's been listening and waiting for requests on the ORB.

OMGCurrent()generates code necessary to:

- get Current Object reference of OMG Current interface without Visigenic additional methods.

VITSCurrent() generates code necessary to:

- get Current Object reference of Visigenic Current Interface with its new methods.

Coordinator() function generates code necessary to propagate explicitly the transaction context to the participating Object, which involves the following steps:

- get a control Object reference
- get Coordinator Object Reference
- get Terminator Object Reference

Appendix 'A' lists the program code for all classes presented earlier, i.e; Translator.java, XMLParser.java, ClientWriter.java and ServerWriter.java.

Chapter 5 Prototype

5.1 Case study: Bank Transfer example

To identify the Listener Objects that are to be registered with the Orb on the Server side and the transactions and objects to be on the Client side we thought of a specification language and a program that interprets those specifications.

In our example Storage and Bank are Listener Object and for us to invoke transactions on these objects they have to be listening on the server side.

Analyzing the application we've identified Storage as the database object that could handle more than one Bank provided that it is a distributed system we are dealing with. Therefore, the Storage Object should be first initialized and the Bank Object(s) is to bind with the storage on the same ORB, also to allow Bank Object to use Storage Object functions to access the database. All that would be handled by the Bank and Storage implementations.

As stated earlier <Specs> is the root element with sub elements defined in a tree structure. A Listener Object is defined as an element with its own subelements defined in sublevels. The name of the element is referred to as 'TagName' and is in this case <Specs>, <Listener Object>, <BindwithObject>, <Tx>, etc.

User's specifications language of a system typically would be like the following example as an XML document:

```
<Specs>
  <package>quickstart</package>
     <Propagation type = "Implicit"></Propagation>
     <CurrentRef type = "OMG"></CurrentRef>
  <ListenerObject name = "Storage">myStorage
```

```
      <BindWithObject name = "Bank">myBank
        <Tx>
              <ReturnValue type = "Account">acc1</ReturnValue>
              <Method>Get_Account()</Method>
        </Tx>
      </BindWithObject>
  </ListenerObject>
<Tx>
      <Object>acc1</Object>
      <Method>Debit(Amt)</Method>
</Tx>
<Commit>yes</Commit>
</Specs>
```

Such specifications can easily be included in the DTD document, the tool interpreter would have to have the logic to handle them as well.

A Bank can handle more than one account and each account could handle more than one banking transaction.

According to the user's specifications above Quickstart is the package name where the skeletons and stubs for the application are located.

Propagation type specified is Implicit indicated the use of Current Interface to manage the transactions.

To get a Current Object Reference the user has specified VITS tag which indicates the user's choice of Visigenic Current Interface as mentioned in section (4.8) of Chapter 4.

```
org.omg.CORBA.Object Obj = orb.resolve_initial_reference("TransactionCurrent");
org.visigenic.VISTransactoins.Current current
= org.visigenic.VISTransactoins.CurrentHelper.narrow(obj);
current.begin();
```

Listener Object to register in the ORB would be Storage, and Bank Objects.

Storage is an object that handles the connection with the database as well as the access to records in the database. StorageServer was assumed to have been given since it requires special handling of code depending on the database type.

Bank Object is implemented to get a reference to stored accounts by name upon request by the client. Bank object has to be registered with the ORB to be made available to the clients for transaction invocations.

```
<ListenerObject name = "Storage">myStorage
    <BindWithObject name = "Bank">myBank
    ...
```

The generated code would be in Server.java file as:

```
Storage myStorage = StorageHelper.bind(orb,"myBank");
Bank myBank= new BankImpl("myBank",orb);
```

As Bank Object creates an instance of Account Object to which the client should have a reference to be able to access its methods such as debit() and credit().

```
    <Tx>
            <ReturnValue type = "Account">acc1</ReturnValue>
            <Method>Get_Account()</Method>
      </Tx>
    </BindWithObject>
</ListenerObject>
```

The generated code would be in the Client.java file as follows:

```
    try{
      Account acc1 = myBank.Get_Account();
      }catch (org.omg.CORBA.SystemException e2){}
```

Since there is an Account Object that Bank instantiates in the Client program, the specifications should include the operations to be invoked on Account object Debit or Credit. The operation is enclosed in <Tx> tag with child tag <Object> indicates the instance or reference name of the responsible object on which the transaction to be invoked. In the following, acc1 is an Account instance created by the Bank's operation Get_account as mentioned above. <Object> tag to identify acc1 as the object for which Debit() function is a member function.

```
<Tx>
    <Object>acc1</Object>
    <Method>Debit(Amt)</Method>
</Tx>
<Tx>
    <Object>acc2></Object>
    <Method>Credit(Amt);</Method>
</Tx>
```

the generated code would be the following:

```
try{
   acc1.Debit(Amt);
   }catch (org.omg.CORBA.SystemException e2){}
try{
   acc2.Credit(Amt) ;
}catch (org.omg.CORBA.SystemException e2){}
```

At the end of specs the user specified the command to commit the transactions performed earlier.

<Commit>yes<Commit>

The code generated is:

```
commit = true;
```

```
if (commit) {
    System.out.println(" *Transactoin Commited* ");
            current.commit(false);
}
else
{
    System.out.println(" Rolling Back Tx  ");
    current.rollback();
}
```

Appendix 'B' a listing of the generated code in Client.java and Server.java files.

5.2 Case Study 2 – Point of Sale Application (POS)

The tool was tested on several other application of which was Point of Sale application. The following shows a tree structure of an XML document, for a point of sale application POS, which adheres to the above DTD rules:

```
<Specs>
  <package>POS</package>
      <Propagation type = "Implicit"></Propagation>
      <CurrentRef type = "OMG"></CurrentRef>
  <ListenerObject name = "Store">store</ListenerObject>
  <ListenerObject name ="StoreAccess ">storeaccess
      <Tx>
      <Method>FindPrice()
      </Method>
      </Tx>
      <Tx>
          <ReturnType type = "float">STotals</ReturnType>
          <Method>GetStoreTotals()</Method>
      </Tx>
```

```
    </ListenerObject>

   <ListenerObject name = "Tax">taxObj
         <Tx>
               <ReturnType type ="float">tax</ReturnType>
               <Method>calculateTax(Amt)</Method>
         </Tx>
   </ListenerObject>
<Object>Visa
         <Tx>
               <Method>GetBalance()</Method>
         </Tx>
</Object>
<Commit>yes</Commit>
</Specs>
```

Appendix 'C' is a listing of the system classes and generated code for this application.

CHAPTER 6 Tool Evaluation

6.1 Advantages of our tool

Both client and server programs for a Transactions system can be generated from partial system specifications. The implementation code generated is a step ahead of what a tool like Jbuilder can generate as it generates smart code in addition to the skeleton code of the Client and Server programs. . Saving a little effort and time in the building of client and server programs. The code generated also counts for fault-tolerance as try-catch wrap automatically around known fault-tolerance points predefined in normal transactions system.

Knowledge of only core elements of the system is required by the programmer/designer to build the DTD production rules, providing easy to use tool to developers.

The tool also shows to be very flexible as new XML elements can be added to or removed from a DTD document, provided it can be handled in the event handler / interpreter program.

XML is very popular and freely available technology as well as Java. The ease of access to elements and their values using XML – Java rich libraries makes it easy and flexible to modify the tool and enhance it.

The tool's interpreter can be implemented in any other language, for example; C++ is a possible candidate. On the other hand, we've tried to implement it with Java script however, we faced the limitation of not being able to write out to a file from a browser due to security issues with the Internet browser.

The concept of our approach seems broad enough that any application can be described using XML specifications.

From another point of view, using our Object-Transaction relationships the tool can be replaced with any specification language, such as; Java cup, lex and yacc, etc.

In relation to modeling tools like Rational Rose, there is potential that Rose can be used to generate an XML DTD from a class diagram where classes/objects can be mapped into XML elements forming a DTD document,(an add-in feature in Rose). The DTD document then can be used as a model DTD for an XML document, which can then be fed to our tool for processing. In other words, Rational Rose Modeling tool can play the role of an interface to our tool.

The major advantage of our approach is error free syntax since the code is automatically generated, provided the user input was syntax error free, in other words, our tool automates the standard coding for ITS programming. It is also worth mentioning that from a programmer's point of view it allows the user to work at higher level and focus on the logic of system rather than having to spend time and effort on coding repetitive and common code to all systems. Relying on a tool to read centralized data and generate related code in two separate programs is a breakthrough to a programmer and the aim of software developing enterprises that try to build such tools.

6.2 Limitations of our tool

The tool seems to have some limitations that are summed as follows:

The order of tags is very important to define the relationship between objects and their methods. However, it may not be the exact order in which the user would like to see it,

thus, the user will have to manually reorder the events or transactions by simply cut and paste segments of code.

Knowledge of XML technology is necessary although is easy to learn and practice.

The Interpreter / code generator needs to be updated if new XML elements where to be added to the tool especially if these new elements require special event handling.

The use of tags for elements of the system could be a long process and could create a huge XML document and endless XML elements and tags.

Dependability on other products; new releases of Visigenic ITS may force a new release of our tool.

6.3 Future work

The research material presented in this document doesn't stop at this stage, however, the tool's expansion and enhancement are definitely in the road ahead. Some of possible enhancement on the implemented tool in this thesis is in brief listed below;

Getting transaction context from a transaction factory can be added to the tool's capability to interpret the user's desires to use a transaction factory in the system.

The opportunity for implementing a handle for synchronization object [8] is also open, as well as implementing coordinator/terminator events, and CORBA Exception Handling.

Incorporating ITS session manager is also envisioned.

[8] Includes methods that are made available for user's implementation such as pre_commit() and post_commit()

6.4 Conclusion

This thesis has shown that specifications of a transaction system can be described with a high level language knowing core elements of a system, such as the listener objects on the server and the their methods to be invoked from the client side. The approach used introduced the Object-Transaction relationship as a parent-child relationship and used XML technology as a specifications-language to such relationships. The approach tested whether XML is sufficient to be used as a specifications language to describe partial system specifications. As a result of testing our approach with a couple of applications, the Bank application and the Point of Sale application, our XML JAVA parser showed success in generating code for the server and client programs. The generated server code registers listener objects with a CORBA ORB, while the generated client code is the standard code required in a client program to locates the objects on the orb and invoke transactions on them as required.

Unlike BNF parsers like YACC the lack of look ahead mechanism with XML was faced in the implementation of the parser of the prototype. To overcome the limitation and to determine the element ahead of time we had to use a two-pass technique to parse the specifications and find specific elements and entities.

Our tool is a prototype of an interpreter for the specifications and a code generator, which would be able to successfully generate JAVA code that is CORBA compliant.

The tool could be enhanced and made more general to handle more complex transactions system. In addition, the tool could be extended to generate code in different languages

other than JAVA. It also can be implemented in different language other than JAVA if desired since XML is programming language independent.

The client program generated would need further modifications by the user if extra logic is required, i.e.; loops, if statements, etc., and that was expected since we used only partial system specifications. Again we clarify that partial system specifications language was sufficient to generate standard and common code to any transactions system like initialization of the ORB and BOA, in addition to smart code such as registering objects on the server side and invoking transactions on the client side. On the other hand, full system specifications would have been cumbersome and the complexity generating the specifications language would defeat the purpose of this thesis.

We believe that the idea and approach presented in this thesis have potential to ease programmer's job to a certain degree and to be of good use to distributed transactions system developers with CORBA and JAVA technologies.

Appendix A

Listing of Translator, XMLParser, ServerWriter and ClientWriter Code

Translator.java : Parser Initiator

```
import java.io.*;


class Translator{

   public static void main (String[] args) throws IOException{

      BufferedReader stdin  = new BufferedReader(new
      InputStreamReader(System.in));

      String filename;
      boolean stop = false;
      String string = new String();
      // instantiate the parser
      XmlParser   xparser = new XmlParser();

      while (!stop){
         System.out.println ("Please enter xml document or 'exit' to
quit.");
         filename = stdin.readLine(); //get tasks.txt from a keyboard

         if (filename.equals("exit")){ //stop the translator program
            stop = true;
            System.out.println("The end!");
            System.exit(1);
         }
         else{
            try{
               File file = new File(filename);
               RandomAccessFile r  = new RandomAccessFile(file, "r");

               xparser.ParseXml(filename);
            }catch(IOException e2){}
         }
      }//while
      }// main
}// translator
```

XMLParser.java: Parser / Event Handler

```
import java.io.*;
import java.io.File;
import com.sun.xml.tree.*;
import org.w3c.dom.*;
import java.lang.*;
import com.sun.xml.parser.Resolver;
import com.sun.xml.tree.XmlDocument;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;


public class XmlParser{

      static final String PACKAGE = "package";
      static final String VITS = "VITS";
      static final String OMG = "OMG";
      static final String PROPAGATION = "Propagation";
      static final String CURRENTREF = "CurrentRef";
      static final String LISTENEROBJECT = "ListenerObject";
      static final String BINDWITHOBJECT = "BindWithObject";
      static final String OBJECT = "Object";
      static final String IMPLICIT = "Implicit";
      static final String EXPLICIT = "Explicit";
      static final String TX= "Tx";
      static final String METHOD = "Method";
      static final String COMMIT = "Commit";
      static final String COMMITROUTINE = "CommitRoutine";
      static final String RETURNTYPE = "ReturnType";
      static final String STR = "String";
      static final String FLT = "float";
      static final String BOOL = "boolean";
      static final String ANY = "Any";
      static final String EMPLICITTOEXPLICIT = "EmplicitToExplicit";
      static final String EXPLICITTOEMPLICIT = "ExplicitToEmplicit";
      static final String INSTANCE = "Instance";

       ClientWriter client = new ClientWriter();
      //create an object of class ServerWriter
      ServerWriter server = new ServerWriter();
      String LO, BWObj, temptype, tempEvalue, LOName, BWObjName,
nonListenObj;
      String Pvalue, Gvalue;
      boolean RegFlag = false, InvokeOnLO = false, InvokeOnBWO = false,
registered = false;
      public XmlParser(){ }; //constructor
      void ParseXml(String FileName)
      {

            InputSource input;
            XmlDocument doc, doc2;
            Element root, OrigRoot;

            //create an object of class ClientWriter
```

```
        try {
            // turn the filename into an input source
            input = Resolver.createInputSource (new File (FileName));

            // turn it into an in-memory object
            // ... the "false" flag says not to validate
            doc = XmlDocument.createXmlDocument (input, false);
            // normalize text representation
            // doc.getDocumentElement ().normalize();


            // access the elements here write server and client programs
 from here.
            // prettyprint
            //  doc.write (System.out);
              root = (ElementNode)doc.getDocumentElement();
              System.out.println(root.getTagName());
              GetData(root);
              secondPass(root);
              server.boaisReady();

        } catch (SAXParseException err) {
            System.out.println ("** Parsing error"
              + ", line " + err.getLineNumber ()
              + ", uri " + err.getSystemId ());
            System.out.println("    " + err.getMessage ());
            // print stack trace as below

        } catch (SAXException e) {
            Exception     x = e.getException ();

            ((x == null) ? e : x).printStackTrace ();

        } catch (Throwable t) {
            t.printStackTrace ();
        }

}

void GetData(Element elem)
{     Node first, last;
      NodeList cnodes, templist;
      String Evalue,tagName,proptype, curtype;
      boolean done = false;
      cnodes = elem.getChildNodes();
      int leng = cnodes.getLength();
            first = cnodes.item(0);
            last = cnodes.item(leng-1);
            tagName = elem.getTagName();
            Evalue = first.getNodeValue(); //substring value as it adds
end space
            if (Evalue != null)
                  Evalue = Evalue.trim();
            if (tagName == PACKAGE)
            {
                  server.includePackage(Evalue);
                  server.initOrb();
```

```
                client.includePackage(Evalue);
                client.initOrb();
                //client.startTx();

            }
            if( tagName == CURRENTREF)
            {     curtype =   elem.getAttribute("type");

                  if (curtype == OMG)
                  {   // use additional methods of Visigenic ITS
                  client.OMGCurrent();
                  }
                  if (curtype == VITS)
                  {
                        client.VITSCurrent();
                  }
            }
            if (tagName == PROPAGATION)
            {
                  proptype =  elem.getAttribute("type");

                  if (proptype == IMPLICIT)
                  {
                  // Use Current Interface
                  // event handled by either OMG or VITS in CURRENTREF
tag
                  }

                  if (proptype== EXPLICIT)
                  {
                  client.Coordinator();
                  }
            }
/*
            if (tagName == EMPLICITTOEXPLICIT)
            if (tagName == EXPLICITTOIMPLICIT")
            if (tagName == STR)
            if (tagName == FL)
            if (tagName == BOOL)

      */
            if (tagName== LISTENEROBJECT )
            {
                  LO = Evalue;
                  LOName = elem.getAttribute("name");
                  if (Evalue != null)
                  {
                        server.RegisterObject(LOName, LO);
                        server.getReady(LO);
                        client.locateObject(LOName,LO);
                  }

            }

            if (tagName ==    BINDWITHOBJECT)
            {
                  BWObj = Evalue;
```

```
                BWObjName = elem.getAttribute("name");
                if (Evalue != null)
                {
                    server.RegisterObject(LOName, LO, BWObj);
                    server.getReady(BWObj);
                    client.locateObject(BWObjName,BWObj);
                }
            }

            while (first!= last)
            {
                // there might be more children here.. go through the
list
                if (first.hasChildNodes())
                {
                    GetData((ElementNode)first);
                }
                first = first.getNextSibling();

            }// while


}//GetData

void secondPass(Element elem)
{   Node firstElem, lastElem,GrandP,parent, next, pnode, method;
    NodeList cnodes, templist;
    String Evalue,Pname,Gname,tagName,proptype, curtype;
    boolean done = false;
    cnodes = elem.getChildNodes();
    int leng = cnodes.getLength();
        firstElem = cnodes.item(0);
        lastElem = cnodes.item(leng-1);
        tagName = elem.getTagName();
        Evalue = firstElem.getNodeValue();
        if (Evalue != null)
            Evalue = Evalue.trim();

        parent = firstElem.getParentNode();
        Pname = parent.getNodeName();
        GrandP= parent.getParentNode();
        Gname = GrandP.getNodeName();

        if (tagName== LISTENEROBJECT )
        {
            LO = Evalue;
            LOName = elem.getAttribute("name");
        }
        if (tagName == BINDWITHOBJECT)
        {
            BWObj = Evalue;
            BWObjName = elem.getAttribute("name");
        }
        if (tagName== TX)
        {

            if (Gname == LISTENEROBJECT )
```

```
                {
                        InvokeOnLO = true;
                }
                if (Gname== BINDWITHOBJECT )
                {
                        InvokeOnBWO = true;
                }
        }


        if (tagName == RETURNTYPE)
        {       tempEvalue = Evalue;
                temptype = elem.getAttribute("type");

        }

        if (tagName == METHOD)
        {
                if (InvokeOnLO)
                {
                  if (tempEvalue !=null && temptype !=null)
                  {
                      client.writeTx(temptype, tempEvalue, LO,
Evalue);
                      InvokeOnLO = false;
                  }else
                      client.writeTx(LO, Evalue);
                }
                else if (InvokeOnBWO)
                {
                   if (tempEvalue !=null && temptype !=null)
                   {
                      client.writeTx(temptype, tempEvalue, BWObj,
Evalue);
                      InvokeOnBWO = false;
                   }else
                      client.writeTx(BWObj, Evalue);
                }
                else if (nonListenObj !=null)
                {
                        if (tempEvalue !=null && temptype !=null)
                        {
                                client.writeTx(temptype, tempEvalue,
nonListenObj, Evalue);
                        }else
                        {
                                client.writeTx(nonListenObj, Evalue);
                        }

                }
                nonListenObj = null;
                tempEvalue = null;
                temptype = null;
        }
        if (tagName == OBJECT)
        {
                nonListenObj = Evalue;
```

```
            }

            if ((tagName == COMMIT) || (tagName == COMMITROUTINE))
            {
                  client.commitTx();
            } // else (just declare the element as a variable with the
element tag as type

            /*else { ************************************
                  declareElement(in the client)at the begining of
main()
                  */
            while (firstElem!= lastElem)
            {
                  // there might be more children here.. go through the
list
                  if (firstElem.hasChildNodes())
                  {
                        secondPass((ElementNode)firstElem);
                  }
                  firstElem = firstElem.getNextSibling();

            }// while

}

}// XmlParser
```

ServerWriter.java

```
import java.io.*;

class ServerWriter{

String str;
File f;
RandomAccessFile sw;

private String bwo, lo, historyObj= "orb";

   public void ServerWriter(){
   }
      void includePackage(String tang)
      {
            try{  f = new File("Server.java");
             sw  = new RandomAccessFile(f, "rw");
             str ="//This is a Server Program automatically generated
\n"+
                  "//the Server performs the following according to
user specifications\n"+
                  "//1) import required libraries and packages\n"+
                  "//2) Initialize an ORB\n"+
                  "//3) register Objects in the ORB\n"+
                  "//4) wait for requests \n\n" +
             "import "+ tang+ ".*;" +"\n" +" import java.io.*;\n import
java.util.*;\n";
             sw.writeBytes(str);
            } catch(IOException e2){}

      }
      void initOrb()
      {
            try{
             str = "public class Server { \n" +
                   "\t public static void main (String[] args) throws
Exception" + "\n" +
                   "\t{ Properties props = System.getProperties();\n"+
                    "\t" +"String services = \"\" ;\n"+
                     "\t"+ "services = (String) props.get(\"
ORBservices \");\n"+
                      "\t"+    "if (services == null ||
services.length() ==0)\n"+
                       "\t\t"+ "{\n" +
                         "\t\t"+     "props.put (\"ORBservices \",
\"com.visigenic.services.CosTransactions \");" +
                         "\t\t"+     "} else { }\n " +
                         "\t"+ "org.omg.CORBA.ORB orb =
org.omg.CORBA.ORB.init(args, props);\n"+
                         "\t"+ "org.omg.CORBA.BOA boa =
orb.BOA_init();\n" +
                         "\t\t"+ "if (args.length != 1)\n" +
                               "{ System.out.println (\"Usage:\"); }\n"
;
```

```
                    sw.writeBytes(str);
                    } catch(IOException e2){}

        }
 void RegisterObject(String LObjClass, String LoInstance, String
 bindwithObj)
        {
              if (LoInstance != null)
                    bwo = bindwithObj;

              if (bindwithObj != null)
              {
                    historyObj = lo;
                    str = LObjClass + " "+LoInstance + " =
 "+LObjClass+"Helper.bind(orb," + "\""+bwo+"\""+");\n";
              try{
                    sw.writeBytes(str);
              } catch(IOException e2){}

              }
        }
 void RegisterObject(String LObjClass, String LoInstance)
        {     if (LoInstance != null)
                   historyObj = "orb";

              str = LObjClass +" "+ LoInstance+ "=  new " +
 LObjClass+"Impl("+"\""+LoInstance+"\""+"," + historyObj+");\n";

              try{
                    sw.writeBytes(str);
              } catch(IOException e2){}

        }
 void getReady(String Lo)
 {
              if (Lo!= null)
              {
                    str = "boa.obj_is_ready("+Lo+");\n" ;
              }
              try{
                    sw.writeBytes(str);
              } catch(IOException e2){}
 }
 void boaisReady()
 {     str = "boa.impl_is_ready(); \n} \n}";
       try{

             sw.writeBytes(str);

       } catch(IOException e2){}
 }
 }
```

ClientWriter.java

```
import java.io.*;


class ClientWriter{
String str = "";
File f;
RandomAccessFile sw;

      public void ClientWriter(){}

      void includePackage(String pack)
      {
            try{  f = new File("Client.java");
             sw  = new RandomAccessFile(f, "rw");
             str ="// This Client Program is automatically generated
\n" +
                   "// according to user specs in XML doc. \n"+
                  "// 1) imports required libraries and packages \n"+
                  "// 2) set System properties \n"+
                  "// 3) locate Object on the ORB\n"+
                  "// 4) invoke transactions\n"+
                  "// 5) Commit if all above return normal\n"+
                  "// 6) Rollback in case of failure \n\n"
                  +"import "+ pack+ ".*;" +"\n" +" import java.io.*;\n
import java.util.*;\n"
                  + "import org.omg.CosTransactions.*;\n import
java.lang.*;\n";
             sw.writeBytes(str);
            } catch(IOException e2){}

      }

      void initOrb()
      {     System.out.println("client!");
            try{
             str = "public class Client{ \n" +
                   "\t public static void main (String[] args) throws
Exception" + "\n" +
                   "\t{ \n\t\t boolean commit= false;"+
                   "\n\t\tProperties props =
System.getProperties();\n"+
                    "\t\t" +"String services = \"\" ;\n"+
                     "\t\t"+ "services = (String) props.get(\"
ORBservices \");\n"+
                     "\t\t"+  "if (services == null ||
services.length() ==0)\n"+
                      "\t\t"+ "{\n" +
                        "\t\t\t"+   "props.put (\"ORBservices\",
\"com.visigenic.services.CosTransactions\");" +
                        "\n\t\t\t\t"+ "} else { }\n " +
                        "\t\t"+ "org.omg.CORBA.ORB orb =
org.omg.CORBA.ORB.init(args, props);\n"+
                        "\t\t"+ "org.omg.CORBA.BOA boa =
orb.BOA_init();\n" +
```

```
                        "\t\t"+ "if (args.length != 3)\n" +
                              "\t\t\t{ System.out.println (\"Usage:\");
}\n\n"
                        + "\t\t"+ "// todo: customize the above check
and Usage msg: \n\n" ;


                  sw.writeBytes(str);
                  } catch(IOException e2){}

      }
void locateObject(String LObjClass, String LObjInstance)
{
      try{ str ="";
            str = "\n\t\t"+LObjClass + " " + LObjInstance + " = " +
LObjClass+"Helper.bind(orb," + "\""+LObjInstance+"\"" + " );\n";
            sw.writeBytes(str);
      } catch(IOException e2){}
}
void startTx()
{
      try
      {     // instance of Current interface
            str = "\tCurrent current;\n" +
            " \t{ org.omg.CORBA.Object initRef =
orb.resolve_initial_references(\" TransactionCurrent \")" +
            ";\n" + "\t\tcurrent = CurrentHelper.narrow(initRef);\n\t
}\n" + "\tcurrent.begin();\n";
            sw.writeBytes(str);
      } catch(IOException e2){}
}

void writeTx(String ReturnType, String ReturnVar, String Obj, String
method)
{

      try
      {
            if (ReturnType != null)
            {

                  str = "\t\t// Tx goes here. Do: set return values and
parameters as appropriate \n\n"
                  + "\t try{ \n\t\t" +
                   ReturnType + " " + ReturnVar+ " = " + Obj + "." +
method + ";\n\n \t\t// other transactions go here\n" +
                  "\t\t}catch (org.omg.CORBA.SystemException e2){} \n";
            }
      sw.writeBytes(str);
      }catch (IOException e2){}
      str = "";
}
void writeTx(String Obj, String method)
{
      try
      {
```

```
            str = "\t\t// Tx goes here. Do: set return values and
 parameters as appropriate \n\n"
            + "\t try{ \n\t\t" +
                  Obj +"." + method + ";\n\n \t\t"+
            "}catch (org.omg.CORBA.SystemException e2){} \n";
            sw.writeBytes(str);
      }catch (IOException e2){}
      str = "";

}
void commitTx()
{
      try
      {
            str = "\t\t// if no exception raised then commit otherwise
rollback \n" +
             "\n\n\t\t //Commit or RoleBack the transaction\n" +
                  "\n\t\t commit = true; \n\n"+
                  "\n\tif (commit)" +
                  "\n\t {\n\t\tSystem.out.println(\" *Transactoin
Commited* \");" +
                  "\n\t\t current.commit(false);" +
                  "\n\t }" +
                  "\n\t else" +
                   "\n\t{\n\t\t\t\t System.out.println(\" Rolling Back Tx
\");" +
                  "\n\t\t current.rollback(); \n \t} \n\t\n} \n}";
                  sw.writeBytes(str);
      }catch (IOException e2){}
}

void VITSCurrent()
{
      try
      {
      str = "\norg.omg.CORBA.Object Obj =
orb.resolve_initial_reference(\"TransactionCurrent\");"+
      "\norg.visigenic.VISTransactoins.Current current "+
      "\n = org.visigenic. VISTransactoins.CurrentHelper.narrow(obj);"+
       "\n current.begin();\n";
      sw.writeBytes(str);
      }catch (IOException e2){}
}

void OMGCurrent()
{
try {
      str = "\norg.omg.CORBA.Object\n"+
      " obj = orb.resolve_initial_reference(\"TransactionCurrent\");"+
      "\norg.omg.CosTransactions.Current "+
      "\n current =
org.omg.CosTransactions.CurrentHelper.narrow(obj);"+
       "\n current.begin();\n";
      sw.writeBytes(str);
      }catch (IOException e2){}
}
```

```
void Coordinator()
{
       try
       {
       str = "\nControl_Var control;"+
       "\nTerminator_var newTranTerminator;"+
       "\nCoordinator_var newTranCoordinator;"+
       "\nNewTranTerminator = control.get_terminator();"+
       "\nNewTranCoordinator = control.get_coordinator();";
       sw.writeBytes(str);
       }catch (IOException e2){}

}
}
```

Appendix B

Listing of Generated Code; Client.java and Server.java for the Bank application

Client.java

```
// This Client Program is automatically generated
// according to user specs in XML doc.
// 1) imports required libraries and packages
// 2) set System properties
// 3) locate Object on the ORB
// 4) invoke transactions
// 5) Commit if all above return normal
// 6) Rollback in case of failure

import quickstart.*;
 import java.io.*;
 import java.util.*;
import org.omg.CosTransactions.*;
 import java.lang.*;
public class Client{
       public static void main (String[] args) throws Exception
      {
             boolean commit= false;
            Properties props = System.getProperties();
            String services = "" ;
            services = (String) props.get(" ORBservices ");
            if (services == null || services.length() ==0)
            {
                  props.put ("ORBservices",
"com.visigenic.services.CosTransactions");
                  } else { }
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,
props);
            org.omg.CORBA.BOA boa = orb.BOA_init();
            if (args.length != 3)
                  { System.out.println ("Usage:"); }

            // todo: customize the above check and Usage msg:


org.omg.CORBA.Object Obj =
orb.resolve_initial_reference("TransactionCurrent");
org.visigenic.VISTransactoins.Current current
 = org.visigenic. VISTransactoins.CurrentHelper.narrow(obj);
 current.begin();

            Bank myBank = BankHelper.bind(orb,"myBank" );
            // Tx goes here. Do: set return values and parameters as
appropriate

       try{
            Account acc1 = myBank.Get_Account(accountname);

            // other transactions go here
```

```
            }catch (org.omg.CORBA.SystemException e2){}
            // Tx goes here. Do: set return values and parameters as
 appropriate

        try{
            Account acc2 = myBank.Get_Account(accountname);

            // other transactions go here
            }catch (org.omg.CORBA.SystemException e2){}
            // Tx goes here. Do: set return values and parameters as
 appropriate

        try{
              acc1.Debit(Amt);

            // other transactions go here
            }catch (org.omg.CORBA.SystemException e2){}
            // Tx goes here. Do: set return values and parameters as
 appropriate

        try{
              acc2.Credit(Amt);

            // other transactions go here
            }catch (org.omg.CORBA.SystemException e2){}
            // if no exception raised then commit otherwise rollback


             //Commit or RoleBack the transaction

             commit = true;


      if (commit)
       {
            System.out.println(" *Transactoin Commited* ");
             current.commit(false);
       }
       else
      {
                  System.out.println(" Rolling Back Tx  ");
             current.rollback();
      }

}
}
```

Server.java

```
//This is a Server Program automatically generated
//the Server performs the following according to user specifications
//1) import required libraries and packages
//2) Initialize an ORB
//3) register Objects in the ORB
//4) wait for requests

import quickstart.*;
 import java.io.*;
 import java.util.*;
public class Server {
       public static void main (String[] args) throws Exception
      { Properties props = System.getProperties();
      String services = "" ;
      services = (String) props.get(" ORBservices ");
      if (services == null || services.length() ==0)
            {
            props.put ("ORBservices ",
"com.visigenic.services.CosTransactions ");            } else { }
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, props);
      org.omg.CORBA.BOA boa = orb.BOA_init();
            if (args.length != 1)
{ System.out.println ("Usage:"); }
Storage myStorage = StorageHelper.bind(orb,"myBank");
Bank myBank=  new BankImpl("myBank",orb);
boa.obj_is_ready(myBank);
boa.impl_is_ready();
}
}
```

Appendix C

Listing of Generated files; Client.java and Server.java for Point of Sale application

Client.java

```
// This Client Program is automatically generated
// according to user specs in XML doc.
// 1) imports required libraries and packages
// 2) set System properties
// 3) locate Object on the ORB
// 4) invoke transactions
// 5) Commit if all above return normal
// 6) Rollback in case of failure

import POS.*;
 import java.io.*;
 import java.util.*;
import org.omg.CosTransactions.*;
 import java.lang.*;
public class Client{
       public static void main (String[] args) throws Exception
      {
             boolean commit= false;
            Properties props = System.getProperties();
            String services = "" ;
            services = (String) props.get(" ORBservices ");
            if (services == null || services.length() ==0)
            {
                  props.put ("ORBservices",
"com.visigenic.services.CosTransactions");
                  } else { }
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,
props);
            org.omg.CORBA.BOA boa = orb.BOA_init();
            if (args.length != 3)
                  { System.out.println ("Usage:"); }

            // todo: customize the above check and Usage msg:


            Store store = StoreHelper.bind(orb,"store" );

            StoreAccess  storeaccess = StoreAccess
Helper.bind(orb,"storeaccess" );

            Tax taxObj = TaxHelper.bind(orb,"taxObj" );
            // Tx goes here. Do: set return values and parameters as
appropriate

      try{
            storeaccess.FindPrice();

            }catch (org.omg.CORBA.SystemException e2){}
            // Tx goes here. Do: set return values and parameters as
appropriate
```

```
        try{
             float STotals = storeaccess.GetStoreTotals();

             // other transactions go here
             }catch (org.omg.CORBA.SystemException e2){}
             // Tx goes here. Do: set return values and parameters as
appropriate

        try{
             float tax = taxObj.calculateTax(Amt);

             // other transactions go here
             }catch (org.omg.CORBA.SystemException e2){}
             // Tx goes here. Do: set return values and parameters as
appropriate

        try{
             Visa.GetBalance();

             }catch (org.omg.CORBA.SystemException e2){}
             // if no exception raised then commit otherwise rollback


              //Commit or RoleBack the transaction

              commit = true;


      if (commit)
       {
            System.out.println(" *Transactoin Commited* ");
             current.commit(false);
       }
       else
      {
                  System.out.println(" Rolling Back Tx  ");
             current.rollback();
      }

}
}
```

Server.java

```
//This is a Server Program automatically generated
//the Server performs the following according to user specifications
//1) import required libraries and packages
//2) Initialize an ORB
//3) register Objects in the ORB
//4) wait for requests

import POS.*;
 import java.io.*;
 import java.util.*;
public class Server {
       public static void main (String[] args) throws Exception
      { Properties props = System.getProperties();
      String services = "" ;
      services = (String) props.get(" ORBservices ");
      if (services == null || services.length() ==0)
            {
            props.put ("ORBservices ",
"com.visigenic.services.CosTransactions ");            } else { }
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, props);
      org.omg.CORBA.BOA boa = orb.BOA_init();
            if (args.length != 1)
{ System.out.println ("Usage:"); }
Store store=  new StoreImpl("store",orb);
boa.obj_is_ready(store);
StoreAccess  storeaccess=  new StoreAccess Impl("storeaccess",orb);
boa.obj_is_ready(storeaccess);
Tax taxObj=  new TaxImpl("taxObj",orb);
boa.obj_is_ready(taxObj);
boa.impl_is_ready();
}
}
```

Appendix D

Listing of CosTransactions.idl interface

```
// From OMG

#ifndef _costransactions_idl_
#define _costransactions_idl_

#pragma prefix "omg.org"

module CosTransactions
{
      // Forward references for interfaces defined later in module

      // In Java we will generate pseudo classes for Current
      interface Current;

      interface TransactionFactory;
      interface Control;
      interface Terminator;
      interface Coordinator;
      interface RecoveryCoordinator;
      interface Resource;
      interface Synchronization;
      interface SubtransactionAwareResource;
      interface TransactionalObject;

      // DATATYPES
      enum Status
      {
            StatusActive,
            StatusMarkedRollback,
            StatusPrepared,
            StatusCommitted,
            StatusRolledBack,
            StatusUnknown,
            StatusNoTransaction,
            StatusPreparing,
            StatusCommitting,
            StatusRollingBack
      };
      enum Vote
      {
            VoteCommit,
            VoteRollback,
            VoteReadOnly
      };

      // Structure definitions
      struct otid_t
      {
            long formatID; /*format identifier. 0 is OSI TP */
            long bqual_length;
            sequence <octet> tid;
      };
```

```
        struct TransIdentity
        {
              Coordinator coordinator;
              Terminator terminator;
              otid_t otid;
        };
        struct PropagationContext
        {
              unsigned long timeout;
              TransIdentity current;
              sequence <TransIdentity> parents;
              any implementation_specific_data;
        };

        // Heuristic exceptions
        exception HeuristicRollback {};
        exception HeuristicCommit {};
        exception HeuristicMixed {};
        exception HeuristicHazard {};
        // Other transaction-specific exceptions
        exception SubtransactionsUnavailable {};
        exception NotSubtransaction {};
        exception Inactive {};
        exception NotPrepared {};
        exception NoTransaction {};
        exception InvalidControl {};
        exception Unavailable {};
        exception SynchronizationUnavailable {};


        // Current transaction
        //interface Current : CORBA::ORB::Current

// In Java we will generate pseudo classes for Current
        interface Current
        {
              void begin()
                    raises(SubtransactionsUnavailable);
              void commit(in boolean report_heuristics)
                    raises(
                          NoTransaction,
                          HeuristicMixed,
                          HeuristicHazard
                    );
              void rollback()
                    raises(NoTransaction);
              void rollback_only()
                    raises(NoTransaction);
              Status get_status();
              string get_transaction_name();
              void set_timeout(in unsigned long seconds);
              Control get_control();
              Control suspend();
              void resume(in Control which)
                    raises(InvalidControl);
        };
```

```
interface TransactionFactory
{
      Control create(in unsigned long time_out);
      Control recreate(in PropagationContext ctx);
};

interface Control
{
      Terminator get_terminator()
            raises(Unavailable);
      Coordinator get_coordinator()
            raises(Unavailable);
};

interface Terminator
{
      void commit(in boolean report_heuristics)
            raises(
                  HeuristicMixed,
                  HeuristicHazard
            );
      void rollback();
};

interface Coordinator
{
      Status get_status();
      Status get_parent_status();
      Status get_top_level_status();
      boolean is_same_transaction(in Coordinator tc);
      boolean is_related_transaction(in Coordinator tc);
      boolean is_ancestor_transaction(in Coordinator tc);
      boolean is_descendant_transaction(in Coordinator tc);
      boolean is_top_level_transaction();
      unsigned long hash_transaction();
      unsigned long hash_top_level_tran();
      RecoveryCoordinator register_resource(in Resource r)
            raises(Inactive);
      void register_synchronization (in Synchronization sync)
            raises(Inactive, SynchronizationUnavailable);
      void register_subtran_aware(in SubtransactionAwareResource
r)
            raises(Inactive, NotSubtransaction);
      void rollback_only()
            raises(Inactive);
      string get_transaction_name();
      Control create_subtransaction()
            raises(SubtransactionsUnavailable, Inactive);
      PropagationContext get_txcontext ()
            raises(Unavailable);
};

interface RecoveryCoordinator
{
      Status replay_completion(in Resource r)
            raises(NotPrepared);
};
```

```
        interface Resource
        {
              Vote prepare()
                    raises(
                          HeuristicMixed,
                          HeuristicHazard
                    );
              void rollback()
                    raises(
                          HeuristicCommit,
                          HeuristicMixed,
                          HeuristicHazard
                    );
              void commit()
                    raises(
                          NotPrepared,
                          HeuristicRollback,
                          HeuristicMixed,
                          HeuristicHazard
                    );
              void commit_one_phase()
                    raises(
                          HeuristicHazard
                    );
              void forget();
        };

        interface TransactionalObject
        {
        };

        interface Synchronization : TransactionalObject
        {
              void before_completion();
              void after_completion(in Status status);
        };

        interface SubtransactionAwareResource : Resource
        {
              void commit_subtransaction(in Coordinator parent);
              void rollback_subtransaction();
        };

};  // End of CosTransactions Module

#pragma prefix ""
#endif
```

BIBLIOGRAPHY

[Abbadi et al. 94] Gustavo Alonso, Radek Vingralek, Divyakant Agrawal, Yuri Breitbart, Amr El Abbadi, Hans-J. Schek and Gerhard Weikum. Unifying concurrency control and recovery of transactions. Information systems vol. 19, No. 1, p 101-119, 1994.

[Anceaume 93] Emmanuelle Anceaume, A comparison of Fault-Tolerant Atomic Broadcast Protocols. IEEE, p 166- 171, 1993.

[Andry 94] Rakotonirainy Andry. Exploiting transction and Object semantics to increase concurrency. Applications in Parallel and distributed computing, IFIP, p 155- 164, 1994.

[Ansari et al. 92] Mansoor Ansari, Linda Ness, Marek Rusinkiewicz, and Amit Sheth, Using flexible transactions to support multi-system telecommunication applications.

[Aslam-Mir98] Shahzad Aslam-Mir. Writing transactional CORBA applications, part 2. Corba Development. P.11-18. Vol. 4, Issue 6. October 1998.

[Bala98] Raghuram Bala. MedLinx Interactive Browser. The portal for healthcare professionals. Corba Development. P. 7-10. Vol. 4, Issue 6. October 1998

[BEA98_a] BEA Systems. BEA M3 below the Water-line. www.beasys.com/

[BEA98_b] BEA Systems. BEA M3 frequently asked questions. www.beasys.com/faq/

[BEA98_c] BEA Systems. Intruducing BEA M3, the world's first component middleware that scales for enterprise. www.beasys.com/

[BEA98_d] BEA Systems. BEA builder for TUXEDO. www.beasys.com/

[BEA98_e] BEA Systems. BEA TUXEDO datasheet. www.beasys.com/

[BEA98_f] BEA Systems. Introduction to the BEA M3 System. Ed. 2.1. June 1998. www.beasys.com/

[BEA98_g] BEA Systems. How the BEA M3 System Works. Ed. 2.1. June 1998. www.beasys.com/

[Beeri 86] C. Beeri, P.A.Bernstein, N. Goodman. A model for concorrency in Nested transaction systems. TR TR-86-03, Wang Institute, 1986.

[Bharg et al. 94] Bharat Bhargava, Yongguang Zhang, Shalab Goel, A Study of Distributed Transaction Processing in an Internetwork. Raid Laboratory, Department of Computer Sciences, Purdue University, IN, USA, 1994.

[Birrell&Nelson 84] Birrell, A.D. and Nelson, B.J., Implementing rmote procedure calls. ACM Trans. Computer systems, Vol 2, pp. 39-59.

[Blaustein et al. 93] Barbara T. Blaustein, Sushil Jajobia, Catherine D. mcCollum and Louanna Notargiacomo, A Model of Atomicity for Multilevel Transactions. IEEE, P 120- 134, 1993.

[Boutros 94] Boutros S. Boutros. Performance Evaluation of a Prudent Two-Phase Commit Protocol. Theses. Concordia University 1994.

[Bracha&Toueg 87] G. Bracha and S.Toueg. Distributed deadlock detection. Distributed computing, 2:127-138, 1987.

[Bradley 2000] Neil Bradley, The XML companion. Addison-Wesley 2000.

[Chappell 97] David Chappell & Patricia Seybold, The MTS Distributed Computing Monitor, June 1997. http://www.microsoft.com/com/wpaper/mtscomp.htm

[Chen 95] Graham Chen, Distributed Transaction processing standards and their applications. Computer Standards and Interfaces, 1995.

[Chen&Dayal 96] Qiming Chen and Umesh Dayal, A Transactional Nested Process Management System. IEEE, p 566- 573, 1996.

[Chung&Mah 97] Soon M. Chung and Pyeong S. Mah, Multidatabase Transaction Management Scheme Supporting Multiple Subtransactions of Global Transaction at a Site. Informatics and Computer Science, p 242- 266, 1997.

[CORBA98] CORBA Development, for CORBA Developers, by CORBA Developers. Volume 4, Issue 6 October, 1998.

[CORBAOTS 97] CORBA specifications OTS service.

[Coulouris et al. 96] George Coulouris, Jean Dollimore and tim Kindberg, "Distributed Systems,

concepts and design". Second Edition, Addison-Wesley 1996

[DCOMDesc 97] DCOM Description

http://diana.ecs.soton.ac.uk/~dem97r/dcom/dcdescription.html

[Deacon et al. 94] Andrew Deacon, Hans-Jorg Check and Gerhard Weikum, Semantics-based Multilevel Transaction Management in Federal Systems. IEEE software, p 452-461, 1994.

[Dogac et al. 98] Asuman Dogac, Cevedet Dengi, and M. Tamer Oszu. Distributed object xomputing platforms. Communications of the ACM. p 95-103, Vol. 41, No. 9, Sept. 1998.

[Dynes&Gruber 93] Laurent Daynes and Olivier Gruber, Nested Actions in Eos. IEEE, p 145-163, 1993.

[Elmagarmid et al. 90] A. Elmagarmid, y.Leu, W. Litwin, and M. Rusinkiewicz. A multidabase trasnaction Model for interbase. Proceedings of the 16th VLDB, August 1990.

[Elmg 86] A.K.Elmagarmid. A survey of distributed deadlock detection algorithms. SIGMOD Record, 15(3), 1986.

[Encina] www.transarc.com/News/press/bmc-9806.html

[Farrag&Abawajy 94] Abdel Aziz Farrag and Jemal Abawajy, Formal Model for Verifying Compatibility among Transactions. IEEE p 76- 81, 1993.

[Floyd 98] Micheal Floyd, Buidling Web Sites with XML. PH PTR.

[Francesco et al. 94] Nicoletta De Francesco, Ugo Montanari and Gioia Ristori, Modelling Concurrent Accesses to Shared Data via Petri Nets. Programming Concepts, Methods and Calculi, IFIP, p. 403- 422, 1994.

[GartnerGroup98] BEA M3 will put components to an enterprise test. InSide GartnerGroup This Week 24, June 1998.

[Geppert&Dittrich 94] Andeas Geppert and Klaus R. Dittrich, Rule-Based Implementation of Transaction Model Specifications. 1994.

[Gligor&Shattuck 80] V. Gligor and S. Shattuck. On deadlock detection in distibuted systems.

IEEE transactions on Software Engineering, SE-6(5), Sept. 1980.

[Goldfarb&Prescod 2000] Charles F. Goldfarb and Paul Prescod, The XML Handbook. Prentice Hall PTR.

[Grasso 97] Ennio Grasso, Implementing Interposition in CORBA Object Transaction Service. CSELT, Italy, 1997.

[Gray&Reuter 92] J. Gray and A. Reuter. Transaction processing, cocepts and techniques. Morgan Kaufmann Edition, 1992.

[Gray81] Gray, Jim. The transaction concept: Virtues and Limitations. IEEE transactions on Software engineering Vol 2. P144-154 1981.

[Grimes 97] Richard Grimes, Professional DCOM, WROX Press 1997.

[Guerraoui 93] Rachid Guerraoui, Toward Modular Concurrency Control for Object-Oriented Distributed Systems. IEEE, p 240-246, 1993.

[Gupta 96] Samir Gupta. Essay in real-time distributed transaction processing and graphical decision. Ph.D. University of Ann Arbor, MI. 1995.

[Haque&Wong 94] Waqar Haque and Johnny Wong, Distributed Readl-Time Nested Transactions. Systems Software, p 85-95,1994,.

[Harder 84] Theo Harder, Observations on Optimistc Concurrency Control Schemes. Information Systems, vol. 9, No. 2, p. 111-120, 1984.

[Harman98] Paul Harman. Distributed component systems. Component development strategies. p 1-16, Vol. 8, No. 8. August 1998.

[Harmon 98] Paul Harmon. Component development strategies Newsletter. Vol. VII, No. 8, August 1998. www.cutter.com/itgroup/

[Hitachi98] TPBroker white paper www.tpbroker.com/tpbroker/product_info/tpwpaper.html

[Hitachi98_a] TP Monitors for electronic commerce, Hitachi's TPBroker a first look. www.tpbroker.com/tpbroker/product_info/

[Hitachi98_b] Hitachi Software. Object Monitors Defined. www.tpbroker.com/tpbroker/pruduct_info/objmon.htm

[Hopkins&Tilden98] Scott. Hopkins and Mark Tilden. Developing a CORBA-Based online reservation system. Corba Development. P2-6. Vol. 4, Issue 6. October 1998.

[Humm 93] Benhard G. Humm. An extended Scheduling Mechanism for Nested Transactions. IEEE, p 125- 134, 1993.

[I/S 94] How middleware can be used to create enterprise and inter-enterprise applications. I/S Analyzer, Vol. 33, No. 7, P 1-15, July 1994.

[IBM98_a] IBM. IBM transactions, Benefits of TXSeries. www.software.ibm.com/ts/txseries/about/

[IBM98_b] IBM. IBM CICS. www.software.ibm.com/ts/cics/about/gim/dfhe2ch1.html

[IBMCICS97] What is CICS?, www.software.ibm.com/is/sw-servers/transaction/whatisCICS.html

[IBMT] Transaction Series, www.software.ibm.com/is/sw-servers/transaction/

[infoworld 97] Test Results http://www.infoworld.com/

[internetworld 97] CORBA and DCOM, How they work with the web

[IONA97_a] IONA Technologies. OrbixOTM Guide. December 1997.

[IONA97_b] IONA Technologies. OrbixOTS white papers. www.iona.com/support/whitepapers/ots

[IONA98_a] IONA Technologies. Transactions. www.iona.com/products/transactions/

[IONA98_b] IONA Technologies. OrbixOTM. www.iona.com/products/transactions/OTM/'

[ISG97] International systems Group, Inc. Middleware White Paper. 1997. www.isg-inc.com

[Kaciste 94] Gerard Lacoste, Distributed Transaction processing in IBC. IBM, France. Lecture notes in Computer Science, Towards a Pan-European Telecommunication Service infrastructure – IS&N'94, Second International Conference on Intelligence in Broadband Services and networks, Germany, September 1994.

[Kindel 98] Charlie Kindel, DCOM protocol. http://premium.microsoft.com/msdn/library/echart/msdn_dcomprot.htm

[Kohler&Hsu 90] Walter H. Kohler and Yun-Ping Hsu, A Benchmark for the Performance Evaluation of Centralized and Distributed Transaction Processing Systems. IEEE, p 488-493, 1993.

[Korth 83] H. Korth. Locking primitives in database system. Journal of ACM, 30(1), Jan. 1983.

[Kunkelmann et al. 97] Thomas Kunkelmann, Hartmut Vgler, Technische Hochschule Dermstads, and Susan Thomas, Interoperability of Distributed Transaction Processing Systems. Digital Equipment GmbH, CEC Karlsruhe., 1997.

[Lia95] R. Lia. A simulation of the ISO trasnction processing protocol. IEEE, p 26-30, 1995.

[Liangkuan et al. 93] Chen Liangkuan, Chen dequing and Zhang Ho, The Object Oriented Distributed transaction. IEEE TENCON, 287-290 1993.

[Litwin et al. 92] Witold Litwin, Dimitrios Georgakopoulos and Marek Rusinkiewicz, Chronological Scheduling of Transactions with Temporal Dependecies. 1992.

[Manola et al. 94] Dimitrios Georgakopoulus, Mark Hornik, Piotr Krychniak, and Frank Manola. Specification and management of of extended trasnactions in a programmable trasnaction environment. IEEE, p 462-473, 1994.

[Marz 84] Marzullo, K. (1984) Maintaining the time in a distributed system. Tech. Report OSD-T8401, Xerox Corporation.

[Menasce&Muntz 79] D.Menasce and R.Muntz. Locking and deadlock detection in distributed data bases. IEEE Tranactions on software Engineering, SE-5(3), May 1979.

[Microsoft97_a] Microsoft. The COM+ programming model makes it easy to write components in any language.

[Microsoft97_b] Microsoft. Windows NT server, DCOM technical overview. 1997.

[Microsoft97_c] Microsoft. Windows NT server, DCOM solution in act. 1997.

[Microsoft97_d] Microsoft. Comparing Microsoft Transaction Server to Enterprise JavaBeans. www.microsoft.com. July, 1998.

[Moss81] Moss, Eliot J., Nested transactions: An approach to reliable distributed computing. PHD Thesis, M.I.T. Department of electrical engineering and computer science, April 1981. Available as M.I.T. laboratory for Computer Science Technical Report 260.

[Mowbray 94] Thomas J Mowbray, choosing between OLE/COM and CORBA. Distribute Object Computing. P 39-45. Nov.- Dec. 1994.

[MSOLE 96] Kraig Brockschmidt, OLE Team, What OLE is really about. Microsoft Corporation, July 1996. http://www.microsoft.com/aboutole.html

[Nett&Weiler 94] Edgar Nett and Beatrice Weiler, Nested Dynamic Actions- how to solve the fault containment problem in cooperative action model. IEEE, p 106- 115, 1994.

[Nygard 94] Mads Nygard. Partial recoverability with distributed transactions. IEEE, p 68-76, 1994.

[Ober 82] R. Obermarck. Distributed deadlock detection algorithm. ACM Transactions on Database Systems, 7:187-208, June 1982.

[OLE 98] OLE automation, http://www.inforamp.net/~kjvallil/t/oleauto.html

[OMG CORBA'95] OMG CORBA Specifications, Section 10 on OTS, 1995.

[OMG RFP99] Draft Request For Proposal by OMG url:\\www.omg.org\RFP\xots99rev

[OMG 2000] Mark Elenko and Mike Reinertsen. XML and CORBA www.omg.org/library/adt.html

[ORBIX OTM Notes 98] ORBIX OTM Seminar, IONA Tech. Montreal –ON, June 1998.

[Orfali96] Orfali J. "CORBA and OLE business Objects. ". Second Edition, Addison-Wesley 1996.

[Pafford 96] Michael Pafford. A survey of distributed transaction processing standards and a proposal for Mobile x/open XA,Thesis. University of Texas at

Arlington.

[Palo 96] Palo Alto, New version of HP Encina/9000 Transaction processing Monitor Enhances HP's Middleware Engineering Initiative. HP Press release,

[Panagos et al. 96] E. Panagos, A. Biliris, H.V. Jagadish and R. Rastogi, Client-Based Logging for High Perfomance Distributed Architectures, IEEE, p 344- 351,1996.

[Pu et al. 93] Calton Pu, Wenwey Hseush, Gail E. Kaiser, Kun-Lung Wu and Philiph S. Yu, Distributed Divergence Control for Epsilon Serializability. IEEE, p 449- 456, 1993.

[Ranganathan 94] Aravindan Ranganathan. Techniques and models for rollback recovery in computer systems. Ph.D. Thesis. University of New York at Buffalo. 1994

[Ravoor&Wong 97] Suresh B. Ravoor and Johnny S. K. Wong, Multithreaded transaction processing in distributed systems. Systems Software:38, p 107-117, 1997.

[Raymond 94] Kerry Raymond, Reference Model of Open distributed processing (ODP-RM): Introduction. CRC of Distributed Systems Technology, Centre of Information Technology Research, University of Queensland, Australia. http://www.dstc.edu.au/AU/research-news/odp/ref-model/papers.html

[Raymond&Berry 94] Andrew Berry and Kerry Raymond, An improved model for transactional operations in RM-ODP, CRC for distributed systems technology. IFIP 1994.

[Resendes&Laukien 98] Robert Resendes and Marc Laukien, Introduction to Corba Distributed Objects. C/C++ Users Journal, www.cuj.com , p 55- 66, April 1998.

[Roes&Burk 88] Marina Roesler and Walter A. Burkhard. Deadlock Resolution and Semantic Lock Models in Object –Oriented Distributed Systems. ACM SIGMOD 17(3), 361-370. 1988.

[Rusinkiewicz et al. 92] Marek Rusinkiewicz, Dimitrios Georgakopoulos and Raj Kumar Batra, A Decentralized Deadlock-free Concurrency Control Method for Multidatabase Transactions. 1992.

[Rusinkiewicz et al. 95] M. Rusinkiewicz, W. Klas, T. Tesch, J. Wasch and P. Muth, Twards a Cooperative transaction model- The Cooperative Activity Model-.

Preceedings of the 21th VLDB Conerence Zurich, Switzerland, p 194-204, 1995.

[Schurmann 95] Gerd Schurmann The evolution from open systems interconnection (OSI) to open distributed processing (ODP). Computer Standards and Interfaces 1995.

[Schuster et al. 94] Hans Shuster, Stefan Jablonski, Thomas Kirsche, Christoph Bussler, A Client/Server architecture for distributed workflow management systems. IEEE, p 253-256, 1994.

[Schwarz 84] P.M. Schwarz. Transactions on Typed Objects. Thesis, CMU, Dec. 1984.

[Shrivastava et al. 93] S.K. Shrivastava, M. C. Little and D. L. McCue, Maintaining Information about Persistent Replicated Objects in a Distributed Systems. IEEE software, P 491-498, 1993.

[Shrivistava 95] Santosh K. Shrivastava, "Lessons Learned from Building and Using the Arjuna Distributed Programming System", Department of Computer Science University of Newcastle upon Tyne, UK

[Siegel 96] Jon Siegel. CORBA fundamentals and programming. Wiley publishers. 1996.

[Siemens98] Siemens. OpenUTM, new funtions, enhancements and changes in Version 5 of the OpenUTM family products. www.OpenUTM.com/

[Skarra 93] Andrea H. Skarra, SLEVE: Semantic Locking for Event Synchronization. IEEE software, p 495-502, 1993.

[Stachnik 93] George Stachnik, Distributed Transaction Processing CICS, Encina and DCE on HP. Proceeding of the 1993 HP Computer Users' European Conference.

[Stok&thijssen 94] P.D.V. van der Stok and P.T.A. Thijssen. Simulation of distributed Real-Time transactions. IEEE, p 82-87, 1994.

[Strigini et al. 1997] L. Strigini, F. Di Giandomenico and A. Romanovsky, Coordinated backward recovery between client processes and data servers. IEE software

engineering, Vol 144, No. 2, April 1997.

[SunRCard 97] Sun report card protocol http://premium.microsoft.com/msdn/library/orbflreport.gif

[Tada et al. 97] Harumasas Tada, Kazuyuki Uchida, Masahiro Higuchi, Mamoru Fujii. A model of Nested transaction with fine granularity of concorrency control. IEEE, p 911-920. 1997.

[Tarr & Sutton 93] Peri Tarr and Stanley M. Sutton, Programming Heterogeneous Transactions for Software Development Environments. IEEE, p 358- 369, 1993

[Tekinerdogan 94] Bedir Tekinerdogan, "Design of an object-oriented framework for atomic transactions", M.Sc. Thesis at the Department of Computer Science, University of Twente – Netherlands

[Thilmany 98] Christian J.Thilmany, Using MTS components on your Web Server with Java. Interactive Magazine Feb. 1998. http://www.microsoft.com/mind/0298/mts.htm

[Tjandra et al. 97] I.A.Tjandra, G. Butler, P. Grogono and R. Shinghal, Modeling Serializability of Distributed Tranactions. 1997.

[Ulusoy 95] Ozgur Ulusoy, A Study of Two Transaction-Processing Architectures for Distributed Real-Time Data Base Systems. System Software, Vol. 31, p 97-108, 1995.

[Veijalainen 94] Jari Veijalaninen, Hetergeneous Multilevel Transaction Management with multiple subtransactions. 1994

[Weihl 84] W.Weihl. Specification and implementation of Atomic data types. PHD thesis, MIT, Mar. 1984.

[Weihl 90] William E. Weihl. Using trasnactions in distributed applications. IEEE, p 366-371, 1990.

[Wong&Ravoor 97] Johnny S. K. Wong and Suresh B. Ravoor. Multithreaded transaction processing in distributed systems. Department of Computer Science, Iowa State University, Ames, Iowa. Systems Software, Issue 38, p 107-117, 1997

[Xiao&Campbell 93] Lun Xiao and Roy H. Campbell, Object-Oriented Transactions in Choices. IEEE software, p 50-59, 1993.

[Zhou92] Wanlei Zhou. A decentralized remote procedure call transaction manager. IEEE Region 10 Conference Tencon 92. Australia. November 1992.

VITA AUCTORIS

Jenane Abouzeki was born in 1973 in **Lebanon**. She received an IDPM Diploma from the **Institute of Data Processing Management** in **London, UK** In 1991. She obtained a B.Sc. in Computer Science from the **University of Windsor** in 1997. She was awarded an honorable mention prize by **ACM** for **JAVA Contest97**. She is currently a candidate for the Master's degree of Science at the University of Windsor and hope to graduate in the Spring of 2000.