Semantic Integrity Control and Interoperability for Component Based Software Development

THESIS

Submitted in partial fulfilment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

M. MADIAJAGAN

Under the Supervision of

Dr. B. VIJAYAKUMAR

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PILANI (RAJASTHAN) INDIA

2009

Semantic Integrity Control and Interoperability for Component Based Software Development

THESIS

Submitted in partial fulfilment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

M. MADIAJAGAN

Under the Supervision of

Dr. B. VIJAYAKUMAR

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PILANI (RAJASTHAN) INDIA

2009

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN)

<u>CERTIFICATE</u>

This is to certify that the thesis entitled **Semantic Integrity Control and Interoperability for Component Based Software Development** and submitted by **M. Madiajagan** ID. No. **2004PHXF439P** for award of Ph.D. Degree of the Institute embodies original work done by him under my supervision.

Signature in full of the Supervisor:	______________________
Name in capital block Letters:	**Dr. B. VIJAYAKUMAR**
Designation:	Associate Professor

Date:

ACKNOWLEDGEMENTS

I would like to express my deepest sense of gratitude, first and foremost, to my Ph.D. Thesis Supervisor **Dr. B. Vijayakumar**, Associate Professor, Computer Science Department, BITS, Pilani – Dubai, United Arab Emirates, for his valuable guidance and encouragement during the course of this research programme. I am extremely grateful to him for his able guidance, valuable technical inputs and useful suggestions. I express my sincere thanks and gratitude to our Director, BITS, Pilani – Dubai, **Prof. Dr. M. Ramachandran** for his motivation, encouragement and moral support to pursue my research.

I would like to place on record my deepest sense of gratitude to the **Research Board, B.I.T.S, Pilani** for giving me an opportunity to do this research work and providing me guidance throughout the research programme. I sincerely thank the Doctoral Advisory Committee, B.I.T.S. Pilani and also Doctoral Advisory Committee, BITS, Pilani-Dubai for its constant monitoring and advisory role that has really helped me very much throughout this programme. I am very much grateful to Dr. Ravi Prakash, Dean Research and Consultancy Division, B.I.T.S, Pilani (Rajasthan), India, Dr. G. Vijaya, Dr. K.K.Singh, Dr. K. Kumar, Mr. Dinesh Kumar, Ms. Monica Sharma for their constant support and encouragement given to me. I am grateful to Dr. K. N. Ramachandran Nair, Dr. V. Santhosh Kumar for their valuable suggestions.

I would like to thank my colleagues Dr. Manjunath, Dr. A. Somasundaram, Dr. S. Vadivel, Dr. Sameen Fatima, Ms. Sujala D Shetty, Ms. S Jeylatha, Ms. Alamelu Mangai, Ms. Susanna Henry, Ms. Mubeena and Mr. G. Rajendran for their interaction. I wish to thank Computer Science Department and Computer Support Group, BITS, Pilani-Dubai for providing me excellent facilities. I also would like to thank all other staff for their kind cooperation.

I acknowledge Ms. Barkha Keni, Mr. Satish Surath, Mr. Ananth Oswal, Mr. Hari Sreenivasan and Mr. B. Praveen for their help, technical inputs and interactions. I wish to thank my wife Mrs. M. Santy, my son Master M. Aroulmurugan, my daughter Miss M. Vithiya and my parents for providing me all support and cooperation in perusing this research programme.

Above all, I thank the Lord for giving me the strength to carry out this work to the best of my abilities.

M. Madiajagan
2004PHXF439P

ABSTRACT

Component-Based Software Development is gaining a lot of importance in the construction of high quality and evolvable software systems, in timely and affordable manner. The present work addresses the Semantic Integrity Control issue for distributed components by providing an environment for defining pre-conditions and post-conditions and enforcing them. It also deals with distribution issues in early phases of application development, namely requirements and specification. Design-time and Run-time interoperability for a distributed environment are considered in the present work.

A component based approach for enforcing Semantic Integrity Control in a distributed multi-database system has been modeled using UML 2.0. The design of core component includes separate interfaces for User, Administrator, and Database Handler. The algorithms for pre-condition and post-condition are formulated and discussed.

The Interoperability of multiple databases in a distributed environment has been realized using a Generic Model for the Database Search Application. The implementation has been carried out using two approaches:

1. JDBC and Java API.
2. ADO .NET Architecture.

A mathematical representation for the queries, for database component has been presented and it makes use of propositions. This will be helpful in the verification of simple as well as complex queries, both mathematically as well as experimentally.

The key factors responsible for maintaining *Semantic Integrity Control* and *Interoperability* are analyzed for three types of components: *Domain, Service* and *Agent*. The block schematic and design steps for *Domain, Service* and *Agent* components are discussed.

The report also summarizes the work, highlights specific contributions and suggests some directions for future work.

Contents

List of Tables

List of Figures

List of Abbreviations

API	-	Application Programming Interface
ADO	-	ActiveX Data Objects
CBD	-	Component Based Development
CCM	-	CORBA Component Model
COM	-	Component Object Model
CORBA	-	Common Object Request Broker Architecture
COTS	-	Commercial Off-the-Shelf
DB	-	Database
DBMS	-	Database Management System
DCOM	-	Distributed Component Object Model
DDBMS	-	Distributed Database Management System
DSS	-	Distributed Software Service
EJB	-	Enterprise Java Beans
GS	-	Generic Search
GUI	-	Graphical User Interface

HTML	-	Hypertext Markup Language
HTTP	-	Hyper Text Transfer Protocol
IDL	-	Interface Definition Language
IIOP	-	Internet Inter-Object request broker Protocol
JDBC	-	Java Database Connectivity
JRMP	-	Java Remote Method Protocol
MSSQL	-	Microsoft Structured Query language
OCL	-	Object Constraint Language
OLEDB	-	Object Linking Embedding Database
OMG	-	Object Management Group
OOP	-	Object Oriented Programming
ORB	-	Object Request Broker
OS	-	Operating System
PSQL	-	Postgres Structured Query Language
QoS	-	Quality of Service
RDF	-	Resource Description Framework
RMI	-	Remote Method Invocation

SOAP	-	Simple Object Access Protocol
SQL	-	Structured Query language
TCP/IP	-	Transmission Control Protocol / Internet Protocol
UI	-	Uniform Identifier
UML	-	Unified Modeling Language
URI	-	Uniform Resource Identifier
VB	-	Visual Basic
Visual Studio IDE	-	Visual Studio Integrated Development Environment
WSDL	-	Web Service Definition Language
XML	-	eXtensible Markup Language

Chapter 1
Introduction

Component-based software development is a *growing field* in computer science. Different frame works and standards for components have been developed. Most of them center on CORBA, COM, *JavaBeans* and EJB. Components are normally described to the environment through some Interface Definition Language (IDL). The *syntactic aspects* are usually well-defined and syntactic constructs that are applicable in all target languages are only allowed. The *semantic aspects* of components, however, are not so well supported and require considerable attention.

Component-based software development is gaining recognition as the key technology for the construction of high quality, evolvable, large software systems in timely and affordable manner. In this new setting, interoperability is one of the essential issues, since it enables the composition of reusable heterogeneous components developed by different people, at different times, and possibly with different uses in mind. Currently most object and component platforms, such as Common Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), or Enterprise Java Beans (EJB) already provide the basic infrastructure for component interoperability at the lower levels, i.e., they sort out most of the "plumbing" issues. However, interoperability goes far beyond that; it also involves behavioral compatibility, protocol compliance and agreements on the business rules. Information systems largely involve networking these days. The development of complex business applications is now focused on an assembly of components available on a local area network or on the net. These components must be localized and identified in terms of available services and communication protocol before any request.

1.1 Semantic Integrity

Informally, *semantics* is what a program means, as opposed to what the program looks like, which is called *syntax* [1]. Syntax is a number of rules defining possible legal ways to combine the words and symbols (tokens) in the language. Semantics is what the different combinations of these tokens actually perform in terms of logical operations. By *Semantic Integrity,* we mean the preservation of the collective semantic properties of a *program*, a *module* or a *system*. If the semantic properties are violated, the system will enter unstable or inconsistent states, which will, eventually, lead to some kind of malfunction. Hence it is necessary to define semantic integrity by way of conditions or assertions and maintain it over time when the *parts* of the system evolve. Some languages such as *Eiffel* [2] have assertion statements to ensure that semantic properties of modules are not violated, but most programming languages leave this task to the programmer. In any case, only a subset of all useful *preconditions* can be tested automatically, so the programmer should always be disciplined in his/her approach to semantics. Semantic Integrity can be enforced using *contracts* or *assertions*. There are different approaches to semantic integrity. The most common approach is to express semantic integrity requirements using *assertions* for an entire system or for parts of a system (pre- and post - conditions).

1.2 Component

Everything ranging from *dll-files* to *stand-alone applications* can be considered as components [3, 4, 5]. Some authors argue that a component needs to be executable on its own whereas others say that a component is simply one or more classes with a common interface. The present work focuses on component design meeting the following requirements:

1. Enforce Semantic Integrity control.
2. Facilitate Interoperability under varying system configurations.

1.3 Interface

Literally, an *interface* refers to whatever is needed (“equipment or programs”) in order to “communicate between different systems or programs”. When used to define *modules, classes and components*, however, it usually means the *syntactic definition* of this interface. This is what is called *signature* in C++, and includes the *function name* and the *number* and *types* of the arguments. It may also include the *return type*, the *exceptions thrown* and a few other *attributes*. This is also the implication of a CORBA IDL Interface Declaration [6]. This kind of interface description contains only an intuitive level of semantic description, through the choice of names for the functions and the formal arguments, and gives only a very limited support to semantic integrity considerations.

1.4 Contract

Contracts take part in the process of preserving the semantic integrity of a software module. A contract in programming terms is very analogous to a contract in real life. It defines certain benefits for both parts in an agreement. It is usually directional, having a provider and a consumer. The consumer must ensure that certain assertions are true before it can use the service offered by the provider. These assertions are called *preconditions* and it may or may not be possible to express them in a programming language [2]. The *service providing function* benefits from a *contract* because it does not have to check that the *precondition* is satisfied inside the service function itself, but can concentrate on providing the service as efficiently and cleanly as possible. The *provider* is obligated to produce the expected service. The *consumer* then knows that if it did satisfy the precondition, it is guaranteed correct service or output, as specified in the post *condition*. Pre- and post conditions were first introduced by Hoare in 1972 in his Hoaretriplets, but have been popularized in later years by Bertrand Meyer and the Eiffel Language [2].

The term *contract* [2] encompasses both pre- and post conditions, but also a clear definition of the *responsibilities* of both the *provider* and the *consumer* of a module. The contract should be viewed as a specification of responsibilities. A *contract* should be *machine-testable* and formalized to the extent that no ambiguities remain. What is often omitted is the fact that not all *contracts* can be formalized or even expressed in a *machine-testable* way. There are also certain *preconditions*, which are too costly to actually test. One example is to find out if a *networked database* can be updated. It might imply that the actual update must take place up until the final step, which is clearly not desirable, just to find out if it is legal to do the update. There are numerous examples of situations where it is not possible to machine-test a *contract*. In Eiffel [2], this kind of "*assertions*" is represented as *comments*.

1.5 Semantic Integrity for Components

This section discusses semantic integrity aspects for components. Five levels of semantic awareness have been identified and are summarized by the keywords "*No semantics"," Intuitive semantics", "Pragmatic semantics", "Executable semantics"* and *"Formal semantics"*.

The level *"No Semantics"* encompasses the discussions that focus exclusively on the *syntactic interface* descriptions. This level covers IDL and similar interface descriptions. It is not concerned with semantic aspects. It is quite common that an interface for a component is described through its signatures alone, and that the interpretation is left more or less to the user's intuition. The component (module) interface is described either textually by means of an interface description language (IDL) or visually / interactively using appropriate tools [7] and an interface is a collection of signatures of services belonging logically together.

Intuitive semantics It is important that the interfaces are semantically consistent when components are to be substituted or when components evolve, but without further specifying what that means. The successful reuse of components requires a good knowledge of the environment and architecture where the components shall be used [8]. This level covers unstructured descriptions and comments about what functions should do. It also covers testing, which is normally based more on an exhaustive attempt to find a wrong answer than on a description of the intended semantics of a system or a module.

"Pragmatic semantics" means that the designers and engineers are highly aware of the semantic implications and requirements of their components, but they do not express these semantics in any particular syntax or formalism. They express the semantic conditions as contracts inserted as comments in the design documentation, the interface descriptions or in the code.

"Executable semantics" means that the semantic aspects and contracts are expressed in some kind of executable language. They can be tested at run time but not used to prove that the program is correct. This covers the Object Constraint Language (OCL) [9] a specification language recently developed for the Unified Modeling Language (UML). OCL is suitable for expressing, for instance, invariants, *pre-conditions,* & *post-conditions*. A common approach to increased semantic integrity in both object-oriented and structured programming communities is to add or use assertions in the programming language. As discussed in [10] and shown in Figure 1.1, four levels of

contracts are specified. The first level is the *syntactic level* where usual interface definition languages and programming languages are used, this is the No Semantic level discussed above. The second level is the *behavioral level* where pre- and post-conditions are defined using Eiffel or similar languages. It corresponds to *"Executable Semantic"* level. This level suggests how to implement the *contracts* in an interface description language. The third level is the *synchronization level* where tools like *path expressions* and *service object synchronization* can be utilized. The fourth and final level is the *quality of service level* which makes use of tools like the adaptive communication environment ORB. Finally, "Formal semantics" cover the area of formal methods used to prove a program's semantic properties.

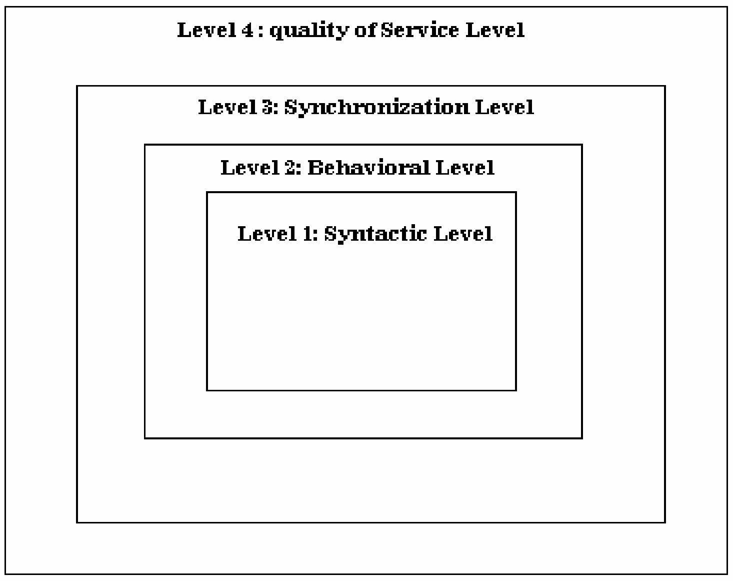
Level 4 : quality of Service Level
Level 3: Synchronization Level
Level 2: Behavioral Level
Level 1: Syntactic Level

Figure 1.1: The Four Contracts Levels

Contracts form a crucial part of a component for composition and inheritance [7]. The specification and implementation for components are treated separately. Components can be combined through their interfaces and the binding of code through appropriate interfaces is done at run-time.

1.6 Interoperability

Interoperability refers to the ability of information systems to operate in conjunction with each other encompassing communication protocols, hardware, software, application, and data compatibility layers. There has been considerable work in industry on the development of component interoperability models, such as CORBA, (D)COM and JavaBeans. These models are intended to reduce the complexity of software development and to facilitate reuse of off-the-shelf components. The focus of these models is syntactic interface specification, component packaging, inter-component communications, and bindings to a runtime environment.

Component-Based Software Development enables development of plug-and-play reusable software, which has led to the concept of '*commercial off-the-shelf*' (COTS) components [11]. COTS-based system development involves composition and reconciliation, whereas custom system development is an act of creation. Traditional development approach uses the waterfall model [12] comprising of system study, analysis, design, coding, testing, implementation and maintenance phases. COTS-based system development starts with a general set of requirements and then explores the marketplace's offerings to see how closely they match the needs; the engineers are consumers, who then integrate the products they buy into a system that meets the need. The approach to system development using COTS-based systems is shown in Figure 1.2.

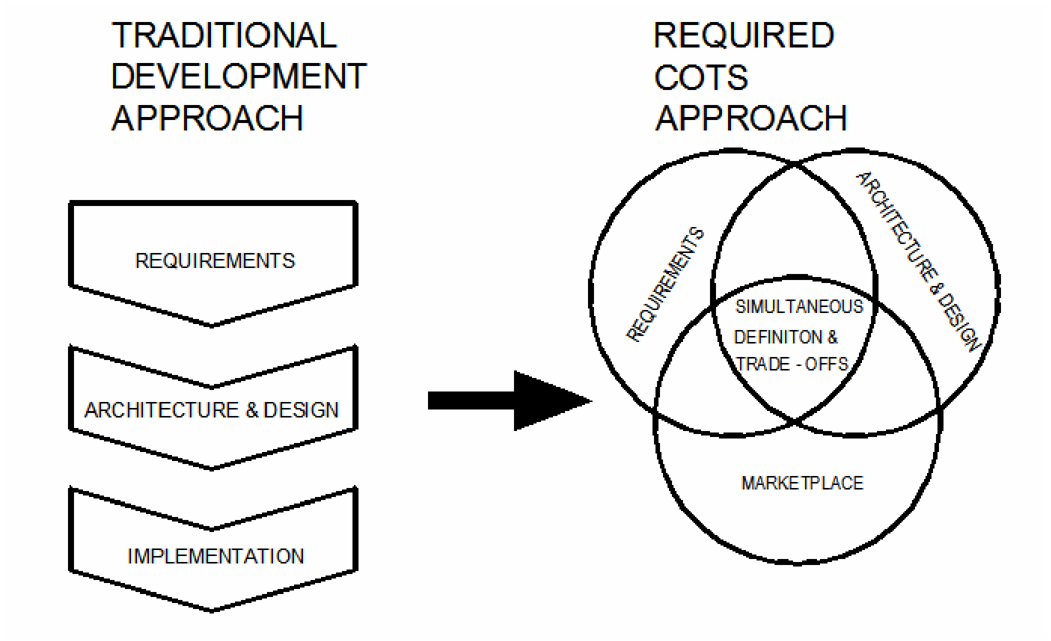

Figure 1.2: COTS Based System Development

The main goal of component based software is to reduce development costs and efforts, while improving the flexibility, reliability, and reusability of the final application due to the (re)use of software components already tested and validated. This approach moves organizations from application *development* to application *assembly*.

Interoperability can be defined as the ability of two or more entities to communicate and cooperate regardless of differences in the implementation language, the execution environment, or the model abstraction.

1.7 Software Architecture and Technologies

The different steps of software system development require one to view the system with respect to several individual perspectives such as those of end-users, analysts, designers and developers. The software architecture encompasses the set of *significant decisions about the organization of a software system* such as:

- selection of the structural elements and their interfaces by which a system is composed, behavior as specified in collaborations among those elements;
- Composition of these structural and behavioral elements into a larger subsystem and architectural style that guides this organization.

Software system history can be distinguished as follows: centralized architecture in 70's, decentralized architecture in 80's, distributed architecture in 90's and web architecture in 2000's. The use of web technologies has allowed more complex functionalities to be offered on the net; from *information publishing to heterogeneous application integration*. Web (enabled/distributed) architecture is based on multi-tier architecture that separates the *presentation, business logic* and *data*. We use the architectural vision to identify and place the different Information Technologies to select for building the system [13]. This approach concerns not only the physical view but also the logical view (*e.g*. code organization, application design.).The basis for web architecture is shown in Figure 1.3.

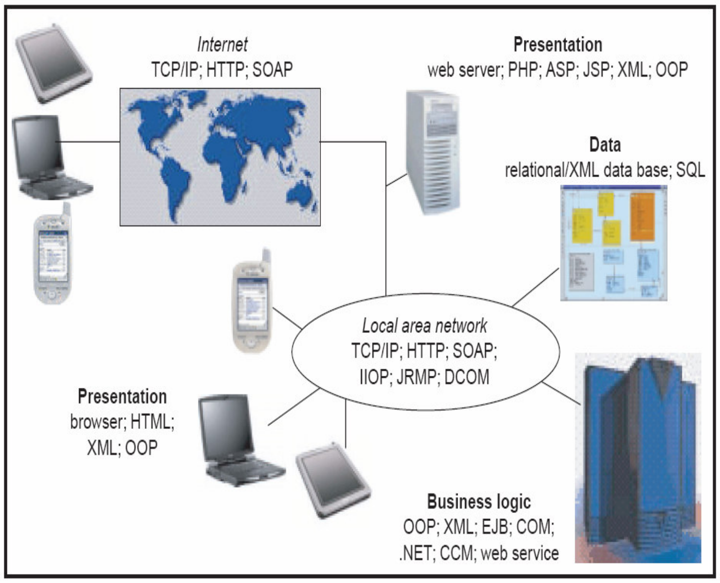

Figure 1.3: Web Architecture Basis

The *presentation tier* allows the graphical interaction with the end-users over the network using a *thin client (*the browser) or a *rich client* (a dedicated GUI). The thin client presentation is performed using web browser-HTML (with script languages and XML if any). The communication with the business logic tier is based on HTTP-TCP/IP. Web dynamic technologies such as PHP, Microsoft ASP and Java JSP work on this principle and these pages are compiled and executed on the web server side to generate just-in-time HTML pages displaying the graphical interface of e-business or other applications. On the other side, the rich client presentation is developed with usual Object-Oriented Programming (OOP) languages such as Java, VB, C++, C#, Delphi, and the communication is carried out by protocols from middleware technology such as CORBA-IIOP, (D)COM, .NET Remoting, Java RMI-JRMP, XML-HTTP and SOAP according to component based programming. The middleware technology is the basic mechanism by which software components transparently make requests to and receive responses from each other on the same machine or across a network.

The business logic tier encloses the application logic representing the enterprise know-how and rules. Usually this side is developed according to a component-based approach with Unified Modeling Language. This approach is based on advanced component models such as EJB from SUN / Java community, COM, DCOM and .NET from Microsoft, CORBA/CCM from OMG or Web services from W3C. These components are mainly implemented following an Object Oriented Programming and component intercommunication is performed by middleware inner protocols. The components run within a software framework called application server that provides a set of technical services such as transaction, identification, load balancing, security, data access, and persistence.

The data tier has the data persistence service with relational, XML and object databases generally using SQL to manage data. In addition to usual data techniques, work is being done on XML-enabled and distributed data management (SQLXML). Table 1.1 categories web architecture into six groups.

Table 1.1: Web Architecture Categorization

Technology	Protocol / Languages	Functions
Modeling Technology	Internet Protocol: TCP/IP,HTTP; XML (data model)	Modeling for object based systems.
Communication technology	Internet Protocol: TCP/IP,HTTP; Internet Inter -ORB Protocol (IIOP) for CORBA; DCOM, .NET Remoting, Java RMI-JRMP (Java Remote Method Protocol), SOAP(Simple Object Access Protocol) , Specific definition programming language such as OMG IDL for CORBA, Microsoft IDL for COM, Java interface for RMI and WSDL for web services.	Data transmission over Internet.
Implementation Technology	Internet Protocol: TCP/IP, HTTP; Internet Inter -ORB Protocol (IIOP) for CORBA; DCOM, .NET Remoting, Java RMI-JRMP (Java Remote Method Protocol), SOAP(Simple Object Access Protocol); Object-oriented programming (Java, C++, Eiffel, C#), Web Programming (HTML, XML, ASP, JSP, PHP, PERL).	Coding using object-oriented web programming languages.
Packaging Technology	Internet Protocol: TCP/IP, HTTP; Internet Inter -ORB Protocol (IIOP) for CORBA; DCOM, .NET Remoting, Java RMI-JRMP (Java Remote Method Protocol), SOAP (Simple Object Access Protocol); EJB, (D)COM, .NET-Programming languages.	This deals with the creation, management and destruction of the business component. With this technology, the component developer no longer needs to write “technical” code that handles

		transactional behavior, security, database connection pooling.
Bridging Technology	Internet Inter -ORB Protocol (IIOP) for CORBA; DCOM, .NET Remoting, Java RMI-JRMP (Java Remote Method Protocol), SOAP(Simple Object Access Protocol); COM-Java RMI, EJB-.NET	Bridging technology allows one to extend the aptitude of a system to interoperate between outer technologies.
Memory Technology	Internet Protocol: TCP/IP, HTTP; relational, object, XML data base, SQL	Provide shared and controlled access to schema information.

1.8 Objectives, Scope and Limitations

The present work is intended to meet the following objectives:

1. Identify the *need* for Semantic Integrity Control and Interoperability in Component Based Software Development.
2. *Design, Implement and Analyze* algorithms to enforce *Semantic Integrity Control and Interoperability for database* search application, using component model.
3. *Analyze* the key design parameters for Domain, Service and Agent Components.
4. *Specify* the design steps for Domain, Service and Agent components.

The component model has been realized using two approaches:

1. Java API and JDBC.
2. ADO. NET Architecture.

This approach can be effectively used for accessing multiple databases in a distributed environment.

The Contracts specification includes four levels, namely, S*yntactic, Behavioral, Synchronization and Quality of Service.* The present work considers only the *Syntactic and Behavioral levels* for contracts

1.9 Research gap

Distributed component-based information systems are becoming one of the major trends in software engineering. Whereas distributed component technologies enable the development of reliable, scalable and secure systems, existing component-based development techniques and methods do not explicitly address distribution. The earlier distribution requirements are considered and integrated with functional requirements; the *least is the risk* that the developed component architecture does not reflect these requirements. UML Components is a widely known CBD method composed of a number of phases targeted at the identification of domain components (system and business), their respective interfaces, and their interaction model to compose the system architecture.

Semantic Integrity Control ensures correct operations over the components. It requires thorough knowledge about the properties of an application. The main problem in supporting automatic semantic integrity control is that the cost of checking assertions can be too high in the case of distributed components.

Design-time interoperability is well within current technological capability for many classes of systems. In run-time interoperability, the constituent systems in a network will be able to support ever-changing demands for service. To meet those demands, the components will continually adapt to new operational contexts. Because the operational context is changing continuously, the developers of those systems cannot know apriori the systems with which they will interoperate. The result is that the difficulty of reaching agreement between developers has been magnified, since agreement can only be reached after the systems have been developed. Since interoperability becomes a run-time issue, it follows that no overall set of agreements can be reached, but that each system must negotiate on a pair-wise basis on the meaning of a particular communication, and do this dynamically, at run-time. Design-time and run-time interoperability exhibit many differences. For instance, design-time interoperable systems achieve their necessary degree of interoperability only by means of tight programmatic control of engineering choices. This approach typically comes at high cost, and involves inflexible agreements about specific requirements (e.g., standards, data semantics, and QoS); very close interaction between the organizations responsible for the systems, and very extensive testing to verify the specific interoperable pathways [14]. The resulting integrations are commonly too inflexible to permit introduction of any new elements into the systems. Also, maintaining such interoperability has its own level of difficulty as system versions change and evolve. More significant to end users, these inflexible integrations limit the users' ability to form ad hoc, creative solutions when necessary.

Interoperability depends to a large extent on common understanding. For two systems to interoperate, hardware pins must align, communication protocols must be consistent, data formats and structure must be understandable, system invocation mechanisms must be shared, and so forth [15]. Yet even with all of the things in place to assure connectivity, there is still no guarantee that either system will be able to the convert signals, bits, and bytes into the information necessary to perform its requisite tasks. Both systems must also make consistent interpretations on the meaning of the data communicated between them; they must exhibit semantic interoperability.

As a trivial example, suppose one system sends the number "5" to another system. What does that communication mean? The answer is that its meaning depends on both systems having agreed that "5" represents a high-priority risk, or that it represents the fifth day of the week, or some other such meaning. In other words, it is necessary to relate the communicated data itself to the meaning of that data. Semantics refers to the meaning of data, providing a way to establish what entities mean with respect to their roles in a system. There is a limited number of ways that agreements on meaning can be achieved. In the context of design-time interoperability, semantic agreements are reached in the same manner as interface agreements between the constituent systems. If the system of systems is a closed system, then the context of those agreements is only that of the system of systems; there is no need for any other entity to share in the agreements, or to understand the implied meaning of the data [16]. However, in the context of run-time interoperability, the situation is more complex, since there is need for some manner of universal agreement, so that a new system can join, adhoc, some other group of systems. The new system must be able to usefully share data and meaning with those other systems, and those other systems must be able to share data and meaning from an unfamiliar new comer.

1. *The present work addresses the Semantic Integrity Control issue for distributed components by providing an environment for defining pre-conditions and post-conditions and enforcing them.*
2. *The present work addresses the distribution issues in early phases of application development, namely requirements and specification. The basic idea is to identify distribution requirements that can be mapped into common distributed services (example: naming, concurrency).*
3. *It also deals with design time and run-time interoperability in a distributed environment.*

1.10 Organization of the Thesis

The thesis is organized into **seven chapters** followed by the **References, Appendices** and **List of Publications**.

Chapter 2 presents a component based approach for software development in a distributed environment for the database retrieval operations.

A *Core Component* for a distributed multi-database system has been proposed. The Core Component is modeled using three interfaces *User, Administrator and Database Handler*. The User Interface is the starting point of access for the Core Component. The Administrator interface deals with access control privileges for users and local databases. The Database Handler facilitates global schema management and site management.

Chapter 3 deals with an *algorithm* for accessing multiple databases in a distributed environment. The implementation involves design of User Interface Component using two approaches:

1. JDBC and Java API.
2. ADO .NET Architecture.

The task carried out here include SQL Query Initiation from client site, Generate Sub-Queries based on Global Schema Information, Execute Sub-Queries in parallel using SQL servers at appropriate sites, assemble the results of the query at the client site and output the query results. The proposed generic approach reduces the effort required to search multiple databases stored at various sites in a network of heterogeneous systems.

Chapter 4 discusses about mathematical representation for SQL SELECT statement using *propositions and predicate* and experimental verification using component based approach. The proposed setup can be very much helpful in verification of simple as well as complex queries, evaluating their correctness and facilitate reliable implementation using database components.

Chapter 5 analyzes the *key factors* responsible for maintaining *semantic integrity control* and *interoperability* in the design of components. The key factors will assist a component designer in understanding the requirements of Domain, Service and Agent based applications.

Chapter 6 presents the design steps involved for three types of components, namely Domain, Service and Agent. The proposed design steps will be helpful to a component designer in developing components pertaining to Domain, Service and Agent applications.

Chapter 7 summarizes the work carried out in this thesis, highlights specific contributions and suggests some directions for future work. The appendices consist of the following information:

- The system configuration.
- Functions and their actions for enforcing semantic integrity control using .NET framework.
- Functions and their actions for enforcing interoperability using Java API and JDBC and .NET Framework.

The relevant articles, books and websites are listed in the **References.**

The **lists of publications** related to the thesis are given at the end.

Chapter 2

Component Approach for Enforcing Semantic Integrity Control in a Distributed Multi-Database System

2.1 Introduction

The Complexity of the software applications has increased considerably over the recent years. There is a growing need for design of software modules that provide reusability and simplify software development efforts. The concept of inheritance is one of the key features for the success of object-oriented programming languages and design methods. Software components are widely applicable to different machines and the users should be available in groups arranged according to precision, robustness, generality and time-space performance. Existing sources of components - manufacturers, software houses, users' groups and algorithm collections - lack the breadth of interest or coherence of purpose to assemble more than one or two members of such groups. Software production in the large would be enormously helped by the availability of spectra of high quality routines, similar to that of mechanical design that is supported by the existence of families of structural shapes, screws or resistors [17].

Component-based software development is gaining importance with the emergence of new frame works and standards. Most of them are focussed on CORBA, COM and JavaBeans. Components are normally specified to the environment through some Interface Definition Language (IDL). The syntactic aspects are usually well-defined and syntactic constructs that are not accessible in all target languages, are not allowed. The semantic aspects of components, however, are not so well supported and require greater attention. In traditional object-oriented programming, the semantic aspects have been a topic of great interest for a long period of time. Since objects and classes have a number of properties in common with components, most of the ideas developed in traditional and object-oriented programming theory will be usable in component-based development also. This chapter focuses on Component-based approach to software development for a Multi-database application. The pre-conditions and post-conditions are enforced using Assertion Manager in the Core Component of the Multi-database system.

A Distributed Multi-Database system involves storage and retrieval of information from independent databases that are spread over multiple sites in a computer network. Each database can be implemented using different DBMS and different architectures that distribute the execution of transactions. The DDBMS gives a unified view of the entire databases to the user [18]. A distribute multi-database system offers scalability and reliability in modern information systems.

2.2 Related Work

The following properties are in general articulated for components:

1. Components may be constructed by third-party vendors.
2. Components may be used in varying application environments.
3. Component source code may not be available for evaluation.
4. Components may be distributed.
5. Component may be used from within different programming environments.

However, in practice, all the above five properties need not be present in a given component-oriented environment. Investigations have shown that most successful component-projects have been done in-house [19], a fact that eliminates the first property listed above. Some common components, such as plug-ins, can only be used with certain applications in certain operating systems. This eliminates the second point and is another example of how difficult properties of component usage are simplified or removed. Even if difficulties may be avoided by reducing some of the problems above, the need to describe semantic properties properly and to preserve the semantic integrity is imperative in component-based development. By the semantic integrity of a software system, we mean that each part of the system respects the intended purpose of any other individual part of that software system. This is a condition to build stable systems and requires that each part be clearly described and that description be maintained as that part evolves.

Substantial amount of work have been done on component based distributed database system over the past few decades and the complexities usually inherent in a distributed database are hidden from the user [20]. The idea of Software Engineering that encourages decomposition [21] of a system into logical units improves the scalability of the system, and it is very much applicable to multi-database system.

2.3 Component Design for a Multi-Database System

The component for the multi-database system is designed to support three main types of interfacing entities. They are Component User, Component Administrator and Database Handler as shown in Figure 2.1. These interfaces can be used for interaction with the other components of the system. Such a design permits the separation of functionality, user interface and data layer from each other. User Interface components from peer systems can interface and interact with the core component concurrently. The separation of User Interface from the core component has an added advantage of being able to connect to Multiple Distributed Database Components.

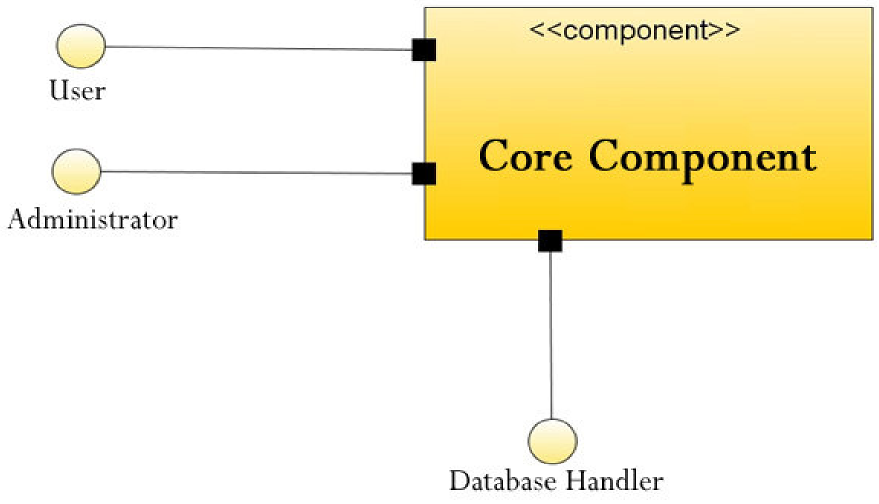

Figure 2.1: Core Component for a Multi-Database System

2.3.1 Component User

The Component User interface acts as an entry point to the core component of a multi-database system. An example of the Component User could be a web based application that provides the user interface to accept a query and then display its result.

2.3.2 Component Administrator

The Component Administrator interface is used to configure, create and maintain local databases. This interface has been separated from the normal User Interface in an attempt to increase the security of the system. The

Component Administrator sets access control privileges for users and local databases.

2.3.3 Database Handler

The Database Handler interface performs the following functions:

- Global Schema Management.
- Site Management.
- Facilitate query execution across various components.
- Managing Assertions for Components.

2.4 Problem Description

The Core Component for Multi Database System (shown in Figure 2.1) performs semantic integrity control and retrieval of information using Database Handler. The Database Handler Component shown in Figure 2.2 consists of the following sub components such as Schema Manager, Assertion Manager, Site Manager, and Execution Manager. The user submits a query to the Core Component through the User interface and the Core Component further connects to the Database handler for retrieval of data by connecting to the appropriate site.

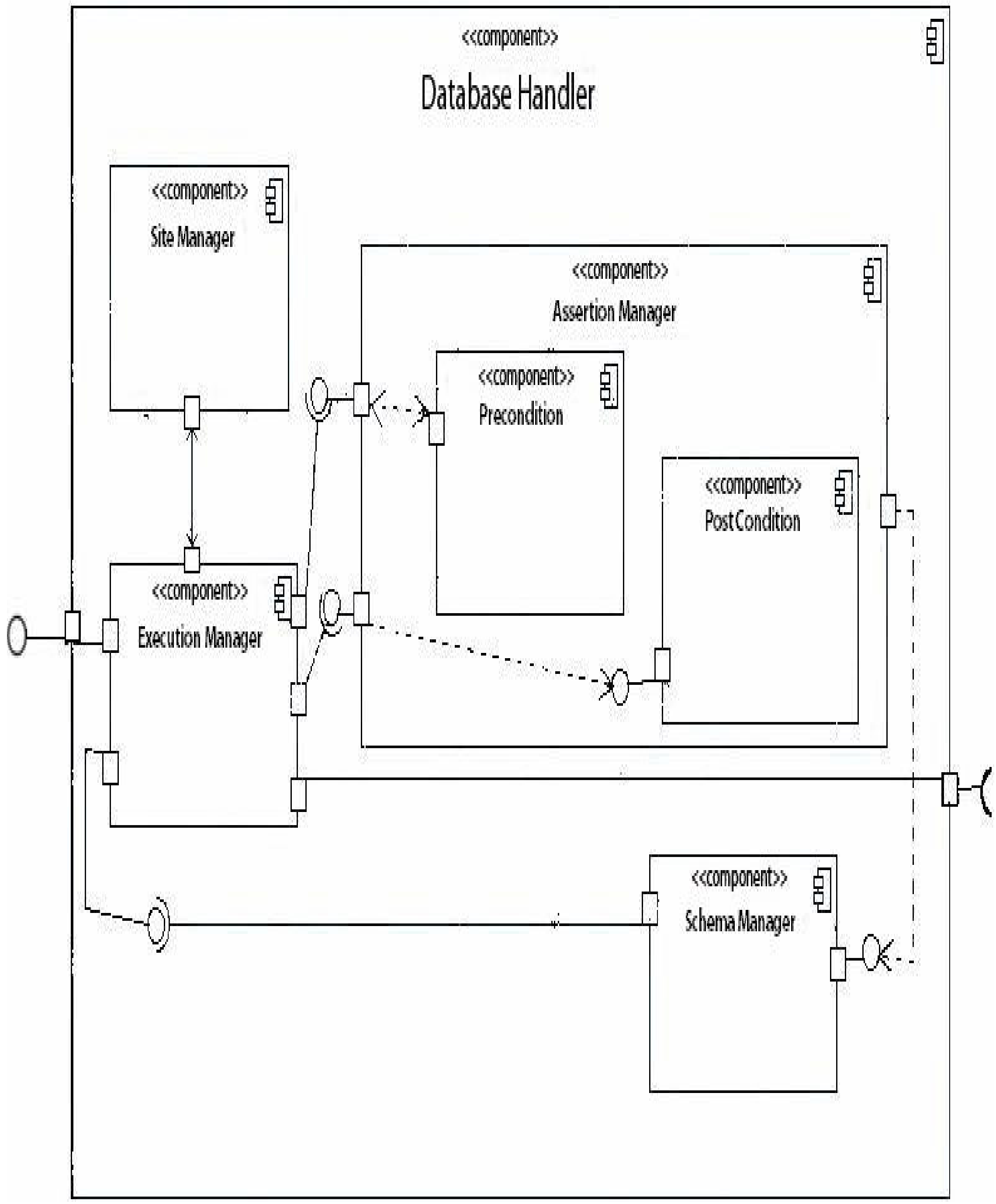

Figure 2. 2: Block Schematic for Database Handler using UML 2.0

2.5 Database Handler

The Database Handler has the following sub-components:

1. Schema Manager.
2. Site Manager.
3. Execution Manager.
4. Assertion Manager.

The following subsection describes their functions in detail.

2.5.1 Schema Manager

The Schema manager is used to store the Global Schema of the Multi-Database system as viewed by the component's users. All the queries handled by the Multi-Database system are designed for this global conceptual schema. The schema manager checks the query for any semantic errors that arise due to representational differences and naming conflicts. It resolves semantic conflicts by providing an explicit translation process. The resolution of the conflict is made possible with the help of an integrated global schema. Information systems often contain components that are based on different schemas of the same or intersecting domains. These different schemas of related domains are described in meta-models that fit certain requirements of the components such as representation power of tractability [21]. For instance, a database may use SQL or an object oriented modeling language. A web service described in XML schema may be enriched with semantics of the domain. All these different types of schemas have to be connected by mappings stating how the data represented in one schema is related to the data represented in another schema. Integrating these heterogeneous schemas requires different means of manipulation for schemas and mappings and the Schema Manager perform this task. It should provide operators such as match that computes a mapping between two schemas. An important issue in a schema management system is the representation of mappings which can be categorized as intentional mapping and extensional mappings [22, 23]. Intentional mappings deal with the proposed semantics of a schema and are used, for example, in schema integration. Extensional mapping is used if the task is data translation or data integration.

2.5.2 Site Manager

The Site Manager is made up of five sub components as shown in figure 2.3:

a) Site Handler.

b) Fragmentation Manager.

c) Replication Manager.

d) Failure Management.

e) Schema Map.

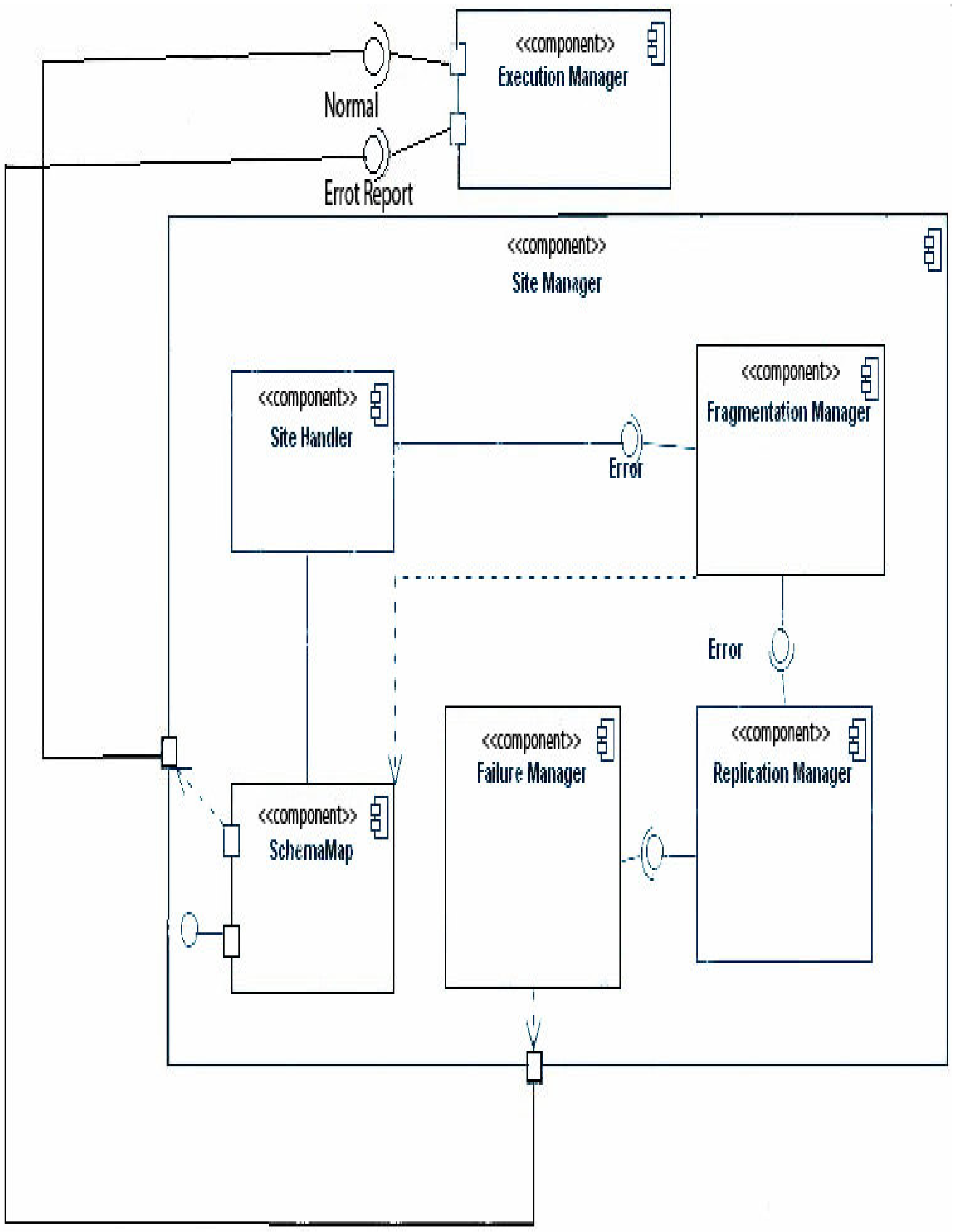

Figure 2.3: Block Schematic for Site Manager using UML 2.0

a) Site Handler

The Site Handler manages connections to the sites of the individual databases and execution of sub queries at a given site. It provides an interface to query the various local sites [24]. Furthermore there is a separate component called the Failure Management component that is responsible for managing temporary as well as permanent failures of any database sites that make up the distributed multi-database system. The same component is also responsible for updating a site that has come online after temporarily becoming offline or unavailable.

b) Fragmentation Manager

The individual databases in a Multi-database system may be partitioned horizontally or vertically based on primary key values. This sub component utilizes the Schema Map to keep track of the multi-database system and constantly updates its log. The schema manager helps in translating query for the global schema to a query for the local schema [25]. This translation is done with the help of the Fragmentation Manager and Replication manager.

c) Replication Manager

The Replication Manager subcomponent manages multiple copies of important database fragments of a multi-database system. It provides an interface to the failure manager. This component is used when a local query fails to execute due to an error (either with the connection or with the site itself.). Some important fragments of the multi-database can be replicated for ensuring availability [26].

d) Failure Management

This component is invoked by the Site Handler when an error is encountered. The component checks with the Schema Map, Replication Manager and Fragmentation Manager to see if the failed query can be executed without compromising the integrity of the multi-database system. It generates new localized queries whenever there are alternatives.

e) Schema Map

The Schema Map sub component provides an interface to the Execution Manager. It plays an important role in converting the global schema [22] based queries into localized queries and passing it on the site handler where the local queries are executed and their results passed on back to the Schema Map [27].

2.5.3 Execution Manager

The Execution Manager provides an interface for the external components to pass on queries to the Database Handler Component. The Execution Manager initially accepts the query from the uniform user interface and then utilizes the Assertion Manager's interface to check the preconditions. After confirming that the preconditions have been satisfied, the Execution manager passes on the query to the Site Manager. The returned results are then processed for any post conditions as applicable and then passed on to the external interface that originally submitted the query. The Site Manager returns the query result to Execution Manager.

2.5.4 Assertion Manager

The Assertion Manager enforces the pre-conditions of the input query involving information retrieval from multi-database system. The post-conditions include connection status, error types, log files related to retrieval operations for each query. The error occurred during the process of information retrieval can be syntactic error, semantic errors. The syntax error arises due to incorrect query specification. The semantic error can be categorized as follows:

1. Errors arising due to naming conflicts [28].
2. Errors arising due to policy conflicts [28].

The Assertion manager checks the query for errors arising out of naming conflicts and representational differences are handled by Schema Manager. The errors arising out of differences in policies and privileges of the multi-database systems are handled by Access Control unit.

"An assertion is a statement containing a Boolean expression that the programmer believes to be true at the time the statement is executed" [29]. In other words this means a facility provided within component to test the correctness or assumptions made by the components for some purpose. Assertions are checks provided within the system to ensure the smooth running of the program. Java exceptions are primarily used to handle unusual conditions arising during program execution. Assertions are used to specify conditions that a programmer assumes to be true. “When programming, if a programmer can swear that the value being passed into a particular method is positive no matter what a calling client passes, it can be documented using an assertion to state it” [30]. Exceptions handle abnormal conditions arising in the course of the program; however they do not guarantee smooth or correct execution of the program. Assertions help state scenarios that ensure the program is running smoothly. Assertions can be efficient tools to ensure correct execution of a program. Assertions are not to replace inputs but to augment them.

2.6 Pre-conditions

The asserting statement undergoes check for preconditions. Preconditions [31] are values or parameters passed to a method, to be used for the functioning of the program. Assert statements can be used to check the validity of the parameters passed before they get used in the body of the method. From server point of view, preconditions express the requirements that clients must satisfy whenever they call a component's code, and are therefore evaluated at their entry point. The pre-conditions can be defined, stored and managed by the component administrator. The user can also have the option of specifying the pre-conditions through "Having Clause" or SET FILTER to command in SQL. The pre-conditions can be read from input file and passed to component administrator. Pre-Conditions can also involve conditional logic using <and> <or> <not> operators. The global schema is consulted while checking the correctness of names relating to DBMS, tables, attributes and key constraints.

2.6.1 ALGORITHM: PRE-CONDITION

The algorithm given below checks all the pre-conditions stored in the input file and enforces them. This can be accomplished by running a code that detects all the key words in SQL, and in the present work SELECT statement has been considered. A proper input is from the starting character of document to the second SQL keyword found in the input. This part is sent to the pre-condition routine to check for the correctness and the algorithm given below checks the syntax of the statement.

Algorithm Precond*(x, indx, indxa)*

INPUT:

x: The input string given by the user, corresponding to the SQL query.

indx: This is of the type integer. It refers to the current processing point of input statement *x.* Its default value is 0.

indxa: This is of type integer. It refers to the current processing point of actual statement *a.* Its default value is 0.

Internal Variables:

a: This is a string variable. It represents a symbol table. It stores the syntax for the required SQL statement. While parsing the input, the tokens are matched with the entries in "*a*".

i: This is of type integer. It refers to the position pointer of the user statement.

ia : It is position pointer for the actual syntax stored in *"a"*.

c: This is a temporary string variable.

z: This is a temporary integer variable.

OUTPUT : Boolean output stating whether the syntax is correct.

I. Read the input query and pre-conditions from the input file and pass them to Component Administrator.

II. Perform parsing for the input query as shown in steps from 1 to 2.4.

```
1.    a <-
  "SELECT$[*|column_name1,column_name2,....]$FROM$tablename1[,tablename2,...]$[WHERE$condition_and|or_condition...]$[GROUP BY$column-list]$[HAVING$""conditions""]$[ORDER BY$""column-list"" ASC|DESC]$"

action: Symbol Table for the instructions. The array with which the input
        symbols should be checked. a has '$' as the key to separate between
        spaces,'['&']' to hold the optional parameters.
      {
2.    while(x==NULL)
action: Loop till the Lexer tokens are empty. i.e Loop untill the whole input
        statement is checked completely.
      {
2.1   i <- next positon of $ in x
action: Counter for input strings.Lexer pointer for the input string.
```

```
2.2     ia <- next positon of $ in a
action: Counter for the array a.Lexer pointer for the string to be checked in
            the symbol table.

2.3       If a[0]=="[" Then
action: Compare the symbol table for its attribute of an optional keyword.

     {
2.3.1           indxa <- indxa + 1
action: Get the next lexer token from Symbol table.This is done because it
          reaches the actual strings to be compared(the position of '[' is
          passed.
          }
2.4       If i>=indxa Then
action: Compare lexer's token of the input and the symbol table for input.
{
2.4.1           length <- ia-indxa
action: Calculate the length of the actual string in the symbol table so that it
          is compared for the completeness.

2.4.2          If length>=14 and x[i] =' ' Then
action: If the length is greater than 14(minimum required length for select
          statements) and the position after the 14th position is empty.

      {
2.4.2.1           If the substring of x between indx and length = The substring
                  of a between indxa and length Then
action: if the above condition is true then it will compare the subsrting
          between lexer pointer of input string and the current pointing
          position with the symbol table elements.

                  {
```

```
2.4.2.2                 indx <- next positon of $ in x + 1
action: if the above comparison is true then go to the next lexer token in the
        input.

2.4.2.3           indxa <- next positon of $ in a + 1
action: Go to the next token in the symbol table.


                  If x[indx+1]=NULL Then
    action: Check if the next position is the end of the input string.

                          return true
action: if yes then "INPUT STRING IS MATCHED" RETURN  A TRUE
         WORD.

          Else
                     precond(x, indx, indxa)
action: if no then call the same function to check the next tokens.

                End If
      }
2.4.2.4           ElseIf length of x is less than 14 Then
      Print error stating that "MINIMUM SYNTAX INPUT FAIL"
action: If the length of input doesn’t exceeds 14 then print an error "INPUT
        IS NOT COMPLETE".
      }
2.4.2.5            Else
action: If the position after the 14th position is not empty then check if
        symbol table has more elements to be checked.
      {
                If indxa > 0 Then
```

```
action: if the symbol table has terms to be checked then check if it is an
        optional condition.

        {
                If substring of a between indxa-1 and indxa is "[" Then
                   precond(x, indx, next index of "[" in a)
action: If it is an optional entity then call the function again by giving the
        symbol  table position pointer as the point of optional entity.

    end if
                BREAK outside the loop.
action: if the above condition fails then Break outside the loop.
             }
             End If

2.4.2.6            Print error at substring (a, 0, ia)
action: Print error that "SYNTAX IS WRONG" position can be mapped by
        giving the substring form start to the lexer pointer of symbol table in
        symbol table.
          }
        Else
2.4.3           print error No proper syntax.
Action:  if the input is empty or doesn't  follow any sequence then print
        error "INPUT IS INVALID"
         }
 }
}
```

substring(X,pos1,pos2)

INPUT: String of characters from which substring is extracted.POS1 position 1 where the extraction should start. POS2 position 2 where the extraction should end.

```
Output: x1 substring of X.
if (pos1<=pos2) then
    s1=X[p1-1]
    substring(X,pos1+1,pos2)
return s1
```

III. Perform lookup in the Global schema to validate database names, table names, attributes and key constraints.

IV. The Assertion Manager transfers control to the Execution Manager.

Complexity:

The Time Complexity is determined by the total number of input pre-conditions (say n) and can be expressed as O(n).

The above algorithm is used for checking the precondition using assertion manager. It starts with the *precond* class which is the class name of the precondition component which takes 3 inputs. The first input corresponds to input query entered by the user. The second and third inputs are set to 0 by default. Here the output will be a boolean value indicating whether the user query is valid and permissible. Finally, the Assertion Manager, transfers control back to the Execution Manger.

2.7 Post-conditions

Just like preconditions, there may be instances where a program needs to execute some post-conditions. Post-conditions need to be evaluated before each exit point in the method. For instance assert statements can be used to check for the validity of the returned values in a method that has multiple *return* statements. Similarly in Web based architecture post conditions inform about what the supplier (i.e. the component's code) guarantees on return, if the precondition has been satisfied on entry. They have to be evaluated at all exit points of the component's code. The server is required to execute the post conditions and client can infer useful information upon execution of post-conditions.

After the successful execution of the user query, the post-condition algorithm returns the following information: DBMS type (MySQL, POSTGRES, MSSQL), Running as user <user-name>, total number of records accessed from all databases stored at the various sites.

2.7.1 ALGORITHM: POST-CONDITION

INPUT: Query Processing at the server.

OUTPUT: log file and the client information after query execution.

```
    1. Connect to the databases using standard OLEDB connections.
    2. IF time out response or error returned from any database.
                Print database / table with given name doesn't exist
                Open the log file (post.log)
                Write to file "current date: ERROR database / table name
                  doesn't exist"
                  ELSE
                      Mount the data on the grid view
                      Open the log file (post. log)
```

```
            Write to file "current date: SUCCESS default
              statement entered.
              print summary information on user name, names of
              databases and total number of records accessed.
3. Exit.
```

2.8 Implementation:

The pre-conditions and post-conditions have been implemented using .NET Component based code. Appendix B lists the functions used in .NET code along with their actions.

2.9 Summary:

This Chapter dealt in detail with a component based approach for enforcing semantic integrity control in a distributed multi-database system. The core component design includes separate interfaces for User, Administrator and Database Handler. The Database Handler includes the sub-components Schema Manager, Site Manager, Execution Manager and Assertion Manger. The pre-conditions and post-conditions are handled by Assertion Manager and are implemented using .NET component based code.

Chapter 3

Interoperability of Multiple Databases in Distributed Environment

3.1 Introduction

A database application requires an access to multiple databases and each database has several views and tables [32, 33]. Whenever one wants to search for information in multiple databases, he/she have to do so using the front end of the particular database server. The user has to spend considerable amount of time and effort in formulation and execution of several queries over multiple databases located in one or more servers. The proposed work simplifies the above two tasks by using any one of the following two approaches:

(1) JDBC and Java API and

(2) ADO .NET Architecture.

Each database in the network reflects the editorial scope and interest of one particular publisher or institution. It is desirable to conduct a comprehensive search of as many relevant sources as possible. To do so will probably require a search of more than one database. It is certainly possible to search each database one-by-one, which is time consuming. However, the uniform interface designed in this present work extends the search to multiple databases spread over the network. The need for searching multiple databases at one time is to ensure broadest coverage of relevant material, save time, save search costs, integrate results into a single search, identify and remove duplicate records across databases.

3.2 Objectives

The present work is intended to meet the following objectives:

- Design of an efficient and interactive user interface that would accept valid SQL query (over multiple databases stored in one or more sites) as input from end user.
- Provide an environment for effective query execution in a distributed environment.
- Allow the end user to view/print the output of the query.

3.3 Related Work

The universal relation as a user interface has been dealt in greater detail in [34]. Here, the data of the entire database are modeled using a single relation. The schema of the universal relation encompasses all the attributes in any of the relation schemes of the database. The advantage of this scheme is that an user need not remember the details of all attributes and their groupings in each relation.

The proposed work considers multiple databases spread over one or more sites in a heterogeneous network. Hence, it is necessary to have efficient Schema Management and Search Techniques to retrieve information in the above mentioned environment and a general approach to database search has been presented in this chapter.

3.4 Problem Description

A generic model for database search application is shown in Figure 3.1 and Section 3.5. The user enters the query and passes it to the GS (Generic Search) module. This module consults global schema which contains the following information:

Site Identifier (IP Address), details of all relations and their respective databases under which they are present, names of DBMS software supported and the operating system environment. The Global Schema is present at all the client sites where the queries are initiated. The GS module maps the relation(s) name(s) in the query with the Database Name and Site Identifier and generates sub-query for each site. The sub-queries are dispatched to the appropriate sites. It then sets up the query for execution by the appropriate SQL server at each site and assembles the results at the client [33]. The GS module is implemented using the following two approaches JAVA code supported with JAVA API and JDBC and ADO .NET Architecture. For experimental setup, three sites have been considered, although this model is applicable to n number of sites.

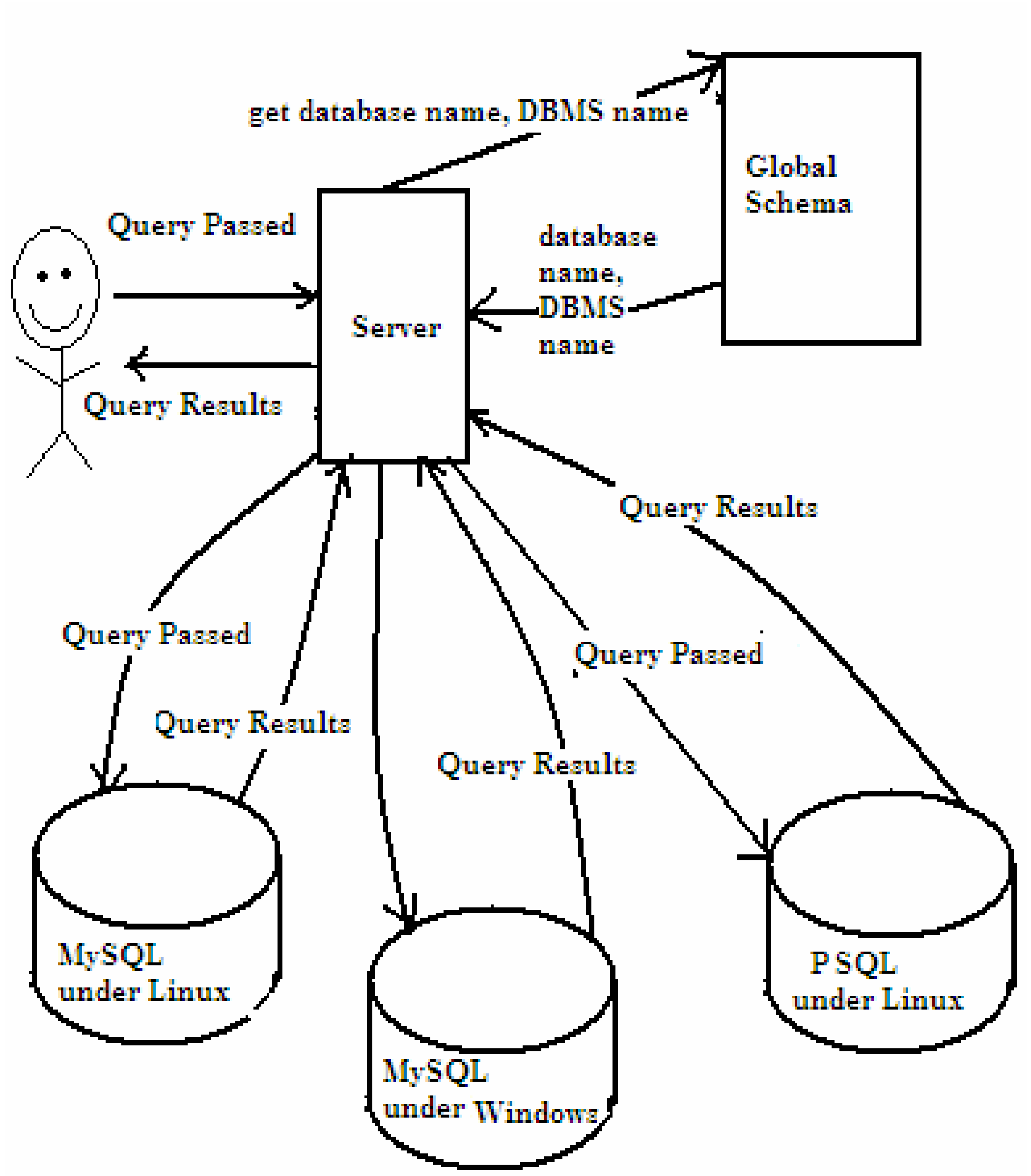

Figure 3.1: Block Schematic for Database Search in a GENERIC MODEL

3.5 GENERIC ALGORITHM

The generic algorithm works as follows:

Algorithm: GS (Generic Search)

Input: A text file containing SQL Query initiated from a client site, Global Schema.

Output: The results of the query (in tabular form) at the client site.

Procedure:

1. The USER enters the SQL query using the UNIFORM INTERFACE.
2. The query is stored in a text file and is checked for correct Syntax using preconditions and the **relation names** & **attribute names** are validated from the **Global Schema**.
3. Open *Global Schema* containing the *database names & relations* under them, *Site Identifier* information, OS name and DBMS name [POSTGRES/MYSQL/MS ACCESS].
4. Associate the names corresponding to *attributes, relations & database* present in the query with the *Global Schema* information.
5. Generate Sub-Query for each site.
6. Dispatch the Sub-Query to the appropriate Site and Connect to the appropriate Database Server at each site, using JDBC & Java API / ADO .NET Architecture.
7. The SQL SERVERS execute the sub-queries in parallel at each site.

8. Assemble the query results in an output file at the client site.
9. The output file at the client site can be viewed as well as printed progressively.

The above algorithm has been implemented using two approaches:

1. The JAVA code supported with JDBC and JAVA API and
2. .NET code supported with ADO.NET.

3.6 Implementing Using JDBC and Java API

The JDBC API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases. The JDBC API provides a call-level API for SQL-based database access. JDBC technology allows you to use the Java programming language to exploit "Write Once, Run Anywhere" capabilities for applications that require access to enterprise data.

The user will enter the query in the space provided in the interface. The program will search for the occurrence of the clause “from”. Extract the table name which follows “from” in the query. The back end server will open the text file and locate the Database Name and Database Server. The JDBC driver will connect to the appropriate database server and database under the server as shown in Figure 3.2. The query undergoes execution in the database server(s) [35] and assembles the results, in a text file at the Client Site. The output generated in this text file can be viewed by the end user.

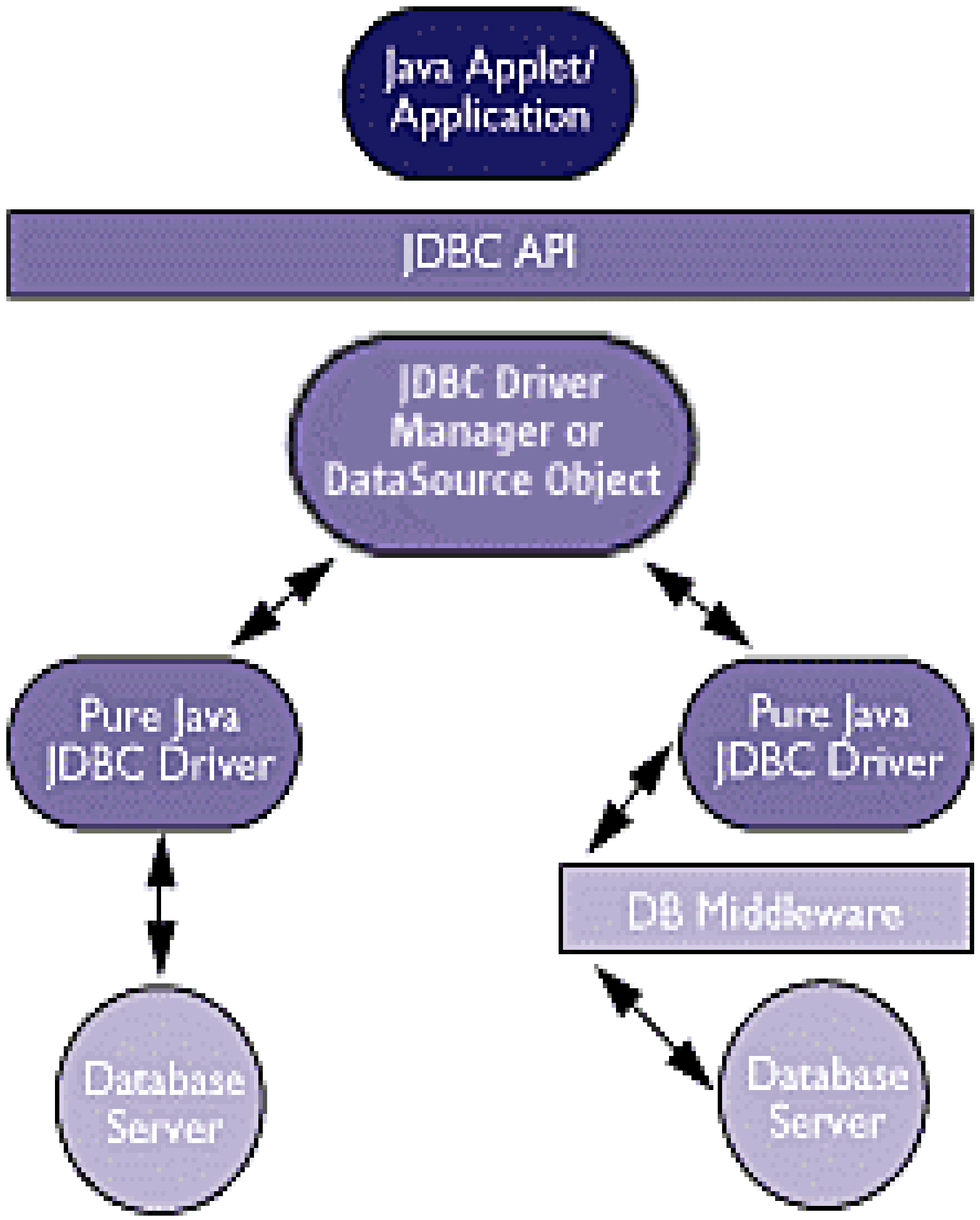

Figure 3.2: Database Access using JDBC Driver

3.6.1 Experimental Setup

POSTGRES, MySQL and MS ACCESS are the DBMS software's that have been considered here.

MySQL [36] and PSQL (POSTGRES SQL) [37] are available on both Linux and Windows Platforms. They are widely used for developing database applications over the web. LINUX and WINDOWS OS have been chosen since both of them are widely used for running many database applications.

The Relational Model for Databases has been considered in this work. The Queries are assumed to be in standard SQL format and are submitted from query files.

Examples used to illustrate this chapter are drawn from the following three databases. Each database has three relations. Each Database is stored at a separate site.

1. MySQL Database under LINUX : library1

   Site 1:- IP Address: 172.16.12.1 (on the intranet)

   The schema of all the tables in the above database is as follows:

   lib_journal (<u>ISBN,</u> Publisher, title, no_of_issues, subprice)

   lib_cse (<u>ISBN</u>, Publisher, Author, title)

   lib_catalog (<u>ISBN</u>, Author, Title, Year, subprice, accno)

   (The primary key field is underlined in each relation.)

2. MySQL Database under WINDOWS: library2

   Site 2: IP Address: 172.16.12.22 (on the intranet)

   The schema of all the tables in the above database is as follows:

   lib_journal(<u>ISBN</u>, Publisher, title, no_of_issues, subprice)

   lib_cse(<u>ISBN</u>, Publisher, Author, title)

   lib_catalog(<u>ISBN</u>, Author, Title, Year, subprice, accno)

   library1 & library2 are horizontally partitioned databases on the primary key: ISBN.

3. PSQL Database under LINUX: student

Site 3:- IP Address: 172.16.12.3 (on the intranet)

The schema of all the tables in the above database is as follows:

Stud_master (<u>idno</u>, name, address, reg_date)

Stud_tran1 (<u>idno</u>, cgpa)

Stud_tran2 (<u>idno</u>,Courseid,t1,t2,q1,q2,ce,total,accno)

3.6.2 Database Creation

The following commands were used to create tables under the library1 & library2 databases used for the generic search over java interface:

```
CREATE TABLE lib_journal (
        ISBN int(10) NOT NULL Primary Key,
        Publisher char(20),
        Title char(25),
        Noofissues varchar(10),
        Subprice real);

CREATE TABLE lib_cse(
        ISBN int(10) NOT NULL,
        Publisher char(10),
        Author char(10),
        Title char(25),
        CONSTRAINT ISBN_ID FOREIGN
        KEY(ISBN)REFERENCES lib_journal (ISBN));
```

```
CREATE TABLE lib_catalog (
        ISBN int(10) NOT NULL,
        Discipline char(10),
        Author char(20),
        Title char(25),
        Year int,
        Subprice real,
        accno char(7),
        CONSTRAINT ISBN_ID2 FOREIGN KEY (ISBN)
        REFERENCES lib_journal (ISBN));
```

The following commands were used to create relations under the student database:

```
CREATE TABLE stud_master (
        id int primary key,
        name varchar(20),
        address varchar(20),
        reg_date varchar(30));

CREATE TABLE stud_tran2 (
        idno  varchar,
        courseid  varchar,
        t1  real,
        t2  real,
        q1  real,
        q2  real,
        ce  real,
        total  real,
        accno  char(7),
        Foreign key (idno) references stud_master);
CREATE TABLE stud_tran1 (
```

idno varchar,
cgpa real,
Foreign key (idno) references stud_master);

3.6.3 Interface Design

The UNIFORM INTERFACE as shown in Figure 3.3 allows the user to enter the query. The interface has been implemented using JAVA API [38] and connection to database is achieved by locating the JDBC driver [39, 40]. Once the user has entered the query and has clicked the "Execute Query" button, the back end will find out the database server and connect to it. The SQL query is then executed and the user can view the query as well as its output results.

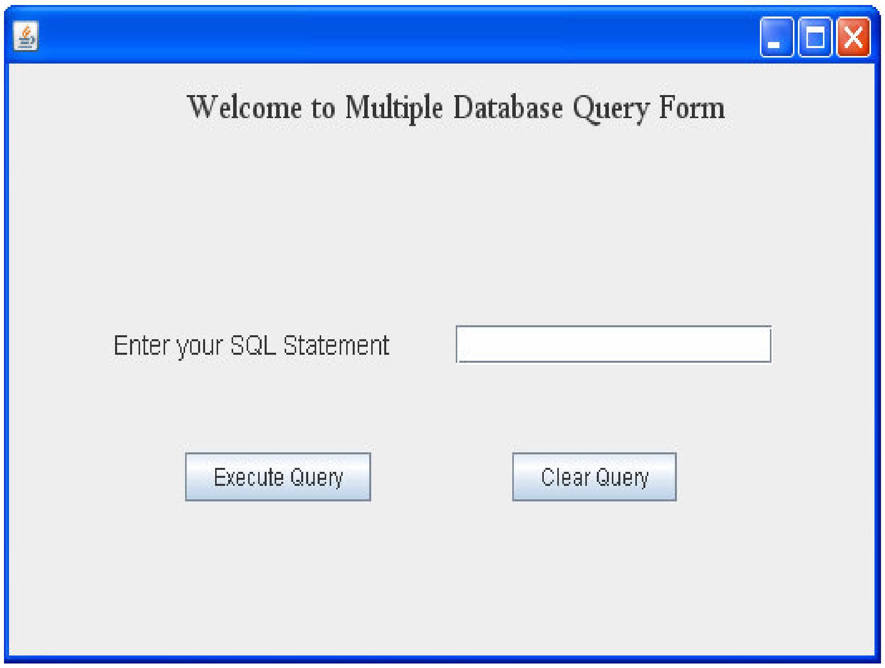

Figure 3.3: Layout of the UNIFORM INTERFACE implemented using JAVA API

3.6.4 Test Scenario

The JAVA program for the present work has been compiled and run using the following sequence of two steps:

```
$ javac Mainclass.java // to create the class file
$ java Mainclass // for execution
```

a) The following query has been entered in the input text file (in1.qry) using the uniform interface:

```
select stud_master.name, stud_tran2.t1, stud_tran2.t2, stud_tran2.q1, stud_tran2.q2 from stud_master, stud_tran2 where stud_master.idno=stud_tran2.idno;
```

A complete trace of the java, program's execution is shown as follows:

Access details:

Site(s)	:	172.16.12.3
DBMS(s) & OS(s):		PSQL under LINUX
Database(s)	:	student

Table 3.1: Results of Sample Query (a).

Name	**T1**	**T2**	**Q1**	**Q2**
Aquarius	45	65	7	6
Venus	67	43	9	9
Ruby	60	66	9	9
……..	……..	…….	…….	……..

b) The following query has been entered in the input text file (in2.qry) using the uniform interface:

select stud_tran2.idno, stud_tran2.accno, lib_catalog.author, lib_catalog.title from stud_tran2, lib_catalog where stud_tran2.accno = lib_catalog.accno;

A complete trace of the java, .NET program's execution is shown as follows:
Access details:

Site(s)	:	172.16.12.1, 172.16.12.2, 172.16.12.3
DBMS(s) & OS(s)	:	MYSQL under LINUX, MYSQL under WINDOWS, PSQL under LINUX.
Database(s)	:	student, library1, library2

Table3.2: Results of Sample Query (b)

IDNO	**ACCNO**	**AUTHOR**	**TITLE**
2002u7ps035	B32562	J. Ullman	Database Systems
2002u7ps023	C40023	Fred Halsall	Multimedia communications
……..	……..	……..	…….
……..	……..	…….	…….

3.7 Implementing Using ADO .NET Architecture

A GS module is implemented using ADO.NET Architecture as shown in figure 3.4. ADO.NET provides consistent access to data sources such as Microsoft SQL Server, as well as data sources exposed through OLE DB and XML. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data. ADO.NET cleanly factors data access from data manipulation into discrete components that can be used separately or in cycle. ADO.NET includes: .NET Framework data providers for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, or placed in an ADO.NET DataSet object in order to be exposed to the user in an ad-hoc manner, combined with data from multiple sources, or remotely send between tiers. The ADO.NET DataSet object can also be used independently of a .NET Framework data provider to manage data local to the application or sourced from XML.

Data Access in ADO.NET relies on two components:

- DataSet and
- Data Provider.

1. **DataSet** [41]

The *DataSet* is a memory-resident representation of data that provides a consistent relational programming model apart from of the data source. It can be considered as a local copy of the relevant portions of the database. The *DataSet* is persisted in memory and the data in it can be manipulated and updated independent of the database. When the use of this *DataSet* is finished, changes can be made back to the central database for updating. The data in *DataSet* can be loaded from any valid data source like Microsoft SQL server database, an Oracle database or from a Microsoft Access database. The elements of DataSet are:

- Data Table Collection.
- DataRelation Collection.

Data Table Collection [41]

The *DataTableCollection* contains all the *DataTable* objects in a *DataSet***.** The *DataTable* is a central object in the ADO.NET library. When accessing *DataTable* objects, they are conditionally case-sensitive. For example, if one *DataTable* is named "mydatatable" and another is named "Mydatatable", a string used to search for one of the tables is regarded as case sensitive.

DataRow Collection [41]

The *DataRowCollection* represents the actual *DataRow* objects in the *DataTable***.** The *DataRow* and *DataColumn* objects are primary components of a *DataTable***.** To insert, delete and update the values in the *DataTable***,** *DataRow* object is used.

DataColumn Collection [41]

The *DataColumnCollection* defines the schema of a *DataTable*, and determines what kind of data each *DataColumn* can contain. The *DataColumnCollection* can be accessed through the Columns property of the *DataTable* object. The *DataColumnCollection* uses the Add and Remove methods to insert and delete *DataColumn* objects.

Constraint Collection [41]

The *ConstraintCollection* is accessed through the DataTable.Constraints property. The *ConstraintCollection* can contain both UniqueConstraint and ForeignKeyConstraint objects for the *DataTable*. A *UniqueConstraint* object makes sure that data in a specific column is always unique to preserve the data integrity. The *ForeignKeyConstraint* determines what will occur in related tables when data in the *DataTable* is either updated or deleted. For example, if a row is deleted, the *ForeignKeyConstraint* will determine whether the related rows are also deleted, or some other course of action.

DataRelation Collection [41]

A *DataRelationCollection* object enables navigation between related parent/child DataTable objects. The *DataRelationCollection* can be created by defining it as a property of either the *DataSet* or the *DataTable*.

2. Data Provider [41]

The Data Provider is responsible for providing and maintaining the connection to the database and it is shown in table. A *DataProvider* is a set of related components that work together to provide data in an efficient and performance driven manner.

The four core objects of .NET Framework Data Provider are as follows:

Table: 3.3: Core Objects of .NET framework Data Provider

Object	**Description**
Connection	Establishes a connection to a specific data source. The base class for all *Connection* objects is the *DbConnection* class.
Command	Executes a command against a data source. Exposes *Parameters* and can execute within the scope of a *Transaction* from a *Connection.* The base class for all *Command* objects is the *DbCommand* class.
DataReader	Reads a forward-only, read-only stream of data from a data source. The base class for all *DataReader* objects is the *DbDataReader* class.
DataAdapter	Populates a *DataSet* and resolves updates with the data source. The base class for all *DataAdapter* objects is the *DbDataAdapter* class.

The .NET Framework currently comes with two *DataProviders*: the SQL Data Provider which is designed only to work with Microsoft's SQL Server 7.0 or later and the OLEDB DataProvider which allows us to connect to other types of databases like Access and Oracle.

Data access with ADO.NET can be summarized as follows:

A connection object establishes the connection for the application with the database. The command object provides direct execution of the command to the database. If the command returns more than a single value, the command object returns a DataReader to provide the data. Alternatively, the DataAdapter can be used to fill the Dataset object. The database can be updated using the command object or the DataAdapter. The DataAdapter comprises of Select, Insert, Delete and Update modules.

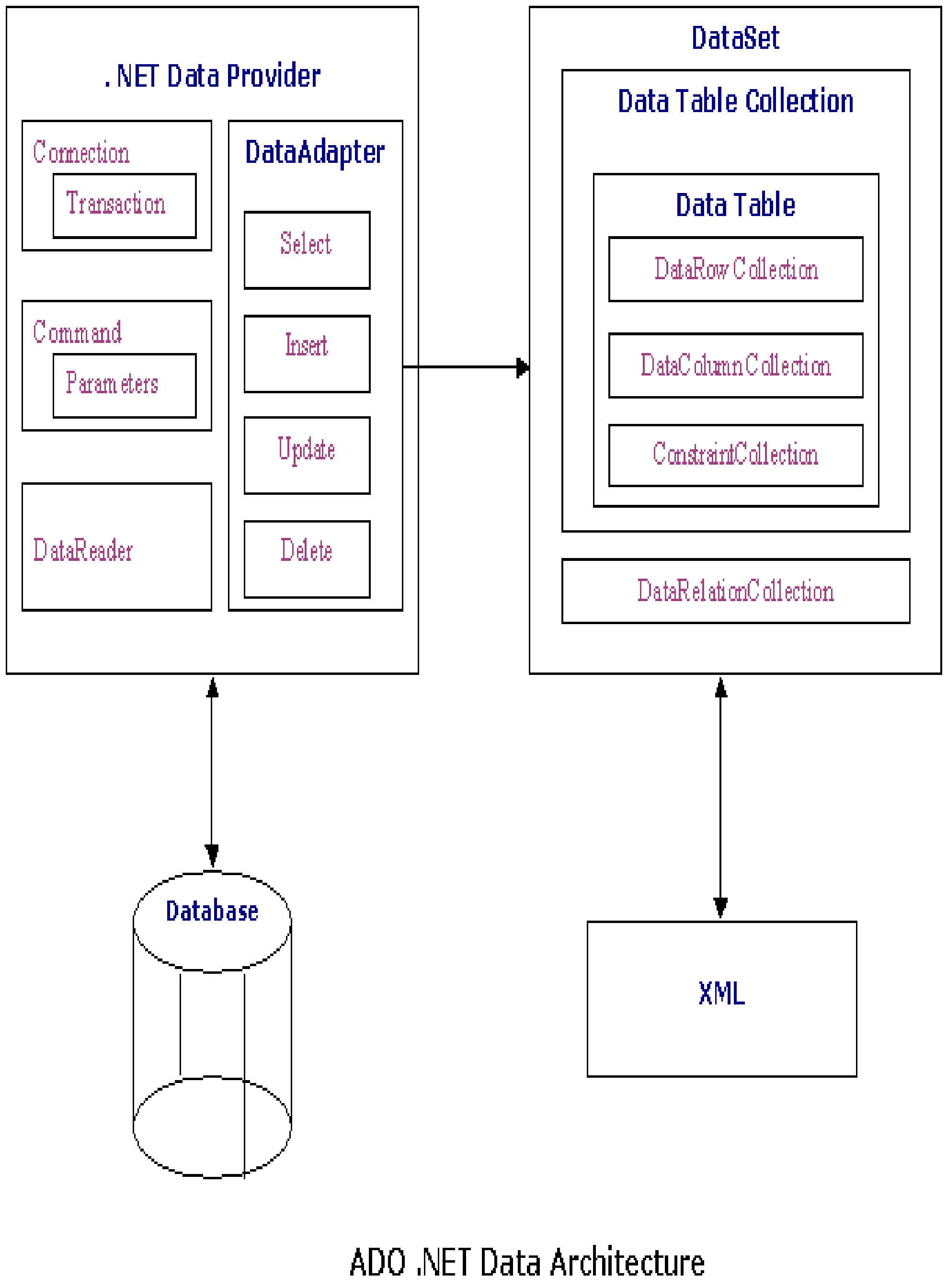

Figure 3.4: ADO .NET Data Architecture [41]

3.7.1 Experimental Setup

MySQL and MS SQL are the DBMS software that have been considered here. The Relational Model for Databases has been considered in this work. The Queries are assumed to be in standard SQL format and are submitted from query files.

Examples used to illustrate this section are drawn from the following three databases. Each database has three relations. Each Database is stored at a separate site.

1. MySQL Database under Windows: Employee 1

   Site 1: IP Address: 76.162.254.156 (ixwebhosting.com)

   The schema of all the tables in the above database is as follows:

   emp_det(no,deptid,desig,salary,dt_join,mailid,address)
   hr_employee (no,deptid,wrkaddr,suboffice,salary,dt_change date)
   empl_master (no,deptid,admin)

2. MySQL Database under Windows: Employee 2

   Site 2: IP Address: 70.87.57.146 (visionwebhosting.com)

   The schema of all the tables in the above database is as follows:

   emp_det(no,deptid,desig,salary,dt_join,mailid,address)
   hr_employee (no,deptid,wrkaddr,suboffice,salary,dt_change date)
   empl_master (no,deptid,admin)

3. MSSQL Database Under Windows: Employee 3

   Site 3: IP Address: 76.162.254.156 (ixwebhosting.com)

   The Schema of all the tables in the above database is as follows:

   emp_det(no,deptid,desig,salary,dt_join,mailid,address)
   hr_employee (no,deptid,wrkaddr,suboffice,salary,dt_change date)
   hr_empl_trans (no,deptid)

3.7.2 Database Creation

The following commands were used to create tables under the **employee 1 and employee 2** databases.

```
CREATE TABLE emp_det (
        no char(10) NOT NULL Primary Key,
        deptid char(10),
        desig char(10),
        salary real,
        dt_join date,
        mailid char(20),
        address char(50));

CREATE TABLE hr_employee (
        No char(10) NOT NULL Primary Key,
        deptid char(10),
        wrkaddr char(15),
        suboffice char(15),
        salary real,
        dt_change date,
        Foregin Key (no) references emp_det);
```

The following commands were used to create relations under the employee database:

```
CREATE TABLE empl_master (
        no char(10) NOTNULL Primary Key
        deptid  char(10),
        admin chat(10),
        Foregin Key (no) references emp_det,
        Foregin Key (deptid) references emp_det );

CREATE TABLE hr_empl_trans (
        no char (10) NOTNULL Primary Key
        deptid char(10),
        Foregin Key (no) references emp_det,
        Foregin Key (deptid) references emp_det);
```

3.7.3 Interface Design

The uniform interface as shown in figure 3.5 allows the user to enter the query. The interface has been created in VB .NET and implemented in ADO .NET and connection to database is achieved .NET Data Provider. Once the user has entered the query and has clicked the "Execute Query" button, the ADO .NET Architecture provides consistent access to data source through OLE DB and XML. The Data-sharing consumer applications can use ADO .NET to connect to these data sources and retrieve. ADO .NET includes .NET framework data providers for connecting to the appropriate database, executing commands and retrieving results.

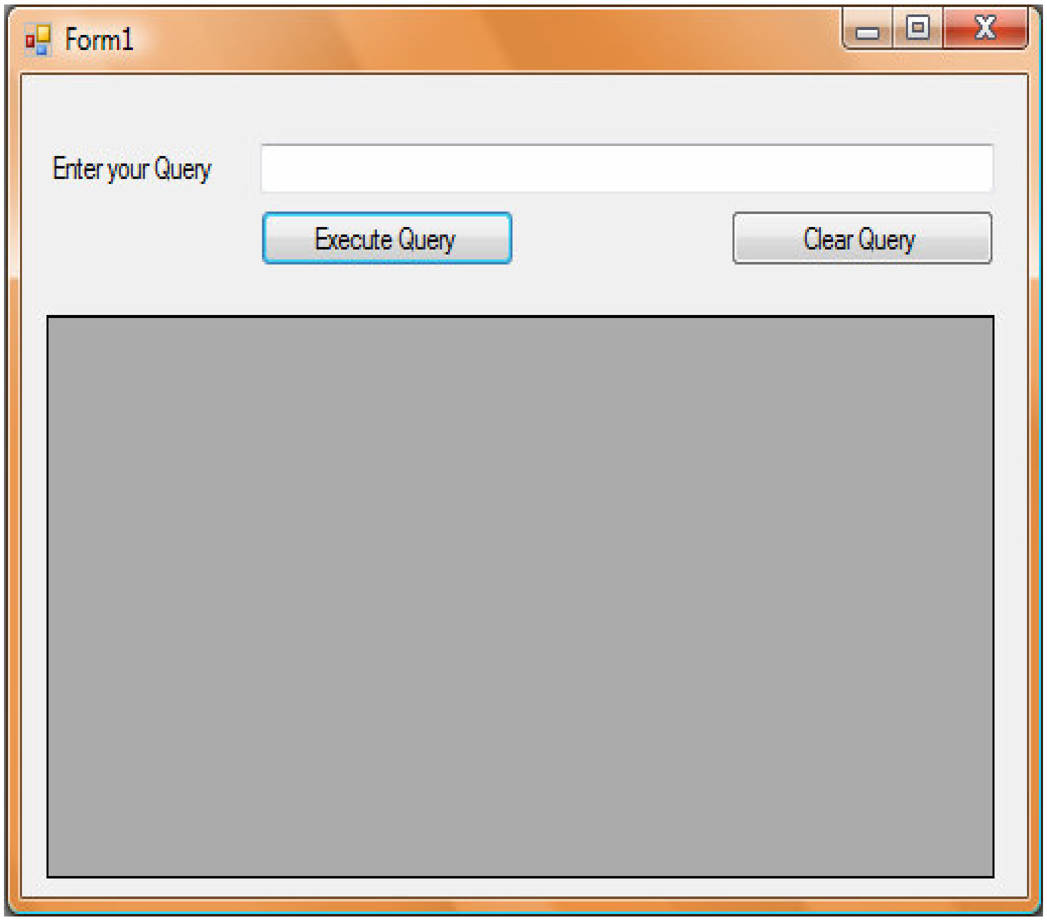

Figure 3.5: Layout of the UNIFORM INTERFACE implemented using ADO .NET

3.7.4 TEST SCENARIO

The .NET program was compiled using the standard Visual Studio IDE.

The following query has been entered in the input text file in3.qry using the uniform interface.

select emp_det.no, emp_det.deptid, emp_det.desig emp_det.amilid, emp_det.address from emp_det, he_employees where emp_det.no = hr_employee.no;

The following table is obtained as a result of the above query.

Table 3.4: Results of Sample Query

NO	**DEPTID**	**DESIG**	**MAILID**	**ADDRESS**
C1035	HR4-1DUB	Senior Advisor	davis@gmail.com	Dubai
D2301	HR1-2SHJ	Finance Controller	mathew@gmail.com	Sharjah
……..	……..	……..	…….	
……..	……..	…….	…….	

The query has been executed for different selectivity factors (number of records selected) and varying network speeds (band widths). The number of records denotes the total percentage of records retrieved from the main database and the execution time (in seconds) was calculated for each pair of selectivity factor and band width. The results are tabulated in Table 3.5.

Table3.5: Execution time (in secs) for a given (selectivity factor, bandwidth)

Number of records	Bandwidth		
	512Kbps	1Mbps	2Mbps
1343	0.81375	0.4860336	0.314117
2686	1.6275	0.9720672	0.628234
4029	2.44125	1.4581008	0.94235
5372	3.255	1.9441344	1.256467
6715	4.06875	2.430168	1.570584
8058	4.8825	2.9162016	1.884701
9401	5.69625	3.4022352	2.198818
10744	6.51	3.8882688	2.512934
12087	7.32375	4.3743024	2.827051
13431	8.1375	4.860336	3.141168

The test database considered here is of size 13.12 MB. The performance graph for the above database is shown in figure 3.6 for varying selectivity factors and band widths. The results obtained were qualitatively similar when the database size is varied in the range 10.0 MB – 100.0 MB. From the graph, it can be inferred that the execution times increases linearly with the number of records, for a given band width. It is also clear that as the band width decreases, the execution time increases. It is quite possible that the execution times can vary to some extent for a given pair of selectivity factor, and band width, depending upon the prevailing network traffic / load and the search algorithm deployed in various databases.

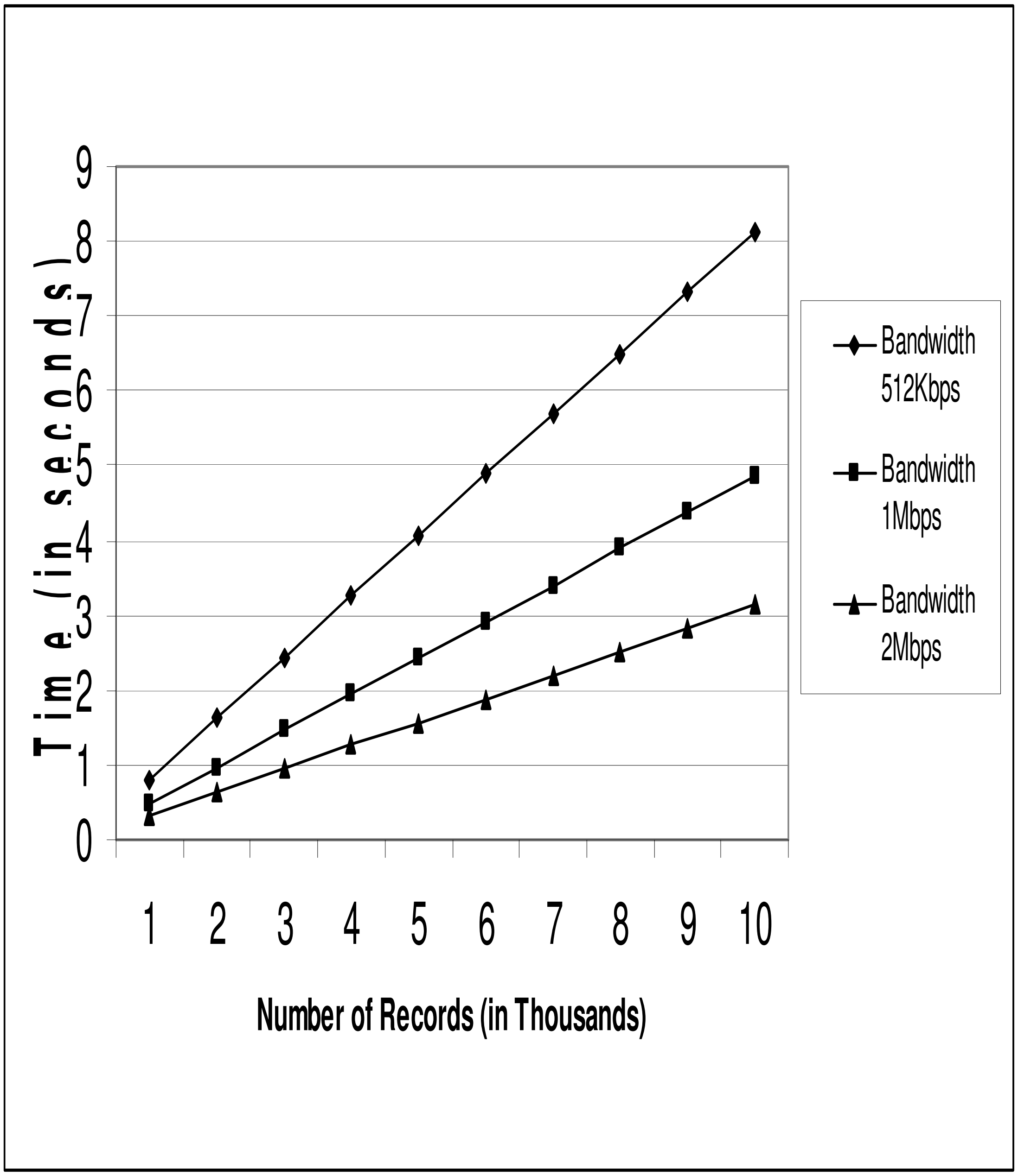

Figure 3.6: Performance graph for Test Database under varying selectivity factor and band width

3.8 Complexity

Table 3.6 summarizes the time complexity of SQL operations at each site, as referred in section 6 and section 7.

Let n be the number of tuples in a relation / fragment.

Table 3.6: Complexity of SQL Operations at each site

Operation	**Complexity**
SELECT	O(n)
PROJECT	O(n)
JOIN (sort-merge)	O(n * log n)

3.9 Summary

An algorithm for searching multiple databases in a heterogeneous environment has been discussed and implemented using two approaches:

1. JDBC and Java API.
2. ADO .NET Architecture.

It has been successfully tested with several queries which have been initiated from multiple client sites.

The programs developed in .NET and JAVA are interoperable as they are created for web based operation and the resulting model can be very helpful in practical applications such as banking systems and airline enquiry systems where there is a need to execute several queries over multiple databases located in different servers.

Chapter 4

Mathematical Representation of Pre and Post Conditions using Propositions for Database Components

4.1 Introduction

The software engineering arena is constantly moving towards a new direction i.e. Component Based Software Development and it is making new advances in this area. The basic idea in this approach is to develop software by assembling off the shelf components which are reusable [42], [43]. A component is a unit of composition with contractually specified interfaces and explicit dependencies [44]. An interface describes the services offered or required by a component without disclosing the component implementation. It is the only means of access to the information of a component. The component reuse is another important characteristic of component-based development. Component based development helps in building software that is reusable [45]. However, reuse without a precise specification is a disastrous risk.

The formal methods refer to the mathematics and modeling applicable to the specification, design, and verification of software [46]. The emphasis is on the creation of theories and tools to aid these activities. The methods are "formal" in the sense that they are precise enough to be implemented on a computer.

4.2 Related Work

The idea behind this chapter is to integrate the formal methods and the frame work for implementation, with regard to database components. Component-based Software Development (CBD) helps in building software in modular way. It is necessary to be able to specify components in such a way that we can reason about their construction and composition, and correctness.

Formal methods have been supported as one of those techniques that are likely, when correctly applied, to result in systems of the highest integrity. The approach of choosing appropriate notations and integrating them with the existing development of systems, being careful to ensure that existing guidelines and procedures are retained, plays a vital role for the successful implementation of formal methods [47].

This chapter deals with experimental work using three database servers and their outputs to the queries, formulated using predicate logic. While formulating the formal definitions for queries the Ten Commandments of formal methods were kept in mind. The Ten Commandments for formal methods [48] highlight the following key points:

- Choose an appropriate notation and the notation used at this stage will have a well defined formal semantics.
- Formal specification, formal development / verification and machine-checked proofs, each of these three levels is useful in itself.

- Cost Estimation: The efforts required by the component designer in estimating development costs, development lead-times and initial set-up costs. The initial set-up costs include investment in support tools, contract, consultancy and training.
- Majority of successful formal methods projects have a consultant who is expert in formal techniques.
- An alternative to integration of techniques is the use of formal methods to review an existing process.
- Formalizing the documentation leads to less ambiguity and thus less likelihood of errors.
- There is nothing magical about formal methods, and ensuring that system documentation meets the quality standards that were set for conventional development.
- Formal methods are one of a number of techniques to result in systems of the highest integrity, and one should not dismiss other methods entirely.
- Formal methods allow us to examine system behavior and to convince ourselves that all possibilities have been anticipated.
- Reuse. The four major factors which conspire against software reuse include: the Very Large Scale Reuse problem, Generality versus Specification, cost for selection of suitable components for future reuse and the reliability of the component reuse. The use of formal methods in system development can help to overcome each of these unambiguously

stating the requirements of a system, or of a component. Thus components that have been formally specified and sufficiently well documented can be identified, reused and combined to form components of the new system.

4.3 Problem Description

The component view of the experimental system is shown in Figure 4.1. The proposed system reads SQL queries submitted by users from input files. The experimental system comprises of Database Servers, User Interface, Query Parser, Authentication Server and Query Generation and Processing module as shown in Figure 4.1. When the clients want to access this component, this component checks its authentication with the help of Authentication Server. If the authentication process is successful then the client is allowed to enter its query for processing and retrieving data from the corresponding server. The query is entered by the client through user interface module and it will be communicated to the Query Generation & Processing Module further for processing. The accepted query is parsed by the query parser and it is send back to the Query Generation & Processing Module. Then the exact database server site is connected and the data are retrieved and forwarded to the requested client. Multiple clients (say, n) can pass the *select* queries to the Database Component, residing at a given site. The database component helps the clients for data retrieval from the appropriate servers located at various sites using the IP addresses.

Components support certain definite set of features and present functional interface to the client application. Applications can be taken away from the details of implementations and managing an advanced technology. Components can be designed to support certain algorithms. The application uses the algorithm by connecting the component to the component manager. Component manager helps the user to specify and register a component type and the communication is established through the component manager.

Component classes that are user-defined must have their functional interfaces specified. The applications must invoke these components as they adhere to the specified definitions and support the interface. Component interactivity is enhanced through defining the integrated functionalities for application invocations. Semantic integrity rules form the basis of preconditions for processing the queries and conditional verification and formal specification methods form the basis for post condition. The post conditions ensure the information absoluteness with higher degree of accuracy after the execution of the call. Therefore the returned value to the user would be program verified and mathematically interpretable.

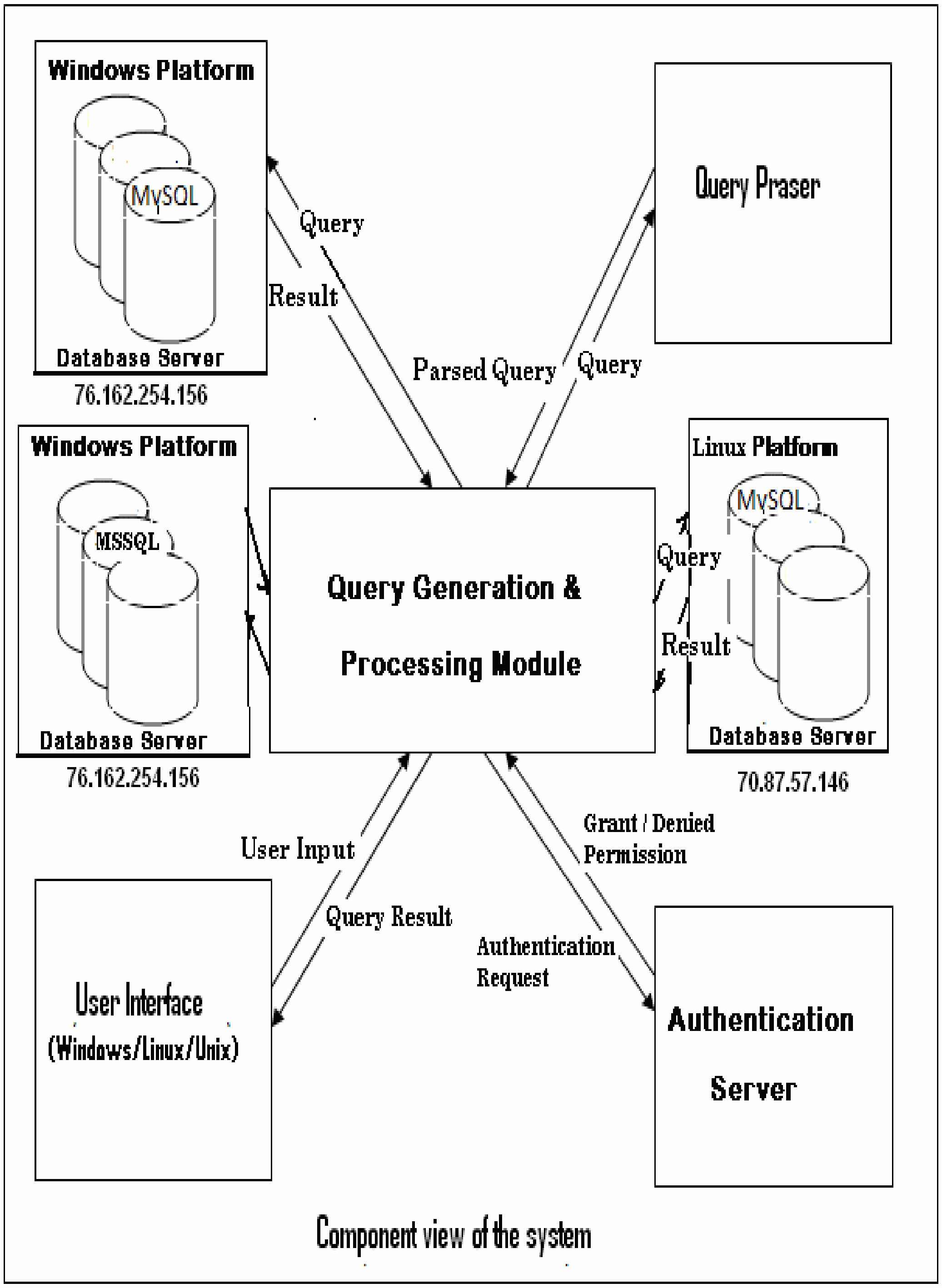

Figure 4.1: Component View of the Experimental System

4.4 SELECT Statement

An SQL SELECT statement returns a result set of records from one or more tables. As SQL is a non-procedural language, SELECT queries specify a result set, but do not specify how to calculate it: translating the query into an executable "query plan" is left to the database system. The SELECT statement has many optional clauses:

- WHERE specifies which rows to retrieve.
- GROUP BY groups rows sharing a property so that an aggregate function can be applied to each group.
- HAVING clause is used in combination with GROUP BY clause to filter the records that a GROUP BY returns.
- ORDER BY specifies an order in which to return the rows.

4.5 Pre-Conditions & Post Conditions

Often a method is written under the assumption that certain things are true about the environment: these are called pre-conditions. We can illustrate the requirements of a function and the conditions that would hold true after its execution. We can do so by specifying the *pre-conditions* and *post conditions* of a function.

The precondition expresses requirements that any call must satisfy in order to be correct, while its post condition expresses properties that are ensured in return by the execution of the call [49].

4.6 Mathematical Representation for SQL SELECT Queries

This section deals with the mathematical representation for SQL SELECT queries and it is based on Demorgan's Laws and the Truth Table method [50].

Consider the following SQL queries:

a) Select * from tbl_company_master

Precondition: zero or more records must exist and one or more fields must exist in the table.

Post- condition: if the above precondition is true the result will be displayed on the screen.

We can express the above precondition in equation 1.

Let p: (set of zero or more records)

q: (set of one or more fields)

A: (the table tbl_company,_master)

^ : *and* operator

→ : implies

~ : negation

v : *or* operator

$(p \wedge q) \longrightarrow A$ (equation 4.1)

The truth table corresponding to equation 1 is shown in Table 4.1. In the truth table, an entry 'T' refers to True and 'F' indicates False for the given predicates.

Table 4.1: Truth table for query (a)

p	q	p^q
T	T	T
T	F	F
F	T	F
F	F	F

The mathematical representation for the above query is verified using the properties of predicate logic.

$p \wedge q \rightarrow A$

p is a statement which expresses a particular condition or situation.

This Logic is represented through a predicate.

Similarly q, A are predicates of the same type either implicitly or explicitly explaining a situation.

$\rightarrow$ can be replaced by " v"

for example : $p \rightarrow q$ can be replaced by $\sim p \vee q$.

Implication can be replaced by "v" which forms the clausal form of a statement or condition [51].

Therefore the above mentioned statement $p \wedge q \rightarrow A$ can be

written as $\sim(p \wedge q) \vee A$

Applying Demorgan's law

$\sim p \vee \sim q \vee A$

which is either (Not preconditions on zero or more records) or (Not preconditions on zero or more fields) or

(table tbl_company,_master). Equation (1) is rewritten as

$p \wedge q \rightarrow A. \equiv \sim p \vee \sim q \vee A$ (equation 4.2)

Table 4.2: Verification of equation 4.2 using DeMorgan's Law for query (a) $\tilde{}p \vee \tilde{}q \vee A$ (Right Hand Side)

p	q	A	~p	~q	~p v ~q	~p v ~q v A
F	F	F	T	T	T	T
F	F	T	T	T	T	T
F	T	F	T	F	T	T
F	T	T	T	F	T	T
T	F	F	F	T	T	T
T	F	T	F	T	T	T
T	T	F	F	F	F	F
T	T	T	F	F	F	T

Table: 4.3: Verification of equation 4.2 using DeMorgan's Law for query (a) $p \wedge q \rightarrow A$ (Left Hand Side)

p	q	p ^ q	A	p ^ q → A
F	F	F	F	T
F	F	F	T	T
F	T	F	F	T
F	T	F	T	T
T	F	F	F	T
T	F	F	T	T
T	T	T	F	F
T	T	T	T	T

Hence equation (4.1) is verified using the DeMorgan's Law, and it is rewritten as $p \wedge q \rightarrow A_. \equiv \tilde{}p \vee \tilde{}q \vee A$ …. equation 4.2 and shown that the right hand side of the expression is equivalent to the left hand side and are shown in Table 4.2 and Table 4.3.

b) Select CompanyId,CompanyName from tbl_company_master

Precondition: CompanyId & CompanyName must exist with zero or more records in the table.

Post- condition: if the above precondition is true the result will be displayed on the screen.

The above precondition in equation 2 can be expressed as

Let p : (set of zero or more records)

q1: (the field CompanyId)

q2: (the field CompanyName)

A : (the table tbl_company_master)

^ : *and* operator

→ : implies

$p \wedge (q1 \wedge q2) \longrightarrow A$ ……… (equation 4.3)

The truth table corresponding to equation (4.3) is shown in Table 4.4.

Table 4.4: Truth table for query (b)

p	q1	q2	q1^q2	p^(q1^q2)
T	T	T	T	T
T	T	F	F	F
T	F	T	F	F
T	F	F	F	F
F	T	T	T	F
F	T	F	F	F
F	F	T	F	F
F	F	F	F	F

The mathematical representation for the above query is also verified using the properties of predicate logic.

$p \wedge (q1 \wedge q2) \rightarrow A$

p is a statement which expresses a particular condition or situation. This logic is represented through a predicate and the implication can be replaced by "v" which forms the clausal form of a statement or condition [51]. Therefore the above statement $p \wedge (q1 \wedge q2) \rightarrow A$ can be rewritten as $\tilde{}(p \wedge (q1 \wedge q2)) \vee A$

By applying Demorgan's Law

$\tilde{}p \vee \tilde{}q1 \vee \tilde{}q2 \vee A$

which is either (Not preconditions on zero or more records) or (Not preconditions on zero or more fields) or (table tbl_company,_master). Equation (2) is rewritten as

$p \wedge (q1 \wedge q2) \rightarrow A \equiv \tilde{}p \vee \tilde{}q1 \vee \tilde{}q2 \vee A$ ...... (equation 4.4)

Table 4.5: Verification of equation 4.4 using Demorgan's Law for query (b)

˜p v ˜q1 v ˜q2 v A (Right Hand Side)

p	q1	q2	q1 ^ q2	p ^ (q1 ^ q2)	˜(p ^ (q1 ^ q2))	A	˜p v ˜q1 v ˜q2 v A
T	T	T	T	T	F	T	T
T	T	T	T	T	F	F	F
T	T	F	F	F	T	T	T
T	T	F	F	F	T	F	T
T	F	T	F	F	T	T	T
T	F	T	F	F	T	F	T
T	F	F	F	F	T	T	T
T	F	F	F	F	T	F	T
F	T	T	T	F	T	T	T
F	T	T	T	F	T	F	T
F	T	F	F	F	T	T	T
F	T	F	F	F	T	F	T
F	F	T	F	F	T	T	T
F	F	T	F	F	T	F	T
F	F	F	F	F	T	T	T
F	F	F	F	F	T	F	T

Table 4.6: Verification of equation 4.4 using Demorgan's Law for query (b)

$p \wedge (q1 \wedge q2) \rightarrow A$ (Left Hand Side)

p	q1	q2	q1 ^ q2	p ^ (q1 ^ q2)	A	p ^ (q1 ^ q2) → A
T	T	T	T	T	T	T
T	T	T	T	T	F	F
T	T	F	F	F	T	T
T	T	F	F	F	F	T
T	F	T	F	F	T	T
T	F	T	F	F	F	T
T	F	F	F	F	T	T
T	F	F	F	F	F	T
F	T	T	T	F	T	T
F	T	T	T	F	F	T
F	T	F	F	F	T	T
F	T	F	F	F	F	T
F	F	T	F	F	T	T
F	F	T	F	F	F	T
F	F	F	F	F	T	T
F	F	F	F	F	F	T

Hence the equation (4.4) is verified using the Demorgan's Law, and shown that the right hand side of the expression is equivalent to the left hand side and are shown in Table 4.5 and Table 4.6.

c) Select CategoryId,BrandId, CompanyId, CompanyName From

```
 tbl_company_master,
 tbl_admin_insert
      Where tbl_admin_insert.CompanyId= tbl_company_master.companyId
```

Precondition: CategoryId,BrandId,CompanyId must exist with zero or more records in the table tbl_admin_insert, CompanyId & CompanyName must exist with zero or more records in table tbl_company_master and zero or more records in tbl_admin_insert.ComapnyId should be equal to tbl_company_master.CompanyId.

Post- condition: if the above precondition is true the result will be displayed on the screen.

The above precondition can be expressed as

Let

p : (set of zero or more records)

q1: (the field CompanyId in tbl_admin_insert)

q2: (the field CategoryId in tbl_admin_insert)

q3: (the field BrandId in tbl_admin_insert)

q4: (the field CompanyId in tbl_company_master)

q5: (the field CompanyName in tbl_company_master)

A: (the table tbl_company_master)

B: (the table tbl_admin_insert)

$$\forall\ (q1=q4): p \wedge ((q1 \wedge q2 \wedge q3) \wedge (q4 \wedge q5)) \longrightarrow A$$

…(equation 4.5)

The above equation (4.5) corresponds to the query (c) over two relations A and B involving join operation over the attribute companyId.

The mathematical representation for the above query also can be verified using the properties of predicate logic, using similar approach as followed for the earlier queries (a) and (b).

4.7 Experimental Verification

The system developed for executing the above queries comprised of the following two tables with the following structures:

1) tbl_company_master

COLUMN_NAME	COLUMN_TYPE
CompanyId	int(10) unsigned
CompanyName	varchar(200)
Address	varchar(200)
FullPageAdv	varchar(200)
URL	varchar(200)

2) tbl_admin_insert

COLUMN_NAME	COLUMN_TYPE
CategoryId	int(10) unsigned
BrandID	int(10) unsigned
CompanyId	int(10) unsigned
BrandLogo	varchar(200)
BannerImage	varchar(200)
FullPageAdv	varchar(200)

The above tables were created on three different servers with MSSQL (76.162.254.156) and MySQL (76.162.254.156, 70.87.57.146) as shown in figure 4.2 and the queries were executed across all database servers.

The users will be authenticated using the authentication module. The user interface for both Linux and Windows operating systems are supported and the results will be displayed to the user. The database server exists as a combination of three different servers and the site IP addresses are 76.162.254.156 (MYSQL under Windows), 76.162.254.156 (MSSQL under Windows) and 70.87.57.146 (MYSQL under Linux).

The following query corresponding to predicate logic as mentioned in problem description has been given as input to the system:

```
Select CompanyId,CompanyName from tbl_company_master
```

The above query executed on the system successfully and the following results are obtained for the given test data and shown in Figure 4.2.

Server and Process Information	
Server Name:	MySQL . ixwebhosting.com(76.162.254.156 under Windows), MSSQL .ixwebhosting.com(76.162.254.156 under Windows), MYSQL .visionwebhosting.com (70.87.57.146 under Linux)
Server Traffic:	Traffic on the Server
Table Involved:	Tbl_company_master
CompanyID	Company Name
4	A M Trading Co LLC
5	A A Juma Plastic Pipes & Fittings Ind. Co. LLC
6	A M Jalal A/c & Ref. Eqpt. Tradg.
16	ABDULLA & BALIAN OFFICE EQPT LLC
26	ABU DHABI MOTORS
37	ACECO
40	Ad Power FZCO
50	ADVANCED IMAGING SYSTEMS EST
59	AHMAD TEA (ME) LTD
108	Al Barq Kitchen Equipments Trading LLC
109	Al Basel Co Heavy Equip Spare Parts Tr
133	Al Ghandi Gen Trdg Co LLC
145	Al Horiya Sand Blasting
146	Al Humaidi Trdg Ent LLC
153	Al Italiah Ind Trdg Co LLC
164	Al Jazirah Eqpt. & Tech. Svcs.
202	Al Naboodah Automobiles LLC

Figure 4.2: Server and Process Information

The same procedure has been followed for query (c) and tested successfully.

4.8 Summary

This chapter discussed in detail an approach to use mathematical representation to express queries for database components. The experimental system has been setup using database components and the SQL queries have been tested successfully.

Chapter 5

Component Design Key Factors for Semantic Integrity Control and Interoperability

5.1 Introduction

This chapter deals with the key factors responsible for maintaining semantic integrity control and interoperability in the design of components. The development of complex business applications is now focused on an assembly of components available on a local area network or the Internet. This chapter focuses on identification of key factors for Semantic Integrity Control and Interoperability and their applicability to Domain, Service and Agent Components.

A survey has been conducted on web-based applications involving phpmyAdmin, using MySQL under Windows and Linux environments. The applications considered are:

1. Online Faculty Feedback System (can be deployed on intranet / Internet) [52].
2. Telephone Directory Information System [53].
3. Library Information System [54].

These systems are more focused on a specific application area. The semantic integrity control and interoperability are implemented separately for each application. Hence, it is necessary to evolve a common approach / design pattern to develop a wide range of applications. Based on the examination of operations performed in these applications, the key factors governing Semantic Integrity Control and Interoperability have been identified and listed below:

The key factors for Semantic Integrity Control are *Authentication, Access to the host program and databases present in the Domain server, Access to Proprietary Programs, Source Manipulation to update assertions, checking for Assertions, Dynamic Data Manipulation, Time Period of Component Availability and Consistency of output*.

The key factors for Interoperability are *Open Protocols, Open APIs, Open Access, Open Source Compatibility, Broad Compatibility and Open Import / Export* [55].

The key factors for semantic integrity are obtained considering simple day to day business architecture in mind. Using the authentication the program is restricted to subscribed users, thus the integrity is maintained externally. If authentication is not required, then there exists emphasis on the internal code's integrity. Source manipulation and updating assertions is a factor that specifies the change in the integrity that the program will encounter and due to which the program's execution technique changes i.e it specifies the dynamic character of the integrity of the component. Dynamic data manipulation specified the capability of the component to change the object or data according to the need by the requested program. Time period of component availability and consistency ensure the component’s integrity and strength for a very long time. During this run of a component, the component is checked for the range to which it can access the programs or the dependencies of the host that it is requesting for. This gives the integrity of the component’s accessibility and its wide extension of service.

The key factors for Interoperability are selected on the basis of the software standards as specified by organizations namely Microsoft, CORBA and IBM. Open protocols define the extent to which the algorithm is standard and thus the extent to which it is widely available for usage. Open access also specifies the same aspect but in a different direction, this specifies the extent to which the program can be accessed by some individual applications. Open APIs provides the set of functions for the developer in the interface design. Open source compatibility specifies the commonness of algorithm and code’s depth between various open source programs. Broad Compatibility specifies whether or not the component is capable of performing its operation in spite of programs restriction with internal coding with the other components .Open import and export specifies whether the object and classes in the interface are interoperable with other components in the distributed environment.

The categories of components considered in the present work include Domain Components, Service Components, and Agent Components. The connection between these components is determined by the language and compiler used in creating and compiling it each time when it is executed. The components generated for web applications act as a platform independent module that supports communication from Client and server machines with the maximum fastness in which it can execute.

All components don't mean the same. They differ in the complexity, scope and level of functionality. The components can be broadly classified as given below:

1. Domain Components,
2. Service Components,
3. Agent Components [55].

5.2 Domain Component

The Domain Component permits reusable classes for an application developed within the same domain. The *key factors* for *semantic integrity control* that are applicable to the design of domain components are discussed in detail in the following paragraphs:

1. **Authentication:**

   Domain component authenticates at user level. This is required in order to facilitate the highest security. It facilitates authorized access from different machines for users running similar processes. The concerned processes can be used to implement the business logic of the domain..

2. **Access to the host program and databases present in the Domain Server:**

   It gives full access rights to all components in the same domain. The host programs can be accessed through the domain component. The host programs can do functions relating to record keeping, by accessing the restricted tables..

3. **Access to Proprietary Programs :**

   The administrator can give permission to the domain component for executing / accessing proprietary programs on a host machine. The proprietary program could be an internal service or distributed computing based service.

4. **Source program Manipulation to update assertions:**

   A domain component cannot manipulate source program pertaining to domain application, as this will result in inconsistent state. However, new assertions can be created and enforced through service component which transfers control to agent and domain component in this order.

5. **Checking of Assertions:**

   The assertions are checked for the correctness of semantics and syntax. A faulty syntax can lead to errors that propagate from the domain to the user application and it is undesirable. At the same time a domain component designer should ensure correct semantics for all assertions.

6. **Dynamic Data Manipulation:**

   Domain components do not allow dynamic data manipulation as it will lead to faulty or misinterpreted result at the receiver's end. It is advisable to transport the data in a standard format and the manipulation is done by the receiving end.

7. **Time Period of Component Availability:**

   The domain component is designed in such a way that it can perform its functions at any time when requested by the user application. It remains active in the host server and runs as a background process. It keeps polling the requests from the user applications and responds to them.

8. **Consistency of Output:**

   The domain component should offer high level of consistency for all the user requests, by providing accurate and timely output. This is applicable even for multiple requests from users, which are identical and the output remains the same.

The *key factors* for *interoperability* that are applicable to the design of domain components are discussed in detail in the following paragraphs:

1. **Open Protocols:**

   Open protocols are not supported as they are built on platform specific environments and use high consistency rate for calculation and transmission of the data obtained.

2. **Open APIs:**

   The domain component as such does not support open API, across all domains. However, the applications within the same domain can use same type of domain component.

3. **Open Access to user and technical documentation:**

   The domain component is allowed for open access to user and technical documentation of components in the same domain only.

4. **Open Source Compatibility:**

   The open source compatibility is also not allowed for the operations involving proprietary information.

5. **Broad Compatibility:**

   The domain component can be designed to provide broad compatibility. This will facilitate in its wider deployment across domain applications.

6. **Open Import / Export**:

   The domain component can be designed to provide export / import facility for data handling. This will support interoperability across different applications in one or more domains.

5.3 Service Component

Service Components are software components such as User Interface (UI), storage, directory services, etc. A service is the basic building block for writing applications using Distributed Software Services (DSS), and is a key component of the DSS application model. The Service Component executes a set of activities in response to a request (event) and delivers some result. Services are executed within the context of a DSS node. A DSS node is a hosting environment that provides support for services to be created and managed until they are deleted or the DSS node is stopped. Services are inherently network enabled and can communicate with each other in a uniform manner. This works regardless of whether they are executed within the same DSS node or across the network. When a service instance is created within a DSS Node, it is dynamically assigned a Universal Resource Identifier (URI), by the constructor service. This service identifier refers to a specific instance of a service identifier running on a particular DSS node. The service identifier is what enables other services to communicate with that service as well as Accessing DSS Services through a Web Browser.

The *key factors* for *semantic integrity control* that are applicable to the design of service components are discussed in detail in the following paragraphs:

.

1. **Authentication:**

   It is required for user identification and service requested by the user. Service component requires authentication in order to facilitate the security and ubiquitous usage between various services offered for different client machines.

2. **Access to the host program and databases present in the Domain Server:**

   The Domain Sever permits access to one or more host programs and databases as per the nature of the service. Some potential services under this category include: executing a remote procedure call and database search.

3. **Access to Proprietary Programs :**

   The service component is implemented using a set of classes. Some of the methods in these classes invoke proprietary program to do extended services such as customization function in an ERP application.

4. **Source Manipulation to update assertions:**

   Source can be manipulated as it can lead to better services which are intended by the user applications. This help in enforcing new assertions (such as assertions to deny inference queries) created in external file.

5. **Checking of Assertions:**

   A service component checks the assertions for the correctness of semantics and syntax. A faulty syntax leads to undesirable exceptions and terminates the intended function abruptly. A service component designer is required to frame the assertions with correct semantics.

6. **Dynamic Data Manipulation:**

   A Service component supports dynamic data manipulation, for handling input and output. For input data, it can do some preprocessing, if necessary (transforming from one format to another format) and so on. The output data can be suitably formatted as per the request of the user application.

7. **Time Period of Component Availability:**

   It is available only for specified time period. Service component can extend or renew the time period of the component upon specific request.

8. **Consistency of Output:**

   The service component offers a higher level of consistency under normal conditions. However, it can vary due to exceptions like node failure or link failure.

The *key factors* for *interoperability* that are applicable to the design of service components are discussed in detail in the following paragraphs:

1. **Open Protocols:**

   Open protocols are supported as it is built on an open platform inter-operable language. The protocols can be used in updation / upgradation related activities.

2. **Open APIs:**

   The service component can be integrated with other application interfaces across all the domains. It can be used as a service whose response can be used in another application run by the user.

3. **Open Access to user and technical documentation:**

   It is allowed for algorithmic change.

4. **Open Source Compatibility:**

   The open source compatibility is not provided as the design of the algorithms is considered to be confidential.

5. **Broad Compatibility:**

   The service component is designed to provide broad compatibility across many applications belong to one more domains.

6. **Open Import / Export:**

   Any forms of data can be imported and exported given by the system to any kind service components that is attached to it. This form of data is interoperable widely for use in any operating environment.

5.4 Agent Component

The Agent Component can be designed to perform pre-defined set of activities and can be regarded as web enabled software component. The *key factors* for *semantic integrity control* that are applicable to the design of agent components are discussed in detail in the following paragraph:

1. **Authentication:**

   It is not required since it receives pre-authenticated messages from Domain / Service Component.

2. **Access to the host program and databases present in the Domain Server:**

The agent component establishes connection to Domain / Service Component. Access to one or more host programs and databases is only through Domain / Service component.

3. **Access to Proprietary Programs:**

Agent component extends access to proprietary program according to the type of component (Domain / Service) as mentioned in section 5. 2 and section 5.3.

4. **Source Manipulation to update assertions:**

Agent component cannot manipulate source program to update assertions, since it is not defined in its scope.

5. **Checking of Assertions:**

Agent component is not required to handle assertions. The assertions are handled by domain and service components.

6. **Dynamic Data Manipulation:**

Agent component cannot perform dynamic data manipulation. However, it can route the formatted output from service component to the user application.

7. **Time Period of Component Availability:**

   The time period of component availability depends on type of component connected (Domain/Service).

8. **Consistency of Output:**

   Agent component offers a higher level of consistency. However, it can vary due to exceptions like node failure or link failure.

The *key factors* for *interoperability* that are applicable to the design of domain components are discussed in detail in the following paragraphs:

1. **Open Protocols:**

   Protocols are restricted to some functions that can be carried on these components. It supports for updating and upgrading, through the service component.

2. **Open APIs:**

   The agent component can be integrated with other interface/service components and it can also be deployed in any user application using open API.

3. **Open Access:**

   It is allowed for customization level changes.

4. **Open Source Compatibility:**

   Agent components can have their programs in a open source environment. This will enable them to inherit one or more classes from the class hierarchy and the agent design can be made quickly and effectively.

5. **Broad Compatibility:**

   It supports compatibility only for similar type of applications with the same domain and not across applications belonging to multiple domains.

6. **Open Import / Export:**

   Agent component cannot directly support open import / export feature for data handling, since it is not defined in its scope. However, the agent component can initiate service / domain component to perform this feature.

The following Table 5.1 and Table 5.2 represent the general component categorization based on *Semantic Integrity Control key factors* and *Interoperability key factors.*

Table 5.1: Summary of *key factors* for *Semantic Integrity Control* for three types of components

Key Factors for Semantic Integrity Control	**DOMAIN COMPONENTS**	**SERVICE COMPONENTS**	**AGENT COMPONENTS**
Authentication	Required for users to authenticate at user level.	Required for User ID and Services requested by Users.	Not required since it receives pre-authenticated messages from Domain / Service Component.
Access to the host program & databases present in the domain server.	Full access rights to all components in the same domain.	Permits access to one or more host programs & databases as per the nature of the service.	Establishes connection to Domain / Service component. Access to one or more host program & databases only through Domain / Service Components.
Access to Proprietary programs	It is accessible only by Domain Administrator.	It is accessible only by allowable services for selected proprietary programs.	It extends privileges according to type of component (Domain/Service).

Source Program Manipulation To update assertions.	It is not allowed.	It is allowed.	It is not allowed.
Checking Assertions.	It is mandatory.	It is mandatory.	It is not required.
Dynamic Data Manipulation.	It does not support dynamic data manipulation.	It supports dynamic data manipulation.	It does not support dynamic data manipulation.
Time Period of Component availability.	There is no restriction.	It is available only for specified time period. Service component can extend or renew the time period of component upon specific request.	The time period of component availability depends on type of component connected (Domain/Service).
Consistency of Output.	It offers a higher level of consistency.	It offers a higher level of consistency under normal conditions. However, it can vary due to exception like node failure, link failure.	It offers a higher level of consistency. However, it can vary due to exceptions like node failure, link failure.

Table 5.2: Summary of *key factors* for *Interoperability* for three types of components

Key factors for Interoperability	**DOMAIN COMPONENTS**	**SERVICE COMPONENTS**	**AGENT COMPONENTS**
Open Protocols The Open protocols have to be openly available to the developer community in a non-discriminatory fashion. The Open Protocols may include protocols that implement industry standards.	Supports for access purposes only.	Supports for updating and upgrading.	Supports for updating and upgrading.
Open APIs The Open APIs must be openly available to the developer community. Open APIs may include APIs that implement industry standards	Not provided.	Allowed for special purpose.	Allowed.
Open Access to User and technical documentation The documentation has to be published so that the developers can access them and reuse them in their application	Allowed for components in the same domain only.	Allowed for algorithmic change.	Allowed for customization level changes.

Open Source Compatibility The product must have the feature of open access to the open source developer for development and for non-commercial distribution of implementation of these products.	Not supported.	Not Supported.	Supported for enhanced operations.
Broad Compatibility The product should have high standards and allows the developers to develop a product of robust, consistent, and interoperable implementation across a broad range of widely deployed products.	Supported.	Supported.	It supports compatibility only for similar type of applications and not across different types of applications.
Open Import/Export It should support the "import" and "export" functions in various products that enable the transfer of user data from one application to another.	Allowed	Allowed for some services.	Not allowed.

5.5 Summary

This chapter dealt with the key factors for maintaining Semantic Integrity Control and Interoperability in the design of components. The *key factors* for *Semantic Integrity Control* include: Authentication, Access to the host program and databases present in the Domain server, Access to Proprietary Programs, Source Manipulation to update assertions, checking for Assertions, Dynamic Data Manipulation, Time Period of Component Availability and Consistency of output. The *key factor* for *Interoperability* include: Open Protocols, Open APIs, Open Access, Open Source Compatibility, Broad Compatibility and Open Import / Export.

Chapter 6

Component Design for Domain, Service and Agent

6.1 Introduction

There are two functional strategies for developing a *Domain component*, either top-down or bottom-up design. Top-down approach provides a good basis to promote a good hierarchical architectural design which can be implemented by a team of component developers in reasonably short time. The bottom-up approach is useful when we have already a collection of classes (developed earlier) and now it is required to integrate them to build the application.

The following are major three categories [56] of components:

1. *Domain Components*.
2. *Service Compon*ents.
3. *Agent Components*.

The design procedures are different for these components since they basically differ in their purpose of use and the place of applicability. The design steps for these components are discussed in details in the following sections.

6.2 Domain Component

A *domain component* can consist of a set of classes. The classes collaborate among themselves to support a set of operations. The operations include: Request information, Request an action to be performed. The classes can belong to any one of the three categories:

Server Class: Receive Messages from Clients.

Client Class: Send Messages to Server.

Client / Server class: Send and Receive Messages.

Some examples of Domain-specific components include: file management packages, device drivers, sorting packages and components pertaining to specific applications like ERP. An expert belonging to a specific domain should produce, maintain and catalog reusable classes and can allow them to be used by application developers.

The d*omain component*s [57] is designed and implemented such that it can be used extensively in multiple applications, and minimizing the variable parts by providing integration with other related components. *Domain component* design and analysis aims at taking full advantage of the reusability of domain models and it is used in multiple applications within the same domain. The block schematic for a *domain component* is shown in figure 6.1. The elements of a d*omain component* include the following:

1. Input from Domain / Agent.
2. *Domain component* Actions.
3. Output to *Domain / Service / Agent Components*.

The *domain component* incorporates restricted activities for Hardware and the Business Logic.

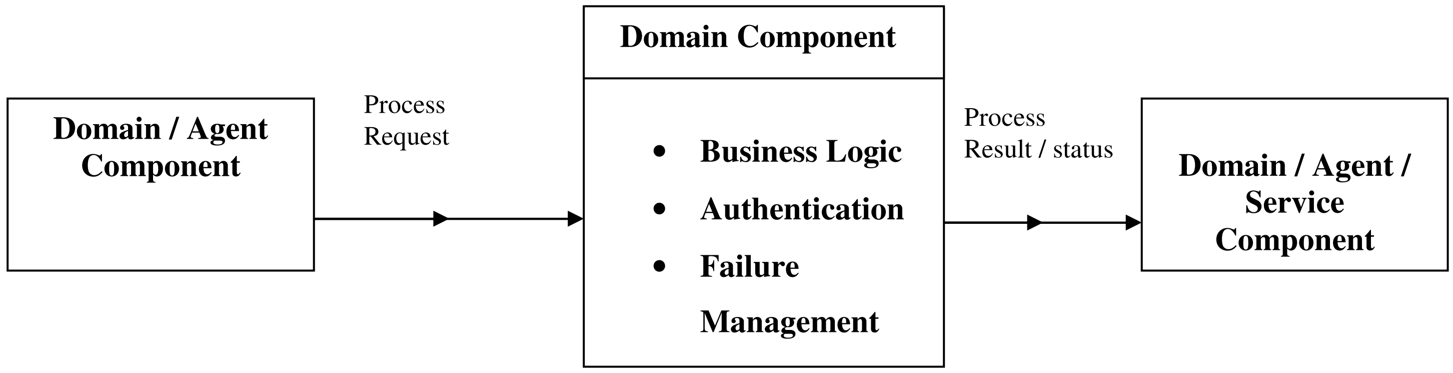

Figure 6.1: Block Schematic of Domain Component

1. **Input from *Domain / Agent Component*:**

   The input(s) for *domain component* can come from another Domain / Agent Component. The inputs possibly include request for processing some information / doing some action / information retrieval from a database. It is more specific on name specification by including some useful types of methods for referencing collections of resources and algorithm.

2. ***Domain component* Actions:**

   The *domain component* performs mainly three actions: Authentication, Business Logic and Failure Management.

It permits only allowable operations pertaining to business logic, authenticates users, and handles failures relating to transactions, nodes and links.

3. **Output to *Domain / Service / Agent Components*:**

The *Domain component* can give the output results, after performing the requisite actions, to Domain / Service / Agent Component(s) according to exports feature. This way it enables the transfer of information from one application to another.

6.3 Tasks in design of Domain Component

The main tasks in the design of *domain component* as perceived in the present work are shown below:

1. Define the Domain.
2. Collect details of existing applications in the domain.
3. Analyze each application in the sample.
4. Look for a set of functions / services that are common across various applications in the same domain and tabulate them.
5. Look for a set of functions/ services that are different / unique to each application in the same domain, and tabulate them.
6. Formulate rules and constraints as per the nature of each application in the domain. The rules could be business rules / policies or the way

the application is to be organized or structured. The implementation constraints can be based on available technologies.

7. It is necessary to define boundaries while dealing with a domain and surrounding environment. The boundaries specify entities, inputs, outputs, information flow between entities and processes within the domain and the semantics of the information.
8. The functional requirements for each application are analyzed.
9. Potential risk / problem areas in the domain are identified and tabulated.
10. Proceed to design classes for the domain, under study, keeping in mind, the reusability asset.
11. Explore the possibility of acquiring reusable classes from existing applications in the same domain.
12. After completing the design of classes in the domain, proceed to catalog and store the reusable classes in an organized way.
13. Maintain the class libraries for the domain under study. This includes tasks similar to software configuration management [58] namely, correction, enhancement and adaptation.
14. Give access rights / permissions for Application developers to use the reusable classes in the same domain

6.4 Service Component

The *Service Component* [57] is designed such that it can be used as a software component for User Interface, Database service and directory services. It acts as a essential block for interfacing with other components applications. Service is naturally network enabled and can correspond with other components that are connected to it through a uniform interface. The service instance is identified by the service identifier which helps the services to communicate with other services through web browsers. *Service Component* is implemented as user interface software module. It has a high potential of connecting to the *domain component*s and the agent components.

The *Service Component* is designed after having a broad analysis of identifying its requirements for a particular service support. The elements of a *Service component* include the following:

1. Accounting feature.
2. Authentication
3. Security.
4. Failure Management.
5. History Backup.

The block schematic for a *service component* is shown in figure 6.2.

The input(s) for *Service Component* can come from *Agent / Domain /* Client. The inputs possibly include service request such as database queries.

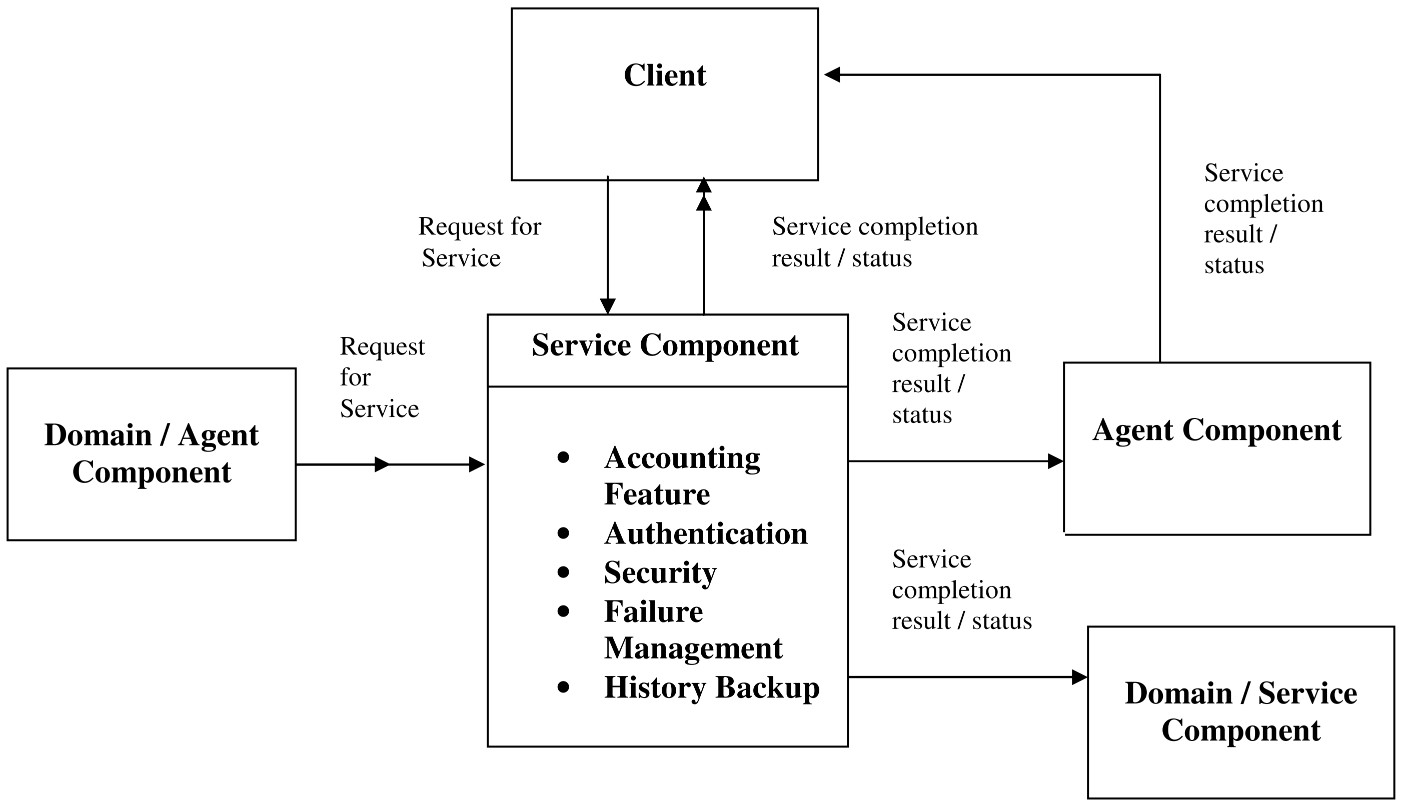

Figure 6.2: Block Schematic of Service Component

1. Accounting feature:

The accounting feature enables recording of information relating to service requests from client / Agent / *Domain Component* and service completion details, along with their time stamps. This is mainly used for billing / system accounting purposes.

2. **Authentication:**

   It permits only allowable services and Client / Component authentication.

3. **Security:**

   The Service component provides high level of information security for the information and service provided by it. The service provided by this component is mostly related to financial view. Therefore an open access to this component is not allowed.

4. **Failure Management:**

   The *Service component* supports the failure management in order to avoid the failure in the midst of processing the service. Therefore the *service component* will have a backup component to take over charge of execution due to failure of the actual service component.

5. **History Backup:**

   The *history backup* mechanism keeps a log of all service activities taking place in the *service component* over a period of time. This Log file will help in identifying any exceptions that might have occurred.

6.5 Tasks in design of Service Component

The main tasks in the design of *service component* as perceived in the present work are shown below:

1. Define the Service to be provided.
2. Look for a set of functions / services that are common across various applications in the same service category and tabulate them.
3. Look for a set of functions/ services that are different / unique to each application in the same service category, and tabulate them.
4. List down the entities, functions to be implemented in the service component and the information flow between entities and the service component.
5. It is necessary to define boundaries while dealing with a service and the operating environment on which it is provided to be deployed.
6. The functional requirements relating to the application and the interface classes are analyzed.
7. Potential risk / problem areas in the service and security features are identified and tabulated.
8. Proceed to design classes for the service component, under study, keeping in mind, the reusability aspect.
9. The possibility of acquiring reusable classes from existing service component is taken into consideration.

10. After completing the design of classes for the service component, proceed to catalog and store the reusable classes in an organized way.
11. Maintain the class libraries for the service component under study. This includes tasks similar to software configuration management [55] namely, correction, enhancement and adaptation.
12. Give access rights / permissions for Application developers to use the reusable classes while designing new domain / agent / service component.

6.6 Agent Component

The *Agent Component* [57] is designed for the pre-defined set of principles and purpose. The *Agent components* act as an intelligent process assistant and moreover they are web enabled software components. The agent components can be integrated with some relevant *domain components* for information retrieval purpose. The block schematic for a service component is shown in figure 6.3. The elements of an Agent Component include the following:

1. Security and Integration with Domain / Service Component.
2. Background active failure Management.
3. Access Control.
4. Interoperability Management.

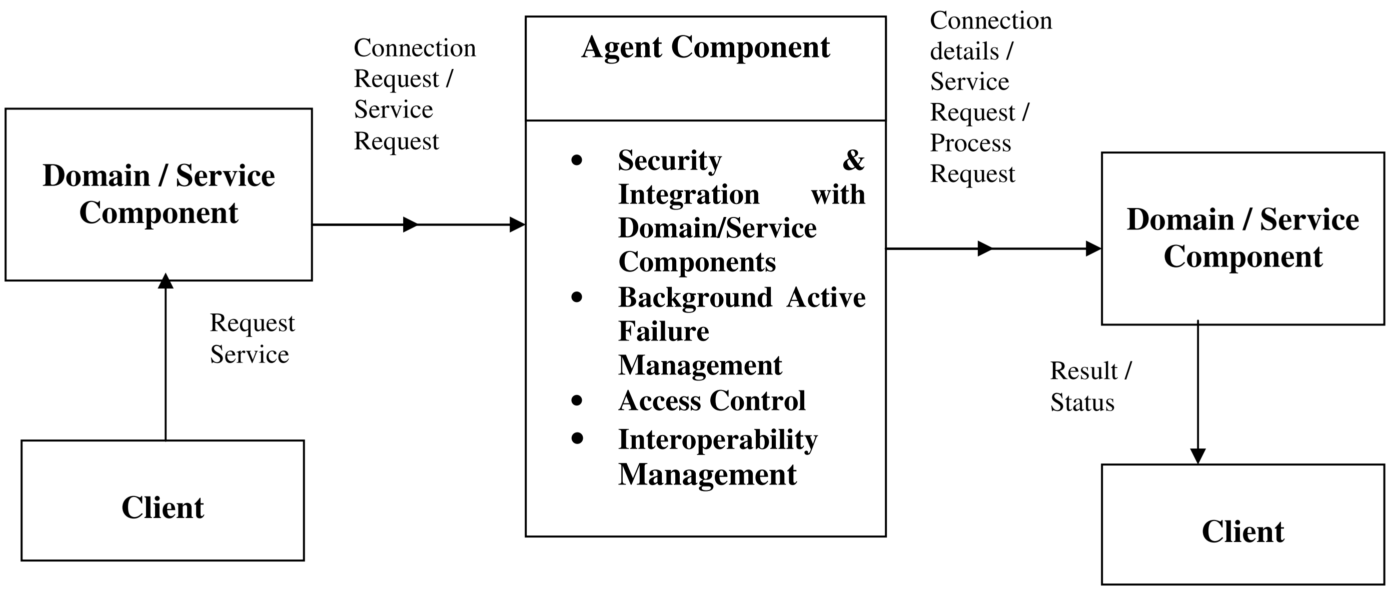
Domain / Service Component
Client
Request Service
Connection Request / Service Request
Agent Component
Security & Integration with Domain/Service Components
Background Active Failure Management
Access Control
Interoperability Management
Connection details / Service Request / Process Request
Domain / Service Component
Result / Status
Client

Figure 6.3: Block Schematic of Agent Component

1. **Security and Integration with Domain / Service Component:**

   A client can request for a service / process request from *Domain Component / Service Component*. The *Domain / Service* Component routes the request to the *Agent Component*. The Agent Component ensures confidentiality by hiding the information of the client to *the Domain / Service Component*. However, it initiates execution of valid client transaction by the *domain / service component* and finally routes the results / status through the *Domain / Service Component* to the client.

2. **Background active failure Management:**

   The *Agent component* supports the failure management in order to avoid the loss of information during the process of execution. Therefore there will be background active failure management run which will take care of this type failure during the processing.

3. **Access Control:**

   The *Agent component* validates the client request routed through *Domain / Service Component* and checks the authenticity and validity of the client request. It establishes connection to the *Domain / Service Component* only for valid users and request. Illegal transactions / users are blocked and exception routines are invoked.

4. **Interoperability Management:**

   The *Agent Component* provides support for interoperability by using open protocols for information storage and transmission across heterogeneous systems in a distributed environment.

6.7 Tasks in design of Service Component

The main tasks in the design of *agent component* as perceived in the present work are shown below:

1. Define the functions to be performed by an Agent.

2. Collect the details of services requested by the client through domain / service components.
3. Look for a set of functions that are common with existing agent components.
4. Look for a set of functions that are different / unique in the design of the current agent, and tabulate them.
5. Formulate the constraints for the agent, under study, as per nature of the application. The rules can be formed for object request broker or protocol security. This implementation constraint can be based on available technologies.
6. A universal type of agent is created to make the communication easier, using open standards for communication and information exchange.
7. Potential risk / problem areas in the agent operation are identified and tabulated.
8. Proceed to design the classes for the agent, under study, keeping in mind, the interoperability feature across heterogeneous computer systems and network configurations.
9. After completing the design of the agent, proceed to catalog and store the reusable classes for distribution in an organized way.
10. Record the protocol used for communication and information exchange for the agent under study.

11. Give access rights / permissions for application developers to use the reusable classes while designing new agent components.

6.8 Summary

This chapter discussed about the design steps involved for three types of components, namely Domain, Service and Agent. The inputs, key functions and the outputs for each type of component are highlighted and the interactions among them are also specified.

Chapter 7
Conclusion, Contributions and Future Work

The Present work has been carried out to meet the *objectives* and *scope* outlined in the first chapter. The significance of *semantic integrity control* and *interoperability* in component based software development has been dealt in greater detail in the successive chapters.

Chapter 2 presented a component based approach for enforcing *semantic integrity control* in a distributed multi-database system. The core component has been designed to handle three separate interfaces namely, *User, Administrator* and *Database Handler*. The user interface is the starting point of access to the core component. The administrator interface handles access control privileges for users and local databases. The database handler performs global schema management and site management. This approach helps a component designer very much in enforcing semantic integrity control for database components.

Chapter 3 focused on a *generic model* for searching multiple databases in a heterogeneous environment. *Interoperability* is achieved at the level of heterogeneity in database type and system configuration (hardware, software) of the host servers in a distributed environment. The proposed approach can be successfully adapted to practical applications involving distributed multi-database systems.

Chapter 4 dealt in detail a *mathematical representation* to verify queries for database components. This will help a database component designer to confirm the correctness of a user's input queries and verify them mathematically as well as experimentally.

Chapter 5 analyzed the *key factors* responsible for maintaining *semantic integrity control* and *interoperability* in the design of components. This approach will be very much useful to a component designer in understanding the requirements of Domain, Service and Agent based applications.

Chapter 6 presented the *design steps* involved for the three types of components*: Domain, Service* and *Agent*. The component designer will be able to specify the inputs, key functions and outputs for each types of component. The interactions among them are also specified.

7.1 Contributions

The specific contributions relating to the present research work are highlighted here.

- A Core Component for a distributed multi-database system comprising of three interfaces namely, User, Administrator and Database Handler has been proposed. The User Interface component provides flexibility for any network user to frame a query through uniform interface or submit a set of queries from a query file. The administrator Interface sets access control privileges for users and local databases. The Database Handler interface allows Insert, Delete and Modify operations on pre-conditions and post-conditions for one or more local databases. The proposed design offers a higher level of modularity and security.

- A generic algorithm for accessing multiple databases in distributed environment has been presented. This generic approach simplifies the search process involving multiple databases, residing at various sites, in a network of heterogeneous systems. This way, a higher level of interoperability is achieved.
- The mathematical representation of pre and post conditions for the database component will help a component designer to frame SQL queries precisely and verify their correctness using predicate logic.
- Identification and Analysis of *key factors* pertaining to *semantic integrity control* and *interoperability* will be helpful to a component designer in the Analysis Modeling phase for *Domain, Service and Agent* components.
- The design steps for *Domain, Service* and *Agent* have been proposed.

7.2 Directions for Future Work

The possible extensions to the present work may be listed as follows:

- Provide schema viewing using multiple windows, for the distributed multi-database system, as per the access privileges available to the component administrator.
- The proposed design for Domain, Service and Agent components can be implemented using Java API & Java and ADO .NET technology.
- Handle queries expressed in informal English-like statements for information retrieval from a distributed multi-database system. In such cases, it is necessary to deduce the predicate logic from the user input specification expressed in English-like statement and then initiate query execution.
- Extend the component framework to handle vagueness for user queries using Fuzzy Logic.
- The service component can enhanced to offer the services of Semantic Web. More specifically, web ontology can be used to describe the following technologies:
  - A global naming scheme (URIs).
  - A standard syntax for describing data (RDF).
  - A standard means of describing the properties of data (RDF Schema).

  - A standard means of describing relationship between the data items (Web Ontology Language).

- Agent component design can be extended to incorporate collaboration, negotiation and reconfiguration capabilities.
- Domain component design can include features to build a knowledge base of domain expertise that can be inferred from domain experts.

Appendix A

System Configuration

The present research work has been carried out using the following system configuration:

1. **Computer Programming Laboratory:** 72 nodes in LAN, comprising of Acer / IBM PCs, Print Server, Windows and Linux OS, Internet access using 100Mbps and Software tools under Ubuntu Linux and Windows.

   The typical configuration for each PC is as follows:

   Intel Pentium 4 : 2.8 GHz ACPI.

   512 MB RAM.

   40 GB Hard Disk.

   1.44 MB Floppy Disk.

   CD ROM Drive.

   Boardcom Net Xtreme Gigabit Ethernet Network Card.

   InterR 82865 Graphics Controller Display Adapter.

2. **IXWebhosting.com (76.162.254.156):** The following are the software loaded in the Server:

   MySQL 300, SQL 300, PSQL ,Java, J2EE, #C, ASP.NET, PHP, CGI.

3. **Visionwebhosting.com (70.87.57.146):** The following are the software loaded in the Server:

   MSSQL300, PSQL, Java, Visual Studio 2008, J2EE.

Appendix B

Functions and their Actions for enforcing Semantic Integrity Control Using .NET Framework

FUNCTIONS	ACTIONS
Public Function **Substring**(ByVal *startIndex* As **Integer**) As **String**	Retrieves a substring from this instance. The substring starts at a specified character position.
Public Function **Contains**(ByVal *value* As **String**) As **Boolean** Member of **System**.**String**	Returns a value indicating whether the specified System.String object occurs within this string.
Public Overrides Function **ToString**() As **String** Member of **System**.**Boolean**	Converts the value of this instance to its equivalent string representation.
Public ReadOnly Property **Length**() As **Integer** Member of **System**.**String**	Gets the number of characters in the current System.String object.
Public Function **IndexOf**(ByVal *value* As **Char**) As **Integer** Member of **System**.**String**	Reports the index of the first occurrence of the specified Unicode character in this string.

public virtual **string ValidationGroup** {public get; public set; } Member of **System.Web.UI.WebControls.TextBox**	Gets or sets the group of controls for which the System.Web.UI.WebControls.TextBox control causes validation when it posts back to the server. Return Values: The group of controls for which the System.Web.UI.WebControls.TextBox control causes validation when it posts back to the server. The default value is an empty string ("").
Public Function **Trim**(ByVal *str* As **String**) As **String** Member of **Microsoft.VisualBasic.Strings**	Returns a string containing a copy of a specified string with no leading spaces (LTrim), no trailing spaces (RTrim), or no leading or trailing spaces (Trim). Parameters: *str*: Required. Any valid String expression. Return Values: Returns a string containing a copy of a specified string with no leading spaces (LTrim), no trailing spaces (RTrim), or no leading or trailing spaces (Trim).
Public Function **UCase**(ByVal *Value* As **Char**) As **Char** Member of **Microsoft.VisualBasic.Strings**	Returns a string or character containing the specified string converted to uppercase. Parameters: *Value*: Required. Any valid String or Char expression. Return Values: Returns a string or character containing the specified string converted to uppercase.
Public Function **StrComp**(ByVal *String1* As **String, **ByVal *String2* As **String,** Optional ByVal *Compare* As **Microsoft.VisualBasic.CompareMethod** = Binary) As **Integer** Member of **Microsoft.VisualBasic.Strings**	Returns -1, 0, or 1, based on the result of a string comparison. Parameters: *String1*: Required. Any valid String expression. *String2*: Required. Any valid String expression.

	Compare: Optional. Specifies the type of string comparison. If Compare is omitted, the Option Compare setting determines the type of comparison. Return Values: If String1 sorts ahead of String2, StrComp returns -1. If String1 is equal to String2, StrComp returns 0. If String1 sorts after String2, StrComp returns 1.

Appendix C

Functions and their Actions for enforcing Interoperability Using .NET Framework

FUNCTIONS	ACTIONS
public enum **UriComponents** Member of **System**	Specifies the parts of a System.Uri. This is used to connect to the component.
public class **TextBox** : **System.Web.UI.MobileControls.TextControl** Member of **System.Web.UI.MobileControls**	Provides a text-based control that allows the user to enter text.
public class **Button** : **System.Web.UI.WebControls.WebControl** Member of **System.Web.UI.WebControls**	Displays a push button control on the HTML based controls.
public class **ExtenderControlDesigner** : **System.Web.UI.Design.ControlDesigner** Member of **System.Web.UI.Design**	Provides UI support for working with extender controls at design time.
System.Web.UI.DataSourceView **GetView**(**string** *viewName*) Member of **System.Web.UI**.**IDataSource**	Gets the named data source view associated with the data source .control.
Public Shared Sub **WriteAllText**(ByVal *path* As **String**, ByVal *contents* As **String**, ByVal *encoding* As **System**.**Text**.**Encoding**) Member of **System**.**IO**.**File**	Creates a new file, writes the specified string to the file using the specified encoding, and then closes the file. If the target file already exists, it is overwritten.

public interface **IStateManager** Member of **System.Web.UI**	Defines the properties and methods any class must implement to support view state management for a server control.
public **System.ComponentModel.ISite** **Site** { get; set; } Member of **System.ComponentModel**.**IComponent**	Gets or sets the System.ComponentModel.ISite associated with the System.ComponentModel.IComponent.
System.Web.UI.DataSourceView **GetView**(**string** *viewName*) Member of **System.Web.UI**.**IDataSource**	Gets the named data source view associated with the data source control.

Appendix D
Functions and their Actions for enforcing Interoperability Using JDBC & Java API

FUNCTIONS	ACTIONS
class FetchFrame extends JFrame public FetchFrame(String tblname)	Fetching the table for data retrieval after relating the global schema with the local site component schema.
ds.getConnection()	The site handler derives the table after querying the local site database table and mapping the attributes with global schema.
Class.forName("mssql.jdbc . driver. mssqldriver")	Database querying and processing table through linking libraries of java implementations.
public void totable(String table) throws SQLException,ClassNotFoundException	Provides SQL exception.
Class.forName("com.mysql.jdbc.Driver").newInstance();	Creation of odbc driver is being established for mysql
DriverManager.getConnection ("jdbc:mysql://localhost/root?user=root&password=password");	The link to MySQL is being established through java database connectivity.

References

[1] Aho, Sethi, Ullman, "Compilers-Principles, Techniques and Tool", Addison Wesley Publing Company, 1996, 32-55.

[2] Meyer, Bertrand, "Eiffel: Programming for Reusability and Extendability", SIGPLAN Notices, vol. 22, February 1987, 85-94.

[3] Brown, Alan W., Wallnau, Kurt C, " The Current State of CBSE", IEEE Software September/October 1998, 37-46.

[4] Broy, Manfred et al., "What Characterizes (Software) Components? Software Concepts & Tools" (1998), 49-56, ISSN 1432-2188, Springer Verlag, June 1998.

[5] Szyperski, Clements, "Component Software, Beyond Object-Oriented Programming", Addison-Wesley, 2002, 23-56.

[6] The Object Management Group, Inc. (OMG), the Common Object Request Broker: Architecture and specification, Chapter 3: OMG IDL Syntax and Semantic, Minor revision 2.3.1: October 2002, http://www.omg.org/cgi-bin/doc.

[7] Weck, Wolfgang, Inheritance "Using Contracts & Object Composition", Proceedings of the second International Workshop on Component-Oriented Programming WCOP'99, Turku Centre for Computer Science, General Publication no 5, 1999.

[8] Wagner F., Wolstenholme P.: "Modeling and Building, Reusable Software. Proc. Of the 10th IEEE International Conference and Workshop on the Engineering of Componet Based Systems, Huntsville, USA, 2003..

[9] Object Constraiunt Language specification: [online] Available from : http://www-4.ibm.com/software/ad/standards/ocl.htm.

[10] Bengnard, Antoine, Je'ze'quel, Jean-Marc, Poluzeau, Watkins, "Making Components Contract Aware", IEEE Computer, July 1999, 37-45.

[11] Gray T. Leavens and Murali Sitaraman, Foundations of Component-Based Systems, Cambridge University Press, 2000.

[12] Peltzer D. .Net & J2EE Interoperability, November 2003.

[13] Browning, D. Integrate .NET Remoting into the Enterprise Windows Server System, Available: http://www.ftponline.com/wss/2002_11/magazine/features.

[14] Lewis, Grace A., Wrage, Lutz. Approaches to Constructive Interoperability (CMU/SEI-2004-TR-020). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.

[15] David Carney, David Fisher, Ed Morris. Some current approaches to Interoperability (CMU/SEI-2005-TN-033). Pittsburg, PA: Software Engineering Institute, Carnegie Mellon University, 2005, 2-7.

[16] Crnkovic I. and M. Larsson. 'Challenges of Component-Based Development'. The Journal of Systems and Software: 2002, 201-212.

[17] Yi Jiao, Baifeng Wu, Kun Zhu and Qiang Yu, " Towards a Systematic Conflict Resolution Policy in Multi-agent System: A Conceptual Framework", Lecture Notes in Computer Science, Springer Berlin, Volume 3865/2006, 274-283.

[18] Rick Neol, "Scale Up in Distributed Databases: A Key Design Goal for Distributed System, May 17, 2004.

[19] Christoph Quix, David Kensche, Xiang Li: "Generic Schema Merging", Advanced Information Systems Engineering, 19th International Conference.

CAISE2007, Trondheim, Norway, Proceedings: Lecture Notes in Computer Science June 11-15, 2007, 127-141.

[20] James F. Kurose and Keith W. Rose "Components of a Distributed Database System, May 2004, 78-91.

[21] Roger S. Pressman, "Software Engineering: A Practitioner Approach", 7th Edition, McGraw-Hill publisher, 2009.

[22] David Kensche, Christoph Quix, Yong Li and Matthias Jarke, "Generic Schema Mapping", Lectre Notes in Computer Science: Conceptual Modeling-ER2007, Springer Berlin, and Volume 4801/2007.

[23] A.J.H. Peddernors and L.O. Hertzberger, "A High Performance Distributed Database System for enhanced internet services", Lecture Notes in Computer Science, Volume 1401/1998, 467-478.

[24] Michale Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pifer, Adam Sah, Jeff Sidell, Carl Staelin, Andrew Yu, Mariposa: A Wide-area Distributed Database System", The VLDB Journal", Volume 5, 1996.

[25] K. P. Birman, T. A. Joseph, "Exploiting Replication in Distributed Systems", ACM Press, New York, NY, 1990.

[26] Andrew S. Tanenbaum and Marten Van Steen, "Distributed Systems, Principles and Paradigms", Chapter 7 Fault Tolerance, 2004, 183, 362.

[27] Thaleheim, B., "Component Construction of Database Schemes", in Proceeding ER'02, Lecture Notes in Computer Science 2503, 2002. pp: 20-34.

[28] Yi Jiao, Baifeng Wu, Kun Zhu and Qiang Yu, " Towards a Systematic Conflict Resolution Policy in Multi-agent System: A Conceptual Framework", Lecture Notes in Computer Science, Springer Berlin, Volume 3865/2006, 274-283.

[29] Sun Microsystems, Java Community Process, [Online] Available from: http://jcp.org/aboutJava/communityprocess/jsr/asrt_prop.html.

[30] BenoyJose, “Javaboutique”, [Online] Available from: http://www.javaboutique.internet.com/tutorials/assertions.

[31] Sun Microsystems, Java Boutique, [online] Available from: http://javaboutique.internet.com/tutorials/assertions/internal_invariants.html.

[32] Johnson,James C, EBSCO Electronic Database, “Black Enterprise”, Vol.35, Iss. 2, Apr 2005, 52-54.

[33] Xi (Michael) Zhang, Tony C. Pan, Umit V. Catalyurek, Tahsin M. Kurc, Joel H. Saltz, "Serving queries to multi-resolution datasets on disk-based storage clusters", Proceedings of the 4th IEEE International Symposium on Cluster Computing and the Grid, 2004 (CCGrid 2004), 2004, 490-498.

[34] Ullman J.D., “Principles of Database Systems”, 2nd edition, W H Freeman & Co., NY, 2001, 45-56.

[35] Ellsworth,Abigail, “Databases”, EBSCO Electronic Database, Radcliff,Carolyn Jerence & User Services quarterly, Vol.41, Issue3,spring 2002, 276-278.

[36] MYSQL, “The MYSQL Open Source Database”, [Online]. Available from: www.mysql.com.

[37] POSTGRESQL, “The POSTGRES Open Source Database”, [Online]. Available from: www.postgresql.org.

[38] George Reese, “Database Programming with JDBC and Java”, OReily Publications, 1st Edition June 1997, 40-57.

[39] JAVA by API, [Online]. Available from: www.java2s.com.

[40] Sun Developer Network (SDN), “The Source for JAVA Developers”, [Online]. Available from: www.java.sun.com.

[41] .NET Framework Developer's Guide, Microsoft Visual Studio 2008 / .NET framework 3.5, “ADO .NET Architecture”, [Online] Available from: http://msdn.microsoft.com/en-us/library/27y4ybxw(VS.71).aspx.

[42] K. Bergner, A. Rausch, and M. Sihling, Componentware The bit picture, 20th ICSE Workshop on Component-Based Software Engineering, Japan, 1998.

[43] S. S. Bhattacharya, P. K. Murthy, and E. A. Lee, Synthesis of embedded software from synchronous dataow speci_cations, J. VLSI Signal Processing, Kluwer Academic Publication, 1999, 155-66.

[44] C. Szyperski, Component Software. ACM Press, Addison-Wesley, 1999, 23-31.

[45] Jezequel, J.M., Meyer, B: Design by contract: The lessons of Ariane. Computer IEEE 30, 1997.

[46] Kung-Kiu Lau and Mario Ornaghi: A Formal Approach to Software Component Specification. Proceedings of Specification and Verification of Component-based Systems Workshop at OOPSLA, 2001.

[47] Sitaraman M, T. J. Long, B. W. Weide, J. E. Harner, and L. Wang. A formal approach to component-based software engineering: education and evaluation. In *Procs. of the International Conference on Software Engineering*, IEEE, Toronto, Canada, 2001, 601-609.

[48] Bowen, J.P., Hinchey, M.G.: Ten Commandments of formal methods. IEEE Computer, Volume 28, Issue 4, ISSN: 0018-9162, 1995, 56-63.

[49] Tom Portfolio and John Russell, PL/SQL User’s Guide and Reference, Release 9.0.1, 2001.

[50] John F. Sowa, Joerg H. Siekmann: Conceptual Structures: Current Practices, Vol.835, Springer-Verlag New York, LLC, 1994, 67-88.

[51] M. K. Venkataraman, “Discrete Mathematics”, The National Publishing Company, 1st Edition, 2000. 9-9.

[52] M. Madiajagan and Taher, “A report on Online Faculty Feedback system”, BITS, Pilani-Dubai, 2005, 15-30.

[53] M. Madiajagan, “Telephone Directory Information System”, 2006, [online] available from : http://www.msav.net/

[54] AUTOLIB, “ Library Management System Software”, AUTOLIB System, Chennai, 2005.

[55] .NET Framework Developer's Guide, Microsoft Visual Studio 2008 “Interoperability principles”, [Online] available: http://msdn.microsoft.com/en-us/library/.

[56] George T. Heineman, William T. Councill, “Component-Based Software Engineering Putting the Pieces Together” Addison-Wesley Pearson Education, May 2001, 79-97 & 641-648.

[57] Hafedh Mili, Ali Mili, Sherif Yacoub, Edward Addy, “Reuse-Based Software Engineering Techniques, Organization and Controls”, John Wiley & Son, Inc. 2002, 124-136.

[58] Mili, A., and S. Yacoub, “A comparative analysis of domain engineering methods: a controlled case study,” 22nd International Conference of Software Engineering, Limerick, Ireland, June 4-11, 2000.

List of Publications

Journal Papers

1. M. Madiajagan, B. Vijayakumar, “Interoperability in Component Based Software Development”, Transactions on Engineering, Computing and Technology, Enformatika, Volume16, November 2006, ISSN 1305-5313, Pages: 207-215.
2. M. Madiajagan, B. Vijayakumar, Anand Oswal, “Testing Prre and Post Conditions with Predicate Logic for Database Components”, CURIE 2009, Birla Institute of Technology and Science, Pilani (Rajasthan), Volume 2, Issue 3, October 2009, ISSN No.: 0974-1305, pages: 53-58.
3. M. Madiajagan, B. Vijayakumar, Sri Hariharan S, “Component Approach to Software Development for Distributed Multi-Database System”, World Scientific and Engineering Academy and Society (WSEAS) Journal of Computing, under review.
4. M. Madiajagan and B. Vijayakumar, “Design considerations for Domain, Service and Agent Components for Web Applications Development”, communicated to Informatica Economica Journal and under review.

Conference Papers

1. M. Madiajagan, B. Vijayakumar, “Semantic Integrity Control in Component Based Software Development”, Emerging Applications of Information Technology, First International Conference on Emerging Application of Information Technology (EAIT 2006), Feb 10-11, 2006, Elsevier, ISBN 10: 81-312-0445-6, pages: 371-374.
2. M. Madiajagan, B. Vijayakumar, Barkha Bhagwant Keni, “ A Generic Model for Querying Multiple Databases in a Distributed Environment Using JDBC and an Uniform Interface”, World Congress on Engineering and Computer Science (WCECS 2007) International Association of Engineers (IAENG), ISBN: 978-988-98671-6-4, pages: 280-284.

Biography of the Candidate

M. Madiajagan holds a M.S., in Software Systems from BITS, Pilani, India. He has 14 years of College / University teaching experience in CSE (Women's College Pondicherry University, India and BITS, Pilani-Dubai, UAE) and 2 years of experience in Blue Chip Software Company. Presently, he is working as Senior Lecturer, CS, BITS, Pilani-Dubai. His areas of interest include Component Based Software Engineering, Distributed Database Systems, Software Architecture, and Theory of Computation. He is a member of Professional bodies ACM, World Enformatica Society and Computer Society of India. He is actively involved in judging committee member in annual students' technical event TECHNOFEST at BITS, Pilani-Dubai. He is also an active member of BITS, Pilani-Dubai Campus Placement Programme and in-charge of Online Feedback System, BITS, Pilani-Dubai.

Biography of the Supervisor

B. Vijayakumar holds a Ph.D. in Computer Science from BITS, Pilani, India in 2001. He has 18 years of University teaching experience in CSE (National Institute of Technology, Tiruchirappalli, India and BITS, Pilani-Dubai, UAE) and 6 years of experience in computer industry. Presently, he is working as Associate Professor, CS, BITS, Pilani-Dubai. His areas of interest include Distributed Database Systems, Component Based Software Engineering, Web Mining, Multimedia Systems and Open Source Software Development. He is member of Professional bodies ISTE (Indian Society for Technical Education), World Enformatica Society and Staff Advisor for Linux User Group, BITS, Pilani-Dubai. He is actively involved as organizing and judging committee member in annual students' technical event TECHNOFEST at BITS, Pilani-Dubai. He has been involved in co-ordination and coaching the students of BITS, Pilani-Dubai for annual UAE National Programming Contest since 2005.