Component Approach to Software Development for Distributed Multi-Database System

Madiajagan MUTHAIYAN, Vijayakumar BALAKRISHNAN,
Sri Hari Haran.SEENIVASAN,
Computer Science Department, BITS-Pilani, Dubai, UAE,
jagan@bitsdubai.com, vijay@bitsdubai.com, bithari.sri@gmail.com

The paper deals with 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 Databases 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.

Keywords: *Assertion, Core Component, Component Based Software Development, Multi-Database, Schema*

1 Introduction

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 [1]. A distribute multi-database system offers scalability and reliability in modern information systems. This paper focuses on a component-based approach to software development involving retrieval operations from multi-databases.

2 Related Work

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 [2]. The idea of Software Engineering that encourages decomposition [3] of a system into logical units improves the scalability of the system, and it is very much applicable to multi-database system. The present work deals with component design aspects for a multi-database system.

3 Component Design for a Multi-Database System

The component design for a Multi-database does involve a core component with three supported interfaces 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. Multiple *User Interface* components 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.

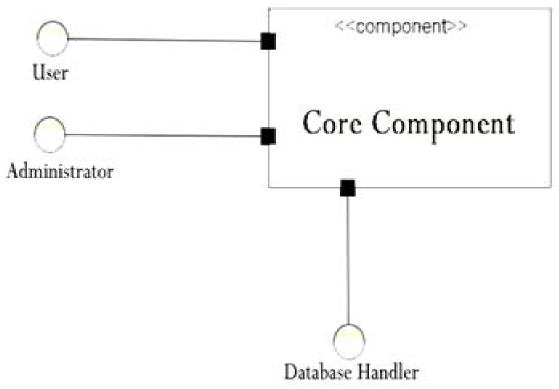

Fig. 1. Core Component for Multi-Database System

The component for the Multi-database is designed to support three main types of interfaces, namely, Component User, Component Administrator and Database Handler as shown in Figure 1.

A. 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.

B. 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.

C. 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.

4 Problem Description

The Core Component for Multi Database System (shown in figure 1) performs semantic integrity control and retrieval of information using Database Handler. The Database Handler Component shown in figure 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.

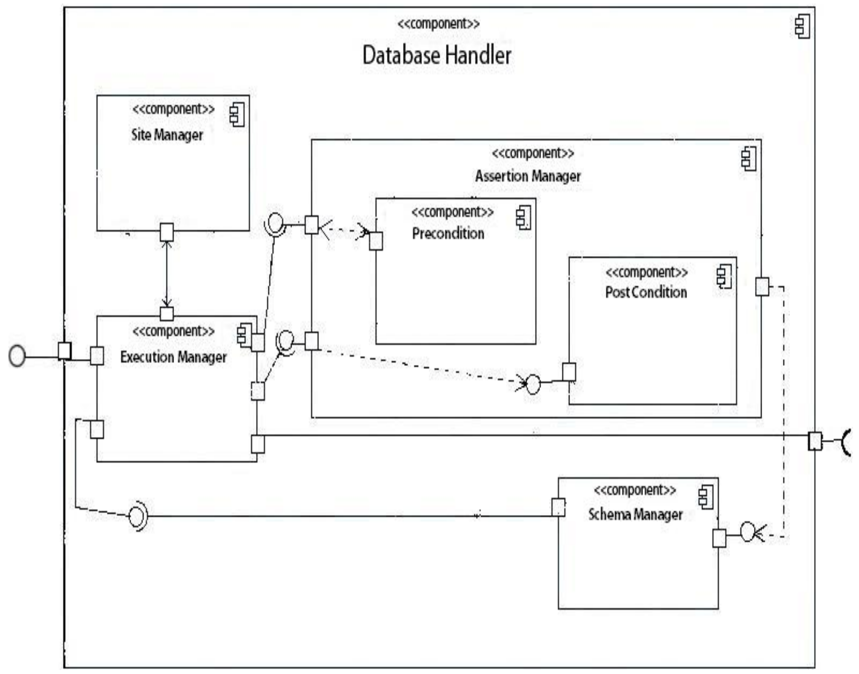

Fig. 2. Block Schematic for Database Handler using UML 2.0

5 Database Handler

The Database Handler has the following sub-components:

- Schema Manager.
- Site Manager.
- Execution Manager.
- Assertion Manager.

The following subsections describe their functions in detail.

A. 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 [4]. 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 and extensional mappings [5]. 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.

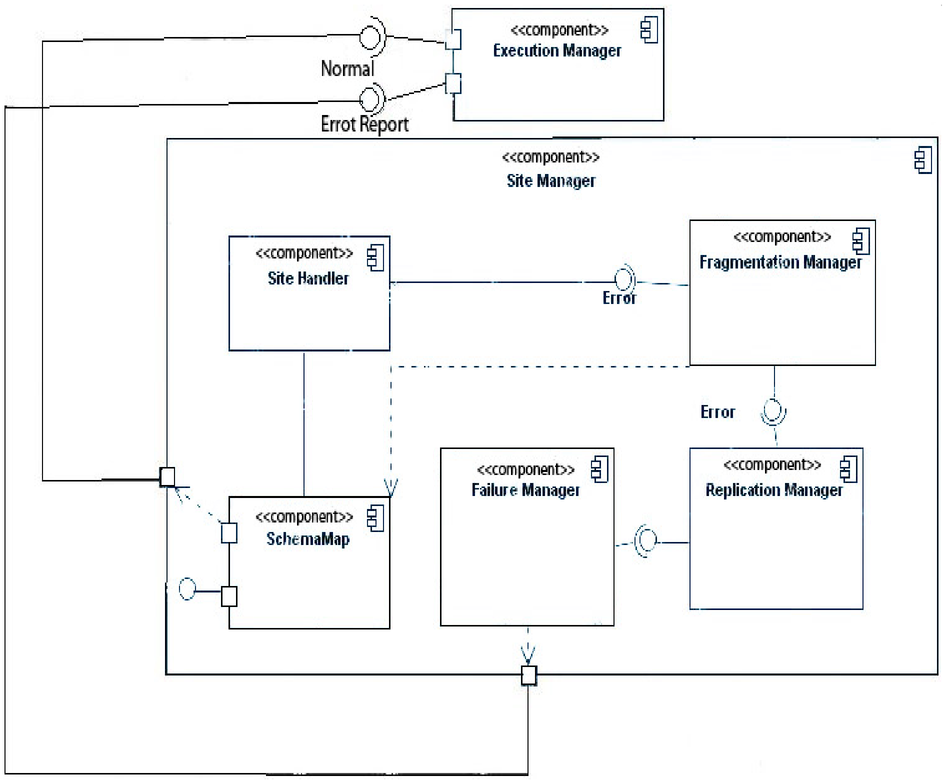

Fig. 3. Block Schematic for Site Manager using UML 2.0

B. Site Manager:
The Site Manager is made up of five sub components as shown in figure 3:
- Site Handler.
- Fragmentation Manager.
- Replication Manager.
- Failure Management.
- Schema Map.

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 [6]. 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.

Fragmentation Manager:
The individual databases in a Multi-database system may be partitioned horizontally or vertically 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 [7]. This translation is done with the help of the Fragmentation Manager and Replication manager.

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 [8].

Failure Management:
This component is invoked by the Site Handler when an error is encountered [9]. 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.

Schema Map:
The Schema Map sub component provides an interface to the Execution Manager. It plays an important role in converting the global schema 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 [10].

C. 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.

D. 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 semantic error can be categorized as follows:
1. Errors arising due to naming conflicts [11].
2. Errors arising due to policy conflicts.
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.

6 Pre-conditions

The asserting statement undergoes check for preconditions. Preconditions [12] 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*> and <*not*> operators. The global schema is consulted while checking the correctness of names relating to DBMS, tables, attributes and key constraints.

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 substring between lexer pointer of input string and the current pointing position with the symbol table elements.

```
{
2.4.2.2 indx <- next position 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 position 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
```

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

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.

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.
```

8 Implementation

The implementation has been carried using the following system configuration:

- Computing Systems Laboratory comprising of 72 nodes in a network of windows and Unix based workstations.
- **IXWebhosting.com (76.162.254.156):** The following are the software loaded in the Server: MySQL, SQL, PSQL ,Java, J2EE, #C, ASP.NET, PHP, CGI.
- **Visionwebhosting.com (70.87.57.146):** The following are the software loaded in the Server: MSSQL, PSQL, Java, Visual Studio 2008, J2EE.

9 Conclusion

This paper dealt in detail 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.

References

[1] R. Neol, *Scale Up in Distributed Databases: A Key Design Goal for Distributed System*, May 17, 2004.
[2] J. F. Kurose and K. W. Rose, *Components of a Distributed Database System*, May 2004.
[3] R. S. Pressman, *Software Engineering: A Practitioner Approach*, 7th Edition, McGraw-Hill, 2009.
[4] C. Quix, D. Kensche and X. Li, "Generic Schema Merging, Advanced Information Systems Engineering," *19th International Conference, CAISE2007*, Trondheim,

Norway, pp. 127-141, June 11-15, 2007.

[5] D. Kensche, C. Quix, Y. Li and M. Jarke, "Generic Schema Mapping," *Lecture Notes in Computer Science: Conceptual Modeling-ER2007*, Springer Berlin, Vol. 4801, 2007.

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

[7] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pifer, A. Sah, J. Sidell, C. Staelin and A. Yu, "Mariposa: A Wide-Area Distributed Database System," *The VLDB Journal*, Vol. 5, 1996.

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

[9] A. S. Tanenbaum and M. Van Steen, *Distributed Systems, Principles and Paradigms*, Chapter 7 Fault Tolerance, pp: 362.

[10] B. Thaleheim, "Component Construction of Database Schemes", *Proceeding ER'02, Lecture Notes in Computer Science*, No. 2503, pp: 20-34, 2002.

[11] A. S Tanenbaum and M. Van Steen, *Distributed Systems, Principles and Paradigms*, Chapter 4 Naming, pp. 183.

[12] Sun Microsystems, Java Boutique, [Online] Available at: http://javaboutique.internet.com/tutorials/assertions/internal_invariants.html.

Madiajagan MUTHAIYAN holds a M.S., in Software Systems from BITS, Pilani, India and a PhD student from BITS, Pilani-India. He has 14 years of College / University teaching experience 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.

Vijayakumar BALAKRISHNAN 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.

Sri Hari Haran.SEENIVASAN is presently a final year student in B.E Computer Science at BITS, Pilani-Dubai. He has experience in IT section at ETA- M&E Company in Dubai during summer term / practice school in the year 2008. He has skills in LINUX and .NET based software applications development.