4.1 The Computational Model for Generic Framework ........................................... 78

4.2 Implementation Scenario ................................................................. 95

4.3 Conclusion .................................................................................... 121

References ....................................................................................... 121
Chapter 4

Detailed Methodology for Development of Framework

This chapter discusses the detailed methodology to implement the generic framework for multi-agent knowledge-based system accessing distributed database grid. The generic framework is leveraging the power of both the technologies: the agent and the grid. Also, on top of agent-enabled data grid environment, the framework integrates the knowledge-based component to provide the analysis and decision making functionalities. This chapter covers implementation details regarding OGSA-DAI (Open Grid Services Architecture - Data Access and Integration) and multiple agents. It also provides necessary screen layouts and code snippets to explain the development details of the framework.

Section 4.1 of the chapter demonstrates the computational model for generic framework. The computational model comprises with three entities: heterogeneous and geographically scattered data resources, data grid core services and multi-agent system environment. Section 4.2 covers the implementation scenario of the framework. The computational model demonstrates the three major types of agents: data grid service agents, domain specific agents and fuzzy interface agent. In this section, the configuration and implementation details about typical data grid service agents and OGSA-DAI are discussed by providing necessary screen layouts and code snippets. The implementation details regarding domain specific agents and fuzzy interface agent will be discussed in the next chapter.

In this chapter, the implementation scenario has been discussed and demonstrated for the data grid typical agents namely resource management agent, data access & integration agent and grid administrative agent. These three agents are generic and can be directly implemented with little modification in any domain specific application. In order to provide the data grid service, they access the OGSA-DAI services. The configuration details for OGSA-DAI are also discussed in detail. Agent is a software entity that works automatically and proactively on behalf of its users. The resource management agent automates the task of deployment and undeployment of heterogeneous and geographically...
distributed databases in the data grid environment through OGSA-DAI server. The data access & integration agent provides the data access, data transformation, data integration and data delivery requests. The grid administrative agent is a proactive agent and it is responsible to provide grid information service and data backup service. These three agents are possessing full or partial characteristics of typical agents.

4.1 The Computational Model for Generic Framework

Today’s modern scientific and commercial applications involve accessing heterogeneous databases in a geographically distributed environment. Therefore, there is an arising demand on integration of different data resources across different administrative domains to provide a uniform access interface for client applications and users. To realize said scenario, the generic framework for integration of multi-agent knowledge-based system and distributed data grid is presented. The agents of the generic framework expose heterogeneous data resources from and to a data grid via grid middleware and provide a uniform access interface and efficient query mechanism.

The available grid middleware technologies have been studied and OGSA-DAI (Open Grid Services Architecture - Data Access and Integration) is used to implement data grid.

OGSA-DAI is an abbreviation of Open Grid Services Architecture - Data Access and Integration. As its name suggests, it is a data access and integration middleware for data grid services using the Open Grid Services Architecture (OGSA). OGSA-DAI provides a way to publish and share data resources across organizational boundaries. It is used to develop applications which use both public and personal data resources, through a secure, extensible framework based on web service standards [6]. OGSA-DAI is an open source Java based middleware that allows to expose distributed data originating from relational databases, XML databases and files to be accessed, updated, transformed and combined. Web services are used as a mediator to provide an efficient access mechanism. The framework provides an extensible interface through which a client application can access the data stored in geographically distributed relational databases without being aware of their physical locations.

The computational model of the generic framework is presented in Figure 4.1. The main components of the computational model are a set of heterogeneous and geographically scattered data resources, the OGSA-DAI core services and the user services provided by
multiple collaborative agents that are working as a part of a multi-agent system. In the following sections, each component of the computational model is discussed in detail.

Figure 4.1: The Computational Model for Generic Framework
4.1.1 Heterogeneous and Geographically Scattered Data Resources

Modern data intensive systems require cross-functional data in near real time. This data resides in a structured form and exist in multiple, heterogeneous and scattered data resources i.e. databases, which may originate from different administrative authority domains. These databases are often accessed by a communication network system including LAN (local area network), WAN (wide area network) and wireless network. These data resources are the minimal element of the generic framework. These data resources are accessed by client applications and users via agents. The data grid environment can contain several relational heterogeneous databases provided by different database technology vendors. These data resources are to be exposed via grid middleware services. Currently, the generic framework is able to integrate the following types of databases:

- Relational data resources, e.g. Oracle, MySQL, SQL Server, DB2 etc.
- XML data resources, e.g. Xindice.
- Files data resources, e.g. files and directories.

4.1.2 The Data Grid Core Services

Nowadays, collaborative projects are seeking for the mechanism which provides data access and sharing of data across organizational boundaries or distributed databases. Instead of requiring, each of these projects individually solves the same data access and integration problem, the common middleware solution should evolve which allows uniform access to data resources using a service-based architecture. The core services offered by middleware solution are used to access the distributed and heterogeneous databases.

Open Grid Services Architecture - Data Access and Integration (OGSA-DAI) is a project that develops middleware solution to assist with access and integration of data from separate data resources via the data grid [4]. The project was funded by the UK e-Science core program and is working closely with the Global Grid Forum Data Access and Integration Services (GGF DAIS) work group [2]. We have used the OGSA-DAI 4.1 Axis version for implementation of data grid basic and core services. It offers a set of services to manage the data resources in the data grid environment. To implement data grid
environment, the grid middleware should be properly configured and used. We have implemented a set of typical agents which automates the configuration tasks of data resources in a grid environment. Section 4.2 discusses about this in detail.

OGSA-DAI aims to provide a uniform service interface for data access and integration of heterogeneous scattered databases exposed to the data grid environment. It also hides differences such as database driver technology, database types, data formatting techniques and delivery mechanisms. The basic and core services provided by OGSA-DAI can be used by the higher level services to offer greater functionality. The OGSA-DAI architecture supports for grouping multiple requests on an OGSA-DAI service into a single message sent to a service. This decreases latency by increasing the granularity of interactions. It also reduces both the number of messages exchanged and the quantity of data transferred [5].

During our study, we have identified that in any type of data grid, there are four important aspects need to be incorporated. They are data access, insert & update, data transformation, data delivery and data security. We have chosen OGSA-DAI as a grid middleware as it provides data access, insert and update in relational, XML and file-based data. We have also identified that it is having a variety of data transformation techniques like XSLT, ZIP and GZIP. Apart from this, it offers different data delivery mechanisms like SOAP over HTTP, FTP, GridFTP and E-mail.

We have identified that, most frequently used data resources in today’s modern scientific and commercial applications are traditional relational databases (Oracle, MySQL, SQL Server, DB2 and PostgreSQL), XML databases (Xindice, eXist) and file-based data. So, we have chosen OGSA-DAI as the core functional component of the computational model to assist in accessing and integration of disparate databases via the data grid. As it supports the above types of data resources, client applications and users are enabled to access and manipulate various databases through secure and transparent data grid system. They are also able to implement the grid interoperability characteristics, while hiding the underlying heterogeneity and dynamics of those databases [5]. OGSA-DAI framework executes workflows which are equivalent to programs or scripts and contain a set of activities. Each activity is a well-defined functional element which accepts data as input, performs processing and generates output in the form of data. The core services are used to build higher-level application specific and customizable services. The core services offer mainly four functionalities:
Data access: It access to structured data in distributed heterogeneous data resources.
Data transformation: It exposes data in schema P to users as data in schema Q.
Data integration: It exposes multiple databases to users as a single virtual database.
Data delivery: It delivers data to where it is needed by using the most appropriate delivery mechanisms. Some of the examples are web service, HTTP, e-mail, FTP, GridFTP [6].

The Figure 4.1 shows the core services offered by OGSA-DAI. The description of main services is as follows:

a) **Data Resource Plug-ins**

Data resource plug-in is used to communicate with data resources. This is a key OGSA-DAI extensibility point. After evaluating a variety of different databases and data resource plug-ins offered by OGSA-DAI, we have identified a set of data resource plug-ins to communicate with heterogeneous and geographically scattered data resources. We have come to the conclusion that the following are the database drivers which are used to access grid database resources. Table 4.1 shows database drivers used for each database type.

<table>
<thead>
<tr>
<th>Database</th>
<th>Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>MySQL Connector J/2</td>
</tr>
<tr>
<td>Microsoft SQL Server 2000</td>
<td>Microsoft SQL Server JDBC Driver</td>
</tr>
<tr>
<td>Microsoft SQL Server 2005, Microsoft SQL Server 2008</td>
<td>Microsoft SQL Server JDBC 2.0 Driver</td>
</tr>
<tr>
<td>Oracle</td>
<td>Oracle JDBC Drivers for Java 1.2+</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>Postgre JDBC Driver</td>
</tr>
<tr>
<td>IBM DB2</td>
<td>DB2 JDBC Driver</td>
</tr>
<tr>
<td>eXist</td>
<td>eXist XMLDB driver</td>
</tr>
<tr>
<td>File system</td>
<td>None</td>
</tr>
<tr>
<td>Resource group</td>
<td>None</td>
</tr>
<tr>
<td>Remote resource</td>
<td>None</td>
</tr>
</tbody>
</table>

*Table 4.1: Database & Related Driver Information*
To configure the access towards the grid data resources, connection URLs are required. These URLs are specific to each database type. After studying variety of such URLs, we have identified a set of database types and associated URLs required to establish a connection towards the grid data resources. Table 4.2 shows the examples of such URLs.

<table>
<thead>
<tr>
<th>Database</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>jdbc:mysql://myhost:3306/mytestdb</td>
</tr>
<tr>
<td>Microsoft SQL Server 2000</td>
<td>jdbc:microsoft:sqlserver://myhost:1433;DatabaseName=ogsadai</td>
</tr>
<tr>
<td>Microsoft SQL Server 2005, 2008</td>
<td>jdbc:sqlserver://myhost:1433;DatabaseName=ogsadai</td>
</tr>
<tr>
<td>Oracle</td>
<td>jdbc:oracle:thin:@myhost:1521:ogsadai</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>jdbc:postgresql://myhost:5432/ogsadai</td>
</tr>
<tr>
<td>IBM DB2</td>
<td>jdbc:db2://myhost:50000/ogsadai</td>
</tr>
<tr>
<td>eXist</td>
<td>xmlldb:exist://myhost:9120/exist/xmlrpc/db/mytestdb</td>
</tr>
<tr>
<td>Linux File System</td>
<td>/home/myDataResourceFiles</td>
</tr>
<tr>
<td>Unix File System</td>
<td>/home/myDataResourceFiles</td>
</tr>
<tr>
<td>Windows File System</td>
<td>\myDataResourceFiles</td>
</tr>
</tbody>
</table>

Table 4.2: Example URLs for Data Resource Connections

b) Activities & Activity Manager

An activity of OGSA-DAI is having a well-defined workflow unit with a specific name. Activities are the components used to perform data related operations. Some of these operations are running an SQL query, performing a data transformation, delivering data etc. Each activity is a well-defined functional unit which accepts data as input, processes it and generates output in the form of data. This is one of the key OGSA-DAI extensibility points. An activity is mapped to a Java class that is responsible for acting on or processing the content resides in that activity. Therefore, new activities can be added according to specific application or requirement.

We have implemented activities to perform actions like Querying, Updating, Transforming and Delivering data. OGSA-DAI offered variety of activities. Therefore, we analyzed a set of activities offered by OGSA-DAI and identified that these all types of activities fall into three major categories:
Statement – Interact with the data resource, e.g. direct an SQL query to a data resource. SQLQuery, SQLStatement, SQLUpdate are some of the examples of relational database activities. AddDocuments, GetDocuments, XPathQuery, XQuery are some of the examples of XML database activities.

Delivery – Deliver data to a third party. CSVToTuple, TupleToByteArrays, TupleToWebRowSetCharArrays are some of the examples of delivery activities.

Transform – Perform transformations on data, e.g. XSL Transform, compression. DeliverToFTP, DeliverToSMTP, ObtainFromFTP are some of the examples of data transformation activities.

The activity manager provides access to activities available on the OGSA-DAI server and this is used by the workflow engine. An activity specification data access object is used by an activity manager to load and save configuration information. The developer can also extend the scope of the activity manager by extending it by their own.

\* An example of an activity: As an example of activity, we have taken SQLQuery into consideration. SQLQuery executes SQL queries on the target data resource and produces a tuple list as an output. Figure 4.2 demonstrates an example of SQLQuery activity. As input, SQLQuery activity takes SQL query expression. The tuples produced by the SQL query are output of this activity. The SQL query expressions are targeted at a relational data resource. One input query is processed by executing the query across the target data resource in each iteration. The result of each iteration is a tuple list. If there is any error occurs at any stage of processing, it may produce partial data.

![Figure 4.2: An Example of SQLQuery Activity](image)
c) Workflow & Workflow Engine

The OGSA-DAI usually completes the tasks by executing workflows. Workflows are sent by clients and executing by the workflow engine on the OGSA-DAI server. The workflow contains more than one activity units. The workflow engine creates activity objects corresponding to those activities reside in a client's workflow. It is also responsible to monitor the execution of a workflow and to update the current status of the execution. The simple example of workflow execution is shown in Figure 4.3.

![Diagram of Workflow Execution](image)

*Figure 4.3: An Example of a Simple Workflow*

The user first sends the workflow to the OGSA-DAI server. The server is responsible to execute the workflow in order to get access the data reside in multi heterogeneous and distributed data resources. The status of the workflow execution is sent back to the user once the execution of workflow is completed.

❖ *Workflow Execution*

As discussed above, workflows are executed on the OGSA-DAI server. Figure 4.4 presents the picture of internal workflow execution performs on the OGSA-DAI server. There are three major entities reside on a server which execute the workflow sent by a client. A client submits its request in the form of workflow to the Data Request Execution Service (DRES).

DRES is a web service which provides the access to a Data Request Execution Resource (DRER). DRER is a component which executes the workflow. It first parses the workflow to instantiate the activities specified in the workflow and provides activities with their target resources (if any). Then after, it executes the workflow and builds a *request status*. Finally, it returns the request status to the Data Request Execution Service (DRES). At last, DRES returns the status of the request to the client.
Workflow Execution Types

Two modes are provided to execute the workflow submitted by a client: Synchronous Execution and Asynchronous Execution.

Synchronous Execution: In synchronous execution mode, the Data Request Execution Service (DRES) resides on OGSA-DAI server waits for the workflow to be completed. It will not return anything to the client until the workflow or the request has been completed. It returns a request status to the client only when the workflow has completed execution. It is recommended to execute simple workflows, which require short time for execution. Figure 4.5 demonstrates the execution of synchronous workflow.

Asynchronous Execution: The beginning of the asynchronous request workflow is totally different from the synchronous request workflow. The Data Request Execution Service (DRES) returns a request status to the client as soon as the workflow starts executing. It is recommended to use, when multiple workflows are executed. Data will be transferred individually by DRES. Figure 4.6 demonstrates an execution of asynchronous workflow.
**Request Status**

In both of the execution mode, synchronous and asynchronous, the request status is returned by a DRES to a client contains the following:

- Status of execution for each activity in the workflow: i.e. completed successfully or returns an error.
- Status of execution for whole workflow: Which is derived from the status of individual activities, i.e. all are completed successfully or any of them completes with errors, or premature termination of the workflow.
- Data: Data are depending on the activity states in the workflow and workflow execution mode i.e. synchronous or asynchronous. An activity DeliverToRequestStatus provided by OGSA-DAI ensures that any data it receives is added to the request status.

**Workflow Types**

Activities are specified in the workflow as discussed above. The pipeline is designed to send a request which contains multiple activities. Pipes are fixed-size buffers. A set of connected activities is termed as a pipeline. A valid pipeline satisfies the following: There must be a path of connected inputs and outputs between each pair of activities in the pipeline. This can be possible via intermediate activities. No activity in the pipeline can be connected to an activity outside the pipeline. Data streams go through the activities in a pipeline-like way. Each activity operates on a different portion of the data stream at the same time. There are three types of workflows as shown in Figure 4.7, Figure 4.8 and Figure 4.9.
1. **Pipeline Workflow:** A set of chained activities is executed in parallel. Data may flow between the chained activities. An example of pipeline workflow is shown in the Figure 4.7.

![Figure 4.7: An Example of Pipeline Workflow](image)

2. **Parallel Workflow:** In parallel workflow, a set of sub-workflows executes in parallel. An example of parallel workflow is shown in the Figure 4.8.

![Figure 4.8: An Example of Parallel Workflow](image)

3. **Sequence workflow:** In sequence workflow, a workflow is divided in sub-workflows. In this set of sub-workflows, each sub-workflow is executed in sequence. For example, a workflow may consist of two sub-workflows: Sub-workflow 1 fetches data tuples from database table and sub-workflow 2 loads the fetched data to another database table. An example of such sequence workflow is shown in the Figure 4.9.

![Sub-workflow 1](image) ![Sub-workflow 2](image)

As mentioned in the explanation about workflow, by default, the activities run parallel as in the OGSA-DAI server as default. Therefore, in the real case, to execute an ordered activities sequence, the activities may add to one or a couple of pipelines which will be submitted to the workflow afterwards.
d) Resources & Resource Manager

Resources are the mechanism which can be accessed, or referred to, by clients during workflow execution to complete the tasks. For example, a client may use Data Resource to access the databases or client may use Data Request Execution Resource during workflow execution. Resources are the powerful mechanism offered by OGSA-DAI to access its functionalities. Every resource is having a unique resource ID. Resource IDs’ are simple name that uniquely identifies the resource. Every resource is offered specific set of activities.

When client wants to use specific activity, resource ID is passed in workflow to access activities offered by specific resource. Each resource has zero or more resource properties with unique names. Resource properties are used to represent the state of a resource and the scope of the resource properties is within the resource. During our research, we have analyzed a variety of resources and their operations and properties. Finally, we have identified five types of resources that can be used frequently to develop domain specific applications. These resources are mainly used to interact with heterogeneous and geographically scattered databases that are integrated into database grid.

The Table 4.3 to Table 4.7 provides the list of these resources with their useful operations and properties.
### Chapter 4: Detailed Methodology for Development of Framework

#### Development of Multi-agent Knowledge-based System Accessing Distributed Database Grid

1. **Resource Name** | **Data Request Execution Resource (DRER)**
--- | ---
**Description** | Executes OGSA-DAI workflows concurrently on a server, Must have at least one DRER.
**Operations** | Execute: Executes a workflow and return the status of the executing or executed request to the client.  
GetResourceProperty: Gets the value of a resource property for a given property name.  
GetMultipleResourceProperties: Gets the value of a set of resource properties given their names.
**Properties** | Activities: Typical activities are data delivery, resource creation, resource destruction and management activities.

*Table 4.3: Description, Operations and Properties of DRER*

2. **Resource Name** | **Data Resource**
--- | ---
**Description** | An OGSA-DAI abstraction of a database or other type of data source, May have 0 or more data resources.
**Operations** | SetTerminationTime: Sets the termination time of the resource.  
Destroy: Terminates the resource immediately.  
GetResourceProperty: Gets the value of a resource property for a given property name.  
GetMultipleResourceProperties: Gets the value of a set of resource properties given their names.
**Properties** | Activities: Examples of activities are query activities and update activities.  
CurrentTime: The resource’s current time.  
TerminationTime: The resource’s termination time.

*Table 4.4: Description, Operations and Properties of Data Resource*
### Table 4.5: Description, Operations and Properties of Session

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Acts as a state container associated with a sequence of workflows, May have 0 or more sessions.</td>
</tr>
<tr>
<td>Operations</td>
<td>SetTerminationTime: Sets the termination time of the session. Destroy: Terminates the session immediately. GetResourceProperty: Gets the value of a resource property a given property name.</td>
</tr>
<tr>
<td>Properties</td>
<td>Activities: Example of activities is to save and load session state. CurrentTime: The session’s current time. TerminationTime: The session’s termination time.</td>
</tr>
</tbody>
</table>

### Table 4.6: Description, Operations and Properties of Request Resource

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Request Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Allows the management of a request submitted to a DRER, Use to monitor the status of execution of asynchronous workflow to determine when it's finished and when any data is available, May have 0 or more request resource.</td>
</tr>
<tr>
<td>Operations</td>
<td>SetTerminationTime: Sets the termination time of the resource. Destroy: Immediately terminates the resource and the request. GetResourceProperty: Gets the value of a resource property for a given property name.</td>
</tr>
<tr>
<td>Properties</td>
<td>RequestExecutionStatus: Execution status of a processing request RequestStatus: Information about the status of a currently executing request Activities: Activities supported by resources CurrentTime: The resource’s current time. TerminationTime: The resource’s termination time.</td>
</tr>
</tbody>
</table>
Table 4.7: Description, Operations and Properties of Data Source

<table>
<thead>
<tr>
<th></th>
<th>Resource Name</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Allows clients to pull data from an OGSA-DAI server. It may have zero or more data sources.</td>
<td></td>
</tr>
</tbody>
</table>
| Operations | **GetBlock**: Gets a single block of data from the data source. The operation will wait until a block is available or the end of the data stream is reached. Problems can occur if this operation waits so long that the client times out. To overcome these problems it is preferable to use **GetNBlocksNB**.  
**GetNBlocksNB**: Gets at most N blocks of data from the data source. The operation does not wait for N blocks to be available. Instead it turns as many blocks (up to N) as are currently available, this may be zero blocks. The ‘NB’ at the end of the operation name stands for 'non-blocking'. The operation also resizes the data source's internal buffer to be at least as big as N.  
**SetTerminationTime**: Sets the termination time of the data source.  
**Destroy**: Terminates the data source immediately.  
**GetResourceProperty**: Gets the value of a resource property for a given property name. |
| Properties | **DataSourceStatus**: The status of the data source.  
**Activities**: Examples of activities are to pull data from a local data source or to stream data to a local data source.  
**CurrentTime**: The data source’s current time.  
**TerminationTime**: The data source’s termination time. |

The resource manager provides access to the resources available on the server at any specific time. This access is provided to activities and applications that access the database grid. Developers may develop their own resource managers for domain specific applications.

e) Server Context

A server context holds components of use across the OGSA-DAI server. Examples of some of the components are activity managers, login providers, resource managers, monitoring components, authorizers and many more. These are configured by a special
context configuration file using the Java Spring framework. All components have access to the server context.

f) Security Context

The security context is populated with security-related information which may pass from the application layer through which a client accesses the OGSA-DAI server. It implements role based access model. For example, client credentials are used to access the data resources. The role mapper takes credentials and a database name and returns a dbRole. dbRole contains the username and password to access the specific database of database grid.

g) Configuration Loader

A configuration loader is used to load the OGSA-DAI configuration files. Configuration files are used to specify the server configuration. These may include the available activities, resources, supported activities, activity implementation classes, resource implementation classes and database usernames & passwords. The default implementation uses configuration files but other implementations could also be developed as per need. Data access objects are provided by the configuration loader to the resource and activity managers.

4.1.3 Multi-agent System Environment

Multi-agent system environment is a kind of Service-oriented Architecture (SOA). A service-oriented architecture is a set of principles and methodologies for designing and developing software in the form of interoperable services. Services are well-defined business functionalities that are built as software components and such services can then be consumed by different business processes or applications. Also, these services can be reused for different applications and purposes. SOA promotes loose coupling between software components or services so that they can be reused.

The grid is said to be a distributed architecture developed in a service-oriented perspective. On the other hand, agents and multi-agent systems are said to be autonomous, intelligent and interactive entities that may use and provide services [3]. To integrate the grid and multi-agent systems together, we have used a service oriented approach. Here, data grid services are provided and used by agents through the data grid middleware.
solution. The data grid core services are wrapped up into agents and different collaborative task agents are working together to provide the core, data and user level services to the client applications and users. As, the generic framework incorporates a multi-agent system which provides scalability and extensibility, new agents would be added and existing agents would be customized as per the requirements of the domain specific applications.

The agents used in the generic framework may categorize in four categories. As shown in the Figure 4.1, they all are working together in multi-agent system environment. They are Data Grid Service Agents, Domain Specific Agents, Fuzzy Interface Agent for fuzzy interface implementation and Interface Facilitator Agent. The data grid service agents are the typical agents, which provide the data grid services to client applications and users. As mentioned in the previous chapter, the knowledge layer of the generic framework consists of a knowledge-based component. To realize it, we have implemented fuzzy set theory and fuzzy logic. As shown in the Figure 4.1, the fuzzy interface agent contains the methodology and algorithms to implement the fuzzy interface. Also, there are several domain agents available in the computational model, which are application specific. The application developers can customize and implement the domain agents according to their requirements. The interface facilitator agent provides the interface to client applications and users in order to access the data grid services.

In this chapter, we have discussed and provided the implementation scenario with necessary screen layouts and code snippets for data grid service agents. These agents are generic and can be directly implemented in any application domain with little modification. The implementation scenario for domain specific agents and fuzzy interface agent is demonstrated in next chapter i.e. implementation of experimental system as these agents are application specific.

As stated above, we have used a service-oriented approach to develop a multi-agent knowledge-based system accessing distributed database grid. The agents are having typical characteristics that make them suitable to perform complex functions. Agents possess the following characteristics: Autonomy, Cooperation, Learning and Reactivity [1]. Web services are having these capabilities to facilitate software agents’ integration within systems [1]. Therefore, to realize the implementation of agents, we have used web services standards.
Web services are software entities which support interoperable machine-to-machine interaction over a network (generally on a web). It has an interface described in a machine-processable format and specifically in WSDL. Other systems interact with the web service in a manner prescribed by its description using SOAP messages. They usually conveyed using HTTP with an XML serialization in conjunction with other web-related standards [8]. Web services are realized through the usage of XML-based open standards, such as UDDI, WSDL and SOAP. These standards provide a common approach for defining, publishing, and using web services [7]. A web service is an abstract notion that must be realized by a concrete agent. This agent is the concrete piece of software entity which sends and receives messages. Furthermore, the service is the resource characterized by an abstract set of functionality that is provided [9].

4.2 Implementation Scenario

The implementation scenario discussed in this section focuses on the role of agents in the generic framework. The following content covers the details about the typical agents used to provide the data grid services, their roles and implementation scenario. We have used OGSA-DAI Axis Version 4.1 as a grid middleware service. We have realized the agents by implementing a set of web services. To implement web services, we have used JAVAX-WS (also, known as JAX-WS) standard.

The domain specific agents and fuzzy interface agent are application specific they are customizable according to the nature and requirements of the application. We have realized the above said agents during the implementation of an experimental system. The implementation scenario for domain specific agents and fuzzy interface agent are discussed in the next chapter (chapter 5).

- Prerequisite Software

- OGSA-DAI Axis Version 4.1

We have chosen OGSA-DAI Axis Version 4.1 as a grid middleware solution. The prerequisites to configure and use OGSA-DAI are as follows:

- OGSA-DAI require Java 1.6 or above
- OGSA-DAI Axis distributions required Jakarta Tomcat 5.5 or above
OGSA-DAI needs ANT 1.6 or above as a build, installation and deployment tool.

Along with this, the following environment variables need to be set to access OGSA-DAI services: CATALINA_HOME, OGSADAI_HOME and CLASSPATH.

To check whether the OGSA-DAI services are deployed on Jakarta Tomcat successfully or not, following are the steps to be performed:

1. Startup Tomcat.
2. Open a browser and write the following line in the address bar: http://localhost:8080/dai/services.
3. If OGSA-DAI is deployed properly, the following screen would be displayed which will contain a list of services and information on their operations and links to their WSDL service description documents, as shown in Figure 4.10.

And now... Some Services

- DataSinkService (read)
  - setTerminationTime
  - destroy
  - queryResourceProperties
  - getMultipleResourceProperties
  - getResourceProperty
  - getVersion
  - influences
  - create
  - putFully
  - putBlocks
  - putBlocksDB
  - putBlock
- DataRequestExecutionService (read)
  - execute
  - queryResourceProperties
  - getMultipleResourceProperties
  - getResourceProperty
  - getVersion
  - influences
  - create
- AdminService (read)
  - AdminService
- RequestManagementService (read)
  - setTerminationTime
  - destroy
  - queryResourceProperties
  - getMultipleResourceProperties
  - getResourceProperty

Figure 4.10: An Example of a Screen Showing Deployed Services
JAX-WS

JAX-WS (Java API for XML Web Services) is a Java programming language APIs for creating web services. It is a part of the Java EE platform from Sun Microsystems. It is a technology for building web services and clients that communicate using XML. In JAX-WS, a web service operation invocation is represented by an XML-based protocol, such as SOAP. The SOAP protocol specification describes the encoding rules, the envelope structure, and conventions for representing web service invocations and responses. Here, calls and responses are transmitted as SOAP messages (XML files) over HTTP.

Java Platform, Enterprise Edition or Java EE is Oracle's Enterprise Java computing platform. This platform offers an API and runtime environment for developing and deploying enterprise software. These may include network & web services and other multi-tiered, large-scale, secure, scalable and reliable network applications.

We have used NetBeans (www.netbeans.org) as an IDE (Integrated Development Environment). NetBeans is an integrated development environment (IDE) for developing primarily with Java. It also offers the environment for developing in other languages such as C/C++, PHP, HTML5 and so forth. It is also an application platform framework for Java desktop applications and others. The NetBeans IDE is developed in Java language and can run on OS X, Windows, Linux, Solaris and other platforms supporting a compatible JVM. The prerequisites to use NetBeans are Java 1.6 or above and Tomcat 6 or above.

**Data Grid Services Provided by Typical Agents:** As mentioned above and shown in the Figure 4.1, there are mainly three typical agents currently implemented which are responsible to provide the data grid services to client applications and users. These agents are namely data access & integration agent, resource management agent and grid administrative agent. Therefore, we have created three web services: data access & integration web service, resource management web service and grid administrative web service. During implementation, we have provided the authorization to access these agents to only administrator. We have assumed here that, an administrator has to work like data grid administrator and he is responsible to manage, control & supervise the entire data grid environment. An administrator is assigned a valid username (i.e. admin) and password. To manage, control and supervise the data grid services, an administrator has to first log in the
system. The system matches the credentials like username and password and if these credentials are valid then the system allows administrator as a valid user.

As mentioned earlier, an interface facilitator agent provides a user interface to end users. The interface facilitator agent enables users to send a request and receive a response to and from the data grid environment and is used to connect end users together with the multi-agent system environment. Therefore, an administrator may get access to data grid services provided by typical agents through interface facilitator agent.

The necessary screen layouts and code snippets for typical agents used to provide data grid services are mentioned as follows.

- **Resource Management Agent**

  Data grid systems generally access existing and real time data stored in several distributed and heterogeneous databases. We have realized a resource management agent, which is responsible to provide the core functionality of the data grid. It is the key component of the data grid environment that manages all actions related to manage and maintain the heterogeneous and geographically scattered database resources. The key task of resource management agent is to deploy (register) and undeploy the data grid nodes within the data grid environment. It also provides the facility to manage and manipulate a specific node in the data grid. When a new node comes, it has to first register itself within the data grid through a resource management agent. The data grid then starts the communication with the newly registered data grid node in order to access the databases resides in that node. As stated above, the resource management agent can be accessed by a data grid administrator through an interface facilitator agent. We have implemented resource management agent through a realization of a web service.

  In order to make available the database resources of a data grid node of the data grid environment, OGSA-DAI use configuration script for configuration and deployment tasks. This configuration script can be modified on need. Configuration script can consist of one or more commands in the form of lines to configure data resources on OGSA-DAI in various ways. This configuration script is deploying on server by executing *ant* commands. The configuration script file is also used by OGSA-DAI to manage the metadata for authentication and authorization of users. An agent is a software entity that acts on behalf of its user. Therefore, the resource management agent is mainly responsible
to generate the configuration script automatically and deploy, undeploy and manage the data resources automatically within a data grid environment with minimal intervention of the user. It hides the complexity of development and execution of the configuration script from user and provides them a standardized interface to manage and maintain the data grid nodes.

The following are the roles played by the resource management agent. As stated earlier also, an agent is a software program or piece of program which acts automatically and proactively on behalf of its user. The resource management agent facilitates the following tasks in a user-friendly and automated way; which otherwise needed to perform manually via the command line through user. To perform the said tasks via the command line is tedious, complex and time consuming and we have eliminated these by realizing an agent. Currently, we are assuming that only administrator (i.e. a typical data grid user) is assigned the privileges to interact with resource management agent. The following are the details about the tasks performed by a resource management agent with necessary screen layouts and code snippets.

1. **Provision of an interface to automate the task of add, update and delete database resources in the data grid environment**

   Upon successful login and adequate privileges, the resource management agent provides a web-based interface to add, update and delete the database nodes within the data grid environment. Figure 4.11 shows the screen layout, after successfully logged into the system to access data grid as an “admin” username. Here, “admin” has assigned the adequate privileges to interact with resource management agent. We can see the ‘Resource’ Tab in the Figure 4.11 to add, update and manage the database resources within the data grid environment.
2. Get database credentials to access a particular database resource along with its type and database connection URL.

Figure 4.12 demonstrated the screen layout provided by a resource management agent to add a resource in the data grid. It is used to get metadata information required to access the specific database resource. This metadata information is used to register, deploy and access the database resource in a data grid environment. The required metadata information are resource ID, Resource Type, Database connection URL, Username and Password. It also takes the information about any backup resource available for the master data resource. The working of backup resource is discussed in further section. After successful submission of these data, the resource management agent automatically developed a configuration script file, which is necessary to expose a database grid node via OGSA-DAI server.
3. Write the above metadata details in the configuration script file.

As mentioned earlier also, the resource management agent automates the process of configuration and deployment of database resources on OGSA-DAI server. For that, it has to develop a configuration script file by using metadata information. This agent automates the following tasks and hides the underlying coding complexity to generate a configuration script file from users.

**Configuration Script File:** After studying about how to deploy and expose heterogeneous and geographically scattered database resources on OGSA-DAI server, we have analyzed that, the OGSA-DAI uses a configuration script file for deploying and exposing database resources. It is a file consisting of one or more lines and each line specifies some configuration or deployment action that is to be completed. There are two important features of configuration script files:

- Each command must be on a single line with no line breaks.
- A line beginning with # is considered to be a comment and is not executed.

A relational resource represents a relational database. On the OGSA-DAI server, a relational resource manages communication between OGSA-DAI and the actual relational database. A variety of commands exist, which may use in configuration script. Among of them, we have identified following commands that can be used by the resource...
management agent for the configuration script file. Table 4.8 summarizes these commands need to write in configuration script in order to deploy different resources on the OGSA-DAI server. These commands are automatically configured by the resource management agent.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Configuration Script Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational Resources</td>
<td>Oracle JDBC deployOracle OracleResource jdbc:oracle:thin:@myhost:1521:ogsadai Login permit OracleResource ANY testuser testpassword</td>
</tr>
<tr>
<td></td>
<td>MySQL JDBC deployMySQL MySQLResource jdbc:mysql://myhost:3306/mytestdb Login permit MySQLResource ANY testuser testpassword</td>
</tr>
<tr>
<td></td>
<td>SQLServer JDBC deploySQLServer SQLServerResource jdbc:microsoft:sqlserver://myhost:1433;DatabaseName=ogsadai Login permit SQLServerResource ANY testuser testpassword</td>
</tr>
<tr>
<td></td>
<td>DB2 JDBC deployDB2 DB2Resource jdbc:db2://myhost:50000/ogsadai Login permit DB2Resource ANY myUser myPassword</td>
</tr>
<tr>
<td></td>
<td>PostGreSQL JDBC deployPostgreSQL PostGreSQLResource jdbc:postgresql://myhost:5432/ogsadai Login permit PostGreSQLResource ANY myUser myPassword</td>
</tr>
<tr>
<td>File System Resource</td>
<td>File Files deploy MyFile D:/location/of/files</td>
</tr>
</tbody>
</table>

Table 4.8: Configuration Script Commands for Deploying Database Resources
Deploy Data Resources through Configuration Script File: The following is an example of configuration script file written to configure resources on OGSA-DAI server.

JDBC deploy RESOURCE_ID URL CLASS Login permit RESOURCE_ID DN USER PASSWORD

Here:

- **RESOURCE_ID** is the ID of the resource.
- **URL** is the database connection URL.
- **CLASS** is the database driver class name.
- **DN** is a credential, attribute or distinguished name. Clients may provide these and these need to be mapped to database usernames and passwords.
- **USER** is the username for database.
- **PASSWORD** is the password for the database.

The following is the code snippet of a web service which realizes the resource management agent. The code snippet demonstrates an automation of creation of configuration script file named “myConfig.txt”. The metadata information is written in this file and saved at the specified location.

```java
//---------------
static String location = "D:/home/ogsa-dai/";
public static void addResource(String type, String resourceId, String connectionURL, String user, String password) throws Exception {
    FileWriter fw = new FileWriter(location + "myConfig.txt", false);
    StringBuffer sb = new StringBuffer("JDBC ");
    sb.append(type);
    sb.append(" ");
    sb.append(resourceId);
    sb.append(" ");
    sb.append(connectionURL);
    sb.append(" \r\n");
    sb.append("Login permit ");
    sb.append(resourceId);
    sb.append(" ANY ");
    sb.append(user);
    sb.append(" ");
    sb.append(password);
    fw.write(sb.toString());
    fw.close();
}
//---------------
```
4. Deploy and Undeploy the database resources on OGSA-DAI server by executing configuration script file commands through automatic execution of ant task.

After generating the configuration script, it should be executed on OGSA-DAI server to deploy the database resources.

**A command used to run Configuration Script File**

```
$ ant -Dtomcat.dir=$CATALINA_HOME -Dconfig.file=CONFIG-FILE [-Djar.dir=JAR-DIRECTORY] [-Dstart.line=LINE] configure
```

Here,

- **CONFIG-FILE** is the location where the configuration script file is saved.
- **JAR-DIRECTORY** is specifying the location of a directory containing any JARs that need to be deployed on the server. It is an optional argument and we can omit this if any JARs we need have already been deployed onto the server.
- **LINE** is specifying the line number of the file at which execution should start. It is also an optional argument.

**Example:**

```
$ ant -Dtomcat.dir=$CATALINA_HOME -Dconfig.file=TestConfig.txt -Djar.dir=TestJARS -Dstart.line=10  configure
```

The above command runs a configuration script file named TestConfig.txt from line 10 and copies the JARs reside in TestJARS to the server.

Resource management agent automates the deployment and undeployment of resources mentioned in the configuration script file by automatically executing an **ant** task on the OGSA-DAI server. For that, the agent consists of very bulky code scripts. A portion of such code script is shown below. This is the sample code snippet for deployment of resources on the OGSA-DAI server. Similarly, there are a variety of scripts generated by the resource management agent to carry out other jobs.

```java
//-------------------
public class AntTask
{
    public static void runTask(File buildFile, String targetName, Map<String, String> properties)
    {
        ProjectHelper projectHelper = ProjectHelper.getProjectHelper();
        Project project = new Project();
```
Chapter 4: Detailed Methodology for Development of Framework

Development of Multi-agent Knowledge-based System Accessing Distributed Database Grid

```java
project.setUserProperty("ant.file", buildFile.getAbsolutePath());

DefaultLogger consoleLogger = new DefaultLogger();
consoleLogger.setErrorPrintStream(System.err);
consoleLogger.setOutputPrintStream(System.out);
consoleLogger.setMessageOutputLevel(Project.MSG_INFO);
project.addBuildListener(consoleLogger);

if (properties != null)
{
    for (String key : properties.keySet())
    {
        project.setProperty(key, properties.get(key));
    }
}
project.init();

project.addReference("ant.projectHelper", projectHelper);
projectHelper.parse(project, buildFile);
try
{
    project.executeTarget(targetName);
} catch (BuildException e)
{
    String err = "";
    for (StackTraceElement ele : e.getStackTrace())
    {
        err += ele;
    }
    throw new RuntimeException("ant task failed \n" + err);
}
//---------------
//-----------------
// ant -Dconfig.file=myConfig.txt -Dwebapp.dir=dai/WEB-INF/ -Djar.dir=myJARs configure
Map<String, String> map = new HashMap<String, String>();
map.put("config.file", location + "myConfig.txt");
map.put("webapp.dir", "D:/home/ogsa-dai/dai/WEB-INF/");
map.put("jar.dir", "D:/home/ogsa-dai/tmp-jars/");
AntTask.runTask(new File(location + "build.xml"), "configure", map);

// ant -Dtomcat.dir=$CATALINA_HOME deploy
map = new HashMap<String, String>();
map.put("tomcat.dir", "D:/apache-tomcat-7.0.29");
AntTask.runTask(new File(location + "build.xml"), "deploy", map);
//---------------
```
By executing web service realized for resource management agent, the heterogeneous and geographically scattered database resources are deployed successfully on OGSA-DAI server without minimal user intervention. Figure 4.13 shows the list of such deployed database resources with their necessary metadata information. User can update the database resources by clicking on the specific resource name. i.e. MBA_Resource needs to be clicked, if user wants to update it. The resource management agent also performs undeployment of database resources from grid environment.

5. **Manage the above details in local database named Resource_Master.**

Apart from configuration file, some of the metadata information is also maintained in the local database for further use. The following is the code snippet of the resource management web service that writes the metadata details in Resource_Master table, which is stored in a local database.

```java
//-----------------
QueryDatabase.executeNonSelect(DatabaseSetting.readSettings(), "INSERT INTO resource_master(`ResID`, ResourceName`) VALUES ('" + resaurceId + ", ', '" + resaurceId + ");";

//-----------------
```
Chapter 4: Detailed Methodology for Development of Framework

Figure 4.13: A List of Deployed Resources with Metadata Information
Data Access & Integration Agent

Data access & integration agent is the heart of any data grid environment. It provides a uniform and standardized interface which allows data resources such as relational databases, file collections, XML databases, RDF resources to be accessed, manipulated, integrated and federated across the network such as LAN or WAN. For that, we have realized a data access & integration agent through a typical web service that uses the services offered by OGSA-DAI – a grid middleware solution. It provides a variety of services described as follows.

- Data access: It provides access to structured data in distributed heterogeneous data resources.
- Data integration: It used to expose multiple databases to users as a single virtual database and thus hides the heterogeneity and provides transparent and single point access of heterogeneous and geographically distributed databases to users. It hides the heterogeneity of the database type, database drivers and database locations from end users.
- Data transformation: It exposes data in schema P to users as data in schema Q.
- Data delivery: It is used to deliver data to by the most appropriate means such as web service, FTP, HTTP, e-mail, GridFTP etc.

As we mentioned earlier in this chapter, the OGSA-DAI completes the execution of tasks by executing workflows. As, data access & integration agent is implemented through data access & integration web service, this web service gets the query execution request from user, executes the workflow on OGSA-DAI server and sends response back to the users [6]. The following code snippet focuses on implementation of a client to run a simple workflow that:

- Runs an SQL query over a database and outputs the results as a list of tuples. It is an OGSA-DAI's representation of rows of relational data.
- Converts this list of tuples into byte arrays (binary format) for efficient transferring.
- Add the byte arrays to the request status for return to the client.
Assumptions:

- Server URL is: [http://localhost:8080/dai/services/](http://localhost:8080/dai/services/)
- Relational data resource ID is *TestSQLDataResource*.
- Relational resource which accesses a database contains a table called *TestMaster*.

**Code Snippet:**

The following code snippet demonstrates the mechanism used by data access & integration agent.

```java
/* importing the following packages from java and ogsa-dai toolkit
java.net.URL;
java.sql.ResultSet;
java.sql.ResultSetMetaData;
uk.org.ogsadai.client.toolkit.ServerProxy;
uk.org.ogsadai.client.toolkit.Activity;
uk.org.ogsadai.client.toolkit.PipelineWorkflow;
uk.org.ogsadai.client.toolkit.RequestExecutionType;
uk.org.ogsadai.client.toolkit.DataRequestExecutionResource;
uk.org.ogsadai.client.toolkit.RequestResource;
uk.org.ogsadai.client.toolkit.activities.delivery.DeliverToRequestStatus;
uk.org.ogsadai.client.toolkit.activities.sql.SQLQuery;
uk.org.ogsadai.client.toolkit.activities.transform.TupleToByteArrays;
uk.org.ogsadai.client.toolkit.exception.ClientException;
uk.org.ogsadai.client.toolkit.exception.ClientToolkitException;
uk.org.ogsadai.client.toolkit.exception.RequestException;
uk.org.ogsadai.client.toolkit.exception.ResourceUnknownException;
uk.org.ogsadai.client.toolkit.exception.ServerCommsException;
uk.org.ogsadai.client.toolkit.exception.ServerException;
uk.org.ogsadai.client.toolkit.messages.Message;
uk.org.ogsadai.resource.ResourceID;
uk.org.ogsadai.resource.request.RequestExecutionStatus;
uk.org.ogsadai.resource.request.RequestStatus;
*/

public class Client {
    public static void main(String[] args) throws Exception {
        // Get a client side server proxy to handle communications with
        // the server.
        ServerProxy server = new ServerProxy();
        String url = "http://localhost:8080/dai/services/");
        server.setDefaultBaseServicesURL(new URL(url));

        // Get a proxy for a Data Request Execution Resource (DRER)
        // DRER executes OGSA-DAI workflows from clients.
        DataRequestExecutionResource drer =
            server.getDataRequestExecutionResource(new ResourceID("DataRequestExecutionResource"));
    }
}
```
// Create activities.
SQLQuery query = new SQLQuery();
TupleToByteArrays tupleToByteArrays = new TupleToByteArrays();
DeliverToRequestStatus deliverToRequestStatus = new DeliverToRequestStatus();

// Connect and Configure activities.
query.setResourceID("TestSQLDataResource");
query.addExpression("SELECT * FROM TestMaster WHERE test_id <15;");
tupleToByteArrays.connectDataInput(query.getDataOutput());
tupleToByteArrays.addSize(20);
deliverToRequestStatus.connectInput(tupleToByteArrays.getResultOutput());

// Create the workflow.
PipelineWorkflow pipeline = new PipelineWorkflow();
pipeline.add(query);
pipeline.add(tupleToByteArrays);
pipeline.add(deliverToRequestStatus);

// Execute the workflow.
RequestResource requestResource = null;
try{
    requestResource = drer.execute(pipeline,
    RequestExecutionType.SYNCHRONOUS);
} catch (ServerCommsException e) {
    System.out.println("Something went wrong between client and server");
    e.printStackTrace();
    System.exit(1);
} catch (ServerException e) {
    System.out.println("Something went wrong server-side");
    e.printStackTrace();
    System.exit(1);
} catch (ResourceUnknownException e) {
    System.out.println("The DRER is unknown to the server");
    e.printStackTrace();
    System.exit(1);
} catch (ClientException e) {
    System.out.println("Something went wrong at the server before the request started execution, that was the client's fault");
    e.printStackTrace();
    System.exit(1);
} catch (RequestException e) {
    System.out.println("Something went wrong when executing the client's request");
    e.printStackTrace();
    printActivityStatus(query);
printActivityStatus(tupleToByteArrays);
printActivityStatus(deliverToRequestStatus);
System.exit(1);
}
catch (ClientToolkitException e)
{
    System.out.println("Something went wrong in the client toolkit");
e.printStackTrace();
System.exit(1);
}

// Get status information and data from the request status.
RequestStatus requestStatus =
requestResource.getRequestStatus();
System.out.println(requestStatus);
RequestExecutionStatus requestExecutionStatus =
requestStatus.getExecutionStatus();
System.out.println(requestExecutionStatus);

if (requestExecutionStatus.equals(RequestExecutionStatus.COMPLETED))
{
    System.out.println("Request completed OK!");
}
printActivityStatus(query);
printActivityStatus(tupleToByteArrays);
printActivityStatus(deliverToRequestStatus);

if (tupleToByteArrays.hasNextResult())
{
    // Get ResultSet.
    ResultSet resultSet =
tupleToByteArrays.nextResultAsResultSet();
    // Get ResultSet metadata.
    ResultSetMetaData metaData = resultSet.getMetaData();
    // Print column names.
    int numColumns = metaData.getColumnCount();
    String columns = "";
    for (int i = 0; i < numColumns; i++)
    {
        columns += (metaData.getColumnLabel(i + 1) + " - ");
    }
    System.out.println(columns);
    // Print rows.
    while (resultSet.next())
    {
        String row = "";
        for (int i = 0; i < numColumns; i++)
        {
            row += (resultSet.getString(i + 1) + " - ");
        }
        System.out.println(row);
    }
    resultSet.close();
}

public static void printActivityStatus(Activity activity)
{
    System.out.println("Activity: " + activity.getActivityName());
    System.out.println(" Status: " + activity.getStatus());
if (activity.hasErrorMessages())
{
    Message[] messages = activity.getErrorMessages();
    for (int i = 0; i < messages.length; i++)
    {
        System.out.println("  " + messages[i]);
    }
}

• SQL Query Across Multiple Databases using SQLBag

Data access & integration agent is also able to retrieve and access the data which reside in multiple heterogeneous and scattered databases. For that, we have used SQLBag, a typical service provided by OGSA-DAI, to execute an SQL query across multiple databases. For that, first we need to create a resource group. A resource group is basically a type of an OGSA-DAI data resource. It represents a set of child resources which are usually hidden from the users. To users, the group appears as a single resource. Thus, we can hide the heterogeneity and provides transparent and single point access of heterogeneous and geographically distributed databases to users. The same SQL query is executed on each database in the resource group and the results are then collected back. For that, resource group must be created which contains all resources that the user wishes to query. Figure 4.14 demonstrates about a SQL query executes across multiple heterogeneous databases.

![Figure 4.14: SQL Query Across Multiple Heterogeneous Databases](image)

The following code snippet shows the SQLBag implementation. It is an example of using CreateResourceGroup which is used to create a resource group containing relational data resources. SQLBag is used to run a query over this resource group [6]. In following
example, we have taken three heterogeneous databases to execute the SQL query. We can add or update more according the requirement of domain specific application.

```java
/* importing the following packages from java and ogsa-dai toolkit
java.net.URL;
java.sql.ResultSet;
java.sql.ResultSetMetaData;

uk.org.ogsadai.client.toolkit.DataRequestExecutionResource;
uk.org.ogsadai.client.toolkit.PipelineWorkflow;
uk.org.ogsadai.client.toolkit.RequestExecutionType;
uk.org.ogsadai.client.toolkit.RequestResource;
uk.org.ogsadai.client.toolkit.ServerProxy;
uk.org.ogsadai.client.toolkit.activities.block.CharArraysResize;
uk.org.ogsadai.client.toolkit.activities.delivery.DeliverToRequestStatus
uk.org.ogsadai.client.toolkit.activities.management.CreateResourceGroup;
uk.org.ogsadai.client.toolkit.activities.sql.SQLBag;
uk.org.ogsadai.client.toolkit.activities.transform.TupleToWebRowSetCharAr
rays;
uk.org.ogsadai.resource.ResourceID;
*/

public class SQLBagExample
{
    public static void main(String[] args) throws Exception
    {
        // Server URL.
        String url = "http://localhost:9020/dai/services/";

        // DRER that executes workflows.
        String drerID = "DataRequestExecutionResource";

        // Resource IDs of relational resources.
        String id1 = "ResourceOracle";
        String id2 = "ResourceMySQL";
        String id3 = "ResourceSQLServer";

        // SQL query to be executed.
        String sql = "SELECT * FROM TestMaster WHERE test_id < 15";

        // SET UP PROXIES

        // Get server proxy.
        ServerProxy server = new ServerProxy();
        server.setDefaultBaseServicesURL(new URL(url));

        // Get DRER proxy.
        DataRequestExecutionResource drer =
        server.getDataRequestExecutionResource(new ResourceID(drerID));

        // CREATE A RESOURCE GROUP

        // Create activities.
        String[] resources = new String[]{id1, id2, id3};
        CreateResourceGroup create = new CreateResourceGroup();
        create.addResourceIds(resources);
        DeliverToRequestStatus deliverToRequestStatus1 =
```
new DeliverToRequestStatus();
deliverToRequestStatus1.connectInput(create.getResultOutput());

// Create workflow.
PipelineWorkflow createWorkflow = new PipelineWorkflow();
createWorkflow.add(create);
createWorkflow.add(deliverToRequestStatus1);

// Execute workflow.
RequestResource requestResource1 =
    drer.execute(createWorkflow,
    RequestExecutionType.SYNCHRONOUS);
System.out.println("Status: " +
requestResource1.getRequestExecutionStatus());

// Extract result data.
ResourceID groupID = create.nextResult();
System.out.println("Resource group ID: " + groupID);

// RUN SQLBag OVER THE RESOURCES IN THE RESOURCE GROUP.

// Create activities.
SQLBag query = new SQLBag();
query.setResourceID(groupID);
query.addExpression(sql);
TupleToWebRowSetCharArrays tupleToWebRowSet =
    new TupleToWebRowSetCharArrays();
tupleToWebRowSet.connectDataInput(query.getDataOutput());
CharArraysResize resize = new CharArraysResize();
resize.addSizeInChars(5000);
resize.connectDataInput(tupleToWebRowSet.getResultOutput());
DeliverToRequestStatus deliverToRequestStatus2 =
    new DeliverToRequestStatus();
deliverToRequestStatus2.connectInput(resize.getResultOutput());

// Create workflow.
PipelineWorkflow pipeline = new PipelineWorkflow();
pipeline.add(query);
pipeline.add(tupleToWebRowSet);
pipeline.add(resize);
pipeline.add(deliverToRequestStatus2);

// Execute workflow.
RequestResource requestResource2 =
    drer.execute(pipeline,
    RequestExecutionType.SYNCHRONOUS);
System.out.println("Status: " +
requestResource2.getRequestExecutionStatus());

// Extract result data.
tupleToWebRowSet.getResultOutput().setResultActivity(deliverToRequestStatus2);
if (tupleToWebRowSet.hasNextResult())
{
    ResultSet resultSet =
tupleToWebRowSet.nextResultAsResultSet();
    ResultSetMetaData metaData = resultSet.getMetaData();
    int numColumns = metaData.getColumnCount();
    String columns = "";
    for (int i = 0; i < numColumns; i++)
Data Browser

Data access & integration agent also offers a generic service called “Data Browser” to provide a user-friendly Graphical User Interface (GUI) for database access and data delivery. It provides a mechanism to query the databases reside in a data grid environment. Figure 4.15 shows the screen layout of data browser. The Tab entitled as ‘Data Browser’ offers two options: Query Browser and Table Information. Query browser uses to pass the query against the data grid nodes. Table information feature provides table’s metadata information. Figure 4.15, 4.16 and 4.17 demonstrates the query browser feature. Figure 4.18 demonstrates the table metadata information feature. The following are the steps to be performed in order to access the query browser feature.

1. To pass a query across the database nodes, the user first need to select the resource id. The resource id provides the list of resources which are currently deployed within the data grid (i.e. deployed on OGSA-DAI server). User can either select ‘All’ resources or a specific resource against whom, he wants to pass a query.

2. We can see a field named as ‘Query Type’ in the Figure 4.15. There are two options available for users. First is DML (Data Manipulation Language) query. Example of DML statement is ‘select’. Second is DDL (Data Definition Language) query. Example of DDL statements is ‘insert’ and ‘update’ (as shown in the Figure 4.15 and 4.16).

3. In the field entitled as ‘SQL Query’, user need to write a SQL query statement and clicks on ‘Go’ in order to get the query result (as shown in the Figure 4.15 & 4.17).
Figure 4.15: A Screen Layout of Query Browser for DML Statement

Figure 4.16: A Screen Layout of Query Browser for DDL Statement
Chapter 4: Detailed Methodology for Development of Framework

Development of Multi-agent Knowledge-based System Accessing Distributed Database Grid

Figure 4.17: A Screen Layout of Result Generated by Query Browser

The Figure 4.18 shows the Table information feature. The user selects any specific resource from the Resource ID field. The Table Name field automatically fetches the tables which are contained in the selected database resource. After selecting a specific table name, the table schema will be displayed in Table Details field.

Figure 4.18: A Screen Layout of Table Information
Grid Administrative Agent

Grid administrative agent is mainly responsible to implement the grid information service (in collaboration with resource management agent) and data backup & replication service.

Data Backup: Data backup & replication is an essential mechanism of the data grid. The data backup mechanism is used to achieve a fault-tolerant design. Fault-tolerant design is a design that enables a system to continue its operation, possibly at a reduced level, when some part of the system fails, rather than failing completely. The fault tolerant system continues more or less fully operational with, possibly, a reduction in throughput or an increase in response time in the event of some partial failure.

Currently, we have realized fault-tolerance design by implementing data backup service at a certain level. In our strategy, there is at least one backup database node maintained for each master database node registered within the data grid. Currently, we are assuming that the database administrator of particular database grid node sets the mechanism for taking regular backup of each master database node to backup database node. As a future extension, we can implement specific domain agent to provide automation of such data backup service by avoiding the intervention of database administrator. Such domain agent is responsible to take back up of master database node at specific interval in order to synchronize the master database node to backup database node without user intervention.

As stated earlier in this chapter, the resource management agent provides the visual interface to set the backup node for each master database node at the time of registering the master database node within the data grid (as shown in the Figure 4.12). Once, the registration completes, the grid administrative agent activates the service to monitor the status of the master database node. To check whether a master database node is active or fail at a certain point of time, it is indeed to check the status of the connection established to that database node. For that, an agent executes a query to each master database node registered in a data grid at every specific time interval. If it finds that a connection to a master database node is fail, it automatically sets the relevant backup database node in the place of the master database node until the master database node will be activated once again. Once it finds that the connection with the master database node is established once again, it again starts to retrieve the data from the master database node. The strategy implemented here is used to avoid a single-point-of-failure. This means if a master
database node fails, a backup component (i.e. database node) automatically "kick in" in its place. It will be “kicking out” once the master database node will be repaired and activated.

**Grid Information Service:** The grid administrative agent provides the visual interface where an administrator (a typical privileged user entitled as ‘admin’) can see the status of the no. of registered nodes, the no. of active nodes and the no. of backup nodes. Figure 4.19 shows the visual interface provided by the grid administrative agent. It shows the number of registered database nodes within the data grid environment as we can see in column name ‘Resource ID’. The column entitled as ‘Resource Status’ demonstrates the status of the resource. The red colored box shows the inactive resource and green colored box shows the active resource. Apart from this, the column entitled as ‘Backup Resource Status’ shows the status of backup resources. Currently, according to the Figure 4.19, we can see that there are three database nodes are registered. They are ‘exam’, ‘MBA_Resource’ and ‘MCA_Resource’. Among of these registered nodes, two of them are also having backup resource nodes i.e. ‘MBA_Resource’ and ‘MCA_Resource’. Also, ‘exam’ resource is not active at this point of time as it is shown in the red colored box. ‘MBA_Resource’ and ‘MCA_Resource’ are active and data are currently retrieving from these original database nodes as they are shown in green colored boxes.

The above said tasks are performed by a grid administrative agent automatically without user intervention. For that, it has to cooperate with resource management agent. Also, it is proactive as it automatically sets the access point to original (i.e. master) and backup database resource nodes when and where required without the actions initiated or performed by users.
Chapter 4: Detailed Methodology for Development of Framework

<table>
<thead>
<tr>
<th>Resource Id</th>
<th>Resource Name</th>
<th>Resource Type</th>
<th>Connection URL</th>
<th>User Name</th>
<th>Resource Status</th>
<th>Backup Resource Status</th>
<th>Delete Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>exam</td>
<td>exam</td>
<td>MySQL</td>
<td>jdbc:mysql://localhost:3306/exam</td>
<td>root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCA_Resource</td>
<td>MCA_Resource</td>
<td>Oracle</td>
<td>jdbc:oracle.thin:@117.239.83.202:1521XE</td>
<td>system</td>
<td>MCA_Resource</td>
<td>MCA_Backup</td>
<td>Delete</td>
</tr>
</tbody>
</table>

Figure 4.19: A Visual Interface Provided by Grid Administrative Agent
Data Replication: Data replication is the user level service generally built upon the core services offered by the data grid environment. Replication is important for all DBMS as it can improve system availability, data consolidation and data distribution. However, a data grid environment is often heterogeneous and highly scattered and large in scale. A data replication service of the middleware distributes data resources into remote sites and keeps them synchronized with the original site on a particular level. Distributed data are usually processed by these remote sites, and sometimes the changes need to be propagated back to the origin site. Currently, the grid administrative agent provides data backup service but as a part of future extension, a specific domain agent to be developed to provide the data replication by applying specific data replication strategy. A lot many replication strategies are available like static replication, best client application, cascading replication and more. Any of these data replication strategies may offer from the above said domain specific agent.

4.3 Conclusion

This chapter discusses the detailed methodology used to implement the generic framework explained in the previous chapter. It shows the sample code snippets and screen layouts necessary to explain the methodology used to implement the framework. It also demonstrates how agents and multi-agent system is used to provide the data grid services to the client applications and users. Moreover, the implementation scenario has been discussed and demonstrated for the data grid typical agents. It also covers the details about how agents communicate with OGSA-DAI services in order to expose heterogeneous and distributed databases in the data grid environment.

The implementation scenario for domain specific agents and fuzzy interface agent will be demonstrated in the next chapter. As, the framework is generic and multi-agent system provides scalability and extensibility, new agents will be added and existing agents will be customized as per the requirements of the client applications and users.

References

1. A. M. Riad, H. A. El-Ghareeb, A Service Oriented Architecture to integrate Web services and Agents in Course Management Systems, retrieved from
http://www.academia.edu/169287/A_Service_Oriented_Architecture_to_Integrate_Web_services_and_Agents_in_Course_Management_Systems on January 2013.


