CHAPTER 3

SEMANTIC DESCRIPTION AND DISCOVERY OF GRID SERVICES

3.1 GENERAL

Grid resources are more often used for hosting several services running as Grid Services which need to be composed and accessed by Grid users. For instance, an education Grid can host numerous services in various domains such as mathematics. In the following research work, a matchmaking system is proposed in this chapter that performs matchmaking of requested Grid services with that of advertised ones to discover the best suitable service. The conventional matchmaking algorithm compares inputs and outputs of the advertised service with that of the requested ones. In this approach, the algorithm can also retrieve irrelevant services just because of their inputs and outputs are matching with that of requested service. The proposed matchmaking system addresses this limitation by providing necessary features for semantic description of services and discovering the best possible services among several advertisements. A matchmaking algorithm is implemented in which the system takes functionality of the required service into account prior to the comparison of inputs and outputs. The semantic similarity of inputs, outputs and functionality of advertised against the requested ones is determined using respective background ontology. Further, one wants to find functionally-similar services and later discriminate among the various retrieved ones using non-functional features such as QoS, security and performance etc. Applications utilizing computational Grid infrastructure require simultaneous allocation of resources, such as high end servers,
memory, disk space, CPU cycles and networks. Hence, selecting resources that ensure user preferred above said end system QoS is important. The matchmaking system proposed in this work uses WSDL-S to add semantics to the Grid service concepts. The service matchmaking algorithm infers knowledge from the description to discover closely related services. We use user-defined weighted QoS factors to obtain the suitable services that meet the user’s requirements. WSDL specification is used to describe service capabilities. However, WSDL lacks the ability to provide semantics to the service descriptions. The matchmaking algorithm proposed here, matches the QoS factors if multiple advertised services meet the requested capabilities. It allows the service requester to associate a weight to the QoS factors defined by the matchmaking system. The QoS factors, namely CPU cycles (CPU), network bandwidth (NBW), disk space (DS), memory buffers (MB), reliability (REL) and network latency (NL) are considered for case study.

3.2 ARCHITECTURE OF MATCHMAKING SYSTEM

The architecture of the proposed matchmaking system is shown in Figure 3.1. It identifies necessary components needed for creating service advertisements and discovery. The annotations to be added to the inputs, outputs and functionalities are obtained from the service provider using an easy to use interface. These annotations are then added to the WSDL file of the Grid service. The WSDL file is then converted into OWL-S description using WSDL2OWLS tool of OWL-S editor.
The OWL-S description describes the Grid services semantically which can be queried using any of available inference engine, for example, Algernon. The information retrieved from the OWL-S description will be used to match with that of the requested requirements. The matchmaking algorithm relies on the power of the concept of ontology. The algorithm compares the functionality, outputs and inputs (FOI) of the requested service with that of advertised one. The semantic similarity between the requested FOI and advertised ones are determined using the concept of ontology.

In order to determine the degree of similarity between the advertised and requested functionalities, a functional ontology is built which establishes the relationship between several functions of a domain of interest. Similarly, domain ontology has been built for determining inputs and outputs similarity. Based on the similarity of FOI, the algorithm retrieves more closely related services with respect to the requested ones. The matchmaking algorithm uses Algernon inference engine to determine the closeness between
the requested and advertised inputs and output types by referring this domain ontology. Similarly, the algorithm determines the closeness between the requested and advertised functionality by referring this functional ontology. A precise definition for both domain and function ontology is given below:

Definition 1: Domain ontology in our context can be defined as an ontology in which all possible datatype are described semantically. In such an ontology, each concept and its subconcept(s) represent a datatype that a function can have as an input and output arguments.

Definition 2: Function ontology in our context can be defined as an ontology in which all possible functionalities are described semantically. In such an ontology, each concept and its subconcept(s) represent a functionality and their names pertaining to the semantics of actions

3.3 CONCEPT OF SERVICE MATCHMAKING

Matchmaking refers to capability matching of the requested services with the advertised services. The capability of a service can be expressed in terms of functionality that the service offers, the kind of input it takes and output it produces. In the proposed algorithm, three parameters of a service are considered for matchmaking process. Also, with domain and functional ontology in the background, it is possible to determine semantic similarity of every parameter and compute different degrees of match. Although, several degrees of match were proposed in the literature (Micheal et al. 2004), the following three types of match are appropriate and refers to closely related services. We consider and define these degrees of matches in our perspective as follows:

Exact Match: This occurs if the advertised FOI of the service A exactly matches with that of requested service R.
Subsume Match: This match occurs if A describes greater functionality than that R requires. Also, A requires less specific input than specified in the request R and S is expected to return more specific output data whose semantics is exactly the same or very close to what has been requested by R.

\[ \text{A(FOI)} \equiv \text{R(FOI)} \rightarrow \{ \text{A(F)} \equiv \text{R(F)} \cap \text{A(O)} \equiv \text{R(O)} \cap \text{A(I)} \equiv \text{R(I)} \} \]

This match is obtained after referring the background ontology. If the requested concept is modelled as subconcept of the advertised concept in the respective background ontology, then we conclude that there exist a subsume match between the advertised and requested capabilities.

Disjoint: The requested service R does not match with the described service A

\[ \text{A(FOI)} \not\equiv \text{R(FOI)} \rightarrow \{ \text{A(F)} \not\equiv \text{R(F)} \cup \text{A(O)} \not\equiv \text{R(O)} \cup \text{A(I)} \not\equiv \text{R(I)} \} \]

The algorithm compares the functionality of the advertised services with that of the requested ones followed by output and input comparison. As a result of functionality comparison, the algorithm retrieves a set of advertised services (Functionally Matched Services (FM)) descriptions with their degree of match that is exact or subsume. By incorporating functionality matching in the initial stage of the algorithm, services performing relevant functions are only selected by the algorithm for further comparison. Also, the time taken to discover suitable services is reduced as most advertised services with irrelevant functionalities are filtered out.

**Algorithm:** FOI MatchMaker

**Inputs:** Advertised_owls A , Requester_Query (RF, RI, RO)
**Outputs:** FOI_Matched_Services FOI, Degree_of_Match M

Parse A(F1,F2,…,Fl), A(O1,O2,…,Om), A(I1,I2,…,In)

For each Ak
do
Degree Df = Match_Functionality(A(F) , R(F))
If Df ∈ {EXACT, SUBSUME}
Fn_match (FM).add(Ak)
end
endfor

Foreach FMk
do
Degree Do = Match_Output(A(O) , R(O))
If Do ∈ {EXACT , SUBSUME }
Output_match (OM).add(FMk)
end
endfor

For each OMk
do
Degree DI = Match_Input(A(I) , R(I))
If DI ∈ {EXACT , SUBSUME }
FOI_Matched_Services (FOI).add(OMk)
M = DI
End

**Module:** Match_Functionality

**Inputs:** A(F) , R(F)

**Output:** D_f

for each reqFn RF ∈ R(F)
do
for each advFn AF in A(F)
do
match m= Find_Degree_of_Match( R(F_i), A(F_j) )
if(m > bestMatch) bestMatch bm = m
bestMatchFn bmf = A(F_j)
if(bm == fail) return fail
if(bm < fail)
DegreeofMatch DM = bestMatch
}
R(F).remove( bmf )
}
return DM
}

Algorithm 3.1 Service Matchmaking Algorithm based on Functionality

In this stage, the outputs of the Functionally Matched services are matched with that of the requested and intermediate Degrees of match and a set of advertised services whose outputs match with the requested ones (Output Matched Services (OM)) are obtained as explained in the algorithm 3.1. Here, to measure the degree of closeness between the outputs, the matchmaker queries the domain ontology using algernon reasoner. Similarly, the inputs of Output Matched Services are compared with that of the requested and degree of match for input is obtained. The service(s) retrieved at this stage will be more related and relevant to the one requested. The algorithm 3.2 computes the final degree of match by aggregating intermediate ranks obtained in earlier comparisons and delivers to the requester.

Module: Find_Degree_of_Match

Inputs: concept C1, concept C2 , domainURI DURI
Output: Degree_of_Match M
if(!Exists(C1,DURI) or !Exists(C2, DURI))
  return fail
if(Equivalent(C1, C2, DURI))
  return exact
else if(Subsumes(C2, C1, DURI))
  return subsume
return fail

Algorithm 3.2 Degree of Match Module

3.4 IMPLEMENTATION

For testing and demonstration purpose, several Grid services offering various arithmetic and some commercial functionalities have been developed. These Grid services were then converted into corresponding OWLS description.

Figure 3.2 Functional Ontology
A sample functional ontology was created by considering different arithmetic and possible commercial operations as concepts so that it is referred while developing Grid services as shown in Figure 3.2. Similarly, a sample domain ontology which expresses possible inputs and outputs to the arithmetic functions was also created as shown in Figure 3.3.

**Matchmaking Algorithm**

The algorithm is implemented in three sequential phases. In every phase, we eliminate irrelevant advertised services by comparing their functionalities, outputs and inputs, respectively with that of the requested ones. It uses Algernon inference engine to interact with the functional and domain ontology to infer the degree of closeness between FOI of advertised and requested services. The advertised services for which the degree of closeness is either exact or subsume is considered for next phase of comparison. Once the algorithm completes the three phases, it delivers a set of services to the requester in which irrelevant set of advertised services
eliminated completely. We consider the following service request description shown in Table 3.1 for illustration of our algorithm. The algorithm starts comparing the parameters of the advertised with that of requested one by one and obtains various degrees of match. As discussed earlier, to infer the semantic information between the requested and advertised parameter, the algorithm refers background service ontologies as shown in Figures 3.2 and 3.3.

Table 3.1 Service Request Description

<table>
<thead>
<tr>
<th>Requested Service Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service.Name</td>
<td>Mathematical Service</td>
</tr>
<tr>
<td>Service.InputsNumber</td>
<td>2</td>
</tr>
<tr>
<td>Service.Inputtype</td>
<td>float,int</td>
</tr>
<tr>
<td>Service.Output</td>
<td>1</td>
</tr>
<tr>
<td>Service.Outputtype</td>
<td>Int</td>
</tr>
<tr>
<td>Service.FunctionalityNumber</td>
<td>2</td>
</tr>
<tr>
<td>Service.Funtionality</td>
<td>Add,Sub</td>
</tr>
</tbody>
</table>

Similarly, we consider a simple advertised service that performs various arithmetic operations viz., addition, subtraction, multiplication and division to demonstrate how semantic similarity is determined in this chapter. Also, we assume that these functions produce an integer output and accept two inputs of type double and integer. The service description file of this service is converted into corresponding OWLS descriptions using WSDL2OWLS tool.

The functionality, outputs and inputs are extracted from the OWLS descriptions using suitable Algernon query. In the initial phase of the algorithm, the functionality of the advertised and requested is compared. In
In our case, the requested functionality is “add” and “sub” whereas the advertised are addition, subtraction, multiplication and division. It means that the advertised functionalities subsume the requested functionalities and this advertised service will satisfy the request. In the second phase, we determine the semantic similarity of the outputs between the advertised and requested. In this case, both the requested and advertised output type is “int”. While referring to the domain ontology, it is found that both the output types are matching exactly. However, in the final phase, while comparing the inputs, it is found that only one of the requested inputs is exactly matching with that of the advertised one. To determine the semantic similarity between the other input variable, the domain ontology is referred. In our domain ontology, the “float” type is modelled as sub concept of “double”. It means that “double” type offers more capability than that of “float” which is requested. Hence, the algorithm concludes that there exists a “subsume” relationship between the advertised and requested inputs.

3.5 QoS BASED MATCHMAKING ALGORITHM

The matchmaking algorithm described in the earlier section has been extended to consider QoS parameters associated with every matching Grid services. Figure 3.4 shows the architecture and the various modules involved in the matchmaking system.

The description module enables semantic descriptions of the Grid services through the WSDL-S tool. The Grid Service Provider generates the WSDL based service description file. This description only supports keyword based discovery of Grid services and lacks the expressiveness of semantic annotations. WSDL-S is used to provide semantics to the concepts involved in the Grid service and thereby creating service advertisements. The concepts considered in this research work includes the data used in the service, input and output arguments of the service, functionality offered by the service. This
approach allows the integration of non-semantic and semantic data in the service description itself.

![Diagram of QoS based Service Matchmaking System](image)

**Figure 3.4 Architecture of QoS based Service Matchmaking System**

We use domain ontology for determining similarity between the advertised and requested concepts. We define domain ontology as an ontology in which all possible concepts are described semantically. In such ontology, each concept and its subconcept(s) represent some form of knowledge present in the service. For example, the datatypes of the input and output parameters of a service can be represented well using the concept of ontology.

The clustering mechanism proposed in the system enables faster discovery of intended Grid services. This module clusters similar WSDL-S descriptions into groups and stored in the UDDI registry. The clustering is based on the functionality a service offers and a unique index is provided to each cluster. This grouping minimizes the number of advertisements to be
compared against the requested advertisements resulting in the quicker
discovery of the service requested.

The discovery module matches the service requests against the
service advertisements and returns the service that meets the requested
capabilities. The matchmaking algorithm first identifies the suitable cluster
containing the services related to the functionality requested. Once a suitable
cluster is obtained, the algorithm retrieves the advertisements from that
cluster one by one and compares the other parameters. With this approach, we
eliminate irrelevant services from being compared and this improves the
accuracy of the end results. Users may have different QoS requirements with
respect to the service requested, for example, cost and reliability in addition to
functionality, inputs and outputs. Hence, when the service requests are
matched by multiple services on a Grid, it becomes difficult to choose the
most appropriate service that meets the requirements. A lot of research is
going on in trying to solve this problem. Devising a generic QoS based
matchmaking algorithm is a difficult task as users have varied QoS
requirements. Our algorithm attempts to solve this problem by using several
QoS factors associated with user-defined weights for matchmaking. The
discovery module allows the users to specify their own priorities in terms of
weights to the QoS factors that the matchmaking system supports. For
example, a user might require a highly reliable service provider more than one
that could process the request fastest. In this case, he would prioritize
reliability by giving it a higher weight than the other factors. Based on the
weighted QoS values, the discovery module computes the overall QoS and
selects the most appropriate service.
3.6 WEIGHTED QoS FACTORS

We model the QoS needed by a service request in terms of six different factors viz., CPU cycles (CPU), network bandwidth (NBW), disk space (DS), memory buffers (MB), reliability (REL) and network latency (NL). These factors can be computed as shown below:

\[
\begin{align*}
\text{CPU} & = \frac{R_{\text{requiredCPU}}}{S_{\text{availableCPU}}} \\
\text{NBW} & = \frac{R_{\text{requiredNBW}}}{S_{\text{availableNBW}}} \\
\text{DS} & = \frac{R_{\text{requiredDS}}}{S_{\text{availableDS}}} \\
\text{MB} & = \frac{R_{\text{requiredMB}}}{S_{\text{availableMB}}} \\
\text{REL} & = \frac{R_{\text{requiredREL}}}{S_{\text{availableREL}}} \\
\text{NL} & = \frac{R_{\text{requiredNL}}}{S_{\text{availableNL}}}
\end{align*}
\]

where \( R_{\text{requiredCPU}} \) is the CPU cycles requested, \( S_{\text{availableCPU}} \) is the CPU cycles available at the service provider and so on.

The parameters representing available resources such as \( S_{\text{availableCPU}} \) can be obtained from the service provider using any Network Monitoring Tool such as Network Weather Service (NWS). We define reliability as the ratio between the number of successful responses from the service provider (NoS) and the total number of service requests to that service provider (ToS).

\[
\text{Reliability} = \frac{\text{NoS}}{\text{ToS}} \quad (4.1)
\]

A separate java module is developed to obtain these QoS parameters periodically from the service provider and will provide the inputs to the matchmaking system.

The parameters required by the service request such as \( R_{\text{requiredCPU}} \) are obtained from the service requester. The requester also specifies the
priorities of the QoS in terms of weights \(W\) to select a single suitable service when the requested capabilities are met by more than one service. The matchmaking system computes overall QoS requested by the user as a function \(f\) of weighted QoS parameters as shown in equation \(2\).

\[
\text{Overall QoS} = \sum (w_1 \times \text{CPU}, w_2 \times \text{NBW}, w_3 \times \text{DS}, w_4 \times \text{MB}, w_5 \times \text{REL}, w_6 \times \text{NL})
\]

where \(w\) is the weight associated with every QoS factor and

\[
\sum w_i = \text{No.of QoS} \times n. \quad n=2, 3, 4,\ldots
\]

In our case, since we considered six QoS factors, \(\sum w_i\) is a multiple of six. This helps the requester distribute the weights in integral values. Also, if “\(n\)” increases, the weights to be assigned to various factors can be distinguished clearly and thereby enables the accurate specification of priority.

### 3.7 ILLUSTRATION

We illustrate our QoS modelling and its effect on the discovery of the best service in this section. For simplicity, we consider the value of “\(n\)” as 2 and hence \(\sum w_i\)=12. We simulate the effect of default as well as user defined weights on the accuracy of service discovery by assuming a set of QoS factors for two different services \(S_1\) and \(S_2\). We clearly distinguish the selection of services in both the cases.

Let \(AS_n\) be the set of QoS factors of the service \(S_n\) as \(AS_n = \{\text{CPU, NBW, DS, MB, REL, NL}\}\). We assume the values of the sets \(AS_1\) and \(AS_2\) for \(S_1\) and \(S_2\) as follows:

\[
\begin{align*}
AS_1 &= \{0.1, 0.3, 0.4, 0.5, 0.9, 0.8\} \\
AS_2 &= \{0.8, 0.1, 0.5, 0.4, 0.3, 0.1\}
\end{align*}
\]
Case I

In this case, the user does not specify any priority towards QoS. Hence, the matchmaking system distributes $\sum w_i$ equally to all QoS. The Overall QoS for both the services $S_1$ and $S_2$ can be calculated using (4.2) as

$$\text{Overall QoS} = \sum (A_{Sn} \times W)$$

For $S_1$,

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>NBW</th>
<th>DS</th>
<th>MB</th>
<th>REL</th>
<th>NL</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{S1}$</td>
<td>0.1</td>
<td>0.3</td>
<td>0.4</td>
<td>0.5</td>
<td>0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>$W$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$A_{S1} \times W$</td>
<td>0.2</td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
<td>0.8</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Overall QoS $= \sum (A_{S1} \times W) = 6.0$

Similarly for $S_2$,

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>NBW</th>
<th>DS</th>
<th>MB</th>
<th>REL</th>
<th>NL</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{S2}$</td>
<td>0.8</td>
<td>0.1</td>
<td>0.5</td>
<td>0.4</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td>$W$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$A_{S2} \times W$</td>
<td>1.6</td>
<td>0.2</td>
<td>1</td>
<td>0.8</td>
<td>0.6</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Overall QoS $= \sum (A_{S2} \times W) = 4.4$

With the overall QoS obtained for $S_1$ and $S_2$, we observe that when the QoS factors are assigned default weights, where the user does not prioritize his requirements, the matchmaking selects $S_1$ as the best match for the user’s requirements. It also infers that the weights considered may not have any effect on the ultimate selection of services.
Case II

Let us consider another case in which the user wishes to give CPU cycles and NBW a higher priority than the other four factors. With the same set of QoS factors for both $S_1$ and $S_2$, we compute overall QoS as shown below:

For $S_1$,

$$\text{Overall QoS} = \sum (A_{S_1} \times W) = 4.2$$

For $S_2$,

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>NBW</th>
<th>DS</th>
<th>MB</th>
<th>REL</th>
<th>NL</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{S_2}$</td>
<td>0.8</td>
<td>0.1</td>
<td>0.5</td>
<td>0.4</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td>W</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$A_{S_2} \times W$</td>
<td>3.2</td>
<td>0.4</td>
<td>0.5</td>
<td>0.4</td>
<td>0.3</td>
<td>0.1</td>
</tr>
</tbody>
</table>

$$\text{Overall QoS} = \sum (A_{S_2} \times W) = 4.9$$

Comparing the overall QoS of $S_1$ and $S_2$, the matchmaking system infers that $S_2$ is a better match for the user's requirements than $S_1$. With this illustration, we argue that the proposed matchmaking system reduces the ambiguity in the discovery of services and gives the user more flexibility in specifying his requirements.

The proposed approach for service matchmaking is not suitable for Grid resource discovery due to the metadata associated with a computational resource is different with that of a service. Thus, a different approach is proposed in the next chapter for representation of Grid resources using ontology and discovery of Grid resources based on the semantics present in the request.