

Dynamic Web Service Selection for Workflow Optimisation

Lican Huang, David W. Walker, Yan Huang, Omer F. Rana

School of Computer Science, Cardiff University

5 The Parade, Roath

Cardiff CF24 3AA, UK

{lican.huang, david.w.walker, yan.huang, o.f.rana}@cs.cardiff.ac.uk

Abstract

This paper investigates dynamic Web Service selection for workflow optimisation, which involves selection from many semantically equivalent Web Services in a service-rich environment. The dynamic selection of Web Services involves discovering a set of semantically equivalent services by filtering available services based on service metadata, and selecting an optimal service based on real-time data and/or data recorded during prior executions of a service. Given that a user is aware of which services are critical to a workflow session, we focus on how these services can be discovered (based on some optimization criteria), and subsequently integrated into an existing workflow. We deal with the above issues and present an architecture and implementation for dynamic Web Service selection for optimising workflow.

1. Introduction

In a service-rich environment, it is possible that multiple copies of a service may exist. For instance, when using Web Services technology, there may be multiple semantically equivalent versions of a service on different machines, each making use of a different implementation (such as programming language or algorithm). Selecting an optimal service from this set of equivalent services is a decision that is often undertaken manually by a user. Most techniques for selecting from a set of such services are determined at design time. Maximilien et al. [1] propose a framework and ontology for dynamic Web Service selection. Liu et al. [2] propose the use of Quality of Service criteria to support service selection. Keidl et al. [3] have proposed the serviceGlobe environment that implements dynamic service selection. There is, however, little support for integrating such dynamic Web Service selection with a workflow engine. We present a mechanism to support dynamic Web Services selection for workflow optimization that invokes the optimal services at runtime. This corresponds to a late binding operation, whereby Web Service instances are resolved based on a user-defined set of optimization criteria.

The motivations for dynamic Web Service selection for workflow optimization are improving fault-resistance and performance based on factors that cannot be determined at design time. When one service instance fails, the workflow engine should be able to utilize another service instance. For computationally

intensive Web Services, selecting services with a specific performance profile is beneficial to the execution of the entire workflow. There are various use scenarios that necessitate the choice of “optimal services” only at run-time, such as an image analysis/visualization service that needs to respond within a particular time. Another scenario involves the choice of services in a changing service environment, where services may not persist over long time periods.

Dynamic Web Service selection for workflow optimization is beneficial especially to scientific workflows. Although scientific workflow is a successor of business workflow, it differs from business workflow in many ways [5]. Many scientific workflow applications are computation-intensive, and may be long-running – lasting weeks or even months. Selection of optimal Web Services among the available ones can shorten the computation time. When running a complete scientific workflow application, if one service fails, the whole workflow must be run again. Dynamic Web Service selection involves the discovery of a list of candidate services, and if one service fails, then the next one can be tried, thereby avoiding the need to repeat the whole workflow. Scientific workflow applications may require heterogeneous computing resources such as supercomputers, clusters, and networks of workstations/PCs. Time-critical applications need a guarantee of completeness within a given period of time, which requires some way of predicting the time it would take for a given computation to complete. Some properties change dynamically, such as machine load.

Therefore, dynamic selection involves choosing a suitable service according to the current conditions and service performance models. To support the selection process, we make use of logging tools that monitor various properties of a Web Service, and record the output into a database. This data can then be analysed for possible patterns of execution behaviour associated with a service in a given context. Scientific problem-solving usually involves the invocation of a number and variety of analysis tools. There is therefore always a need for logging the execution of Web Services. Dynamic web service selection can be based on the use of data recorded into a database.

Often, there are some Web Services which are critical to the operation of a workflow, such as experimental simulations, whilst others such as data format transformations may not be so critical. Due to the extra delay of dynamic selection and invoking of Web Services, only critical activities in workflow applications are required to be dynamically selected. We assume here that such critical services are long running, and therefore the time taken to choose between them is significantly less than the time involved in their execution. In a workflow script, determining which services are to be dynamically selected is often undertaken manually. Assuming the users are aware of the structure of their workflow, the users can easily decide which services are critical. We use a proxy service as a placeholder for a Web Service to be dynamically selected in a workflow script.

This paper presents results of the EPSRC-funded Workflow Optimisation Services for e-Science Applications (WOSE) project. In the WOSE prototype the dynamic selection of services is performed through a Proxy Service interacting with a Discovery Service and an Optimisation Service. The structure of the rest of this paper is as follows. Section 2 presents an overview of dynamic Web Service selection for workflow optimization; Section 3 describes the Discovery Service; Section 4 presents the Optimization Service; Section 5 describes our prototype implementation of dynamic service selection; and finally Section 6 presents conclusions and ideas for future work.

2. Dynamic Web Services Selection for Workflow Optimization

The dynamic selection of Web Services includes finding a list of candidate services from one or more registries, selecting an optimal service from this list based on real-time

and historic data, and invoking the selected service dynamically. Our approach to dynamic Web Service selection for workflow optimization integrates these functions into the workflow process by using a Proxy Service as the late-binding service placeholder. The Proxy Service itself is a Web Service. In a workflow script an abstract service description is passed as a parameter to the Proxy Service, which selects and invokes a matching concrete service implementation that is optimal relative to some specified criteria. The parameters input to the Proxy Service also include the input parameters of the late-binding service. The output from the Proxy Service is simply that of the dynamically selected service, and this can be then input to other activities in the workflow. Abstract workflow description in a language like SCUFL is useful for the users who do not have knowledge of particular implementation technologies, such as Web Services. Such an abstract description can also be translated to different technology-specific implementations. In our abstract workflow description, the late-binding Web Services and the metadata for service discovery and optimal service selection are described. An XSLT conversion tool developed by us is used to transform an abstract workflow description and insert it into a BPEL4WS script. This simplifies the design of workflows by application scientists.

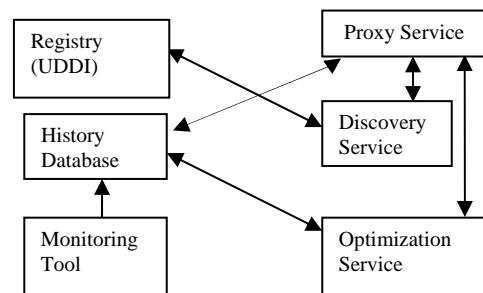


Figure 1 Architecture of dynamic Web services selection for workflow optimisation

Figure 1 shows the WOSE architecture of dynamic Web Service selection for workflow optimisation. Currently, the Universal Description Discovery and Integration (UDDI) registry is used to host service descriptions. This registry primarily provides an identifier for a service, a service metadata for semantic definition, and the location of the WSDL file describing the service interface (via a URL). The database shown in Fig. 1 contains the history data on previous service invocations, such as the response time of services. When

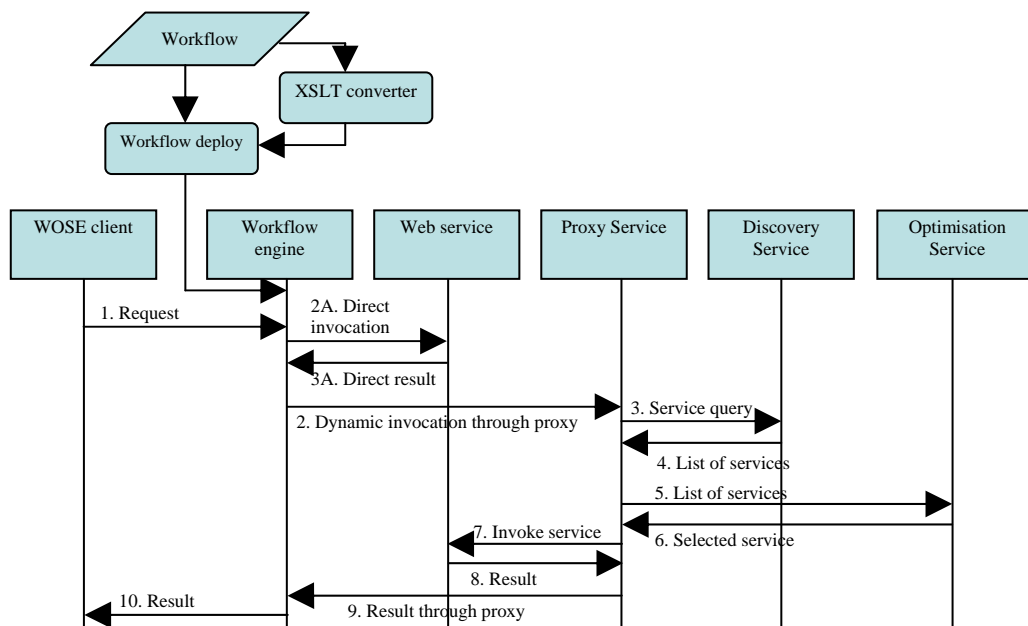


Figure 2. Message sequence of workflow incorporating dynamic Web Service selection.

many semantically equivalent copies of a service are found, the optimization service selects an optimal service based on the history database, and/or real-time data such as the peak speed, current load, and current available memory of the machine hosting the service. Figure 2 shows the sequence diagram for workflow execution, which integrates the Proxy Service in the workflow.

2.1 Abstract workflow description

In general, the process of workflow design can be viewed as progressing from abstract functionality to concrete implementation. Both abstract and concrete services need to be described by metadata in order to support this progression [9]. Web Service workflow standards, such as BPEL, do not support the levels of abstraction needed by most scientists. In scientific workflow applications, varying levels of abstraction should be supported [10]. Web Service standards do not support specification of processes or resources at a semantic level, and do not support provenance or authoring and versioning (although some progress is being made on this in the “Web Services Modelling Ontology” (WSMO) project, the results of this project are still not available for use and deployment). An abstract workflow description can add these properties and capabilities. An abstract workflow description describes tasks in terms of functions, not the executable resources. Other tools can transform these abstract workflow descriptions into concrete workflow scripts,

such as BPEL scripts. The abstract workflow description used in the WOSE project is based on OWL-S and SCUFL [6], and describes dataflow and control flows and service metadata, and allows the specification of which abstract services are to be dynamically selected. Our abstract workflow description uses XML as its language. An XSLT conversion tool can transform an abstract workflow description into a concrete workflow description by adding data type conversions at design time, and placing proxy services as placeholders for services that need to be discovered dynamically in the workflow graph.

2.2 Proxy Service

Workflow scripts make use of a workflow description language to control the data flow and control flow of a workflow application. In a workflow script, most of the Web Services may be chosen at design time. However, for certain critical Web Services (i.e., those services which could be a potential bottleneck), dynamic selection of Web Services is beneficial to the whole workflow optimization. This is based on the premise that choosing the same service in all instances is likely to be less beneficial than taking dynamic properties of the service and its hosting environment (such as machine load) into account. A user must therefore specify which Web Services in the workflow are to be invoked dynamically. We use a Proxy Service to act as a placeholder for a service that has to be invoked dynamically. The Proxy Service is also considered to be a standard service in our

workflow system. Its input parameters include the semantic metadata for the service description and parameters of the Web Service to be selected. Its output is the result of the dynamically invoked service.

The Proxy Service is used as an adaptor for dynamically selecting and binding to Web Service in a workflow script. In a workflow application, some activities are critical in terms of their execution time or fault-tolerance properties. We use a Proxy Service to replace these activities at the appropriate place in the workflow script, and make use of semantic metadata, from which semantically equivalent services are discovered. Among these semantically equivalent services an optimal service is selected and invoked (if multiple services are found, the first service is selected), and its output is passed to the next activity in the workflow.

2.2.1 Proxy Service interface

A workflow is a composition of Web Services, within which concrete Web Services are selected at design time. However, some workflow applications require late-binding of Web Services for optimal performance and/or fault-resistance. Without changing or extending the workflow languages currently in use, WOSE uses a Proxy Service to dynamically discover and invoke services. Instead of a concrete Web Service, a Proxy Service is substituted that discovers and selects the optimal service based on service metadata, and invokes this service dynamically.

The WSDL interface of the Proxy Service is as follows:

```
<wsdl:message name="ServiceProxyResponse">
  <wsdl:part name="ServiceProxyReturn"
    type="xsd:string" />
</wsdl:message>
<wsdl:message name="ServiceProxyRequest">
  <wsdl:part name="servicemeta" type="xsd:string"/>
  <wsdl:part name="querymethod" type="xsd:string"/>
  <wsdl:part name="optimizationMETA"
    type="xsd:string"/>
  <wsdl:part name="optimizationMode" type="xsd:string"
    />
  <wsdl:part name="operation" type="xsd:string" />
  <wsdl:part name="parameters" type="xsd:string" />
</wsdl:message>
```

Here, *servicemeta* gives a semantic definition the abstract service, which is used by the Discovery Service to find semantically equivalent services. *querymethod* specifies the query mechanism used by the Discovery Service to search for services. The *byNAME* method is used by the Discovery Service to discover services with the same name in UDDI

registries. The *byMETA* method is used by the Discovery Service to discover services which match the *servicemeta* in UDDI registries. The *byONTOLOGY* method is used by the Discovery Service to utilise ontology-based methods. The *optimizationMETA* gives a description of the problem-specific optimization model. *optimizationMode* is the mode for selecting the optimal Web Service, such as execution time, degree of trust in results, etc. *operation* corresponds to the actual function performed by the late binding service. *parameters* are the parameters to be passed to the late binding services. *ServiceProxyReturn* is the result returned by the late-binding service.

2.2.2 Service metadata

Semantic metadata for service description is expressed in RDF, and includes the service name, its function, service parameters and type, and operation name. This metadata can be used by the Discovery Service to find matching services in the UDDI registry, and includes enough information to invoke the selected service dynamically. Because of the limited storage available for items in UDDI, we use a URL that points to an XML file which defines the service metadata of a service in the UDDI registry.

2.2.3 Dynamic invocation of Web services

The URL of the WSDL file of the optimal service can be obtained using the Optimization Service (see Section 4). The WSDL file is downloaded from this URL and parsed. The service is invoked using Dynamic Invocation Interface (DII) technology based on the WSDL file of the service. The required information, such as the Web Service name, its namespace, operations, ports, and parameters, can be found from the WSDL file associated with the service. The input and output of the Proxy Service are of type String. The actual contents of the input and output data structures are described in XML. We have many adaptors to transform the String type of input or output data of a Proxy Service into various data types to match the ports of other Web Services which link the Proxy Service to other services in the workflow script.

2.2.4 Fault-tolerance of Web services

The Proxy Service enables the fault-tolerant execution of workflow scripts. If a service fails, then an alternative semantically equivalent service will be substituted. The pseudocode of the Proxy Service is as follows:

```

Public String ServiceProxy(String servicemeta, String
querymethod,String optimizationMETA, String
optimizationMode , String operation, String parameters){
    Set values of uddiregistries;
    Servicelist = discoveryService(servicemetadata,
querymethod, uddiregistries);
    While (true)
    {
        Service= Optimizationservice(Servicelist);
        If result =dynamicInvoke(service) fails{
            Updates fail-timestamp in Database;
            Logging;
        }
        Else
        {
            Update average_response_time in
            database;
            Logging;
            Stop;
        }
    }
    Return result;
}

```

2.2.5 Logging of processes via the Proxy Service

In scientific workflow applications, we often want to record which services and data were used to reach a scientific conclusion. For example, a workflow application that locates a gene in a DNA sequence may include several gene analysis Web Services. The trace (or provenance) of which Web Services and gene banks were used is important to scientists. Because the Proxy Service is independent of the workflow engine used, specific logging data can be obtained and stored in databases. In general, the logging data includes the optimal Web Services that were invoked, the source of the input data, etc. Scientists can judge the trustworthiness of the scientific conclusions obtained through the execution of the workflow by examining the logged data.

3. Discovery Service

A variety of Web Services with semantically equivalent functionality may co-exist. Moreover, these Web Services may be updated over time. The representation used for discovery includes items such as service name, interface, performance, and various parameters associated with quality of service issues. If we use UDDI to find Web Services, to catalogue these services, and to register service properties, such as performance, we are not able to record such properties directly into the UDDI registry. Using P2P technologies, we can use a domain-specific ontology to classify Web Services and their properties.

The output from the Discovery Service is a list of Web Services that match the properties of the service metadata definition. The Optimization Service uses this list to choose the optimal Web Service. If multiple services are found by the Optimization service, the first one is selected.

Due to the current limitations of UDDI for semantic description, we have extended UDDI by adding metadata and ontology data attachments for service description – this approach is similar to the work of Miles et al. [4]. The Discovery Service uses these metadata or ontology data to match Web Services with service metadata defined in the workflow script. The metadata attached in the UDDI registry should define at least the following items.

```

Service Name and its semantic definition
Array of Operations and semantic definitions {
    Input parameters interface
    Output parameters interface
}

```

The Discovery Service interface is as follows:

```

<wsdl:message name="ServiceDiscoveryResponse">
<wsdl:part name="ServiceDiscoveryReturn"
type="xsd:string" />
</wsdl:message>
<wsdl:message name="ServiceDiscoveryRequest">
<wsdl:part name="servicemeta" type="xsd:string" />
<wsdl:part name="querymethod" type="xsd:string" />
<wsdl:part name="uddiregistries" type="xsd:string" />
</wsdl:message>

```

3.1 Web Service matchmaking methods

querymethod is the method used by the Discovery Service to find and filter the services available. There are three methods: byNAME, byMETA and byONTOLOGY. The pseudocode for the Discovery Service is as follows:

```

Public string ServiceDiscovery (String servicemeta,
querymethod, String uddiregistries)
{
    Switch querymethod
    Case byNAME
        Look up the matched service name in
        uddiregistry(ies)
    Case byMETA
        Look up the matched metadata in the description
        entity of businessService entity of
        uddiregistry(ies).
    Case byONTOLOGY
        Look up the URL of semantic definition XML
        File and matchmake the semantically equivalent
        service.
}

```

3.1.1 Discovery by service name

The byNAME method is used to query all semantically equivalent services with a specified name. The byNAME method would

typically be used where the services are registered by the same business entity but with different ways to access the service (i.e. the existence of different bindingTemplate in UDDI. No extensions are needed to the information contained in the UDDI registry.

3.1.2 Discovery by metadata

The byMETA method is used to query all services by examining the services' metadata. The byMETA method would typically be used where all service providers conform to a particular metadata specification, and have the ability to publish their own services in the UDDI registries. By using the metadata, the Discovery Service can find the semantically equivalent services.

This method needs to register metadata information in the UDDI registry. This information includes service name, service ID, list of operation names, operation IDs, and their input, as well as output, and data types. These items are registered in the vector of description entities in the business service entity. The description entity uses <wosemeta> and </wosemeta> to indicate that the description is for metadata information. The registration of metadata is as follows:

```
<businessService serviceKey ="37B49950-90A0-11D9-9950-BB91FEE6D433" businessKey = "5D4F01A0-8FE2-11D9-81A0-832EB7272640" >
  <name>optdemo</name>
  <description><wosemeta serviceName= "optdemo"
serviceID="1239-3514-6196-8373"/></wosemeta>
</description>
<description><wosemeta operationName
="optimizationdemo" operationID="1351-3612-7812-9156"><inputtype>string</inputtype><outputtype>string</outputtype></wosemeta></description>
...
</businessService>
```

The Discovery Service is passed the service metadata through the input parameter *servicemetadata* of the Proxy Service. The *servicemetadata* parameter of the Proxy service includes serviceName, serviceID, operationName, operationID, and input as well as output types. The Discovery Service uses these metadata to match items stored in the UDDI registry. serviceID and operationID are used to identify the semantically equivalent services and operations. The service name and operation name can be omitted. Data types are used to match the input and output data types of connected workflow activities.

3.1.3 Discovery by ontology

The byONTOLOGY method is used to query all semantically equivalent services by semantic matchmaking. This would typically be used where there are many service providers who publish the services according to the schema encoded in a service ontology. The service providers are loosely connected or without any relationship. How many services and where these services are located cannot be decided at the workflow design time. For example, if we want to book the lowest priced hotel, we want our query to consider as many hotel reservation Web Services as possible. A Discovery Service using the byONTOLOGY method can find all the hotel reservation Web Services. The Proxy Service can invoke every service in the list to find the lowest priced hotel. This requirement is difficult to satisfy by using traditional workflow technologies. But using the Proxy Service it is simple to accomplish this function. Another case is the choice of an optimal scientific data analysis service, where a local data set needs to be processed. We also need to find as many analysis Web Services as possible, and invoke every service, compare the values and find the optimal value. This cannot be done at design time, but the Proxy Service will do it quite effectively.

Due to the generally large size of an ontology for defining services and WSDL items, we register the URL of an XML file for the ontology into the description of business services, as follows:

```
<businessService serviceKey ="14E69DE0-D28D-11D9-80990AF8150A" businessKey = "7D92E5E0-D1C2-11D9-A5E0-CF2D2BB38AE7" >
  <name>optdemo</name>
  <description><woseOntology>http://131.251.47.147:8080/axis/optimaldemo.xml</woseOntology></description>
...
</businessService>
```

The Discovery Service byONTOLOGY method deals with two issues: access to ontologies through registries, and matchmaking services. The ontology uses an XML file, or other storage mechanism, to store the ontology contents and the URL of this storage location is published, usually by the service owner. Matchmaking is based on the OWL-S [7] ontology. OWL-S supplies Web Service providers with a core set of markup language constructs for describing the properties and capabilities of their Web Services in unambiguous, computer-interpretable form. OWL-S builds on top of OWL. OWL-S includes ServiceProfile, ServiceGrounding and

ServiceModel. ServiceProfile is concerned with what the service does and is related to our service discovery approach. ServiceProfile provides a particular class of service capabilities, while adhering to some client-specified constraints. The service ontology includes a service domain, an interface, and other properties such as QoS. The matchmaker finds the suitable services by using the *servicemetadata* passed as an input parameter to the Proxy Service, and the service ontology descriptions from the XML files.

4. Optimization Service

The Optimization Service selects the optimal service from the list of semantically equivalent services based on the criteria for optimization. The criteria may cover many aspects such as performance, quality of service, trust, cost, etc. The input to the Optimization Service is the list of semantically equivalent services. The output is the selected optimal service. Quality of service requirements for Web Services are availability (is it running), accessibility (can I run it now), integrity/reliability (will it crash while I run/how often), throughput (how many simultaneous requests can I run), latency (response time), regulatory (conformance to standards), and security (confidentiality, authentication). Web Service performance is affected by many factors such as algorithms, coding languages, Web Service container, hosting machine CPU power, load, bandwidth, etc. Microsoft releases a WS Test 1.1 Benchmark[8] to test Web Service performance on different platforms such as .NET 2.0, .NET 1.1, Sun JWS DP 1.5 and IBM WebSphere 6.0. These platforms are quite different in performance. Other factors affecting Web Service performance are payload size (the total amount of bytes exchanged in each transaction), simultaneous users accessing the system, encoding styles (RPC-style, document-literal, and services with attachments), etc.

When the Discovery Service finds multiple semantically equivalent services in registries, such as UDDI registries, the implementation of these services (such as algorithms, coding languages, encoding styles, and environmental factors such as CPU power, service container, etc) are already defined. We call these “static factors” as they do not change for a particular service. However, factors such as problem scale, payload size, number of simultaneous users, CPU load, and bandwidth are changeable. Therefore, the model for evaluation of the performance of a Web Service will include

these factors. Problem scale is relative to the specific problem and related with the computational complexity of the problem. For example, if we calculate the norm of a matrix, the order of the matrix will be an important factor in performance. Therefore, the performance model needs to consider the problem scale as a specific factor which is different in different Web Services. Problem scale should be published by Web Service providers.

Performance can be modelled as a function of the above factors, but the exact relation between these factors still requires further research. Because the static factors are difficult to represent in a metric, we can use average-response-time to represent them. Average-response-time, for instance, is calculated by the response time divided by the number of invocations. (The Proxy Service has a strategy to record response times).

The description entity of the binding model of the business service entities in the UDDI can be used to store the initial optimal ratings such as response time.

An example of such a description is as follows:

```
<optimization><wsdlURL>http://131.251.47.227:8080/active-
bpel/services/optdemo?wsdl</wsdlURL><ratings>15</ratings></optimization>
```

The pseudocode for the Optimization Service is as follows:

```
Public string ServiceOptimization (String serviceURL,
String optimizationMode, String optimizationMETA)
{
    Swith optimizationMode
        Case byPERFORMANCE
            Look up the average-response-time and
            real-time computer load,etc.
        Case byTRUST
            Look up the trustworthiness of the services.
}
```

optimizationMode is the method that the Optimization Service uses to select the optimal service. *byPERFORMANCE* is the default mode, which means that the optimization is based on the performance of Web Service. The *byTRUST* mode indicates that the optimization is based on the trustworthiness of the Web Services. Real-time data such as machine load, bandwidth, and number of simultaneous users can be obtained by third party tools, or by specific monitoring services. The history database consists of the following items for each service:

Service name
URL of WSDL file
Fail timestamp
Frequency of failure
Trust ratings
Average-response-time
Average load

The Optimization Service can use both real-time data and history data to select the optimal service.

5. Implementation of Dynamic Service Selection

We have implemented the ideas presented in the preceding sections in the WOSE prototype. The Proxy Service includes support for service discovery, optimal selection, and dynamic invocation of the selected Web Service. The Proxy Service is a standard Web Service implemented in Java. The Proxy Service acts as a placeholder for a service instance, and contains a description of the service instance that needs to be discovered (such as input/output types). The Proxy Service is also responsible for invoking the discovered service, and marshals data values to/from the service instance. The Discovery Service is responsible for finding all the semantically equivalent Web Services from extended UDDI registries, based on the service description. Based on the list of discovered services, an Optimization Service is now responsible for interacting with a performance history database and real-time data to choose between them (based on user criteria). The URL of the WSDL file for the selected optimal service is returned to the Proxy Service. The Proxy Service reads the selected Web Service's WSDL document to find information such as Web Service name, its namespace, operations, ports, and parameters. Invocation of the service uses a Dynamic Invocation Interface (DII) approach based on the WSDL file of the selected service, and outputting the result to the next Web Service within the workflow.

In our implementation, we use ActiveBPEL as the workflow engine, BPEL4WS as the workflow language, and Tomcat and Axis as Web Service containers. The history database is a MySQL database. The JUDDI registry maintained by the Welsh e-Science Centre is used as the UDDI registry. The example workflow scripts we used are based on the SCUFL language. An XSLT converter is used to transform these scripts into BPEL4WS.

6. Conclusions and Future Work

We have presented an architecture for dynamic Web service selection within a workflow session and its prototype implementation as part of the WOSE project. We integrate critical activities which are to be selected optimally into the workflow scripts using a Proxy Service. The Proxy Service enables the discovery of semantically equivalent services. Thus, instead of putting a specific Web Service in the workflow script, we instead insert a service proxy into the workflow script at design time. In contrast to previous work on dynamic selection of Web Services, we demonstrate how the approach can be used within a workflow enactment process. The Discovery Service has three query methods. The Optimization Service selects the optimal service based on history data and real-time data which are obtained by the use of third party monitoring tools to monitor execution of a Web Service and to gather workload information from a particular machine. Our future work will focus on refining the optimization model, specifying an abstract workflow description, and implementing a tool to transform it into a BPEL script. We also intend to apply our work to real examples of scientific workflow applications.

References

- [1] E.Michael Maximillien, Munindar P.Singh, *A Framework and Ontology for Dynamic Web Services Selection*, IEEE Internet Computing, 2004 September.
- [2] Yutu Liu, Anne H.H.Ngu, Liangzhao Zeng, *QoS Computation and Policing in Dynamic Web Service Selection WWW2004*, 2004 New York, USA.
- [3] Markus Keidl, Alfons Kemper, *Towards Context-Aware Adaptable Web Services WWW2004*, 2004, USA.
- [4] Simon Miles, Juri Payne, Michael Luck and Luc Moreau, *Towards a protocol for the attachment of metadata to grid service descriptions and its use in semantic discovery*, Scientific Programming 12(2004) 201-211.
- [5] Munindar P. Singh and Mladen A.Vouk *Scientific workflows: Scientific computing meets transactional workflows*, <http://www.csc.ncsu.edu/faculty/mpsingh/papers/databases/workflows/sciworkflows.html>
- [6] <http://taverna.sourceforge.net/>
- [7] DAML- Coalition. OWL-S 1.1 Release <http://www.daml.org/services/owl-s/1.1/>
- [8] Microsoft, *Comparing Web service performance*, http://msdn.microsoft.com/vstudio/java/compare/webservice_perf/default.aspx
- [9] James Blythe, et.al. *Types of editors and specifications*, GriPhyN technical report 2004-23 <http://pegasus.isi.edu/pegasus/publications/editors.pdf>
- [10] Matthew Addis, et.al. *Experiences with e-Science workflow specification and enactment in bioinformatics*, <http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/108.pdf>