Dynamic Invocation, Optimisation and Interoperation of Service-oriented Workflow

Licam Huang, David W. Walker, Omer F. Rana and Yan Huang
School of Computer Science, Cardiff University
5 The Parade, Roath
Cardiff CF24 3AA, UK

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

Abstract This paper presents the WOSE workflow framework, which features runtime optimisation, dynamic Web Service invocation, and interoperation between different workflow languages and engines. We discuss strategies for choosing the optimal service and workflow engine from several semantically equivalent Web Services, and several possible workflow engines. The WOSE framework divides a workflow application composed of many interacting services into several small optimization blocks. We use an XSLT converter that converts from one workflow language to another and enables the interoperation between different languages and workflow engines. We describe a Web Service proxy through which optimal services are invoked dynamically and may be manually updated.

Keywords: workflow, e-Science, Web Services, dynamic invocation.

1 Introduction

Web Service based workflow technologies are actively being used in industry, commerce, and research institutes. A variety of Web Service composition languages such as PDL, XPDL, BPSS, EDOC, BPMI, WSCI, ebXML, and BPEL4WS are available today [1]. There are also a variety of workflow engines and Grid Computing Environments [2,3,4,5]. These generally differ in their configuration, specific deployment mechanism, and performance. It is also often the case that workflow undertaken to support scientific computing has many differences from workflow approaches used for business applications. Scientific workflow often requires support for large data volumes, and is used to support parameterized execution of a large number of jobs, to monitor and control workflow execution including dynamic configuration of services, to execute in a changing environment where resources are not known in advance, and in which workflows may be composed hierarchically.

There are therefore three issues of concern: (1) mechanisms to discover and invoke Web Services dynamically, (2) mechanisms to optimize workflow performance by choosing Web Services from those available, and (3) mechanisms to interoperate between workflow languages and engines. All of these are essential to realize the Grid vision, which necessitates interoperation between services across multiple administrative sites. A key challenge, therefore, is to enable such interoperability to take place seamlessly.

2 The WOSE Framework

The WOSE framework attempts to solve the three issues in the above section (see Fig. 1). The XSLT converter transforms workflow scripts in different languages, enabling interoperation of workflow languages. The Optimizer makes a dynamic selection from the multiple available services, and workflow engines based on a pre-defined set of criteria specified by a user. The Service proxy enables dynamic selection of optimal services (see Fig. 2).

Figure 1: WOSE framework

2.1 Workflow language converter

As Web Services share the same core model based on a directed graph, we can find methods to transform between them. The transformation between languages has several advantages. Scientists will not need to learn multiple workflow description languages, and the generated workflow scripts can be re-used. An XSLT converter is used to transform workflow languages because they are
based on XML. XSLT is particularly useful as it provides a high-level declarative programming language that can allow frequent changes to be made in the XML document describing the service. The XSLT converter is also beneficial in dealing with aspects beyond the current workflow languages such as the overall performance of all services that constitute the workflow. It is used to translate between the intermediate representation used in WOSE into BPEL4WS, Scufl\(^1\), or SWFL (we target these languages because of their widespread use, and the relevance of their schema for Grid applications), and execute it using publicly available workflow engines. By choosing different WOSE services and XSLT rules, we can produce a transformation of scripts, which may give the same results but differ in performance.

![WOSE sequence diagram](image)

2.2 **Service Discovery, Optimizer, Dispatcher and The WOSE engine**

A variety of Web Services with similar functionality may co-exist. Moreover, these Web Services may be updated over time. Therefore, it is necessary to provide some mechanism for optimizing the workflow by selecting optimal services (based on some user criteria). Service discovery within the WOSE framework is used to discover the Web Services based on their definitions in the intermediate representation used. The service discovery module may access a number of pre-defined UDDI registries using the description of the requirements (WOSE Service Requirement Description Document SRDD, which includes items such as the service name, its interface, its execution performance, etc.). The output from the service discovery module is a Web Service description list giving the available matching Web Services and their properties.

Using the workflow script and semantic definitions, the Web Services description list from the service discovery process, and the transformations of scripts by the XSLT converter, the optimization module divides the whole workflow into several workflow blocks. In addition, workflow may be optimized through task re-design [6]. The output of the Optimizer is a task description which describes properties of workflow blocks and the workflow engines to which they are dispatched. After division into blocks, the Dispatcher distributes these blocks to the selected workflow engines, and the WOSE-shell to the WOSE engine. The WOSE engine will invoke these third party engines just as standard Web Services. The WOSE engine is a very simple workflow engine, which sequentially invokes workflow blocks in remote workflow engines just like invoking normal Web Services (it may therefore be seen as a “meta-engine”). The WOSE-shell workflow describes the sequence of invoking other blocks, the data input/output types of these blocks, and the dataflow between them. In this instance, WOSE is therefore not attempting to find some least common denominator solution, but allow aggregation of different solutions.

2.3 **WOSE services**

WOSE services are Web Services specific to the WOSE Framework. For the purpose of extending existing workflow languages and engines, WOSE services include several standard Web Services. These can be classified into man-machine interface service catalogue, logging service catalogue, proxy services catalogue, and workflow engine wrapping service catalogue. The WOSE man-machine interface services catalogue includes “head” services and “tail” services. These Web Services get external data and add information to allow different modes of display. The head and tail services are automatically invoked by the WOSE engine. The head service imports data from an external XML-based parameter file using the Getparameters operation. This approach has the advantage of allowing the external data to be changed at run-time. Similarly the tail service adds a display format XML tag which may be used to display results in customized formats, such as HTML or Scalable Vector Graphics (SVG)\(^2\). Logging services support the recording of status and error information for the execution of Web Services and workflow engines. Proxy services are used to invoke services dynamically and to make it easier to determine where in the workflow an error has occurred in the event of failure. For example, the Web Service proxy allows dynamic discovery of a Service (see Fig.2). Other proxy services are used to wrap existing scientific tools or clients implemented in Perl or Python, for instance. Alternatively, a wrapped workflow engine is seen as a single service, which is invoked with a workflow script relevant to that engine. This type of approach allows a complete deployment environment to be wrapped as a single service.

---

1. http://taverna.sourceforge.net/ -- Scufl is part of the Taverna project
2. http://www.w3.org/TR/SVG/
2.4 WOSE Prototype

We have developed the WOSE prototype that initially demonstrates the WOSE framework. The WOSE prototype at this stage takes a Scufl script as input, and enacts it on a BPEL4WS workflow engine (ActiveBPEL).

XSLT converter: The XSLT converter transforms Scufl scripts into BPELScript using an XSLT transformation script we developed using the Xalan XSLT product.

WOSE shell service: This is a Web Service that listens to, and responds to, the client. The WOSE shell service uses accepts the parameter file name from the client.

Getparameters service: This service gets parameter data from an external XML parameter file after the workflow scripts are deployed. Thereby avoiding the need to modify, deploy and re-run the workflow script when the source data changes. This is extremely useful for a group of users who want to share the same workflow, but use different parameters.

Web Service proxy: With Dynamic Discovery and Invocation (DDI), a service proxy discovers and invokes a Web Service at run-time. This requires three steps:

1. Discovery of the Web Service’s details from a UDDI or other registry, and finding the location of suitable services based on the Service Requirement Description Document (which may be a URL of the WSDL document describing the service).
2. Reading the WSDL document to find information about the Web Service, such as its namespace, operations, ports, and parameters.
3. Invocation of the service using Dynamic Invocation Interface (DII) technology.

Soaplab proxy (part of the Scufl/Taverna project): Soaplab is a set of Web Services providing programmatic access to applications on remote computers. We use one Soaplab proxy to invoke a set of Soaplab Web Services.

Sink: The results of a workflow application may be of many types and may be stored in many different formats. The WOSE framework uses an XML file to store results. From the XML format and semantic description, we can use XSLT to transform the results to different display formats such as HTML.

3 Discussion and Conclusion

We have presented the WOSE framework for dynamic invocation, optimization, and interoperation in a Web Service-based workflow framework. The goal of the WOSE framework is to optimize workflow, to interoperate among workflow engines and scripts, and to dynamically invoke on-line Web Services. Semantic definitions of Web Services are provided using the OWL language 3 (in the most complex instance), and simple categories and service identities in the simplest instance (which makes use of the UDDI registry).

Dividing large workflow applications into small blocks brings about many benefits. Small blocks fit the principle of modern software design methods. If a small block fails on one engine it can be re-executed at low cost on another engine. Small blocks can be executed in parallel. Moreover, a small block script in one language can be executed by the optimal workflow engine by transforming the script into the engine’s language.

Workflows engines differ in many respects, such as the type of enactment approach they provide, whether they use a centralised co-ordinator (or a distributed one), and their overall performance. Our aim in the WOSE project is to attempt to unify them in some way.

Acknowledgement: The WOSE project is funded by the EPSRC, and involves Cardiff University, Imperial College and Daresbury Laboratory.

References


3 http://www.w3.org/TR/owl-features/