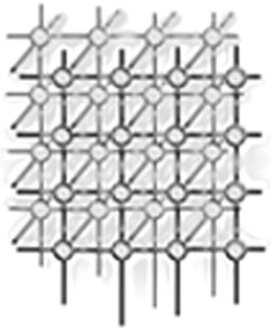
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2000; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

A Workflow Portal Supporting Multi-Language Interoperation and Optimisation



L. Huang[1], A. Akram[2], D.W. Walker[1,*,†], R.J. Allan[2], O.F. Rana[1] and Y. Huang[1]

[1] *Computer Science, Cardiff University Queen's Buildings, 5 The Parade, Roath, Cardiff CF24 3AA, UK*
[2] *e-Science Centre, CCLRC Daresbury Laboratory, Daresbury, Warrington, WA4 4AD, UK*

SUMMARY

This paper presents a workflow portal for Grid applications; which supports different workflow languages and workflow optimisation. We present an XSLT converter that converts from one workflow language to another and enables the interoperation between different workflow languages. We discuss strategies for choosing the optimal service from several semantically-equivalent Web services in a Grid application. The dynamic selection of Web services involves discovering this set by filtering available services based on metadata, and selecting an optimal service based on real-time data and/ or historical data recorded during prior executions. Finally we describe the framework and implementation of the workflow portal which aggregates different components of the project using Java portlets.

KEY WORDS: workflow, portal, portlets, Web services, optimization, Grid

1. INTRODUCTION

Web service technologies are actively being used in industry, commerce, and research institutes. The Web service standards, SOAP, WSDL and UDDI, simplify the integration and interaction between heterogeneous service providers and consumers. A variety of Web service composition languages such as PDL, XPDL, BPSS, EDOC, BPML, WSCI, ebXML, and BPEL4WS are available today [1] with corresponding workflow engines, which however all differ in their

*Correspondence to: Computer Science, Cardiff University Queen's Buildings, 5 The Parade, Roath, Cardiff CF24 3AA, UK
†E-mail: d.w.walker@cs.cf.ac.uk

Received 31 January 2000
Revised 19 September 2002

configuration, deployment specific description, 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 [9]. Scientific workflow often requires support for large data volumes, parameterized execution of large numbers of jobs, dynamic configuration of services at runtime, adapting to a changing environment, and hierarchical and complex workflows.

Three issues of concern are therefore mechanisms to: (1) discover and invoke Web services dynamically; (2) optimize workflow performance by choosing suitable Web services from set of available services; and (3) interoperate between different workflow languages and corresponding engines. A key requirement is to allow interoperability between different workflow systems, without violating security or performance. Dynamic Web service selection for workflow optimization is beneficial especially to scientific workflows mainly because applications are computationally intensive, and may be long running – some scientific procedures last weeks or even months. Selection of optimal Web Services among the available ones can shorten the overall computation time.

This paper presents results of the EPSRC-funded Workflow Optimisation Services for e-Science (WOSE) in developing a WOSE portal.

2. WORKFLOW PORTAL

The WOSE project consists of different components, e.g. a client to upload workflow, language converter, proxy service and result displayer, all of which are tightly coupled together and need user interaction. The Workflow portal integrates all these high level services providing component views on a single portal page. For instance a portlet imports source data from an external XML-based parameter file uploaded by the user with the Getparameters Web service. This approach has the advantage of allowing the sources data to be changed at runtime and makes a uniform interface possible. At the sink of the workflow another service adds the results to a file with a display format XML tag. The aim of the Workflow portal is to extend the functions of the existing workflow languages and enactment engines by co-coordinating additional services at runtime, to minimize the changes in workflow and its deployment descriptor. The only changeable sections are the data and control flows. The WOSE portal encapsulate the following services:

WOSE shell service: The WOSE shell service is a Web service that listens to, and response to, the client. It uses the WoseWebService to accept parameter file name from the client used as data source.

Getparameters service: This service retrieves parameter data from an external XML file specified by the client. The source data can be written into a workflow script with an assign statement in BPEL4WS or a stringconstant processor in Scufl. Normally changes in the source data require modification in workflow and re-deployment, but putting the data in the XML parameter avoids this.

Sink: The workflow results may be of many types and can be stored in many different formats. The WOSE framework uses XML to store results. From the format and semantic

description, we can use XSLT to transform the results to different display formats such as HTML or Scalable Vector Graphics (SVG) [†].

The WOSE portal includes a source input portlet to upload workflow scripts to be transformed and deployed, and a client portlet to upload the parameter file. The latter is a unified interface for inputting source data and getting the result back. The WOSE client portlet will invoke a workflow engine (such as an ActiveBPEL engine) and displays results in the client browser.

3. INTEROPERABLE WORKFLOW FOR PROBLEM SOLVING ENVIRONMENTS

A variety of workflow management systems for Grid computing have been reported in the literature. These range from portal-based interfaces to connect components together, to non-HTTP systems that allow composition and deployment of a set of services. Often these systems are categorized as being a Problem Solving Environment (PSE). In many ways, a PSE is seen as a mechanism to integrate software construction and management tools and application specific libraries, within a particular problem domain. One can therefore have a PSE for financial markets [2], for gas turbine engines [3], etc. Focus on implementing PSEs is based on the observation that in the past scientists using computational methods wrote and managed all of their own computer programs. However, now computational scientists use libraries and packages from a variety of sources, and those might be written in many different computer languages. Engineers and scientists now have a wide choice of computational modules and systems available, so that navigating this large design space has become a challenge in its own right. A survey of 28 different PSEs by Fox, Gannon and Thomas (as part of the Grid Computing Environments RG of the Global Grid Forum) can be found in [4], and practical considerations for implementing PSEs can be found in Li et al. [5]. Both of these papers indicate that such environments generally provide some backend computational resources, and convenient access to their capabilities. Furthermore, workflow features significantly in both of these descriptions. In many cases, access to data resources is also provided in a similar way to computational resources. Often PSE and Grid Computing Environment are interchangeable terms, as PSE research predates the existence of Grid infrastructure. We believe that a key element within a PSE is the provision of a workflow engine that enables a set of services (provided by one or more scientists) to be executed, in the context of a particular application. We therefore envisage the workflow to utilize a set of pre-defined components (services) discovered via a registry service such as UDDI.

The large number of PSE projects over the years has resulted in a range of language specifications for describing workflow, and an even greater number of execution environments to host the services (based on different implementation technologies) that are part of the workflow description. This has led to a problem of interoperability between different systems, making

[†]http://www.w3.org/TR/SVG/

it difficult for a scientist to effectively construct an application by combining services from different sources, where services may themselves be composed hierarchically from other services and the composition expressed in terms of a particular workflow language. Interoperability issues also arise due to differences in data types used to describe service interfaces, and in configuration parameters needed to initiate execution within a particular environment. To overcome these issues we use Web services as the key implementation technology for invoking and describing services. To achieve this, it is necessary for a translation mechanism to be provided that takes existing service descriptions and converts these into a format that can be used. The workflow descriptions are therefore specified in XML. This also permits translation between different workflow languages to be achieved using an XSLT converter [‡] to achieve interoperation of different workflow engines.

3.1. Architecture of WOSE framework

Figure 1 shows the overall architecture of the WOSE framework including the portal. The workflow deployment portlet imports the workflow scripts and semantic descriptions of Web services and workflows. An XSLT converter transforms one workflow language to another based on pre-defined rules. The resulting workflow script is then deployed in the workflow engine compliant with the chosen language. Once execution finishes, a portlet encapsulating the sink service displays output according to formatting metatag and associated rules. The current focus of the project has been to evaluate such a translation for the Taverna/ Scufl [§] workflow system, and integrate this with a publicly available BPEL4WS engine, such as ActiveBPEL [¶].

In general, all languages based on Web services share the same core model based on a directed (usually acyclic) graph, making transformation between them quite easy. XSLT is particularly useful for translation 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 performance, QoS, etc. and transforming them into the implemental scripts of publicly available workflow engines.

Consider the transformation from Scufl to BPEL4WS as an example. The Simple Conceptual Unified Flow language (Scufl) is a high-level conceptual workflow language. The Business Process Execution Language for Web service (BPEL4WS) resulted from a merger of Microsoft XLANG [‖] and IBM's WSFL [**], and is a block-structured programming language. Table 1 shows the correspondence between Scufl and BPEL4WS elements. The XSLT-rules column shows the XSLT rules applied to transform from Scufl to BPEL4WS. The Assign rule takes the value from a string constant processor in the Scufl script and assigns it to a variable in the BPEL4WS script. The arbitraryInvoke rule takes the endpoint and operation from the Scufl

[‡] http://www.w3.org/TR/xslt
[§] http://taverna.sourceforge.net/
[¶] http://www.activeBPEL.org/
[‖] http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm
[**] http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

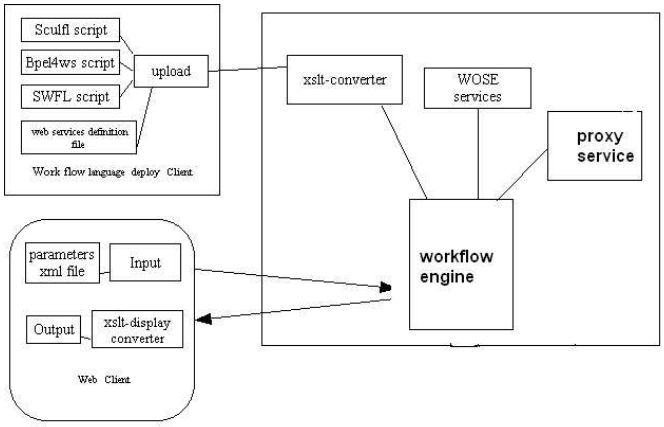


Figure 1. Workflow Portal Framework

Table I. Correspondence between Scufl and BPEL4WS

Scufl	BPEL4WS	Xslt-rule	WOSE Service
stringconstant	Assign	Assign	
Arbitrarywsdl	Invoke	arbitraryInvoke	
Source	Receive	DataInput	Getparameter
Sink	Reply	OutputResult	Tail service
Data link	Sequence activity	DatalinkAnalysis	
Concurrency constraint	Control activity	ControllinkAnalysis	

script and uses these as parameters for invoking Arbitrarywsdl Web service in the BPEL4WS script. The DataInput rule gets the parameters from an external parameter file by invoking the getparameter service. The OutputResult rule adds an XML display format tag into the results file. The DatalinkAnalysis rule analyses the sequence of activities and dataflow in Scufl and generates the corresponding sequence of activities in BPEL4WS. The ControllinkAnalysis rule analyses the control structure of activities in Scufl and generates the control structure of the corresponding activities in BPEL4WS.

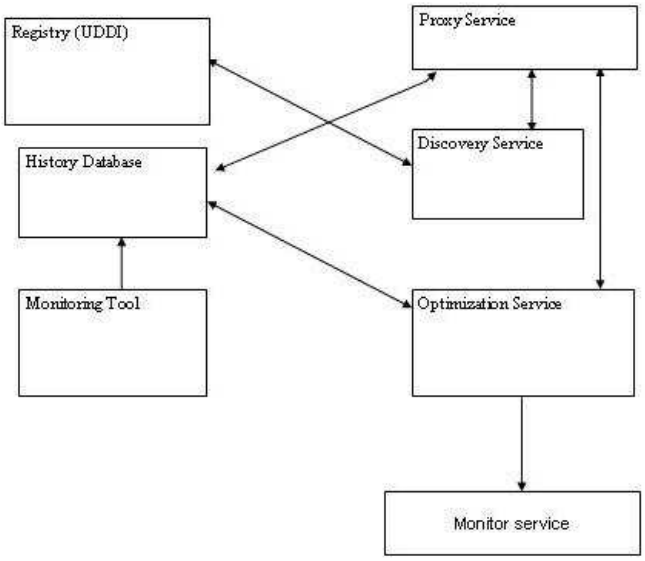


Figure 2. Dynamic Web Service Selection for Workflow Optimisation

4. WORKFLOW OPTIMIZATION STRATEGY

In a rich service environment, "similar" copies of the same service can exist with different performance or other user defined set of criteria; and best suitable service from these multiple instances cannot be selected at design time due to unpredictable performance parameters at the actual time of execution. Workflow optimisation is achieved by selecting optimal web services at run-time and integrating dynamic selection of web service into workflow.

Figure 2 shows the architecture of dynamic Web service selection for workflow optimisation [7, 8]. 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. 2 contains the history data on previous service invocations, such as the response time of services. Proxy service is an adaptor for dynamic selected service in workflow. When many semantically equivalent copies of a service are found by Discovery service, the optimization service selects an optimal service based on the history database, and real-time data such as the peak speed, current load, and current available memory of the machine hosting the service by invoking monitor service.

Proxy service: The Proxy Service is an adaptor for dynamically selecting and binding 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, an optimal service is selected and invoked and its output is passed to the next activity in the workflow. The Proxy Service is also a Web service. Its parameters include servicemeta, which gives a semantic definition to the abstract service and is used by the Discovery Service to find semantically equivalent services, querymethod, which specifies the query mechanism used to search for services, optimizationMETA, which gives a description of the problem-specific optimization model, optimizationMode, which is the mode for selecting the optimal Web Service, such as execution time, degree of trust in results, etc. operation, which corresponds to the actual function performed by the late binding service, and parameters, which are the parameters to be passed to the late binding services. ServiceProxyReturn is the result returned by the late-binding service.

Discovery service: A discovery service discovers a list of services from registries. 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 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. 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 by similar method to the work of Miles et al. [4]. 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 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. 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. Matchmaking is based on the OWL-S ontology [7].

Service selection: The Optimization Service selects the optimal service from the list of semantically equivalent services based on the criteria for optimization set by user. The criteria may cover many different 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. Optimisation service uses real-time data (by invoking monitor service) and history data to evaluate the scale of optimisation (such as performance) to select the optimal service, and return the optimal service location to the Proxy service.

Service invocation: When Proxy service gets optimal service location, it invokes the selected service dynamically. The WSDL file is downloaded from this URL and parsed. And

the service is invoked using Dynamic Invocation Interface (DII) to achieve late binding. The input parameters are passed to optimized service from Proxy service; and optimized service sends the output result to Proxy service. Proxy service will output the result to the Web service in the next step. 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.

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

5. IMPLEMENTATION

In this section we present the implementation of different components of WOSE Framework and Portal. The XLST converter we implemented is transformation form Scufl script to BPEL4WS and enacts it in a BPEL4WS workflow engine (ActiveBPEL), which is installed in Tomcat.

5.1. Scufl to BPEL4WS converter

Scufl is different from BPEL4WS in vocabulary and structure. Moreover, BEPL4WS has partnerlink types, variables and different namespaces. We retrieve the port types and namespaces of Web services from WSDL files, which are downloaded in the transformation process. For every operation of a standard Web service, we use a pair of variables to hold the requested input parameters and returned output data. The flow chart of the XSLT converter, which transforms from a Scufl file into BPEL4WS, begins with generating namespaces and ends with generating activities (see Figure 3).

After importing the Scufl script using the WOSE source input portlet, the XSLT converter transforms it into the BPEL4WS definition and process files, and other two files specific to ActiveBPEL.

5.2. Workflow input portlet and WOSE portal

The WOSE source input portlet is for inputting Scufl scripts via a Web browser. Once all files are transformed, into BPEL4WS, the workflow script is deployed in the ActiveBPEL engine. Before executing the workflow, the user uploads the parameter file via another portlet, and then the portal invokes the ActiveBPEL engine by inputting the parameter file name and workflow service endpoint of this workflow application into the ActiveBPEL engine. This will start the execution according to the BPEL4WS script in the ActiveBPEL engine. Finally, the ActiveBPEL engine will generate results and send these back to the WOSE portal. The example

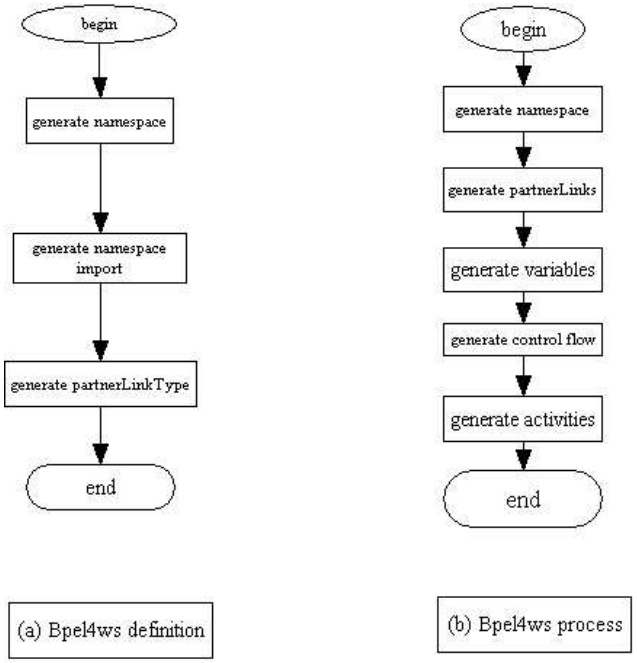


Figure 3. XSLT flow chart for transformation of SCUFL to BPEL4WS

used here involves three Web services. InvokeBrowser gets raw protein data, getproteinseq gets pure protein sequence, and blastall retrieves protein information from protein database. In our example the Blastall service is dynamically selected. In the workflow script, the Proxy service is then replaced by the blastall service.

We have used Tomcat and Axis as our Web service container. The historical database of different services is stored in mySQL database. The JUDDI registry maintained by the Welsh e-Science Centre is used as the UDDI registry. We have installed a blastall service in 7 different machines–ygrid01, ygird01, ygrid03, ygrid04, ygrid05, ygrid06. Desktops running Linux and laptops using windows are used as clients for testing. Different instances of the blastall service are deployed on different machines and registered with jUDDI with their unique access points.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented the design and first trial of a portal supporting interoperation and optimization of workflows. We used an XSLT converter to translate workflow languages

to achieve workflow interoperation. We used a Proxy service as adaptor of dynamic selected service to optimise the workflow. We discovered semantically equivalent services by a strategy of UDDI extensions. We used a Performance service to select optimal services by getting real-time data with a monitoring service. We used the portal to achieve multi-language support (workflow source input) and make it parameter-independent and display-independent (WOSE client).

In future work we plan to extend BPEL4WS to provide more steering capabilities at runtime to enable graceful recovery from unpredictable situations such as connection failure or service un-availability. We will also develop a a workflow monitoring service. Scientists will then be able to stop/ pause/ resume workflows or parts of a workflow, add breakpoints, monitor status at breakpoints, and log input/ output parameters. For a portal based monitoring service there are limitations in Web Clients [11] which may mean not all user requirements can be fulfilled and an interactive desktop may be preferable. The Web Service Invocation Framework (WSIF) [12] will be investigated for this purpose.

REFERENCES

1. van der Aalst W.M.P., *Don't go with the flow: Web services composition standard exposed*, IEEE Intelligent Systems, Jan/Feb 2003.
2. Bunin O., Guo Y. and Darlington J., *Design of Problem-Solving Environment for Contingent Claim Valuation*, Proceedings of EuroPar, LNCS 2150, Springer Verlag, 2001.
3. Fleeter S., Houstis E., Rice J., Zhou C. and Catlin A., *GasTurbnLab: A Problem Solving Environment for Simulating Gas Turbines*, Proceedings of 16th IMACS World Congress, 104-5, 2000.
4. Fox G., Gannon D. and Thomas M., *A Summary of Grid Computing Environments*, Concurrency and Computation: Practice and Experience (Special Issue), 2003. Available at: http://communitygrids.iu.edu/cglpubs.html.
5. Li M., Rana O.F., Walker D.W., Shields M. and Huang Y., *Component-based Problem Solving Environments for Computational Science*, Book chapter in Component-based Software Development (Ed: Kung-Kiu Lau), World Scientific Publishing, 2003.
6. Dewan R., Seidmann A. and Walter Z., *Workflow Optimization through Task Redesign in Business Information Processes*, Proceedings Of The Thirty-First Hawaii International Conference On Systems Sciences (HICSS'98), 1998.
7. Huang L., Walker D.W., Rana O.F. and Huang Y., CCGrid 2005, http://dsg.port.ac.uk/events/conferences/ccgrid05/wip /schedule/Paper13.pdf http://dsg.port.ac.uk/events/conferences/ccgrid05/wip/talks/talk13.ppt
8. Huang L., Walker D.W., Huang Y. and Rana O.F., *Dynamic Web Service Selection for Workflow Optimization*, in the proceedings of AHM2005, http://www.allhands.org.uk/2005/proceedings/papers/423.pdf, http://www.nesc.ac.uk/talks/ahm2005/423.ppt
9. Singh P.P. and Vouk M.A. *Scientific workflows: Scientific computing meets transactional workflows*, http://www.csc.ncsu.edu/faculty/mpsingh/papers/databases/workflows/sciworkflows.html
10. Akram A., Chohan D., Wang X.D., Yang X. and Allan R.J. *A service oriented architecture for portals using portlets*, UK e-Science AHM2005, September 2005.
11. Akram A., Chohan D., Wang X.D., Meredith D., and Allan R.J. *CCLRC portal infrastructure to support research facilities*, GGF Workshop on Science Gateways, GGF14, Chicago, IL, USA, June 2005.
12. *Web Service Invocation Framework* http://ws.apache.org/wsif/ [1st December 2005]