Automatic Portal Generation Based On XML Workflow Description

Dashan Lu

15 January 2010

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Contents

DECLARATION

This work has not previously been accepted in substance for any degree and is not concurrently submitted in candidature for any degree.

Signed .... (candidate)
Date ....28/01/2010................

STATEMENT 1
This thesis is being submitted in partial fulfillment of the requirements for the degree of PhD.

Signed .... (candidate)
Date ....28/01/2010....................

STATEMENT 2
This thesis is the result of my own work/investigations, except where otherwise stated. Other sources are acknowledged by explicit references.

Signed .... (candidate)
Date ....28/01/2010................

STATEMENT 3
I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organizations.

Signed .... (candidate)
Date ......28/01/2010.............

ACKNOWLEDGEMENT

I would like to thank Prof David Walker and Dr Coral Walker, my supervisors, for giving me the opportunity to do PhD research under their supervision, and for their support and encouragement, and above all for their friendship.

I also would like to thank my parents, Sheng Lu and Huiyun Fang, and my fiancée, Meng Wang.

Summary

This dissertation investigates the automatic generation of computing portals based on XML workflow descriptions. To this end, a software system is designed, implemented and evaluated that allows end-users to build their own customized portal for managing and executing distributed scientific and engineering computations in a service-oriented environment. The whole process of the computation is represented as a data-driven workflow. The portal technique provides a user-friendly problem-solving environment that addresses job assignment, job submission and job feedback. An advantage of this approach is that the complexity of the workflow execution in the distributed environment is hidden from the user. However, the manual development and configuration of the application portal requires considerable expertise in web portal techniques, which most scientific end-users do not have. This dissertation address this problem by describing a tool chain consisting of three tools to achieve automatic portal generation and configuration.

In addition, this dissertation presents a mapping of each element of WSDL to the UDDI data model, the conversion from the data-flow workflow to control-flow workflow by using XSLT, an implementation of a drag-and-drop visual programming environment for the generation of a workflow skeleton, and a methodology for the automatic layout of portlets in a portal framework.

Chapter 1

Introduction

1.1 Background

1.1.1 Grid Computing

Grid computing [1] originated in the early 1990s with the aim of making computer power as easy to access as electricity from a power grid. In contrast to traditional supercomputing, in which many processors are physically connected by a high-speed computer bus [2], grid or distributed computing relies on stand-alone computers, often in different administrative domains, connected by a conventional network interface such as Ethernet. Loosely-coupled computers linked by a network can perform a large number of tasks that may not be possible with limited local resources. Grid computing has been applied in academia to computationally-intensive scientific and mathematical problems, as well as being used in finance to analyze complicated economical models, in pharmacy to discover new drugs, and in engineering to perform multidisciplinary simulations.

1.1.2 Web Portals

Scientific and engineering computer applications on a service-oriented architecture are often represented as a data-driven workflow. A user can conveniently compose applications with workflow abstractions in a visual programming environment, and execute them by passing them to a workflow execution engine. Web portals made up of JSR168-compliant portlets have become popular as a means of providing an interface to allow users to access different resources, including data and services in a distributed environment. JSR-168[1] refers to the Java Portlet Specification V1.0, which provides a basic portlet programming model, and an Application Programming Interface (API) based on this model. Portlets that are built using this API are compatible with any portal framework that complies with JSR-168. A web portal provides a user-friendly interface for distributed systems, and is well-suited as a visual environment for users to compose computational applications represented as data-driven workflows. It is a component in the distributed environment that can enable users to access the diverse resources which are located different geographic sites.

1.1.3 Web Services

In the emerging distributed computing paradigm, a standard technique is needed for defining interfaces to software components, methods to access these components via interoperable networks, and discovering methods for identification, authorization and authentication of these components' providers and consumers. A Web service has two of the characteristics above. Firstly, a Web service enables machine-to-machine interoperable interaction over the network. It supports dynamic discovery and composition in het-

[1] http://www.jcp.org/en/jsr/detail?id=168, accessed 12 January 2010.

erogeneous environments by the registration and discovery of its interface definition. The details on how to implement registration and discovery will be introduced in Section 2.2.1. Secondly, their platform independence has led to the web services framework being widely adopted to exploit local services as remotely accessible services across different platforms (e.g., Microsoft .NET, IBM WebSphere, BEA WebLogic and the open source Apache AXIS). Web Service Description Language (WSDL) plays an important role in web service description, development, discovery and registration. WSDL uses an XML file to specify a hierarchical model of a Web service. This is done because the design of WSDL complies with an object-oriented object model. WSDL separates the interface from the implementation of functions. It can make it easy to pose more advanced queries to a service repository. The details of these queries are introduced in Chapter 2.

1.1.4 Distributed Workflows

In general a service-oriented scientific application in a distributed environment can be viewed as being composed of several web services physically located on different servers. WSDL is only responsible for describing a web service's interface, including what the web service does, what the parameters of the web service are, which protocol the web service can be accessed through, and where the web service can be accessed. It cannot support the description of co-operation between web services and the transfer of data between web services sequentially or in parallel. Instead a workflow description language is used to describe the coordination and composition of the services, which may be categorized as either local or remote services. In general, a large-scale scientific computing application can be considered as being composed of a set of specific services, and so is expressed as a form of service integration. The ability to link and compose diverse distributed software components and data to specify a workflow is critical in many application domains. Thus, several problem-solving environments (PSEs) for enabling users to compose workflows have been developed. These support large-scale workflows of distributed data and software components, which cannot be merged into a single system for maintenance and execution. SWFL[3], an extension of WSFL[1] from IBM, is designed to support web service cooperation, and data flow and control flow, in service-oriented applications. However, a workflow composed of a set of web services and described by SWFL, contains detailed information about the structure of the workflow, namely all services involved in the workflow and the data flow between pairs of web services. It does not distinguish between abstract and concrete workflows, which are usually defined as follows:

- An *abstract workflow* specifies what the services that make up the workflow do, but does not specify particular implementations for these services.

- A *concrete workflow* specifies not only what the services in the workflow do, but also specifies their service implementations.

The separation of these two types of workflow can bring more flexibility to service-oriented applications generated by a portal generator. For this reason, a new lightweight abstract workflow description language based on XML is introduced in this dissertation. This language inherits some features from SWFL and focuses on the description of the structure of the workflow. The other important feature of this abstract workflow description language is the capability of extending the workflow description by adding information about the file or service corresponding to each node of the workflow. The extended workflow with the file and service instances specified is called a concrete workflow, and it is executable on a compatible workflow execution engine. In the implementation discussed is this dissertation, the ActiveBPEL[2] workflow engine is responsible for workflow execution. In order to create a concrete workflow executable in this specific workflow execution environment, a convertor is needed to transform the workflow description to BPEL. This convertor is inserted into the portal generator, and is easy to extend to support multiple transformations for different workflow execution engines. The multiple workflow description scripts supporting the convertor enable the portal generator to be flexible to support workflow execution in multiple workflow execution engines.

1.1.5 Distributed Scientific Computing

Grid computing provides the underlying software infrastructure for carrying out secure distributed scientific applications. Such applications are becoming increasingly important as the components (such as data, sensors, and compute resources) that make up an application often cannot feasibly be placed at a single geographical location. This is particularly true in collaborations in which a number of different organizations are contributing resources to a distributed application, or where a multinational organization wishes to make coordinated use of resources that are distributed around the world. Workflows are a convenient abstraction to represent large-scale distributed applications in which the workflow nodes are tasks usually implemented by Web services. A workflow can be composed in a visual programming environment, which can then be executed in different workflow execution engines. Web-based portals can provide a user-friendly virtual environment that hides

[2] http://www.activebpel.org, accessed 12 January 2010.

the complexities of creating and executing a workflow application. Thus, the emerging field of distributed scientific computing builds on grid computing, in conjunction with workflow, portal, and service-oriented technologies [4].

1.2 Research Problem and Motivation

Although a portal provides a user-friendly environment for creating and executing a workflow application, a major problem is that the creation and configuration of an application portal requires considerable expertise in portal technologies, which scientific end-users generally do not have. Even for people with expertise in portal technologies, the manual creation and configuration of application portals for different workflows representing distributed scientific and engineering computations is still often tedious and error-prone work. Thus, the major motivation for the research presented in this dissertation is to provide a means whereby scientific end-users, with no expertise in portal or Web service technologies, can create portals automatically which they can then use to specify and execute distributed applications. This avoids the difficult, costly, time-consuming, and error-prone process of building a portal manually. For example, consider the case of the GECEM portal discussed in more detail in Section 2.4.1.The GECEM portal manages the execution of a workflow that is a pipeline of three tasks, and is static in the sense that any change in the structure of the workflow would require the portal code to be substantially rewritten. The GECEM project employed two postdoctoral research associates for two years each to implement the GECEM portal, and cost £367,250. With the approach proposed in this dissertation based on the automatic generation of portals from abstract workflows the effort and the cost to implement the GECEM portal would be much less. Indeed, once the

Web services for carrying out the tasks, and the file and service repositories (see next section) are in place, the GECEM portal can be generated in a few minutes. Furthermore, if the structure for an existing workflow changes, a portal for the new workflow can be quickly made by editing the original workflow (for example, by using the Design Tool discussed in Chapter 4), and then using the new workflow to automatically generate the corresponding portal.

1.3 Research Methodology

The main aim of this research is to investigate the technical feasibility of developing a software system which can automatically generate a portal for running a workflow based on an abstract description of the workflow. The generated portal should allow end-users to specify the files to be input to the workflow and the specific services to be invoked, thereby converting the abstract workflow into a concrete workflow. The portal should also support the execution of this concrete workflow. All the input and output files, including intermediate output, have to archived in a virtual data store (or *file repository*). In addition, the service implementations which are available for the user to browse and access are stored in a virtual service store (or *service repository*). This store must provide sufficient information to distinguish services of different types. The system can simplify composition of scientific workflows that can then be embedded into the portal framework. It can also let users specify workflows and generate a portal that supports the selection of specific files and services for the execution of one particular instance of this workflow. Additionally, two extra systems are developed to support the generation of the portal by the portal generator. One is a light-weight workflow structure generator, which is a visual graphical workflow structure editor to enable users to make a workflow skeleton easily and efficiently by dragging and dropping components. The other is an interface to allow users to query information about files and services that are registered in the file and service repositories.

1.4 Research Contributions

The research described in this dissertation contributes to looking for an efficient way of integrating portal technology into the construction and execution of large loosely-coupled distributed applications. The automatically generated portal acts as a graphical user interface for users to build loosely-coupled distributed applications by orchestrating web services registered in the web service repository, and specifying the data inputs registered in the file repository. The procedure of construction and execution of each loosely-coupled application is completely different due to differences of the structure of the workflow representing the application and the web services composing the application. It is difficult and inefficient for users to build a portal interface for each different application by hand. An alternative, more efficient approach promulgated in this dissertation is to use an XML description of the abstract workflow to generate a portal automatically, rather than manually. The XML description conforms to a specific XML schema, which can be used to represent a large application with different workflow instances.

1.5 Research Challenges and Novelty

The main research challenge considered in the research presented in this dissertation is whether it is technically feasible to automatically generate, from an abstract workflow description, a portal that allows users to specify the file and service instances in a workflow, and to execute the resulting concrete workflow on a third-party execution engine. This challenge has been addressed by the design, implementation, and evaluation of a prototype software system. Specific challenges are:

1. To ensure that the generated portal is easy to use and provides the functionality required by users.
2. To design an XML schema to describe skeleton workflows that can then be extended to represent abstract and concrete workflows.
3. To ensure the design of the software architecture of the prototype system supports interoperation with other XML-based workflow descriptions, and with other portal frameworks and workflow execution engines.

Interoperation is supported mainly through the use of XSLT scripts. First consider how the prototype system, outlined in Chapter 3, would interoperate with third-party workflow description languages that express a workflow as a Directed Acyclic Graph (DAG) in XML. For such languages it is generally possible to use XSLT to convert a workflow description into a skeleton workflow description that conforms to the XML schema used by the Design Tool and Label Tool (see Section 4.1.4). This is a reasonable expectation since every workflow description must contain the basic structure of the DAG. In general, a different XSLT script is required to convert each XML-based workflow description language into a skeleton workflow that conforms to the XML schema in Section 4.1.4. The resulting workflow description can then be input to the Label Tool.

In the prototype system the automatically generated portal runs in the GridSphere portal framework. However, other portal frameworks that comply with the JSR-168 specification could also be used by generating the appropriate portal configuration files. In the prototype system XSLT scripts are used to generate the configuration files for GridSphere (these are the group.xml. layout.xml, portlet.xml, and web.xml files described in Sections 6.6.4 and 6.7.9). The use of a portal framework other than GridSphere would require XSLT scripts that produce the configuration files for that framework - one set of XSLT files would be required for each portal framework. For example, for JetSpeed-2[3] it would be necessary to generate a configuration file with extension psml[4], which defines how portlets are aggregated and laid out on a portal page. In addition, the ANT script that controls the generation of the configuration files would also need to be modified to create a different directory structure for each different portal framework.

The prototype system uses the ActiveBPEL workflow engine for the execution of concrete workflows. As described in Section 6.8, the files needed to execute a workflow on the ActiveBPEL workflow engine are created by a Web service referred to as the Conversion Web Service. This involves the application of a set of XSLT scripts to the concrete workflows specified by the user in the automatically generated portal. The resulting files are compressed into a file with suffix bpr which is deployed on the ActiveBPEL workflow engine - the bpr file is specific to ActiveBPEL. The use of a workflow execution environment other than ActiveBPEL would require a Conversion Web Service for that particular environment - for each workflow execution environment a different Conversion Web Service is needed.

[3] http://portals.apache.org/jetspeed-2/, accessed 12 January 2010.

[4] Portal Structure Markup Language.

The novelty of this research lies in the *automatic* generation of a portal for selecting file and service instances in a workflow. As noted in Section 2.4, previous research has not attempted to achieve this, but has instead tended to focus on standalone environments for workflow construction and execution. The approach adopted in this dissertation provides better support for sharing workflows within and between communities since users can exchange abstract workflows that can be converted to concrete workflows, using the automatically generated portal. These concrete workflows can then be executed using file instances and service implementations of which the original creator of the abstract workflow may be unaware.

1.6 Outline of Dissertation

The following chapters show the structure and implementation of the web portal generation system. The evaluation of the system and future work are also included. Chapter 2 introduces the techniques and software technologies which are relevant to this research, and some systems based on the portal technique which are similar to the portal generated by the portal builder described here. Chapter 3 describes the framework of the portal builder, and relevant technologies and tools used in the development of this tool. Chapters 4, 5, and 6 demonstrate the implementations of three components of this system. In Chapter 7 the evaluation of the entire system is presented. This evaluation tests the stability of the system and cooperation with third-party software for some real test cases. The last two chapters present some reflections and lessons learned from this research.

Chapter 2

Background Research

Chapter 1 has given a brief overview of the concepts and technologies that underlie the use of Web services to compose distribution applications in the form of data-driven workflows, and the use of portals to provide a user interface for the management and execution of such applications. This chapter gives a more detailed description of these technologies, which is necessary to fully understand the research discussed in later chapters. This chapter also describes related research that is relevant to this dissertation.

2.1 Service-Oriented Computing

Service-oriented computing (SOC) makes use of loosely-coupled software services to build higher-level distributed processes and applications [5]. An architecture for service-oriented computing has three main parts: a provider, a consumer, and a registry in which providers publish details of their services, and whereby consumers can discover the services they wish to use and how to invoke them. A number of service-oriented architectures conform to this abstraction, including Web services and the related WS-* software stack, CORBA[1], and Java-RMI[2].

The CORBA is a set of specifications for a distributed object-oriented architecture published by the Object Management Group[3], and is a widely-accepted standard for integrating legacy systems and applications, and for providing interoperability between them [6, 7]. CORBA is commonly used in distributed business environments, particularly within corporate intranets and for e-commerce applications. CORBA provides a standard mechanism for defining the interfaces between distributed components, as well as tools to facilitate their implementation using a variety of languages. CORBA also provides several standard services, such as directory services, event services, transaction services, and security services. In addition, CORBA provides a mechanism for transparently communicating between components.

Java-RMI [8] supports distributed computation by allowing methods of remote Java objects to be invoked on other Java virtual machines (JVM). Like CORBA and Web services, Java-RMI provides platform independence to the extent that methods can be invoked on any host computer that runs a JVM, however, it is not language-independent (although C and C++ code can be executed through JNI[4]). Both Java-RMI and CORBA are object-based, in contrast to Web services in which there is no concept of an object reference. Instead services are defined more simply through an end-point reference that supports a set of operations. A further distinction is that Web services are stateless.

[1] Common Object Request Broker Architecture.

[2] Java Remote Method Invocation.

[3] http://www.omg.org/, accessed 12 January 2010.

[4] Java Native Interface.

CORBA and Web services are broadly similar in their high-level, service-oriented architecture [9]. Both provide mechanisms for specifying service interfaces, communicating between services, and publishing and discovering services. Service interfaces are specified with IDL[5] in CORBA, and WSDL for Web services as discussed in Section 2.2.4. CORBA uses the IIOP[6] to communicate between services, whereas Web services generally use SOAP (see Section 2.2.2). Finally, CORBA Naming/Trading services correspond to repositories, such as UDDI (see Section 2.3), that are widely-used for Web services. Both CORBA and Web services can be viewed as providing a platform-independent and language-independent (or language-agnostic) approach to service-oriented computing, and on a purely technical level both could have been used to provide the service implementations discussed in this dissertation.

In spite of the similarities between CORBA and the Web services stack, there are important distinctions. CORBA has the advantage of being a more mature software platform, and it is also more efficient in its communications. On the other hand, Web services usually communicate through HTTP, and so messages can pass through firewalls. This is more difficult to achieve with CORBA's IIOP. Furthermore, CORBA technology has to be available at both ends of a communication, whereas Web services just assume the existence of a SOAP processor, which is much more lightweight and inexpensive. CORBA messages are in binary format, in contrast to the use of XML for messages between Web services, which allows the content of Web service messages to be processed by a number of widely-available tools.

[5] Interface Definition Language

[6] Internet Inter-ORB Protocol.

In the context of research presented in this dissertation, Web services technologies were adopted, rather than CORBA, because the latter are more open and lightweight. CORBA software is proprietary and expensive to license, in contrast to Web service software, which is usually free and open-source. Finally, Web services are more accepted by the distributed scientific computing and Web communities, and hence their use makes the research presented in this dissertation of more relevance to these communities.

2.2 Web Services

2.2.1 Web Service Overview

A Web service, as described by IBM [10], is an interface that describes a collection of operations that are network accessible through standardized XML messaging. A Web service fulfills a specific task or a set of tasks, and is described using a standard, formal XML notation, called its service description. This provides all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location.

The nature of the interface hides the implementation details of the service so that it can be used independently of the hardware or software platform on which it is implemented, and independently of the programming language in which it is written. This allows and encourages Web service based applications to have loosely coupled, component-oriented, cross-technology implementations. A Web service can be used alone or in conjunction with other web services to carry out a complex application or a business transaction [10].

The above definition of a "Web service" given by IBM is not universally agreed upon. Many other Web service infrastructure providers have published their own Web service definitions. For example, in Sun's definition, Web services are software components that can be spontaneously discovered, combined, and recombined to provide a solution to a user's problem/request. The Java language and XML are the prominent technologies for Web services [11].

Although, the definitions of a Web service given by different infrastructure providers are varied, these definitions identify some common features of a Web service. First of all, a Web service is a platform- and implementation-independent software component that is critical to the cooperation of different services across various platforms in a distributed environment. A Web service is described using a service description language, and also it can be published to a registry of services, and discovered from a registry of services, through a standard mechanism. Web services are invoked through a declared API, usually over a network, and can be composed with other services.

2.2.2 The SOAP Communication Protocol

From the definition of Web services given in Section 2.2.1, it is clear that Web service technology is built around XML. A Web service itself is described in XML by a Web Service Description Language (WSDL) document, as discussed in Section 2.2.4. Like other common technologies based on the Web, a Web service has a client-side and a server-side part. The client sends a request to the server. The server receives this request and takes some corresponding action, finally sending a response back to the client. How the communication between client and server happens, and how the data are represented in XML format, is governed by the XML-based SOAP protocol. The most important part of SOAP protocol is the specification of the SOAP envelope framework, which is the kernel of data transmission between the client side and server side. The structure and extensibility mechanisms provided by the SOAP envelope framework makes SOAP well-suited as a fundamental part of XML-based distributed computing. The SOAP envelope framework defines a set of mechanisms for identifying what information is in the message, what entity should deal with the information, and whether this is optional or mandatory. The SOAP message is completely composed of XML documents that define the unit of communication in a distributed environment.

In a SOAP message, the root of the document is the "Envelope" element. In this root element there are two child elements: "Header" and "Body". As the primary extensibility mechanism in SOAP, the "Header" provides the means by which additional facets can be added to the SOAP-based protocol. Typical areas that the "Header" affects are authentication and authorization, transaction management, tracing and so on. The "Body" element surrounds the information that is the core of the SOAP message. The "Body" element can contain arbitrary XML documents, which could be the parameters required by a service located on the server side, or the response sent back to the client side.

2.2.3 Web Service Description

Service description and discovery are very important in a Service-Oriented Architecture (SOA). The discovery of services is about how to both describe and identify services. In the SOA approach, illustrated in Figure 2.1, the service provider publishes a service description to one or more service registries. It should be noted that the item in the registry is not the code for the service itself, but is a service description that tells all service requestors the details they need to know in order to properly invoke the web service. In order to be invoked a web service has to be described in sufficient detail.

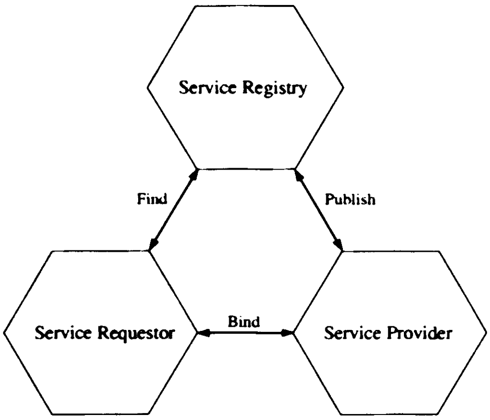


Figure 2.1: Service-oriented architecture.

The service description stack represented in Fig. 2.2 provides a way to describe various aspects of a Web service. The layers of Web service description stack are divided into functional and non-functional groups. The bottom three layers belong to the functional group. They describe what a web service does, how to invoke the web service, and where it is located. The upper two layers are in the non-functional group, as they do not directly provide information about invocation, but rather provide other information to influence whether a service requestor should choose to invoke a particular Web service. As shown in Figure 2.2, the service implementation definition and the service interface definition both use the WSDL standard, which is described in more detail in Section 2.2.4. The service implementation definition describes where the web service is located, in other words, to which network address the request should be sent to invoke the web service. The service interface definition exactly describes what information should be sent to the service, how to encode the message, and what Internet protocols to use. For the non-functional group, Web Services Endpoint Language (WSEL) is used to describe service properties such as cost and quality-of-service. Finally, a Web service description language, such as WS-BPEL[7] [12] or SWFL[8] [3], is used to describe a collection of Web services and how these orchestrated into a higher-level business or scientific computing process in a distributed environment.

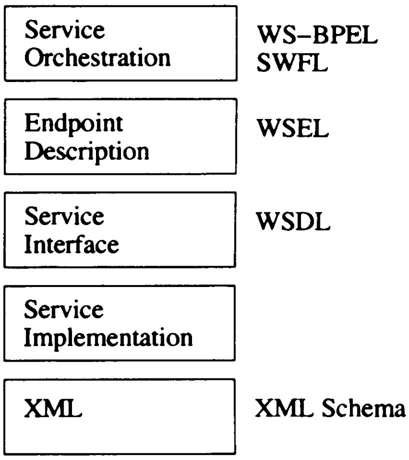


Figure 2.2: Web service description stack.

2.2.4 WSDL

Web Service Description Language (WSDL) is a proposed standard to describe the technical invocation syntax of a Web service. WSDL is an pure XML document conformant to the WSDL schema definition. As stated in Section 2.2.3, the WSDL description describes a Web service in term of three properties:

[7] Web Services Business Process Execution Language.

[8] Service Workflow Language.

- What a web service does, i.e., the operations (methods) the service provides.

- How a service is accessed, i.e., details of the data format and protocol necessary to access the service.

- Where a service is located, i.e., details of the protocol-specific network address of the service, such as a URL.

The WSDL schema defines the following set of high-level XML elements:

- portType: this is a Web service's abstract interface definition, in which each nested child operation element defines an abstract method.

- message: this defines a set of parameters referred to by the methods or operations. A message can be further decomposed into parts.

- types: this defines the collection of all data types used in the Web services that are referenced by various message part elements. The definition of the data types can be imported from other schema documents.

- binding: this contains details of how the elements in an abstract interface (portType) are converted into a concrete representation in a particular combination of data format and protocol.

- port: this expresses how a binding is deployed at a particular network endpoint.

- service: this is a collection of ports

Figure 2.3 shows the organization of the WSDL information model. The relationship between the abstract and concrete notions of message and operation, as contained in the portType and binding elements, are clearly shown in this figure.

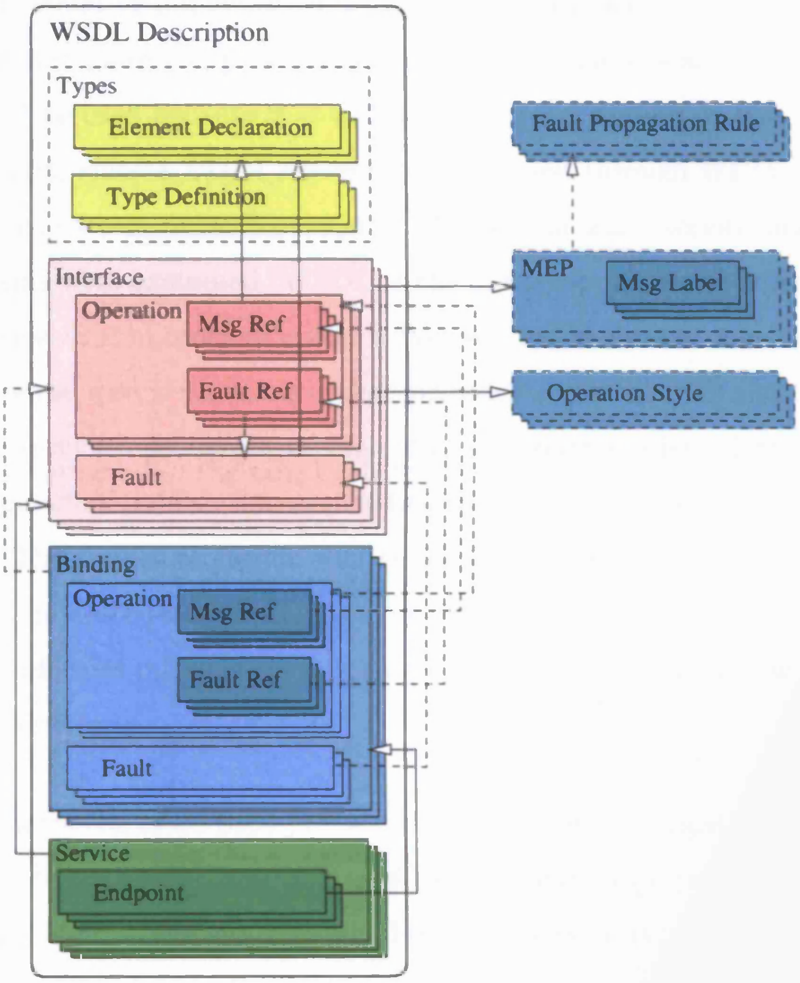


Figure 2.3: WSDL information model

2.3 Web Service Discovery

This section introduces the topic of Web service discovery and, in particular, the role of a service registry in a service-oriented architecture. There are several alternative service registries, but this section focuses on the Universal Description, Discovery and Integration (UDDI) standard. The service repository could be implemented using an SQL database, for example, which would permit more complex queries and may be more scalable. However, UDDI will be used because it is widely used in the context of Web services. Furthermore, since a UDDI repository is accessed through HTTP it is less likely to encounter firewall problems. In Section 2.2.4 service description mechanisms were examined. WSDL is the most popular way of describing a Web service. But in order to invoke a Web service, it must first be discovered through some query mechanism that provides access to information about the Web service. As shown in Fig. 2.1, in a service-oriented architecture service providers publish the service description of a Web service to a service registry. The service requestor queries the service registry to retrieve one or more service descriptions that meet certain criteria. The service description contains sufficient information to let a service requestor to bind to and invoke the Web service.

UDDI works as more than just a business and service registry. It also defines a set of data structures and an API specification for programmatically registering and finding services, bindings, and service types. UDDI in general, whether used for software components, services, or otherwise, has the following basic requirements:

- A set of data structure specifications for the metadata to be stored in the registry. The metadata gives information on data ownership and categorization, and provides a logical referencing mechanism.
- A set of create, read, update, delete operations specification for storing, deleting and querying the data in the registry. Authentication is re-

quired for operations that change information and for public registries, whilst open access is permitted for read and query operations.

2.4 Related Research

2.4.1 GECEM portal

The idea of creating an automatic portal generation system, which is the focus of this dissertation, arose from the earlier GECEM[9] project [13], which developed a portal for distributed computational electromagnetic (CEM) simulations. The GECEM portal supports scientists in accessing distributed resources for the solution of CEM problems [14] using Grid technologies. The distributed resources include input files specifying the model geometry, and proprietary software and hardware for mesh generation and CEM simulation. The web-based GECEM portal allows users to interact with these resources for submitting jobs, monitoring and executing distributed grid applications, and collaboratively viewing the result of a CEM simulation within a visualization environment. The workflow supported by the GECEM portal is a pipeline style, in which the nodes represent the surface mesh generation, volume mesh generation, and CEM migration and simulation services. The starting point for this workflow is the node representing the surface mesh generation, which takes as input a file specifying the geometry of the problem. This input is assumed to be generated outside the portal, and stored in a specified file repository that can be browsed and accessed from within the portal by authorized users. The output generated by each service is automatically stored into the file repository at a user-specified location. In the portal the service at each node is controlled by a portlet that the user

[9] Grid-Enabled Computational Electromagnetics

interacts with. In each portlet multiple semantically equivalent services are available for a particular task, and the user can select one of these available services. The GECEM workflow is divided into three steps. The output from the first service, the surface mesh generation service, is passed to the volume mesh generation service, and the output generated by this, in turn, is passed to the CEM migration and simulation service. The final output, and all intermediate files from the surface and volume mesh generation services, are archived in the specified file system and are automatically registered in the file repository.

In the GECEM application, the first three stages of the GECEM workflow are responsible for the performance of the main numerical tasks, and the last one is used to visually represent the final output. The workflow is a pipeline with a fixed number of nodes, where each node is associated with a service to perform a particular activity. The static structure of the GECEM workflow is restrictive if the user is interested in dynamically composing a service-based application by inserting, deleting or reordering nodes in the workflow. However, if this is done, the whole process of chaining together the services in the workflow can only be accomplished by hand-coding the new portlets and editing the portal configuration files. Thus, in effect, the portal has to be rebuilt, and this requires a high degree of expertise in using the portal framework and portlet programming skills, which most end-users do not have. Thus, the portal builder introduced in this dissertation aims to address the dynamic composition of service-based applications without the need for any expertise with portal frameworks.

The GECEM portal is an example of a *vertical portal* which provides a web-based entry point for a particular application domain, in this case CEM simulation. In general, the processing logic of a vertical portal includes composing, running and monitoring computational tasks by invoking Web services to perform actions on the back-end resources. The user needs to access data repositories, to configure the computational tasks and submit the tasks, and finally to monitor the progress of the tasks and access the results. Some data and computational tasks or independent of the application domain and can be reused between different vertical portals [15]. Examples of other successful vertical portals are GEON[10] and NEESgrid[11]. Other horizontal portal environments provide generic middleware execution management [16, 17] to reduce the complexity of implementation for application developers, and to hide the complexities of service use and management from the end-users.

2.4.2 P-GRADE Portal

The P-GRADE portal [18] is similar to the GECEM portal in some respects and is also developed with the GridSphere portal framework, with the use of nodes to represent the input and output data files as a workflow. A workflow application is developed in a workflow editor provided by the P-GRADE portal. This is based on Java "Web Start" technology, enabling it to independently run on the client-side and communicate with the portal server. The user has to associate the particular services and data files with each node in the workflow by using drop-down list boxes that are configured by the portal administrator. The transfer of files to execute workflows, and the submission of jobs to Grid resources, is handled by the DAGMan workflow scheduler[12], whereas the GECEM portal drives the workflow by itself.

The P-GRADE portal does not make use of the concept of an abstract

[10] http://portal.geongrid.org, accessed 12 January 2010.

[11] http://www.nees.org, accessed 12 January 2010.

[12] http://www.cs.wisc.edu/condor/dagman/, accessed 12 January 2010.

workflow during the workflow construction. Instead users directly specify the binary executable for each node, and also specify the data file locations by choosing them from dropdown lists. Furthermore, the user builds the complete workflow application within the P-GRADE portal. This is distinct from the approach adopted in this dissertation in which the development of an executable workflow is divided into three steps, producing in turn the skeleton workflow, abstract workflow, and concrete workflow, as outlined in Chapter 3. In this approach the structure of the workflow is determined once the skeleton workflow is constructed, and this structure is subsequently automatically embedded into the portal by the Build Tool. Then, within the portal, the user needs only to specify the data files and service instances. The building of the workflow structure is done outside the portal using the Design Tool, or other third-party software.

2.4.3 ASC Portal

The Astrophysics Simulation Collaboratory (ASC) Portal [19] is another collaborative environment in which distributed projects can perform research. The ASC is a web-based problem solving framework designed for the astrophysics community. The ASC portal supplies specific tools for the management of large-scale numerical simulations and the resources these run on.

2.4.4 Abstract Workflows

The concept of an abstract workflow was introduced in the GriPhyN Virtual Data System (VDS) and its interface portal, Chiron [20]. Virtual Data described by XML is stored in a Virtual Data Catalog (VDC). An XML style language called the Virtual Data Language (VDL) can query the VDC. Chiron allows users to query the VDC by namespace and/or attributes, metadata annotations, and provenance history. Once the query is completed successfully, the VDS locates all the data and procedures required to derive the requested data and generates an "abstract" derivation workflow. The term "abstract" in this context means each node in this workflow is not tied to specific executables or datasets, and it is represented in XML format, which can be read and executed by various workflow engines.

In the VDL nomenclature, a *transformation* represents an executable program, and contains all the information needed to invoke it. A *derivation* represents an execution of a particular transformation, and specifies the files that provide input and output to the associated transformation, as well as any other arguments that are needed to execute the transformation [21]. Thus, a derivation provides a specification of how to generate a particular file (file A, say) by stating (in the simplest case) which transformation to apply to a set of specific input files in order to produce it. There is no need to explicitly store complete workflows, because if a particular file is not available the VDS will determine the derivation for the missing file and first execute the transformation associated with it to generate the missing file. Applying this process repeatedly will, in effect, create the workflow needed to generate file A. A transformation in which the name and location of the executable program are explicitly given can referred to as a concrete transformation. However, it is also possible to have an abstract transformation in which the executable details are absent, and where the name of the transformation plays a similar role to the concept of a service type (introduced in Section 3.1). The Pegasus system [22] interoperates with the VDS by taking the abstract workflow generated by the VDS and mapping it onto Grid resources. Pegasus selects a particular executable for each abstract transformation, thereby creating a concrete workflow which is converted to Condor's DAGMan[13] format ready for execution.The Pegasus approach focuses on using just-in-time scheduling algorithms and AI techniques to automatically map an abstract workflow onto a set of distributed service instances [23]. In the approach adopted in this dissertation, this mapping corresponds to the conversion of an abstract workflow to a concrete workflow, which is carried out by the user within the automatically generated portal. Thus, this dissertation addresses the requirement of user choice, rather than attempting to automatically minimize some execution metric, such as makespan.

It is possible write a computer program to convert a VDL or Pegasus workflow into a workflow that conforms to the XML schemas discussed in Sections 4.1.3 and 5.5. An abstract workflow, as conceived in Section 5.5, is similar to the corresponding concept in VDS, although VDS is not based on Web services. However, VDS has no concept of a file type – in VDL workflows can only be constructed to produce specified output files from specified input files.

Other efforts to implement abstract workflow management solutions include WebFlow [24], a system with multiple levels for high performance distributed computing, and GridFlow [25] which has a two-layered architecture with global Grid Workflow management and local Grid sub-workflow scheduling.

[13] http://www.cs.wisc.edu/condor/dagman/, accessed 12 January 2010.

2.4.5 GRED Graphical Editor

GRED [26] is a graphical editor that is used as part of the integrated programming environment GRADE. It is used to design, debug, and performance tune message-passing programs running on a network of computers. The GRED editor hides the cumbersome details of the underlying low-level message-passing system (which is currently the PVM[27] system) by providing visual abstractions, but allows the programmer to define local computations of the individual processes in C or in Fortran independently from the visually supported process management and inter-process communication activities.

2.4.6 Visual Programming Environments

There are several visual programming environments, such as Taverna [28], Kepler [29] and Triana [30], for composing service-oriented distributed application in a graphical environment. Taverna is a workbench for workflow composition and enactment developed as part of the myGrid project. In comparison with Kepler and Triana, it does not provide an interactive workflow editor, nonetheless it provides a static workflow viewer, where the workflow is saved as an image file. In contrast, the Triana project enables users to visually construct workflows in a graphical environment, and can be considered as a workflow-based graphical problem-solving environment. Triana is independent of any particular problem domain and incorporates a range of Web services and Grid jobs. Within Triana, a local file can be connected with a remote file and the link performs a file transfer. The submission of a job is performed by entering details of the executables. These systems provide users with powerful support for the composition and execution of distributed workflows, with an intelligent and user-friendly interface to compose service-oriented applications and to monitor the whole execution process. The disadvantage of these systems is the steep learning curve for users to become proficient in their uses and also potential difficulty when porting service-oriented applications between environments as their installation and maintenance may prove to be a significant challenge.

2.4.7 Transforming Between Multiple Workflow Languages

The workflow execution engine responsible for distributed application execution usually only accepts as input files in one specific workflow description language. It is often difficult to execute a workflow written for one workflow execution engine on a different workflow execution engine. In order to avoid manually reconstructing a workflow with the same structure in different workflow languages, it is necessary to automatically translate the workflow from one workflow language to another. For example, an XSLT workflow language converter [31] can transform one workflow to another with a set of predefined rules expressed in one or more XSLT stylesheet files.

2.5 Summary and Review

This chapter has introduced the concept of service-oriented computing, and justified the use of Web service technologies for implementing the prototype software system evaluated in Chapter 7. Related research has also been reviewed, and a key difference between the approach adopted in this dissertation and other approaches has been identified, namely that this dissertation investigated the *automatic* generation of a portal based on an abstract workflow description. This portal is then used to create a concrete workflow that can be executed from within the portal on a third-party execution engine. This approach differs from the P-GRADE portal which does not make a distinction between abstract and concrete workflows, and which does not produce a separate portal for each abstract workflow. The approach in this dissertation also differs from the VDS/Pegasus system which lacks the concept of a file type, and is more concerned with the selection of service instances, rather than enabling user-directed selection. Other software environments for workflow composition and execution, such as Taverna, Kepler, and Triana, also do not dynamically generate portals, and do not feature both abstract services and files through service and file types.

Chapter 3

A Framework for the Automatic Generation of Portals

3.1 Introduction

As discussed in Chapter 2, two important drawbacks of an integrated development environment for the composition and execution of distributed workflows are the length of time it may take to become a proficient user of the environment, and potential problems in porting the composed application between different computer platforms. To address these drawbacks an alternative approach will be detailed in the rest of this dissertation. The focus is on developing lightweight tools capable of performing small, clearly-defined tasks that are interoperable with other existing workflow tools. These tools make it possible to separate workflow composition and workflow execution. The composition of a workflow is divided into three steps that create a workflow skeleton, an abstract workflow and a concrete workflow in sequence, which are defined as follows:

- A *skeleton workflow* specifies the structure of a workflow as a directed acyclic graph (DAG) made up of nodes representing files and services. The arcs in the graph represent the flow of data between a file and a service, or between one service and another. The content of the files, and the tasks carried out by a service, are not specified in a skeleton workflow.

- An *abstract workflow* specifies what the services that make up the workflow do, but does not specify particular implementations for these services.

- A *concrete workflow* specifies not only what the services in the workflow do, but also specifies their service implementations.

In the approach promulgated in this dissertation, the last stage of generating the concrete workflow, and then executing it, are both functions that are integrated within the portal environment, which is generated automatically from a description of the abstract workflow. This portal can still be used as a high-level integrated user interface.

The type of distributed application that benefits from the automatic portal generation framework investigated in this dissertation has the following features:

1. The application can be represented as a DAG workflow.

2. The application's inputs and outputs can be regarded as files.

3. The application's users require that it can be run multiple times with different input and output files [32].

4. The application's users require that a task can be performed with different algorithms by choosing from a set of semantically-equivalent service implementations in a service repository.

5. The application must be extensible. This means that the application can easily be edited to modify its original structure to address different user requirements.

The first two of these features are intrinsic to the application, and the last three are user requirements for the software environment in which the application is created and executed.

The first four features above match the GECEM application and portal, with the only drawback of the GECEM portal being that it does not support users wishing to modify its structure. Once the portal is generated, the application structure is fixed and the portal would have to be redesigned in order to modify it. The Portal Generator presented in this dissertation allows a portal to be automatically generated based on a given data-driven abstract workflow, which is described in XML format with a specific XML schema. When the portal is generated, the system creates a set of corresponding files for configuring the portal. The server hosting the portal can notify potential users where and how to access portal, and the layout of each portlet in the portal is specified by the portal configuration files. In order to initialize the execution of the workflow, the portal allows users the abstract workflow into a concrete workflow by specifying:

- The input files.

- The name and location of the intermediate and final output files.

- The service instances corresponding to the nodes of the abstract workflow. The service instance for each service node is selected from a set of semantically-equivalent services stored in the service repository.

The portal integrates this information with the abstract workflow and generates a concrete workflow, bàsed on the BPEL workflow description language. Finally, this concrete workflow is passed to a workflow execution engine, while the portal continues to monitor the whole process of execution.

The Portal Generator is a tool chain that automatically generates a web-accessible portal based on an abstract workflow. This tool chain consists of three major parts to implement workflow skeleton design, abstract workflow specification, and web portal generation. The first component in the system is the Design Tool, which enables users to visually design a skeleton workflow in a graphical environment. The details of skeleton workflow generation will be described later in Chapter 4. The second component is called the Label Tool, which accepts as input the output of the Design Tool and aids users in creating an abstract workflow by specifying the input and output file types, and the semantically-equivalent services from the file repository and the service repository respectively. The last component, the Build Tool, is the major component of this system and accepts as input the abstract workflow output by the Label Tool and automatically generates a portal. The Build Tool's main task is to generate a set of portlets to support users in selecting the specific service and file instances in the concrete workflow. In addition, it configures the portal so the portlets can be laid out properly and are accessible by authorized users of the portal. A schematic representation of this tool chain, and the generated portal, is shown in Fig. 3.1.

Figure 3.1 implies that the user of the Design Tool knows what is contained in the service repository, since it would not be productive to compose a skeleton workflow that cannot be executed subsequently because the services needed are not in the repository. If this is not the case then Fig. 3.1 is an oversimplification, and the process of designing and labeling a workflow would be an iterative process. Another solution would be to embed a repository browser into the Design Tool so that user can check on what service types are available when composing the skeleton workflow. However, in this case it might be better simply to merge the Design Tool and Label Tool into a single interface. A number of third-party UDDI browsers are available[1].

It should be noted that the service and file repositories, as key components of the service-oriented infrastructure, would be established and managed by an administrator with appropriate expertise. The end-users would not be directly involved in managing the repositories. Thus, the administrator would identify the type for each service and file, perhaps in consultation with the users.

In discussing abstract workflows it is important to clarify what is meant by "semantically-equivalent services" and "file types". In the context of this dissertation, two or more service instances are said to be *semantically equivalent* if they have the same typed interface and perform the same high-level computational task, although the low-level details of the algorithm for carrying out the task, and other implementation details, may be different for different services [33]. Services that are semantically equivalent in this sense are said to have the same *service type*. Similarly, a set of files are said to have the same *file type* if their content conforms to the same specified template or schema. For example, in image processing JPEG files can be viewed as

[1] For example, http://sourceforge.net/projects/uddibrowser/, accessed 12 January 2010.

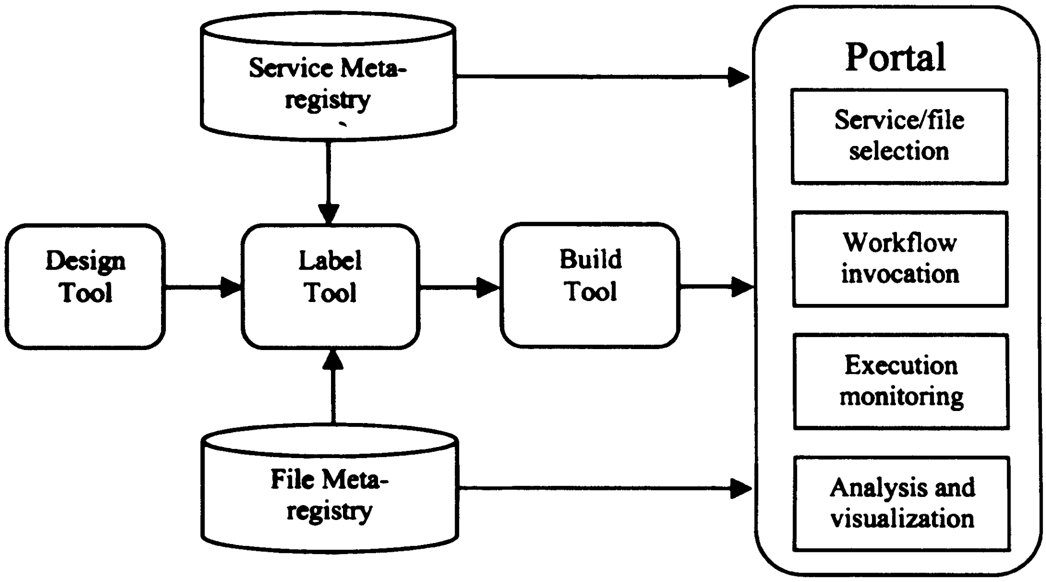


Figure 3.1: Design Tool, Label Tool, Build Tool and the generated portal.

being of the same file type. If two services are semantically equivalent, in the above sense, then the file types of their corresponding inputs and outputs are the same. Thus, an abstract workflow is an extension of a skeleton workflow in which each file and service node has been labeled with a file or service type. A concrete workflow is an extension of an abstract workflow in which each file and service node has been associated with a particular file or service instance.

3.2 Design Tool

The first component in the tool chain, the Design Tool, generates a skeleton workflow and provides users with a graphical interface that enables them to draw the workflow structure by simply dragging and dropping specific components onto a canvas. The skeleton workflow is saved as an XML file with a specific schema. In order, to indicate the dataflow, directed connections are drawn between the tasks and files used. The Design Tool also enables users to modify an existing skeleton workflow by converting the skeleton's workflow description file back to a graphical representation in the tool's workflow editor. The Design Tool also makes it possible to publish skeleton workflows through the network and share them with other users. The Design Tool is an standalone Java application, and does not require any external data source.

3.3 Label Tool

The Label Tool is the second component in the tool chain, and provides an interface that allows users to associate a specific service type, or file type, with each node in an input skeleton workflow. File and services types are explained in Section 3.1. A common IO service is used to read from, and write to, files. The Label Tool can take a skeleton workflow output from the Design Tool as input. It first validates the input against the abstract workflow XML schema, then analyzes the file nodes and service nodes, and displays them on a graphical canvas. At the same time, this tool accesses a specified online service repository and file repository. Thus, when the user wants to associate a file type or service type with a node in the skeleton workflow, the tool checks whether the specified type is compatible with the node in the current workflow, since its type may be constrained by the other nodes to which it is connected. Subsequently, the Label Tool outputs a new XML file to describe the abstract workflow, which derived from the input skeleton workflow by labeling each file node with a file type, and each service node with a service type.

3.4 Build Tool

The Build Tool is the most important and innovative component in the tool chain. It generates the portal application including a set of portlet codes, several configuration files, and corresponding JSP pages. These automatically generate files zipped into a WAR file, which can easily be deployed into a Java web server, such as Tomcat. The Build Tool takes the abstract workflow output by the Label Tool and parses it, and for each service task, generates a corresponding portlet in order for the user to select a service instance registered in the service repository. Optionally the user can also specify the name of the intermediate output file generated by the invocation of this service if a copy needs to be kept; for example, for checkpointing. Several portlets are also generated to specify the initial parameters for the service tasks with rank 0[2], and these parameters can be used to initialize the entire workflow execution, but excluding the file and service selection portlets. Additionally, the Build Tool generates a portlet, that is responsible for generating a concrete workflow in a suitable workflow description language, which is then deployed onto a specific workflow execution engine. Another portlet is also generated to enable the user to execute the workflow once it is successfully deployed with its initial parameters for the services with rank 0 as specified by the file selection portlets.

In the portal, all portlets are located in several tabbed panes. The first pane is for the initial parameter specification, and may contain one or more different parameter specification portlets. After the initialization of the file specifications each portlet is automatically connected to the file repository so users can select a file name compatible with the node's file type. A file

[2] Note: The definition of rank is introduced in Chapter 4.

contains all the information needed to invoke a service. The number of portlets in this pane depends on the number of parameters required to initialize the execution of the workflow. The layout of the service selection portlets depends on the ranks of the services in the workflow description, with the selection of services with the same rank being aggregated into one pane. These portlets access the service repository to get a list of the semantically equivalent services complying with the service type specified in the abstract workflow. Users can select a service instance from this list to insert into the workflow, and optionally specify the intermediate output file name. The generation of the concrete workflow description file and the deployment portlet enables users to generate a workflow and deploy it onto a selected workflow execution engine. The last portlet in the final pane is an interface between the user and the workflow execution engine. The final outputs, and any intermediate outputs, are automatically saved into the file system with the names specified by the user in the file and service portlets, and are also automatically registered in the file repository.

Chapter 4

Skeleton Workflows and the Draw Tool

4.1 Skeleton Workflows

This dissertation is concerned with distributed applications that can be represented by data-driven workflows by means of a directed, acyclic graph (DAG). In an acyclic graph there are no loops or iterations in the flow along directed arcs of the graph. The nodes in such a workflow represent either data sources/sinks (in the form of files) or computational tasks (in the form of services). The directed arcs flowing into a task node represent inputs for that task. The term *data-driven* means that a task will begin execution if, and only if, all of its inputs are available to it. In a data-driven workflow the flow of control is determined by the flow of the data.

As discussed in Chapter 3, the process of generating a service-oriented application workflow can be divided into the following three steps.

- Generate a skeleton workflow using the Draw Tool. This gives the structure of the workflow, with no reference to what the nodes of the workflow actually do, beyond each node representing either a file or a service.

- Generate an abstract workflow from a given skeleton workflow using the Label Tool.
- Generate a concrete workflow from an abstract workflow. This is done by user selection within the portal generated by the Build Tool.

There are other software tools that aggregate some, or all, of the above functions together. However, the main advantage of distinguishing between these three steps, and performing each with distinct software, is to allow ample opportunities for interoperation between the tool chain proposed in this dissertation (and shown in Fig. 3.1) and third-party software for generating workflows. The separation of concerns embodied in the above three steps also enables clearer and more elegant software design.

A skeleton workflow is described as a DAG, and each vertex node must be one of the following three types: an input node that has no predecessors; an output node has no successors; or an interior node that has both predecessors and successors. In a skeleton workflow input and output nodes can be considered as placeholders for files, and for consistency a service node having no external input is assumed to have a preceding null input node. Similarly, a service node that produces no external output is viewed as having a null output node. An interior node can be either a file or a service node. Interior file nodes provide a means of checkpointing the output of a service, as shown in Fig. 4.1, and can be removed without affecting the logical correctness of the workflow.

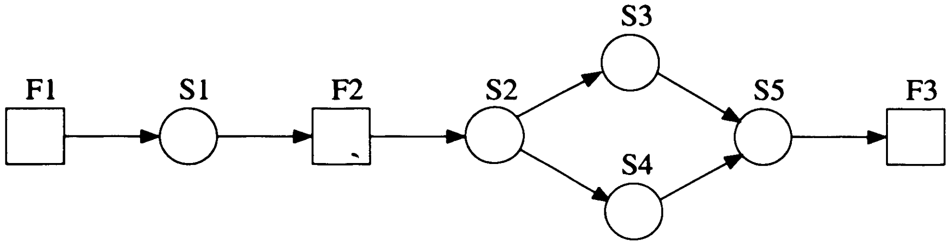


Figure 4.1: A skeleton workflow. File and service nodes are represented by squares and circles, respectively. F1 and F2 are the input and output files, and F3 is a file that stores the intermediate output produced by service S1.

4.1.1 Mathematical Background

A directed acyclic graph, $G = (X, U)$, consists of

- a finite set $X = x_1, x_2, x_3, \cdots, x_n$, of nodes, and
- a subset U of the Cartesian product of $X \times X$, the elements of which are called arcs. The notation $(x_i, x_j) \in U$ means there is a directed arc from node x_i to node x_j.

In the graph $G = (X, U)$, a node x_j is said to be a successor of a node x_i if $(x_i, x_j) \in U$. The set of all successors of x_i is denoted by $S(x_i)$. Similarly, a node x_j is said to be a predecessor of a node x_i if $(x_j, x_i) \in U$. The set of all predecessors of x_i is denoted by $P(x_i)$.

The rank of node x_i, $r(x_i)$, is defined recursively as

$$r(x_i) = 1 + \max_{x_j \in P(x_i)} (r(x_j)) \tag{4.1}$$

where the rank of any node with no predecessors is taken to be 0. Figure 4.2 depicts a DAG, in which all nodes of the same rank lie on the same vertical line. Based on the above concepts, each node in a DAG has a rank, and can be assigned a unique ID number by successively numbering the nodes with rank 0, followed by the nodes with rank 1, and so on, as shown in Fig. 4.2.

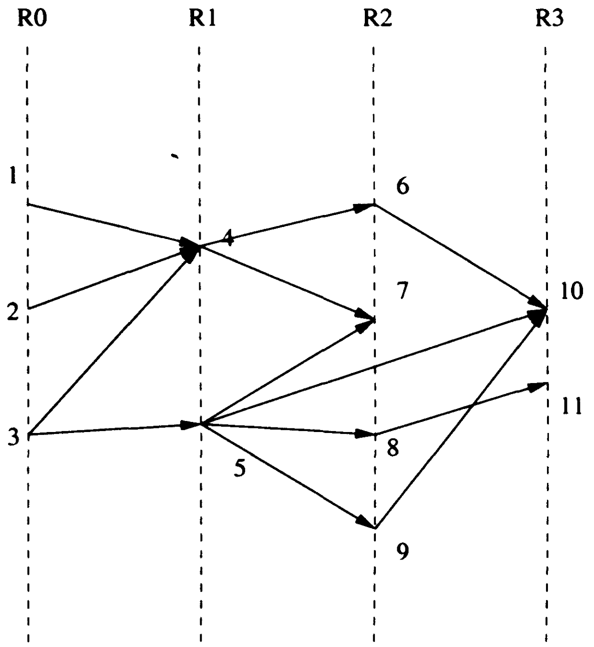


Figure 4.2: An acyclic graph

4.1.2 Representing a DAG

A computational application in a distributed environment may involve a large amount of interrelated activities (for instance, different web services located on different computers). In general, some activities can take place concurrently, but others must take place after a certain activity has completed.

Table 4.1 shows how the relationships between the activities of in an application workflow may be specified. The top and bottom activities with index 1 and 10 are "dummy" activities, which represent the start point and termination point of application, respectively, and take zero time. For each of the other activities i, the table gives their duration, d_i, and a list of their predecessors, that is, a list of activities which must all have finished before activity i commences.

Activity	Duration	Predecessors
1	0	–
2	4	1
3	10	1
4	6	2
5	2	2
6	11	2
7	22	4,5
8	3	5
9	17	3,6,8
10	0	7,9

Table 4.1: Tabular representation of a workflow application.

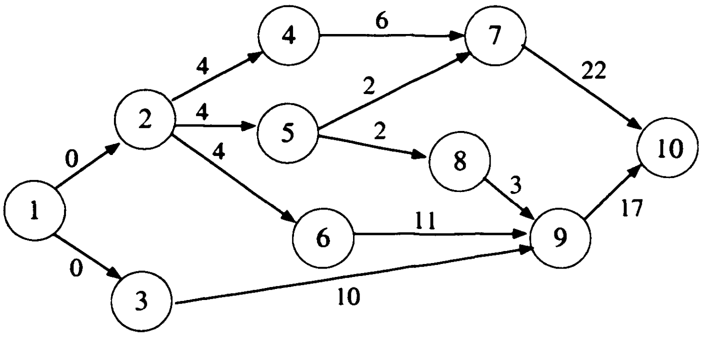


Figure 4.3: The workflow from Table 4.1 represented as a directed acyclic graph.

Based on the information in Table 4.1, the activity graph shown in Fig. 4.3 can be constructed to represent this application. In this graph, the nodes correspond to the activities, that is, the web services composing the distributed application. Each arc of the graph is labelled with the duration of the activity at its origin. Specifically, if activity i cannot commence before activity k has finished, that is to say until at least d_k time units after activity k commenced, this condition is represented by an arc (x_k, x_i), which is labelled with d_k.

4.1.3 The Structure of a Skeleton Workflow

In the standard way of representing a service in XML format the interface of the service is described indtependently from its implementation details. This interface description in provided in a WSDL document, as described in Section 2.2.4. A workflow composed of services can also be represented in XML so that the workflow specification is distinct from its implementation and execution. This allows the same workflow to be ported across different user communities and execution environments by associating different service instances with different tasks in the workflow. Thus, it is necessary to design an XML-based language to specify the model described in Section 4.1.2.

In Section 4.1.2, a skeleton workflow is defined so that each task node has a rank and a unique ID, and no cycles exist in the structure. The directed arcs between two nodes are the paths by which data flows from one workflow node to another. In practice, the task nodes are not enough to create a complete workflow, as at least one file node is required to initialize the skeleton workflow, and another is required to act as the final output. Moreover, the file nodes also require an ID to identify them. Therefore, a complete skeleton workflow is a set of tasks, each with its own unique ID and a rank, which are connected to at least one predecessor file input node, and one successor file output node. The file nodes also have their own unique ID, and the directed arcs describe the data's source and destination. The data source for an arc can either be the output generated by a service or data directly read from a file. All arcs must have their own unique ID to distinguish them from other arcs. The components and relevant information required in defining a complete skeleton are shown in Table 4.2

In Table 4.2, each task node has a unique ID number, which is an integer

Name	Property
Task node	ID Type Name (optional)
File node	ID Type Name (optional)
Arc	ID Source ID Destination ID Name (optional)

Table 4.2: The components in a skeleton workflow.

value indexed from 0, to specify the node's type and an optional name value, with default value null. The naming conventions of a skeleton workflow allows two or more tasks nodes to share the same name and this will not cause any conflicts, since the system only identifies a task node by its ID. Therefore, the ID can be used as the primary key for each task node, and this convention is also applied to the file nodes. A task node and file node also can have the same ID number, since the system can distinguish these nodes by their type values. In contrast, the arc component only consists of an ID indexed from "0", and two values that identify the source and destination nodes. Therefore, this convention only allows three different types of arcs, which are "file-to-task", "task-to-task", and "task-to-file". A "file-to-file" arc is not allowed, since in a virtualized context, a "file-to-file" type would not entail any action.

XML is used to represent the structure of a skeleton workflow in a way that is completely cross-platform and independent of programming language. Almost all the programming languages, such as C/C++, Java, Perl, and so on, provide facilities for XML processing in several ways, which makes XML

ID	Name	Element type	Property
0	null	task	Task node
0	null	file	Source input
1	null	file	Destination output
0	null	activity	An arc in the workflow
1	null	activity	An arc in the workflow

Table 4.3: Simple skeleton workflow.

a very attractive representation for describing service composite applications in a distributed environment. Since service interfaces are also described by XML (using WSDL), it is relatively easy to expand a given skeleton workflow into an abstract workflow, and then into a concrete workflow. Thus, by merging the XML service information into the skeleton workflow an executable workflow can be produced for input to a workflow execution engine. An XML document complying with the skeleton workflow XML schema (discussed in Section 4.1.4) represents a specific skeleton workflow. A very simple example will now be given to show how to map a skeleton workflow into an XML document. In this simple example, there is only one task node with one predecessor (an input file node), and one successor (an output file node), as shown in Table 4.3. The corresponding XML document has a root element called SkeletonWorkflow, which contains all the sub-elements representing file nodes, task nodes, and arcs (these are represented by activity elements). The XML document is for the simple example is shown in Fig. 4.4.

The skeleton workflow described by this XML document is shown in Fig. 4.5.

In the XML document, the SkeletonWorkflow element is the root element. It contains two file elements that each map to file nodes in the skeleton workflow structure. Each file element has, in turn, a child element called ID, which holds the ID value of each file node in the workflow. The root

```
<?xml version="1.0" encoding="UTF-8"?>
<SkeletonWorkflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:noNamespaceSchemaLocation="SkeletonWorkflow.xsd">
    <file>
        <ID>0</ID>
    </file>
    <file>
        <ID>1</ID>
    </file>
    <task>
        <ID>0</ID>
    </task>
    <link>
        <ID>0</ID>
        <source isFile="true">0</source>
        <target>0</target>
    </link>
    <link>
        <ID>1</ID>
        <source>0</source>
        <target isFile="true">1</target>
    </link>
</SkeletonWorkflow>
```

Figure 4.4: XML document for the simple skeleton workflow.

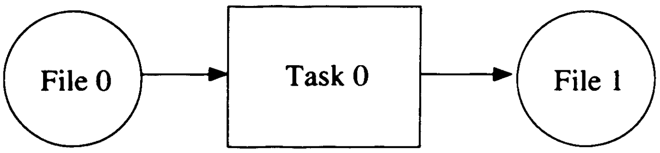


Figure 4.5: Simple skeleton workflow.

element also contains one task element that represents the task node in the workflow. The task element is similar to the data element, and has an ID subelement to hold the task ID value. The final two subelements nested in the SkeletonWorkflow element are link elements. A link element represents an arc connecting a file node and a task node, or two task nodes, and is the most complicated type of child element of SkeletonWorkflow. There are three completely different arcs possible in a skeleton workflow structure: file-to-task, task-to-file and task-to-task. In this simple case, only two types are used, and the last one will be introduced in a more detailed example later on. However, for any kind of arc, the corresponding activity element has to have a unique ID, and source and destination subelements to indicate which nodes are connected by the arc. In addition, the source and destination elements have an isFile optional attribute (with default value "false"), which indicates if the source/destination is a file node. Thus, in Fig. 4.4, the first link element has an ID of 0. Its source corresponds to the file element with ID of 0, since the source subelement has ID = 0 and isFile is "true". The destination of this link element is the task element with ID of 0, since the the destination subelement has ID = 0 and isFile is "false", by default. The second link element has an ID of 1. Its source corresponds to the task element with ID of 0, since the source subelement has ID = 0 and isFile is "false", by default. The destination of this link element is the file element with ID of 1, since the the destination subelement has ID = 1 and isFile is "true".

In the preceding simple example, only two types of arc connections are involved in the skeleton workflow, file-to-task and task-to-file, and the type task-to-task is missing. Task-to-task connections are the most important connection type in a workflow, as usually data are exchanged between pairs of services, rather than between a file and service. The most complicated arc connection is a composition connection, in which a task node receives input from multiple sources, which may be file or service nodes. A more complicated example is now introduced to demonstrate other types of arc connections. This example workflow contains three file nodes, two task nodes and four arc connections, as shown in Fig. 4.6.

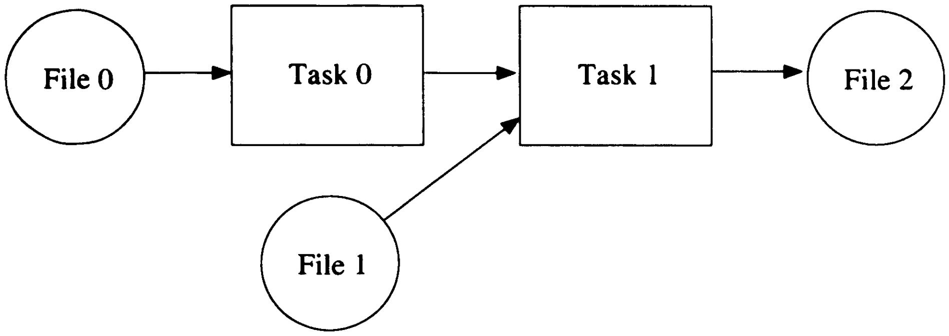


Figure 4.6: A more complicated skeleton workflow.

Figure 4.7 shows the XML representation of the workflow in Fig. 4.6. The arcs between File 0 and Task 0, and between Task 1 and File 2, are specified in the same ways as in the earlier example (see Fig. 4.4). The new feature of Fig. 4.6 is the fact that Task 1 has multiple inputs, from Task 0 and File 1. In the corresponding XML document in Fig. 4.7 these multiple inputs are handled by two inports which are associated with Task 1, which are linked to the outputs of Task 0 and File 1 through the link elements with ID = 1 and 2, respectively. Link 1 has a source subelement with value 0 and no isFile attribute, and a target subelement with value 1, a port attribute with value 0, and no isFile attribute. This indicates that this link connects the task with ID = 0 to port 0 of Task 1. Link 2 has a source subelement with value 1 and an isFile attribute with value "true", and a target subelement with value 1, a port attribute with value 1, and no isFile attribute. This indicates that this link connects the file with ID = 1 to port 1 of Task 1.

In discussing the representation of distributed applications as workflows the issue of how data are actually transferred between tasks is not addressed since this will be the responsibility of the workflow execution environment, and will also depend on the capabilities of the services that provide the implementation of the tasks. Thus, data may be streamed between tasks if both the workflow execution environment and the service implementations support this mode of transfer.

It should be noted that more complicated workflows are considered in Chapter 7.

4.1.4 XML Schema for Skeleton Workflow

An XML schema constrains the content and structure of an XML document, and so can be regarded as providing a model for the data in the document. An XML schema shows the basic structure of an XML document in terms of the nesting of its elements. In this dissertation all XML documents representing skeleton workflows have to comply with a particular schema. Document Type Definitions (DTD) are an alternative to XML schema for describing XML documents. In this dissertation XML schema are used, rather than DTDs, because the XML schema language itself is XML-based and supports richer data types than DTDs. Thus, it is possible to use a number of common XML tools to edit and process XML schema, although XML schema implementations require much more that just the ability to read XML. An XML schema can also be manipulated with XML DOM or JDOM, or translated by XSLT. Another important feature of XML schema

```
<?xml version="1.0" encoding="UTF-8"?>
<SkeletonWorkflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:noNamespaceSchemaLocation="SkeletonWorkflow.xsd">
    <file>
        <ID>0</ID>
    </file>
    <file>
        <ID>1</ID>
    </file>
    <file>
        <ID>2</ID>
    </file>
    <task>
        <ID>0</ID>
    </task>
    <task>
        <ID>1</ID>
        <inport>0</inport>
        <inport>1</inport>
    </task>
    <link>
        <ID>0</ID>
        <source isFile="true">0</source>
        <target>0</target>
    </link>
    <link>
        <ID>1</ID>
        <source>0</source>
        <target port="0">1</target>
    </link>
    <link>
        <ID>2</ID>
        <source isFile="true">1</source>
        <target port="1">1</target>
    </link>
    <link>
        <ID>3</ID>
        <source>1</source>
        <target isFile="true">2</target>
    </link>
</SkeletonWorkflow>
```

Figure 4.7: Workflow involving task input from more than one source.

that is useful for this work is its extensibility. This is a key feature because the skeleton workflow will need to be extended to an abstract workflow, and subsequently to a concrete workflow. At each stage, it must have a different corresponding XML structure to describe the workflow. Thus, it must have different XML schema to describe these structures in an XML document. Hence, with the extensibility of XML schema, the development of an XML schema becomes simple, as the new XML schema file can be conveniently developed by importing the other XML schema files as external sources, rather than re-editing the original file.

Since an XML schema document is an XML document, it has to import its own schema by namespace declarations. The XML schema namespace is usually declared like this: xmlns:xs="http://www.w3.org/2001/XMLSchema". Then it is necessary to define the elements and their properties in the skeleton workflow structure. The root element is a complex element, because it contains several nested elements, as shown in Fig. 4.8. SkeletonWorkflow is a complex element, which contains elements or attributes of either a simple or complex nature. In this particular case, it contains three other complex

```
<xs:element name="SkeletonWorkflow">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="file" type="fileType" minOccurs="0"
                                          maxOccurs="unbounded"/>
            <xs:element name="task" type="taskType" minOccurs="0"
                                          maxOccurs="unbounded"/>
            <xs:element name="link" type="linkType" minOccurs="0"
                                          maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 4.8: The root element in a skeleton workflow.

elements, which are data, task and activity. Note that these three child elements are surrounded by a <sequence> element, which indicates that these child elements must appear in the same order as they are declared. The first child element is "file" which represents the file node in a skeleton workflow and is also a complex element; the details of this element are introduced later. The next child element is "task", and the last one is the "link" element representing a connection between a file and task or between two tasks. Each of these child elements has two attributes, namely "minOccurs" and "maxOccurs" which constrains how many times each element appears in a workflow to ensure the skeleton workflow has at least the simplest form. The simplest workflow contains one task, two files, one file-to-task and task-to-file arc, as in Fig. 4.5.

```
<xs:complexType name="fileType">
    <xs:sequence>
        <xs:element name="ID" type="xs:int"/>
    </xs:sequence>
    <xs:attribute name="isFile" type="xs:string" default="false"/>
    <xs:attribute name="name" type="xs:string" use="optional"/>
</xs:complexType>
```

Figure 4.9: fileType definition.

The child element "file" nested in the "SkeletonWorkflow" element is also a complex element, as shown in Fig. 4.9. This defines a complexType called "file", which can be referred to by an element "file". This complexType includes a nested child element called "ID" of integer type. The "ID" child element is used to indicate the unique index of the file node. This complexType also has an optional attribute called "name", which can be left as a null value.

```
<xs:complexType name="taskType">
    <xs:complexContent>
        <xs:extension base="fileType">
            <xs:sequence>
                <xs:element name="inport" type="xs:int" minOccurs="0"
                                               maxOccurs="unbounded"/>
                <xs:element name="outport" type="xs:int" minOccurs="0"
                                               maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Figure 4.10: taskType definition.

The next child element nested in the "SkeletonWorkflow" element is "task" as shown in Fig. 4.10. This is derived by extending the file element in the last paragraph, to have optional "inport" and "outport" subelements

```
<xs:complexType name="linkType">
    <xs:sequence>
        <xs:element name="ID" type="xs:int"/>
        <xs:element name="source" type="lType"/>
        <xs:element name="target" type="lType"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="lType">
    <xs:simpleContent>
        <xs:extension base="xs:int">
            <xs:attribute name="isFile" type="xs:string" default="false"/>
            <xs:attribute name="port" type="xs:int"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

Figure 4.11: linkType definition.

The last child element is the most complicated one, and is responsible for representing arc connections, as shown in Fig. 4.11. It has three subelements:ID, source, and target. The ID subelement gives the unique ID of the link, and the source and target specify the nodes that are connected by the link. Both the source and target subelements may have "isFile" and "port" attributes. The "isFile" attribute indicates whether the node at the start (for source) or the end (for target) of the link is a file, and has default value "false". The "port" attribute indicates the inport (for target) or outport (for source) of the connection.

4.2 System Design and Implementation

4.2.1 System Structure

The Design Tool is a visual programming environment in which a user can draw a skeleton workflow by dragging and dropping the skeleton workflow components onto a canvas. The tool can automatically generate the corresponding XML file, which complies with the XML schema described earlier in this chapter. Also, the Design Tool enables users to edit an existing skeleton workflow by reading in its XML description file and drawing each component of the workflow on the canvas. Users then have the capability of modifying this existing workflow similar to creating a new workflow by dragging components on the canvas and/or adding new components. The revised workflow can then be saved as an XML file in the file system. The user diagram is shown in Fig. 4.12.

The layout of the Design Tool was designed to be a well-managed and simple GUI, to allow users to become quickly accustomed to its use in order to be able to draw a skeleton workflow.

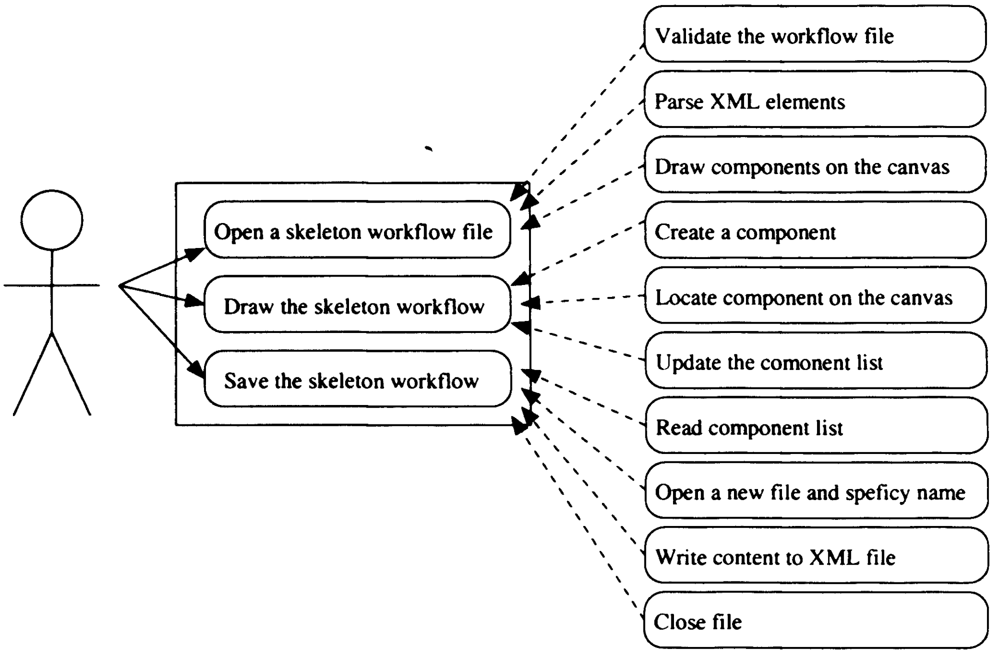

Open a skeleton workflow file
Draw the skeleton workflow
Save the skeleton workflow
Validate the workflow file
Parse XML elements
Draw components on the canvas
Create a component
Locate component on the canvas
Update the comonent list
Read component list
Open a new file and speficy name
Write content to XML file
Close file

Figure 4.12: Design Tool user case diagram.

Each function that the Design Tool provides is split into several subtasks. The modification of an existing skeleton workflow file is split into three subtasks. The file is first validated against the XML schema to make sure it is a valid skeleton workflow. If the file is valid, the data are read into memory and a list of components representing the structure of the skeleton workflow is generated. The tool automatically positions each workflow component on the canvas. The user is able to carry out further editing on the canvas by dragging and dropping the components, creating new components, or deleting existing components. Drawing a new skeleton file on a blank canvas is simpler than modifying an existing workflow. A user simply drags the new components onto the canvas, and meanwhile the tool will keep updating the list of components in memory. Finally, a valid XML file to represent the new skeleton workflow is generated. The process of saving a file is split into four steps. The tool asks the user to assign a name to the new skeleton workflow. Each component in the component list is retrieved from memory, and all the components are written into the output XML file. After the IO operation, the tool automatically flushes the memory reserved for this workflow and IO operation, and then closes the file.

The Design Tool is implemented with Java in an object-oriented model, and has two major parts. The first one is the visual editing environment, which is in charge of the skeleton workflow presentation and its user-friendly interface. Users can directly interact with the Design Tool through this interface which passes the input from the users to the data processing part of the tool. After the data is processed the components of the skeleton workflow represented on the canvas are updated. The data processing is the second major part of the tool, and is responsible for storing information about the skeleton workflow components and updating the component list.

There are seven major classes in the design of the visual environment, and Fig. 4.13 presents the class names and variables. The largest class is the Workflow class, which is responsible for generating the layout of the Design Tool to provide a user-friendly interface that makes the generation of a skeleton workflow a simple and easy process. The workflow class contains a list of the Java SWING [34] components used to construct the interface. Based on the structure of the skeleton workflow described in Section 4.1.3, the Design Tool must have three graphical self-defined classes: TaskNode, FileNode and Arc. These are used to represent a task node, a file node, and an arc between a task node and a file node, or between two task nodes, and inherit from the JPanel class. The TaskNode and FileNode are both derived from their common parent class called Node. This makes use of class inheritance in the object-oriented (OO) programming model [35], and reduces the amount of unnecessary redundant code by avoiding the tedious process of re-declaration of common variables and the definition of common methods in different classes. The technique used in an OO programming model, is to declare the common variables and define the common methods in one parent class. The child classes, which require these variables and methods, can then be implemented by using a keyword "extends" in the Java programming language. Thus, the parent class "Node" defines the common variables "ID", "selected" and "boundColor", and there are three pairs of methods used to assign and retrieve these variables. The "ID" variable is used to identify each node and the "selected" variable is used to indicate the status of the node. If a node is connected to another node and it is unavailable, the "selected" variable will hold the value "true"; otherwise it will be "false". The "boundColor" variable is important to indicate the type of node. The workflow class is not derived from the Node class itself; the "boundColor" variable takes effect after some child class inherits from this parent class. When the taskNode class inherits from the Node class, this variable is reloaded to represent the task node. If it is derived by a fileNode, this variable takes the file node color.

The Connector class also inherits from the Node class, and is capable of representing the inflow and outflow of data for a task node. In this class there are only two variables declared: "connected" and "color". The "color" variable is used to show the status of the "Connector" to inform the user whether the node is connected or not. There are two child classes, "output" and "input", and one variable represents the direction of the flow of data. If the data flow enters a task node, this data is treated as flow-in data, and

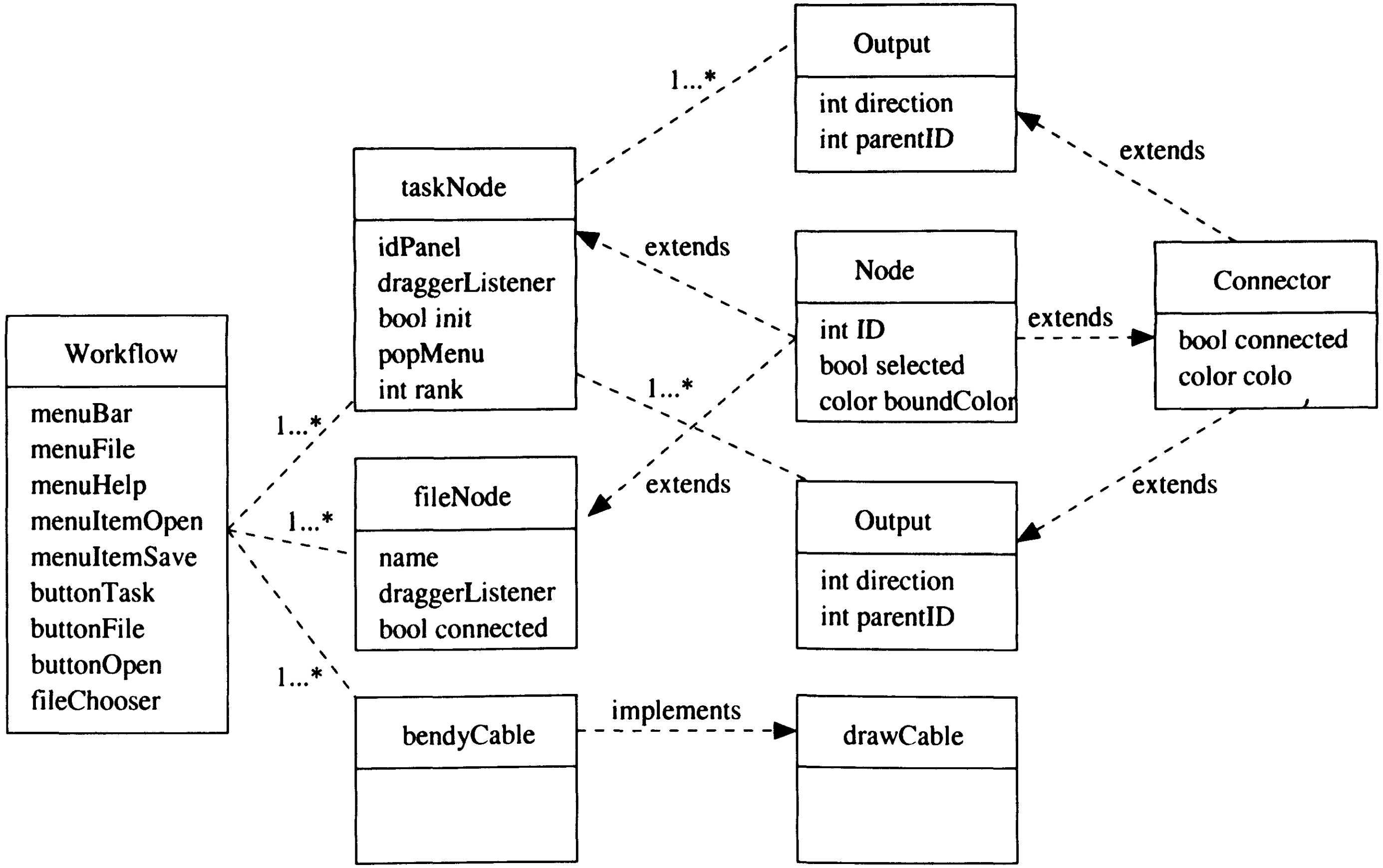


Figure 4.13: Design Tool class diagram part 1

in the Design Tool this is represented by "Input". If the data leaves a task node, the data is considered as an outflow of data and it is represented as "Output" in the Design Tool. The "input" and "output" classes also have one variable called parentID that is used to indicate which task node the "Input" or "Output" is related to, as one task node is allowed to have more than one "Input" as a parameter and more than one "Output", each of which is a copy of the task node execution result. Another feature that the node has is that the user can drag and drop them around the canvas. Therefore, nodes have an event listener, which monitors actions taken by the users and triggers an update of the data processed. There exists an important Java Swing component called "BendyCable", which is responsible for representing the connection between any two node objects. This implements an interface called the "DrawCable" interface, which provides methods for drawing a connection between two nodes. In the "BendyCable" class all these methods are implemented.

In the Workflow class there is a toolbar containing several function buttons, which serve as shortcuts. These represent the functions that are frequently used by users, including creation of a task node or file node, and deletion of a task node or file node. In order to create a new file node, the user can click the "File Node" button and a new file node will appear on the canvas of the Design Tool, along with a new ID. The same technique is applied when the user wants to draw a task node on the canvas. Once this button is pressed, the task node will not appear immediately. First, the interface will display an option panel to prompt users to specify the number of inputs and outputs. Only when the user has clicked the confirmation button, will the task node appear on the canvas. The Design Tool also provides a button to delete nodes. When a user wants to delete either a file or task node, he/she must first select the node by clicking on the node, which makes its outline become red. Then the user may click on the "Delete" button, which triggers two events: one is that the selected item is deleted from the component list, and the other is that the Design Tool repaints the canvas. The Design Tool provides a button for users to create the output skeleton workflow description file. Once the user decides to create an output of the XML file by pressing this button, the Design Tool displays a file chooser for the user to specify the name of output file, and then converts the component list saved in memory to an XML file.

In the design of the Design Tool the data processing is separated from the interface, because this separation makes it simpler for improvements to be made in the future. When making improvements to the data processing part, such as changing to a more efficient algorithm to check the validation of the skeleton workflow, all the modifications will occur only in the data processing part. Conversely, if the interface is enhanced aesthetically, this will not lead to any changes in the data processing part.

Marshalling a Java object (JavaBean [36]) means converting it to XML format for storage or sending. Unmarshalling means converting the XML elements back to the corresponding Java objects. It is a great advantage to work with unmarshalled XML documents because the elements can be converted to Java Beans. Since Java regular classes are much easier and natural to work with, and code can be maintained efficiently and edited in comparison with parsing a vast bundle of XML documents. The unmarshalled Java Beans object can even keep the validated attributes based on the original XML schema. Therefore, in order to read or write data from the skeleton workflow to an XML file, there are three corresponding bean classes, which hold the information to describe a file node, a task node and a connection between them. The FileSource class holds the file node information to indicate the id and name of a file node. The TaskSource class holds the task node's ID, rank, name and its ports. The DataLink class is used to hold information on the connection between two nodes. The design of DataLink addresses two cases: one where a file node works as a target node, and the other where the target node is a task node.

The IO tasks performed by the Design Tool are to write a skeleton workflow into an XML file, and to read a skeleton workflow from an XML file. Given the simplicity of the XML schema for the skeleton workflow, DOM or JDOM will not be used in the Design Tool, because XML element features, such as adding a new element or deleting an existing element, are not required. This avoids the unnecessary memory requirements of DOM or JDOM, because DOM and JDOM reads the entire XML file document into memory and generates a tree structure for presenting all the elements in the XML file. Hence, when the user wants to modify an existing skeleton workflow, the XML file is parsed by SAX (Simple API for XML), which is the best way to map data to Java objects with minimum memory use. SAX is an event-based XML parser implementing the SAX API. It generates a set of events that respond to different features found in the XML parser. In comparison, with DOM (or JDOM) it avoids having to read the entire XML document into memory to generate a huge tree structured model to represent the XML document. To summarise, even though DOM and JDOM seem to be more beneficial and have more advanced features, they cause inefficiency which may affect some performance-sensitive applications. For example, if the number of task nodes is large, the DOM or JDOM object model API will cause the Design Tool to be extremely slow.

When a user is drawing a skeleton workflow the Design Tool detects whether the skeleton workflow is valid in order to be converted into an XML form. The validation of the workflow is done prior to any IO operation because it avoids any IO exception errors that would occur if the workflow did not comply with the XML schema. Thus, the validation of the data is a key consideration in the design of the Design Tool. The simplest skeleton workflow consists of at least one task node, two file nodes for input parameters and output, and two data links. With this constraint in mind, suppose a user tried to save a skeleton workflow having either only one task and file node. The Design Tool will respond with a popup window to indicate that the conversion of this workflow is forbidden. When the user wishes to generate a connection to either another file node, or port located on another task node, the Design Tool will prompt a window panel with a list of all task nodes with at least one available port to connect to. The Design Tool scans all the components in the component list and makes this list available to the user. This avoids deadlock in the skeleton workflow. In addition, the Design Tool will automatically check whether there is still any task node with an unconnected port or any unconnected file node. All task nodes and file nodes have to be fully connected by arc connections before the skeleton workflow is converted to an XML document. The file node background colour will automatically change to indicate that it is connected, when the arc connection is created. The colour of a port indicates whether it is connected to another port located on another task node or file node.

Chapter 5

Abstract Workflows and the Label Tool

5.1 Introduction

Chapter 4 has described how skeleton workflows can be created using the Design Tool, and how these are represented in XML as DAGs made up of file and task nodes connected by links that represent the flow of data. This chapter shows how a skeleton workflow can be extended to produce an abstract workflow using the Label Tool. With the Label Tool the user specifies the type of each file and task node in the workflow. As defined in Section 3.1, a set of files are said to have the same file type if their content conforms to the same specified template or schema. Similarly, a set of services are said to have the same service type if they have the same typed interface and perform the same high-level computational task. In this context, saying that a set of services have the same service type is the same as saying that they are "semantically-equivalent services". The Label Tool interacts with a file repository and a service repository. The file repository stores information about the type and location of files. The service repository stores information about service types and implementations, including the location of the WSDL file and service endpoint of-service instances. With the Label Tool a user can label every task node in a skeleton workflow with a service type. This is a manual process since it is specifying what the workflow actually does. However, some degree of automation is possible, For example, since the labeling of the task nodes determines the details of the input and output of every task node, when this labeling process is complete the file type of every file in the resulting abstract workflow will be determined. Furthermore, if task nodes with lower rank are labeled first, then this determines the file types of some or all of the inputs to the higher rank task nodes, which in turn restrictions the service types that those nodes could be labeled with. Thus, the Label Tool presents to the user only those service type that are feasible for a particular task node, given its known input file types. In addition to producing an output abstract workflow as an XML file, the Label Tool also performs consistency checks to ensure that linked tasks have compatible inputs and outputs, and that the file repository contains files with the required file types.

The remainder of this chapter is organized as follows. Section 5.2 describes the use of a UDDI repository to store a registry of Web services. The mapping between WSDL data and UDDI data is described, and forms the basis of querying a UDDI repository based on the characteristics of a service as stored in its WSDL file. Section 5.2.5 outlines have to set up a UDDI repository using the open-source JUDDI implementation, while Section 5.2.6 briefly describes how to query such a repository. In Section 5.3 the padZero Web service is used to illustrate how a Web service is registered and discovered in a UDDI repository. Some important points in the design and implementation of the Label Tool are given in Section 5.4. Section 5.5 presents details of the XML Schema used for describing an abstract workflow. Finally, a review of the chapter is given in Section 5.6.

5.2 The Web Service Repository

The Universal Description, Discovery, and Integration (UDDI) specification provides the standard way of storing the detail of Web services in a repository, and so it is assumed that UDDI is used as the Web Service repository with which the Label Tool (and hence the user) interacts.

WSDL was introduced in Section 2.2.4 as the standard way of describing a Web service in terms of its interface and location. In Section 2.3, UDDI was introduced as a widely-used Web service repository that uses the tModel data abstraction to store Web service information. The use of UDDI as the Web service repository used by the Label Tool requires some way of interrelating the WSDL and UDDI data models. This issue will now be considered in Sections 5.2.1 – 5.2.4.

5.2.1 UDDI Data Structure

The UDDI specification allows entities, such as businesses and organizations, to register public information about themselves and details of the services they offer. It also allows entities such as businesses and industry groups to register information about the types of services they have defined, and to refer to them with unique identifiers. These identifiers are Universal Unique Identifiers (UUIDs) that follow the OSF Distributed Computing Environment (DCE) [37] convention. UDDI has two fundamental data structures: businessEntity and tModel. Business entity information is divided into White, Yellow and Green pages. The tModel data structure provides a reusable abstraction, through the tModel element, that specifies reusable abstract definitions of service types that others can use and combine. Thus, the tModel data structure gives a signature for a service.

5.2.2 Integrating WSDL into UDDI

As outlined in Section 2.3, the UDDI specification provides a platform-independent way to publish and discover Web services and service providers. The UDDI data structures provide a framework for the description of the basic Web service information, and an extensible mechanism to specify service access information in a standard description language. Many such languages exist in specific domains, however, WSDL is the standard, general-purpose, XML-based description language for describing the interface, protocol bindings, and deployment details of network services. WSDL complements the UDDI standard by providing a uniform way to describe the abstract interface and protocol bindings of arbitrary network services. This subsection clarifies the relationship between WSDL and UDDI, and to describes an approach to mapping WSDL descriptions to UDDI data structures. The goal of mapping the WSDL to UDDI data structures is to enable the automatic registration of WSDL into UDDI, and also to enable flexible UDDI queries based on WSDL. This mapping enables various types of queries for both design time and run time discovery. The queries supported by this mapping are listed in Table 5.1.

Given Information	Supported Query
The namespace and/or local name of a wsdl:portType	The tModel that represents that portType
The namespace and/or local name of a wsdl:binding	The tModel that represents that binding
A tModel representing a portType	All tModels that represent bindings for that portType
A tModel representing a binding	All bindingTemplates that represent implementations of that binding
The namespace and/or localname of a wsdl:service	The businessService that represents that service

Table 5.1: The queries supported by the mapping.

5.2.3 The WSDL and UDDI Data Models

The WSDL and UDDI data models have been introduced in the previous sections. WSDL contains import, portType, binding, service and port elements. The WSDL structure and the relationship between elements are shown in Fig. 5.1. In the UDDI data model, tModel elements are often referred to as service type definitions. The tModel element plays an important role in the UDDI data model. In mapping WSDL to UDDI a tModel is used to represent the technical specification, such as service types, bindings and network protocol, and it is also used to create a category system to classify technical specifications and services. When a particular specification is registered in a UDDI registry as a tModel, it is assigned a unique key, called a tModelKey. The tModelKey is used by other UDDI entities to reference the tModel. Services are represented in the UDDI structure by the businessService element, and details of how, and where, the service is accessed are provided by one or more binding structures. Figure 5.2 schematically shows the UDDI data structure. The businessService can be thought of as a logical container of services. The bindingTemplate structure contains the accessPoint of the service, as well as a reference to the tModel that implements it.

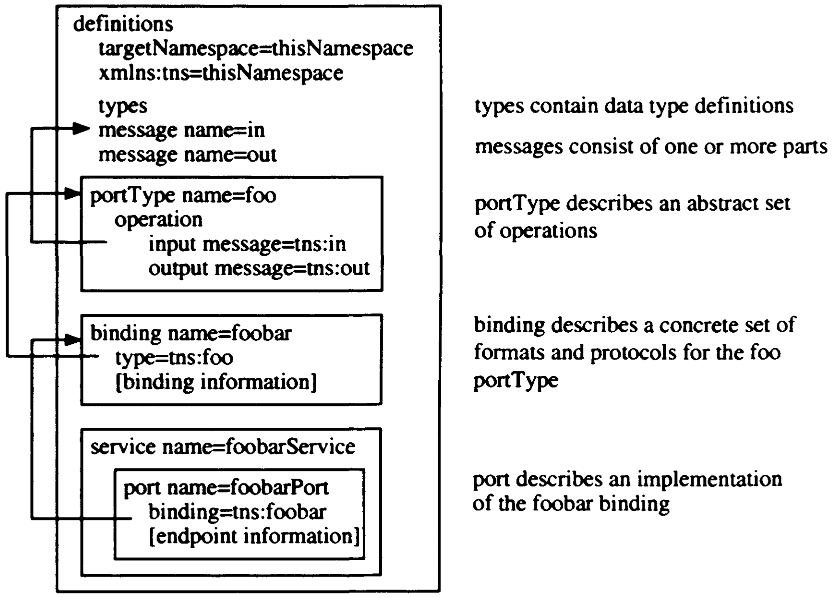

definitions
targetNamespace=thisNamespace
xmlns:tns=thisNamespace
types
message name=in
message name=out
portType name=foo
operation
input message=tns:in
output message=tns:out
binding name=foobar
type=tns:foo
[binding information]
service name=foobarService
port name=foobarPort
binding=tns:foobar
[endpoint information]
types contain data type definitions
messages consist of one or more parts
portType describes an abstract set of operations
binding describes a concrete set of formats and protocols for the foo portType
port describes an implementation of the foobar binding

Figure 5.1: WSDL data structure and the relationship between elements.

5.2.4 Mapping WSDL to UDDI

WSDL is capable of supporting reusable and modular definitions, and each element has a certain relationship with other elements. The goals of mapping WSDL into UDDI are to enable the automatic registration of WSDL definitions in UDDI, and to support precise and flexible UDDI queries based on WSDL artifacts and metadata. In order to support queries based on WSDL artifacts and metadata, the mapping must be able to represent the individual WSDL artifact and the relationship between artifacts. The methodology

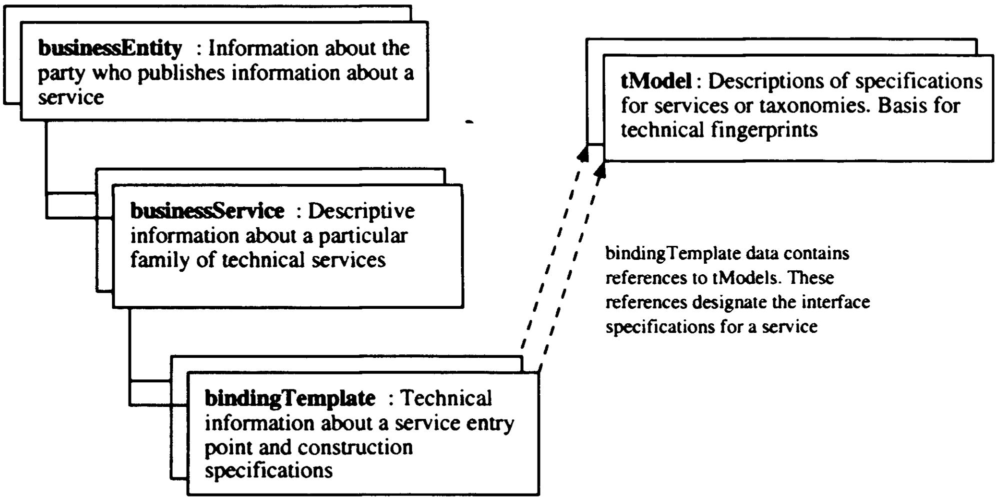

businessEntity : Information about the party who publishes information about a service
businessService : Descriptive information about a particular family of technical services
bindingTemplate : Technical information about a service entry point and construction specifications
tModel : Descriptions of specifications for services or taxonomies. Basis for technical fingerprints
bindingTemplate data contains references to tModels. These references designate the interface specifications for a service

Figure 5.2: businessEntity, businessService, bindingTemplate and tModel elements in the UDDI data structure.

of the mapping is to map each WSDL artifact to a separate UDDI entity. The wsdl:portType and wsdl:binding elements map to uddi:tModel entities. keyedReference elements provide a mechanism to express additional metadata and also to represent the relationship between UDDI entities. Figure 5.3 shows how WSDL artifacts are mapped to UDDI entities.

Mapping wsdl:portType to uddi:tModel

The portType element in WSDL is modeled as a tModel in UDDI. The information which has to be captured from portType is its entity type, its local name, its namespace, and the location of the WSDL document that defines this portType. The mapping of portType information to a UDDI tModel is as follows: the uddi:name element of tModel is used to represent the local name of the portType. The tModel contains a categoryBag which must contain a keyedReference with a tModelKey of the WSDL entity category system and a key value of "portType". In addition, the tModel has to have an overviewDoc with an overviewURL that contains the location of the WSDL

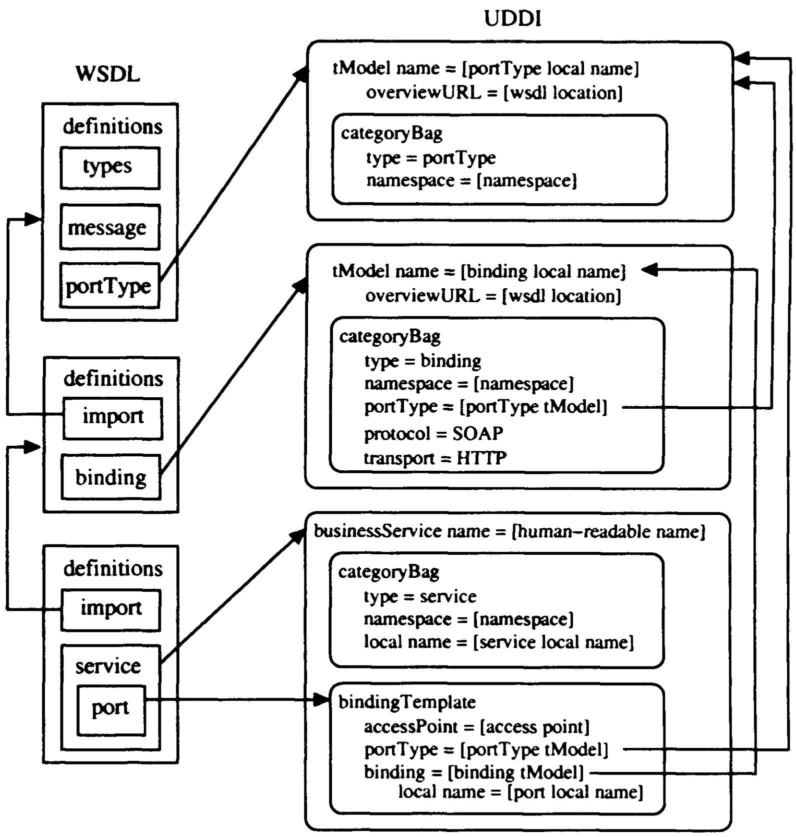

UDDI
WSDL
definitions
types
message
portType
definitions
import
binding
definitions
import
service
port
tModel name = [portType local name]
overviewURL = [wsdl location]
categoryBag
type = portType
namespace = [namespace]
tModel name = [binding local name]
overviewURL = [wsdl location]
categoryBag
type = binding
namespace = [namespace]
portType = [portType tModel]
protocol = SOAP
transport = HTTP
businessService name = [human-readable name]
categoryBag
type = service
namespace = [namespace]
local name = [service local name]
bindingTemplate
accessPoint = [access point]
portType = [portType tModel]
binding = [binding tModel]
local name = [port local name]

Figure 5.3: Mapping WSDL artifacts to UDDI entities.

WSDL	UDDI
portType	tModel (categorized as portType)
Namespace of portType	keyedReference in categoryBag
Local name of portType	tModel name
Location of WSDL document	overviewURL

Table 5.2: Mapping of the wsdl:portType element.

document to describe the portType. Table 5.2 gives a summary of how the wsdl:portType element is mapped.

Mapping wsdl:binding to uddi:tModel

The binding element in WSDL is modeled as a tModel in UDDI. The information captured from the binding element is its entity type, its local name, its namespace, and the location of the WSDL document defining the binding. This is exactly same as the information captured for the portType element. The only difference is that it adds the portType which this binding implements, and its protocol. In the mapping of the binding information to the tModel the uddi:name element of tModel holds the value of the name attribute of the wsdl:binding. The tModel for the binding also has to have a categoryBag which should have at least four keyedReferences:

1. A keyedReference with a tModelKey of the WSDL entity category system and a keyValue of wsdl:binding.

2. A keyedReference with a tModelKey of the WSDL portType reference category system and a keyValue of the tModelKey that models the wsdl:portType to which the wsdl:binding relates.

3. A keyedReference with a tModelKey of the UDDI type category system and a keyValue of "wsdlSpec" for backward compatibility.

WSDL	UDDI
binding	tModel(categorized as binding and wsdl-Spec)
Namespace of binding	keyedReference in categoryBag
Local name of binding	tModel name
Location of WSDL document	overviewURL
portType binding relates to	keyedReference in categoryBag
protocol from binding extension	keyedReference in categoryBag
transport from binding extension (if it exists)	keyedReference in categoryBag

Table 5.3: Mapping of wsdl:binding to uddi:tModel.

4. One or more keyedReferences are required to indicate the protocol and optionally the transport information, such as soap:binding and http:binding.

Table 5.3 gives a summary of the mapping of wsdl:binding to uddi:tModel.

Mapping wsdl:port to uddi:bindingTemplate

The port element in WSDL is modeled as a bindingTemplate in UDDI. The information that needs to be captured about a port is the binding that it implements, the portType that it implements, and its local name. By capturing the binding, users can search for services that implement a specific binding. By capturing the portType, users can search for services that implement a particular portType without necessarily knowing the specific binding implemented by the service. Figure 5.4 gives a summary of mapping port to bindingTemplate.

WSDL	UDDI
port	bindingTemplate
namespace	captured in keyedReference of the containing businessService
Local name of port	tModel name
Location of WSDL document	overviewURL
binding implemented by port	tModelInstanceInfo with tModelKey of the tModel corresponding to the binding
portType implemented by port	tModelInstanceInfo with tModelKey of the tModel corresponding to the portType

Table 5.4: Mapping of wsdl:port to uddi:bindingTemplate.

5.2.5 The JUDDI Implementation of UDDI

In the Label Tool, JUDDI[1] is chosen to implement the UDDI repository for Web services. JUDDI is an open-source Java implementation of the UDDI specification, and hence is platform-independent. JUDDI requires an external data store to manage and persist the registry data, and is compatible with any relational database which complies with the ANSI standard for SQL. As a pure Java web application, JUDDI can be deployed to any application service or servlet container that supports the servlet API (version 2.1 or later). In the Label Tool implementation, MySQL is used as the external data store, and Jakarta Tomcat serves as the web application to host the JUDDI application. The details on the deployment of JUDDI on MySQL, and the configuring of JUDDI, are introduced in the Section 5.4.2.

JUDDI has a core request processor that can unmarshal UDDI requests into several Java objects, and invoke the appropriate UDDI function to process them and then marshal UDDI responses into XML format. In JUDDI, to invoke a UDDI function employs the services of three configurable modules

[1] http://ws.apache.org/juddi/, accessed 12 January 2010

that deal with data persistence, authentication, and the UDDI generator. The authentication module authenticates users (by userID and password), and provides each authenticated user with an authorization token, which is needed to publish or make queries. In the UDDI specification, each UDDI entity, such as businessService, bindingTemplate, and tModel, needs a unique token to identify itself. Therefore, JUDDI provides a UDDI generator to generate the unique ID for different UDDI entities.

JUDDI contains three groups of APIs. The first group enables users to access the UDDI registry information and manage the publishers. The APIs in the second group support different UDDI queries. For example, the find_tModel function can return a tModel list that contains zero or more tModelInfo structures matching specified criteria. The get_tModelDetail function enables users to request full information about the known tModel data. A user can retrieve the full description of a specific tModel through the corresponding value of its key. The last group of APIs is responsible for managing existing UDDI entities, or adding/deleting UDDI data.

5.2.6 The UDDI4J Class Library

Web service technology allows applications to communicate with each other through an XML-based protocol. UDDI provides a registry where service providers can store information about the Web services they provide and users can make queries on these available services and invoke them. The open-source software UDDI4J[2] defines a set of Java APIs that provide access to a UDDI registry. These can be used to query the information about Web services in the registry from within a Java application. UDDI4J wraps XML elements used by UDDI and then sends them to the registry to perform the

[2] http://uddi4j.sourceforge.net/, accessed 12 January 2010

querying and publishing.

In a typical implementation of UDDI, the SOAP protocol[3] is used to access a UDDI registry. Each registry provides a set of SOAP access points. In the Label Tool, UDDI4J provides a Java implementation of the UDDI API specification, and is used to access the UDDI registry. In the core of UDDI4J, each UDDI entity is mapped to a corresponding Java object - this is quite similar to how Java Beans describes specific data structure in a Java object. For example, each UDDI tModel data has the same data structure as shown in Fig. 5.4: Each tModel entity in the UDDI registry has three attributes, "authorizedName" "operator" and "tModelKey", together with a number of nested subelements: "name", "description", "overviewDoc" and "categoryBag".

Figure 5.5 shows the relevant parts of the Java code that uses UDDI4J to represent the corresponding tModel entity. The code consists of a set of methods for setting and getting attributes. Other data structures in the UDDI specification are handled in a similar way. The UDDIProxy class in UDDI4J enables a client program to access a UDDI registry. UDDIProxy also supports network access to an actual UDDI registry server. The access requests and responses are wrapped as SOAP objects by the UDDIProxy and are transmitted between the client program and UDDI registry server. The details of implementations of UDDI queries based on UDDIProxy are specified in Chapter 6.

[3] Formerly referred to as the Simple Object Access Protocol, see `http://www.w3schools.com/soap`, accessed 12 January 2010.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <tModelDetail generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
      <tModel authorizedName="dashan lu" operator="jUDDI.org"
              tModelKey="uuid:C7E7A310-4F65-11DC-A310-F1F60770805C">
        <name>Powerof2Porttype</name>
        <description>Description of the abstract Powerof2 web service</description>
        <overviewDoc>
          <overviewURL>
            http://chlorin.cs.cf.ac.uk:8080/axis/new1wsdl/Powerof2Porttype1.wsdl
          </overviewURL>
        </overviewDoc>
        <categoryBag>
          <keyedReference keyName="portType namespace"
                          keyValue="http://dlu1.FFTWebservice1.powerof21/definition"
                          tModelKey="uuid:E792EDD0-015B-11DC-BE3D-A55DD9166EA6"/>
          <keyedReference keyName="WSDL type" keyValue="portType"
                          tModelKey="uuid:B824DBE0-015A-11DC-BE3D-E2747DAFDB82"/>
        </categoryBag>
      </tModel>
    </tModelDetail>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 5.4: An example tModel expressed as an XML document.

```
public class TModel extends UDDIElement {
public TModel(Element base) throws UDDIException {
        super(base);
        tModelKey = base.getAttribute("tModelKey");
        operator = getAttr(base,"operator");
authorizedName = getAttr(base,"authorizedName");
...
}

public void setTModelKey(String JavaDoc s) {
        tModelKey = s;
     }

public void setOperator(String JavaDoc s) {
      operator = s;
     }

     public void setAuthorizedName(String JavaDoc s) {
        authorizedName = s;
     }

...
}
```

Figure 5.5: JAVA code for tModel data structure in UDDI specification.

5.3 Building a UDDI Web service repository

The Label Tool is a Java client application that allows a user to associate each node of a workflow with a specific type of Web service from a Web service repository. To build a Web service repository based on the UDDI specification, the system and software have to include the JUDDI web application, MySQL[4], the Axis SOAP engine[5], and the Apache Tomcat web server[6]. Tomcat is a Java servlet container that is used to host the JUDDI web application. MySQL serves as the external relational database system to store all UDDI entities that represent Web services. Apache Axis, which is the Apache implementation of SOAP, acts as the Web service development platform and host.

5.3.1 Web service development

In order to build a Web service registry, several Web services first have to be prepared. In this case, the Web services are all developed with Apache Axis. The procedure for developing a Web service with Axis is divided into two parts. The first part is to develop a Web Service Description Language file for the Web service. The second part is to generate the Java code to implement the Web service. The Web service padZero will be used to demonstrate the procedure of Web service development, deployment and registration. The padZero Web service takes one double array and one integer value as input and returns another double array. The WSDL document for this specific Web service is separated into two files. As described in Fig. 5.3, a description of a Web service in WSDL is based on object-oriented concepts. In

[4] http://www.mysql.com/, accessed 12 January 2010.

[5] http://ws.apache.org/axis/

[6] http://tomcat.apache.org/, accessed 12 January 2010.

the first file, an abstract description of the Web service is created through the "portType" element. In the second file, the details of the Web service implementation, such as transport protocol and SOAP access point, are specified. Moreover describing a Web service with two separate WSDL files is more convenient when registering Web service information as UDDI entities, and making UDDI queries. More details of the advantages of separated descriptions of a Web service will be explained in the next section. To create a WSDL file for the padZero Web service, the first WSDL file defines the "portType" of this Web service as shown in Figure 5.6.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:def="http://dlu.FFTWebservice.dustCloud"
xmlns:tns="http://dlu1.FFTWebservice1.zeropad1/definitions"
targetNamespace="http://dlu1.FFTWebservice1.zeropad1/definitions">
<types>
<xs:schema>
        <xs:import namespace="http://dlu.FFTWebservice.dustCloud"
schemaLocation="doubleArray1.xsd"/>
</xs:schema>
</types>
        <message name="zeroPadInput">
<part name="in0" type="def:doubleArray"/>
                <part name="in1" type="xs:int"/>
        </message>

        <message name="zeroPadOutput">
                <part name="out" type="def:doubleArray"/>
        </message>

        <portType name="PadZeroService">
                <operation name="zeroPadedData">
                        <input name="zeroPadRequest" message="tns:zeroPadInput"/>
                        <output name="zeroPadResponse" message="tns:zeroPadOutput"/>
                </operation>
        </portType>
</definitions>
```

Figure 5.6: The portType part of the padZero Web service.

In the WSDL file in Fig. 5.6, an external XML schema file is imported to define an array filled with double values. Note that although the WSDL specification supports arrays of primitive datatypes in Java, the ActiveBPEL workflow engine which is used in the prototype system developed in this research, does not fully support these elements and could cause a serious error when the concrete workflow containing the padzero Web services is executed. After the external XML schema is imported, two message elements are used to define the inputs and output of this Web service. In turn, the portType element refers to two messages. The next step is to define "binding" and "service" elements in the second WSDL file, as shown in Fig. 5.7.

The WSDL file in Fig. 5.7 imports the first WSDL file containing the "portType" for the padZero Web service. In the "binding" element, the value of the "type" attribute is the name of "portType", qualified by its namespace. In the soap:binding element nested in the binding element, the SOAP style is specified as rpc. That means that this Web service is an RPC type of Web service. As this is declared as an RPC-style Web service [38], the client part that invokes this Web service must create an element with the qualified name of the operation, and then add each message part listed in the soap:body element as its child element. So in the SOAP envelope to send a request to the Web service there is still a single element. The RPC style is not good for interoperability, especially when exchanging complex datatypes between Web services implemented with different languages and hosted on different platforms. Therefore in the prototype system, all Web services that are registered in the Web service repository are all implemented in the Java language and only support primitive datatypes, and arrays of such datatypes. In the "service" element, the Web service access point is specified in the soap:address child element in order to tell the client side where to send

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
         xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
             xmlns:xs="http://www.w3.org/2001/XMLSchema"
             xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:defs="http://dlu1.FFTWebservice1.zeropad1/definitions"
             xmlns:tns="http://dlu1.FFTWebservice1.zeropad1/service"
             targetNamespace="http://dlu1.FFTWebservice1.zeropad1/service">
  <import namespace="http://dlu1.FFTWebservice1.zeropad1/definitions"
          location="PadZeroPorttype1.wsdl"/>
  <binding name="PadZeroServiceSoapBinding" type="defs:PadZeroService">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="zeroPadedData">
      <soap:operation soapAction=""/>
      <input name="zeroPadRequest">
        <soap:body parts="in0 in1" use="literal"/>
      </input>
      <output name="zeroPadResponse">
        <soap:body parts="out" use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="PadZeroServiceService1">
    <port name="PadZero1" binding="tns:PadZeroServiceSoapBinding">
      <soap:address location="http://chlorin.cs.cf.ac.uk:8080/axis/services/PadZero1"/>
    </port>
  </service>
</definitions>
```

Figure 5.7: The binding and service elements of the padZero Web service.

the Web service invocation request.

After the WSDL of the Web service has been developed the next step is to generate a set of Java code files, and add into these the code to implement the Web service. Then the Web service is deployed on Apache Axis. To simplify the process of Web service development Axis provides developers with a code generation system , WSDL2Java, to automatically convert a valid WSDL file into Java code for Web service implementation. In the typical case of the padZero example, the client-side code generated by the WSDL2Java tool is as follows:

- Java interface for the WSDL portType element: padZeroService.java
- Java class for the WSDL binding element: padZeroSoapBindingStub.java
- Java interface for the WSDL service element: padZeroService.java
- Java class for the WSDL service element: padZeroServiceLocator.java

It should be noted that the WSDL2Java tool generates two files named padZeroService.java, but these are distinguished by being stored in different directories. When generating the server-side code from the binding element in the WSDL file, WSDL2Java generates a padZeroImpl.java file for the developer to fill in the implementation code. In addition, XML files are generated for deployment and undeployment of the Web service. The padZeroImpl.java implements the padZeroService interface, and the code to do the data processing for the service must be added into the padZeroImpl.java file. After finishing the implementation of the client and server side code, a Web Service Deployment Descriptor (WSDD) is used to do custom deployment of the Web service on Axis. The deployment descriptor contains information about the Web service to be deployed that needs to be made available to the Axis engine. Consider the deployment descriptor for the padZero Web service shown in Fig. 5.8. The root element in Fig. 5.8 tells the Axis engine that this XML document is a WSDD deployment, and defines the "java" namespace. The service element defines the service. In this case, the Web service provider is java:RPC, which indicates a Java RPC service. In fact, the org.apache.axis.providers.java.RPCProvider should instantiate and call the class, padZeroService, by giving the service two parameters through <paramater> tags, and telling the service of the public method of that class which can be called via SOAP. A correct WSDD file for a service can then be used to deploy the Web service on Axis and make it accessible via SOAP.

The whole procedure of development and deployment of a Web service on Axis has been explained with the padZero example. Other required Web services can be implemented and deployed on the same Axis server in the same way. All WSDL files for these Web services are split into two parts - one containing the abstract description of the Web service interface, and the other is to define the "binding" and "service" elements. The Web service is described in two separate WSDL files to allow the UDDI registry to support more advanced queries.

5.3.2 Registering a Web Service into UDDI

In order to make a Web service visible to service consumers, all Web services have to be published in a UDDI Web service repository. Sections 5.2.5 and 5.2.6 have explained how to register the WSDL for describing a Web service into a UDDI repository. As explained in those sections, and in Section 5.2.4, each WSDL element maps to a specific corresponding UDDI data entities so that JUDDI can support advanced UDDI queries based on WSDL

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
          xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="PadZero1" provider="java:RPC" style="rpc" use="literal">
    <requestFlow>
      <handler type="soapmonitor"/>
    </requestFlow>
    <responseFlow>
      <handler type="soapmonitor"/>
    </responseFlow>
    <parameter name="wsdlTargetNamespace"
               value="http://dlu1.FFTWebservice1.zeropad1/service"/>
    <parameter name="wsdlServiceElement"
               value="PadZeroServiceService1"/>
    <parameter name="schemaQualified"
               value="http://dlu.FFTWebservice.dustCloud"/>
    <parameter name="wsdlServicePort" value="PadZero1"/>
    <parameter name="className"
        value="zeropad1.FFTWebservice1.dlu1.service.PadZeroServiceSoapBindingImpl"/>
    <parameter name="wsdlPortType" value="PadZeroService"/>
    <parameter name="typeMappingVersion" value="1.2"/>
    <operation name="zeroPaddedData" qname="zeroPaddedData" returnQName="out"
               returnType="rtns:doubleArray"
               xmlns:rtns="http://dlu.FFTWebservice.dustCloud"
               returnItemQName="tns:item"
               xmlns:tns="http://dlu.FFTWebservice.dustCloud" soapAction="" >
      <parameter qname="in0" type="tns:doubleArray"
                 xmlns:tns="http://dlu.FFTWebservice.dustCloud"
                 itemQName="itns:item"
                 xmlns:itns="http://dlu.FFTWebservice.dustCloud"/>
      <parameter qname="in1" type="tns:int"
                 xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
    </operation>
    <parameter name="allowedMethods" value="zeroPaddedData"/>
    <arrayMapping xmlns:ns="http://dlu.FFTWebservice.dustCloud"
        qname="ns:doubleArray"
        type="java:double[]"
        innerType="cmp-ns:double" xmlns:cmp-ns="http://www.w3.org/2001/XMLSchema"
        encodingStyle="">
  </service>
</deployment>
```

Figure 5.8: Web service deployment descriptor for padZero Web service.

artifacts for Web services. To map a WSDL document artifact to UDDI entities requires the creation of a set of UDDI entities to represent the various WSDL elements. The WSDL Entity Type tModel provides a typing system for this purpose. This category system is used to identify that a UDDI entity represents a particular WSDL element. This category system can be generated through the save_tModel interface provided by the JUDDI web application, and the new tModel entity is stored in the external MySQL database. The details of this tModel are used to identify which type of WSDL each UDDI entity represents. The categoryBag nested in the tModel has a keyedReference:

```
<keyedReference keyName="WSDL type"
                keyValue="portType"
                tModelKey="uuid:B824DBE0-015A-11DC-BE3D-E2747DAFDB82"/>
```

In a similar way, there are two more category systems that represent the namespace and localname for WSDL. Before mapping the WSDL file into UDDI entities, it is also necessary to create the WSDL portType reference tModel, SOAP protocol tModel, and HTTP protocol tModel. The portType reference tModel is used to refer to the UDDI entity representing the specific portType. The WSDL portType reference tModel is first to be defined. There exist different internal relationships between the WSDL entities, such as a wsdl:binding describes a binding of a specific wsdl:portType, and a wsdl:port implements a particular wsdl:binding. All such relationships must be fully and exactly expressed in UDDI entities. The representation of these relationships in UDDI could be achieved by a built-in mechanism provided by UDDI and the reference category systems which were generated before. The portType reference tModel provides this kind of capability to indicate that a UDDI entity has a relationship with a certain wsdl:portType tModel. The content of the keyValue attribute in the keyedReference to this tModel is the tModelKey of the wsdl:portType tModel being referred to. Adding this keyedReference in the categoryBag can be used to indicate that a wsdl:binding implements a specific portType. Figure 5.9 is a WSDL portType reference tModel. Similarly, the SOAP protocol tModel and HTTP protocol tModel are both used to categorize the binding tModel representing the wsdl:binding that supports the SOAP and HTTP protocols. Note that further tModels must be created for different transport protocols, such as SMTP and TCP, but in this system communication between all Webs services in the workflow are based on the HTTP protocol.

```
<tModel authorizedName="dashan lu" operator="jUDDI.org"
        tModelKey="uuid:AC8C65D0-0161-11DC-BE3D-E961C56CC9A0">
  <name>uddi-org:wsdl:categorization:protocol</name>
  <description>
    Category system used to describe the protocol supported by a wsdl:binding
  </description>
  <overviewDoc>
    <overviewURL>
      http://www.oasis-open.org/committees/uddi-spec/doc/tn/tc-tn-wsdl-v2.htm#protocol
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference keyName="uddi-org:types" keyValue="checked"
                    tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"/>
    <keyedReference keyName="uddi-org:types" keyValue="categorization"
                    tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"/>
  </categoryBag>
</tModel>
```

Figure 5.9: tModel for portType reference

The WSDL file for the padZero Web service will be taken as an example to show the complete procedure for registering a WSDL file into a UDDI repository. In the WSDL files in Figs. 5.6 and 5.7 for describing the padZero Web service, there is only one portType and one binding to implement this portType. To register this WSDL portType element into the UDDI repository a tModel representing this specific portType must be created, and a unique tModel key generated for it. Several mandatory attributes for this tModel must be specified. The name of this tModel must be the same as the value of the name attribute of the portType element in the WSDL file. The overviewURL must be the URL to the WSDL itself. In its categoryBag, two keyedReference elements must be added. One of these is "portType namespace" and the content of keyValue in this keyedReference refers to the namespace tModel which is the targetNamespace of the WSDL file. The name of the other keyedReference item is "WSDL type", and the content of its keyValue is "portType", which is used to specify that the type represented by this tModel is portType. The tModel representing the portType of the padZero Web service is shown in Fig. 5.10. The binding tModel, shown in Fig. 5.11, representing the binding element of the WSDL for the padZero service, is a little more complicated than the one representing the portType. It has more keyedReference elements to specify which transport protocol the binding uses, and which portType the binding implements. Moreover, the content of the "WSDL type" keyedReference is specified as "binding" instead of "portType".

As shown in Fig. 5.2, the service and port components in a WSDL document are represented as businessService and bindingTemplate entities, respectively, in UDDI. The bindingTemplate holds all the information for a Web service to be actually invoked. Usually a logical Web service may have several bindings, such as the SOAP-based-on-HTTP binding and SMTP

```
<tModel authorizedName="dashan lu" operator="jUDDI.org"
        tModelKey="uuid:1799B4C0-4F66-11DC-B4C0-99FD0C8084A7">
  <name>PadZeroPorttype</name>
  <description>Description of the abstract padZero web service</description>
  <overviewDoc>
    <overviewURL>
      http://chlorin.cs.cf.ac.uk:8080/axis/new1wsdl/PadZeroPorttype1.wsdl
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference keyName="portType namespace"
                    keyValue="http://dlu1.FFTWebservice1.zeropad1/definitions"
                    tModelKey="uuid:E792EDD0-015B-11DC-BE3D-A55DD9166EA6"/>
    <keyedReference keyName="WSDL type"
                    keyValue="portType"
                    tModelKey="uuid:B824DBE0-015A-11DC-BE3D-E2747DAFDB82"/>
  </categoryBag>
</tModel>
```

Figure 5.10: The tModel for the padZero Web service portType.

bindings. Each binding should be described in a bindingTemplate structure. In the case of the padZero Web service, the bindingTemplate structure should specify the access point of the Web service, and two the tModelInstanceInfo subelements, nested in the tModelInstanceDetails element, indicate that the service implements the padZero portType and uses a SOAP/HTTP binding. The creation of the bindingTemplate of the padZero Web service comes with the creation of the businessService element of this Web service by taking the save_service operation instead of creating the bindingTemplate by the save_binding operation. This is because the save_binding operation is usually used to save a new bindingTemplate to an existing service registered in UDDI. Similar to publishing a bindingTemplate, a service can be published along with its binding template details and a categoryBag containing a set of keyedReferences. In this particular case, the keyedReferences are used to identify which type of element is mapped to, namely the targetNamespace

```
<tModel authorizedName="dashan lu" operator="jUDDI.org"
        tModelKey="uuid:C8BF5E10-4F68-11DC-9E10-A3E74E7E43A1">
  <name>PadZeroSoapBinding</name>
  <overviewDoc>
    <overviewURL>
      http://chlorin.cs.cf.ac.uk:8080/axis/wsdl/PadZeroService.wsdl
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference keyName="binding namespace"
                    keyValue="http://dlu.FFTWebservice.zeropad/service"
                    tModelKey="uuid:E792EDD0-015B-11DC-BE3D-A55DD9166EA6"/>
    <keyedReference keyName="WSDL type"
                    keyValue="binding"
                    tModelKey="uuid:B824DBE0-015A-11DC-BE3D-E2747DAFDB82"/>
    <keyedReference keyName="portTypeReference"
                    keyValue="uuid:1799B4C0-4F66-11DC-B4C0-99FD0C8084A7"
                    tModelKey="uuid:4AF30FF0-021E-11DC-BE3D-C00BA74C8AB8"/>
    <keyedReference keyName="SOAP protocol"
                    keyValue="uuid:84C6C6E0-0160-11DC-BE3D-C858FF055FD9"
                    tModelKey="uuid:AC8C65D0-0161-11DC-BE3D-E961C56CC9A0"/>
    <keyedReference keyName="HTTP transport"
                    keyValue="uuid:9C1C49A0-0160-11DC-BE3D-AEDAD8606D3D"
                    tModelKey="uuid:617CE4B0-0162-11DC-BE3D-831AED3D34F1"/>
    <keyedReference keyName="uddi-org:types"
                    keyValue="wsdlSpec"
                    tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"/>
  </categoryBag>
</tModel>
```

Figure 5.11: The tModel for padZero Web service binding.

and local namespace. After publishing, the service should contain the information shown in Fig. 5.12. All the Web services required for workflow execution are registered in the UDDI Web service repository in a similar way. The Web service registry is now ready to support UDDI queries from the Label Tool.

```
<businessService businessKey="03927110-4F02-11DC-B110-C30F49B92B9C"
                 serviceKey="D808ECE0-4F6F-11DC-ACE0-D65BFED22EBF">
  <name>ZeropadService</name>
  <bindingTemplates>
    <bindingTemplate bindingKey="7D6A14D0-105D-11DD-94D0-AB68C79209FA"
                     serviceKey="D808ECE0-4F6F-11DC-ACE0-D65BFED22EBF">
      <accessPoint URLType="http">
        http://chlorin.cs.cf.ac.uk:8080/axis/new1wsdl/PadZeroService1.wsdl
      </accessPoint>
      <tModelInstanceDetails>
        <tModelInstanceInfo tModelKey="uuid:C8BF5E10-4F68-11DC-9E10-A3E74E7E43A1">
          <description xml:lang="en">
            The wsdl:binding that this wsdl:port implements.
          </description>
        </tModelInstanceInfo>
        <tModelInstanceInfo tModelKey="uuid:1799B4C0-4F66-11DC-B4C0-99FD0C8084A7">
          <description xml:lang="en">
            The wsdl:portType that this wsdl:port implements.
          </description>
        </tModelInstanceInfo>
      </tModelInstanceDetails>
    </bindingTemplate>
  </bindingTemplates>
  <categoryBag>
    <keyedReference keyName="WSDL type" keyValue="service"
                    tModelKey="uuid:B824DBE0-015A-11DC-BE3D-E2747DAFDB82"/>
    <keyedReference keyName="service namespace"
                    keyValue="http://dlu1.FFTWebservice1.zeropad1/service"
                    tModelKey="uuid:E792EDD0-015B-11DC-BE3D-A55DD9166EA6"/>
    <keyedReference keyName="service local name" keyValue="Zeropad Service"
                    tModelKey="uuid:EE022810-015C-11DC-BE3D-AA4640C03962"/>
  </categoryBag>
</businessService>
```

Figure 5.12: The padZero service in the UDDI repository.

5.4 System Design and Implementation

5.4.1 System Structure and Design

After building a UDDI Web service repository the Label Tool can now be designed as a client program to make queries to retrieve information from the repository, and then the user can then use this information to associate workflow nodes with a specific service type. In order to associate a service type with a node of a skeleton workflow drawn by the Design Tool, the Label Tool has to have the same ability as the Design Tool to read in an XML workflow description file, unmarshall the XML elements into corresponding Java objects, and display these Java objects on the canvas of its graphical interface. In addition, the Label Tool also works as a client program of the UDDI repository to enable users to query it to select the matching service types for each node. The user use diagram for the Label Tool is shown in Fig. 5.13.

Each function of the Label Tool is split into several subtasks. The key functionality is to query the UDDI repository to find the specific Web service types. A UDDI proxy is used to create a connection with the UDDI repository and retrieves information about the Web services registered in the repository. The proxy class has a set of methods to support different queries based on UDDI data structures. The queries made by the Label Tool to retrieve the Web service type are based on UDDI tModels representing the portType of Web services as expressed in their WSDL files. The Label Tool is also capable of identifying how many parameters (inputs) each node in a workflow has by analyzing the specified skeleton workflow description file. If a Web service type is matchable for a task node in the workflow, the tool allows the user

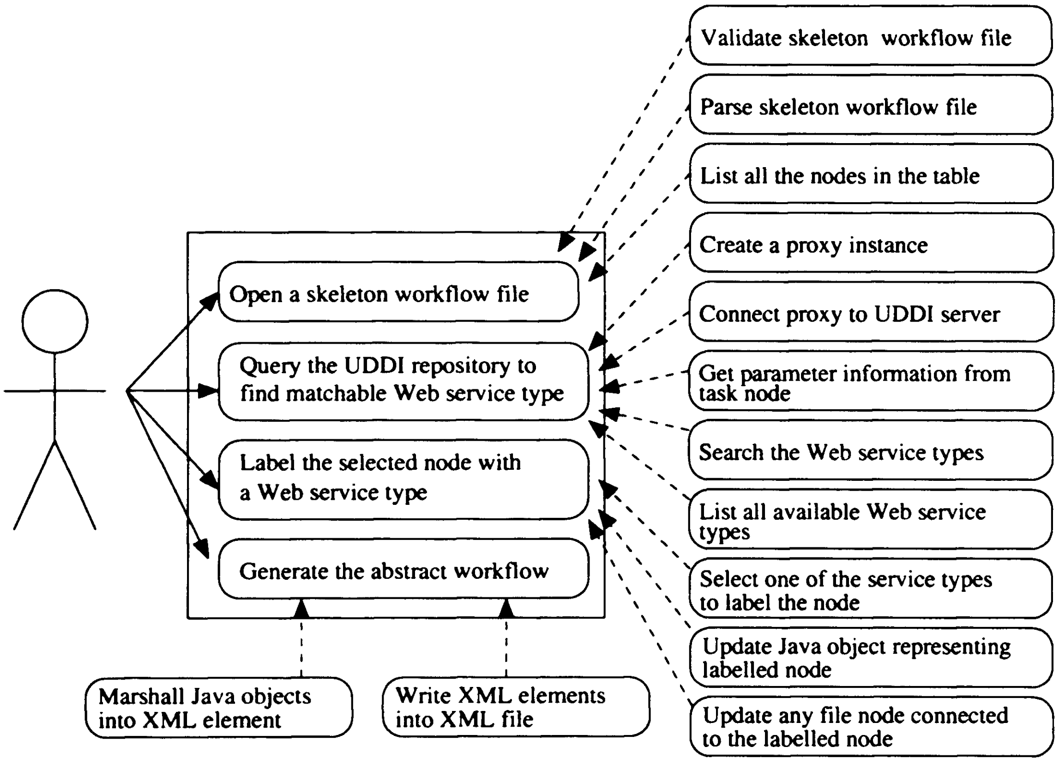

Validate skeleton workflow file
Parse skeleton workflow file
List all the nodes in the table
Create a proxy instance
Open a skeleton workflow file
Connect proxy to UDDI server
Query the UDDI repository to find matchable Web service type
Get parameter information from task node
Search the Web service types
Label the selected node with a Web service type
List all available Web service types
Generate the abstract workflow
Select one of the service types to label the node
Update Java object representing labelled node
Marshall Java objects into XML element
Write XML elements into XML file
Update any file node connected to the labelled node

Figure 5.13: User case diagram for the Label Tool.

to associate this specific Web service type with the task node. Once the task node is labeled, the status of all file nodes connected to this task node is fully determined and so can be labeled with a appropriate file type.

The workflow structure itself is not modified by the Label Tool, so the Label Tool uses the simpler Java swing component JTable [39], instead of the canvas used in the Design Tool to visualize and manipulate the workflow components. The JTable simply displays information for each node of the workflow, including ID, rank, and status. Once the status of all file nodes and task nodes is indicated as "labeled", the Label Tool can output the abstract workflow description file for the workflow.

As in the Design Tool, in order to avoid the existence of loops, the output port of a task node is not allowed to be connected to the input port of a task node with a lower ID. The Label Tool adopts the convention that one task node is not allowed to be associated with a matchable service type until all task nodes with lower ID have been labeled. This is because only the task nodes with rank 0 can determine their input types just by themselves. The task nodes with higher ID cannot fully determine their input types by themselves, because their input ports may connect to the output port of a task node with lower ID, and the data type of an input port must be identical to the data type of the output port to which it is connected. Therefore, until the lower-ranked task nodes have been labeled, the input data type of a task node cannot be determined. In addition, the user is not allowed to label file nodes since this can be done automatically once the task nodes they connect with are labeled. This principle guarantees that the data type associated with a file node always matches that of the input or output port of the task node to which it is connected.

Unlike the XML document processing done by the Design Tool, in the Label Tool the processing is more complicated since it involves tasks such as adding new elements or attributes and deleting an existing element or attribute. For this reason JDOM is chosen instead of SAX to do XML processing in the Label Tool. In contrast to the SAX event-driven model, the JDOM approach converts the whole XML document into a tree structure of Java objects and keeps them in memory. Although this way has definitely more memory overhead, it can provide advanced operations on XML documents more easily than SAX does.

5.4.2 System Implementation

Access the Web service repository with UDDI4J

The key functionality of the Label Tool is to retrieve information about service types by querying to UDDI Web service repository. The UDDI4J API is used to access the UDDI Web service registry and query the information about Web services. UDDI4J defines a straightforward and convenient mapping of the operations and data structures of the UDDI specification to Java objects and methods. It makes it simple to incorporate UDDI operations as part of any Java application, and hence is well-suited for use in the Label Tool. The Label Tool performs a find operation that retrieves all tModels representing a portType that is matchable with the task node selected by the user. The message sent to the UDDI server by the proxy to specify this query is shown in Fig. 5.14.

The tModel key of the service corresponds to the "WSDL type" tModel, and its value is specified as "portType". The UDDI4J code fragment for enacting the query is shown in Fig. 5.15. This code first sets up the inquiry

```
<find_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
  <categoryBag>
    <keyedReference tModelKey="uuid:B824DBE0-015A-11DC-BE3D-E2747DAFDB82"
                    keyName="WSDL type" keyValue="portType" />
  </categoryBag>
</find_tModel>
```

Figure 5.14: SOAP message for tModel query.

URL to tell the proxy where to send the query, and then creates a categoryBag instance to store the keyedReference in order to make the UDDI query. The keyedReference indicating "WSDL type" with the value of "portType" is added into the categoryBag. Finally the proxy makes the query based on this categoryBag by calling the find_tModel method to retrieve all tModels representing portTypes for all Web services. The return value of the method is an instance of TModelList that contains a list of tModel candidates. In this case, this list should contain all tModels representing the portType element. Once the list is retrieved successfully, the Label Tool will access each tModel instance in the list by calling the getTModelInfoVector() method. In each tModel representing the portType element, the most important information that the Label Tool wants to collect is overviewURL, which is the URL where the WSDL file for this specific portType is located. By accessing this URL, the Label Tool can retrieve the WSDL describing details of this portType, such as the message elements for the inputs and output. This information is then used to identify if the portType of the selected Web service is matchable for the specific task nodes in the skeleton workflow.

Matching a Web service type to a Task Node

Labeling a task node with a specific service type involves the following five steps:

```
proxy = new UDDIProxy();
proxy.setInquiryURL("http://chlorin.cs.cf.ac.uk:8080/juddi/inquiry");
proxy.setPublishURL("http://chlorin.cs.cf.ac.uk:8080/juddi/publish");
cb = new CategoryBag();
cb.add(new KeyedReference("WSDL type",
                          "portType",
                          "uuid:B824DBE0-015A-11DC-BE3D-E2747DAFDB82"));
TModelList tml = proxy.find_tModel(null, cb, null, null, 10);
```

Figure 5.15: Java code for proxy to access the UDDI Web service repository.

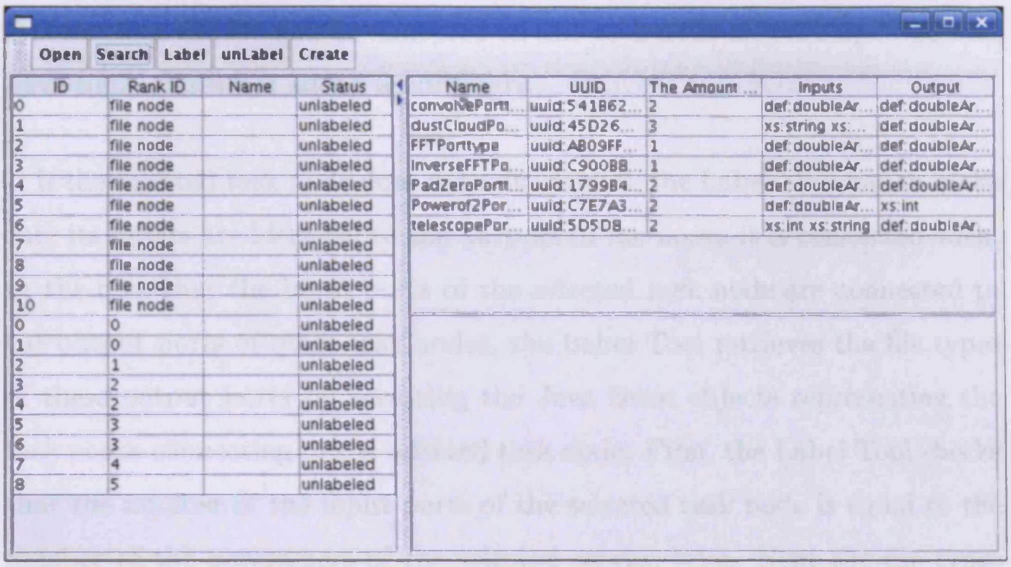
1. Select a task node in the skeleton workflow.

2. Select a service type from the UDDI Web service repository.

3. Check whether the selected task node and service type are matchable (if not, repeat steps 1 and 2 until a match is found).

4. Update the task by adding Web service information, and update all file nodes connected with this task node.

For steps 1 and 2, the user selects the task node and available Web service type from two different tables provided by the Label Tool graphical interface, as shown in Fig. 5.16. In Fig. 5.16, the lefthand table is used to list all nodes in the skeleton workflow. Each has four attributes displayed in the table: its ID, rank, optional name, and status. The first three attributes cannot be modified by the Label Tool, as they are fixed on creation of the skeleton workflow. The status attribute is used to indicate whether this node has been associated with a specific service or file type. It is not necessary for the user to label a file node with specific file type, as it is automatically labeled when the task node it is connected to is labeled. All Web service types in the UDDI repository are displayed in the table on the right of the interface. Each row of this table indicates an available service type, and shows users the name of



ID	Rank ID	Name	Status
0	file node		unlabeled
1	file node		unlabeled
2	file node		unlabeled
3	file node		unlabeled
4	file node		unlabeled
5	file node		unlabeled
6	file node		unlabeled
7	file node		unlabeled
8	file node		unlabeled
9	file node		unlabeled
10	file node		unlabeled
0	0		unlabeled
1	0		unlabeled
2	1		unlabeled
3	2		unlabeled
4	2		unlabeled
5	3		unlabeled
6	3		unlabeled
7	4		unlabeled
8	5		unlabeled

Name	UUID	The Amount ...	Inputs	Output
convolvePortt...	uuid:541B62...	2	def:doubleAr...	def:doubleAr...
dustCloudPo...	uuid:45D26...	3	xs:string xs:...	def:doubleAr...
FFTPorttype	uuid:AB09FF...	1	def:doubleAr...	def:doubleAr...
InverseFFTPo...	uuid:C900BB...	1	def:doubleAr...	def:doubleAr...
PadZeroPortt...	uuid:1799B4...	2	def:doubleAr...	def:doubleAr...
Powerof2Por...	uuid:C7E7A3...	2	def:doubleAr...	xs:int
telescopePor...	uuid:5D5D8...	2	xs:int xs:string	def:doubleAr...

Figure 5.16: The layout of graphical components in the Label Tool.

a service type, a unique identifier (UUID), the number of parameters, and the input and output data types. The number of parameters, and the list of input and output types are retrieved by the Label Tool using the WSDL4J[7] API. For each labeling, the user takes one unlabeled task node from the node table, and then selects one of the available service types from the Web service type table. Once the "Label" button is pressed, the Label Tool checks the rank ID of the selected task node. If the task node has rank 0, all of its input ports are connected to the corresponding file nodes, thus the Label Tool needs to check that the number of input ports of the selected task node is identical to the number of parameters of the service type selected by the user. If they match, the information of this specific Web service type is associated with the selected task node. This means that a Java Bean object representing this specific task node updates its attributes, and adds some new attributes to hold the service type information. Meanwhile one or more Java Bean objects

[7] http://sourceforge.net/projects/wsdl4j/, accessed 12 January 2010.

representing the file nodes connected to this task node also are updated and have some attributes added accordingly.

If the selected task node does not have rank 0, the Label Tool has to make sure its inputs are identical to the outputs of the nodes it is connected with. In the case that the input ports of the selected task node are connected to the output ports of other task nodes, the Label Tool retrieves the file types of these output ports by accessing the Java Bean objects representing the task nodes connecting to the selected task node. First, the Label Tool checks that the number of the input ports of the selected task node is equal to the number of the parameters of the selected service type. Next the file types of the output ports of the task nodes connecting to the selected node are stored into a list in order, and the parameters of the selected service type are also stored in order in another list. Each item in the list of parameters of the service type must be identical to the corresponding entry in the list of connected output ports as this means that the specified service type is matchable to the selected task node.

A more complicated case is where the input data of the selected task node are not only from other task nodes with lower rank, but also from one or more file nodes. The file node of a file node can be determined from the service type of the task node to which it is connected. Therefore, not only does the list save the file types of the output ports of task nodes providing inputs for the selected task node, but also the file types of the file nodes connected to it. The other list does in a similar thing to save the types of the parameters of the specified service type. During comparison of corresponding items in the two lists, any items representing the type of a file node are ignored, as these types can be determined from the selected service type. Fig. 5.17 illustrates the difference when matching inputs that come solely from task nodes, and from a mixture of task and file nodes.

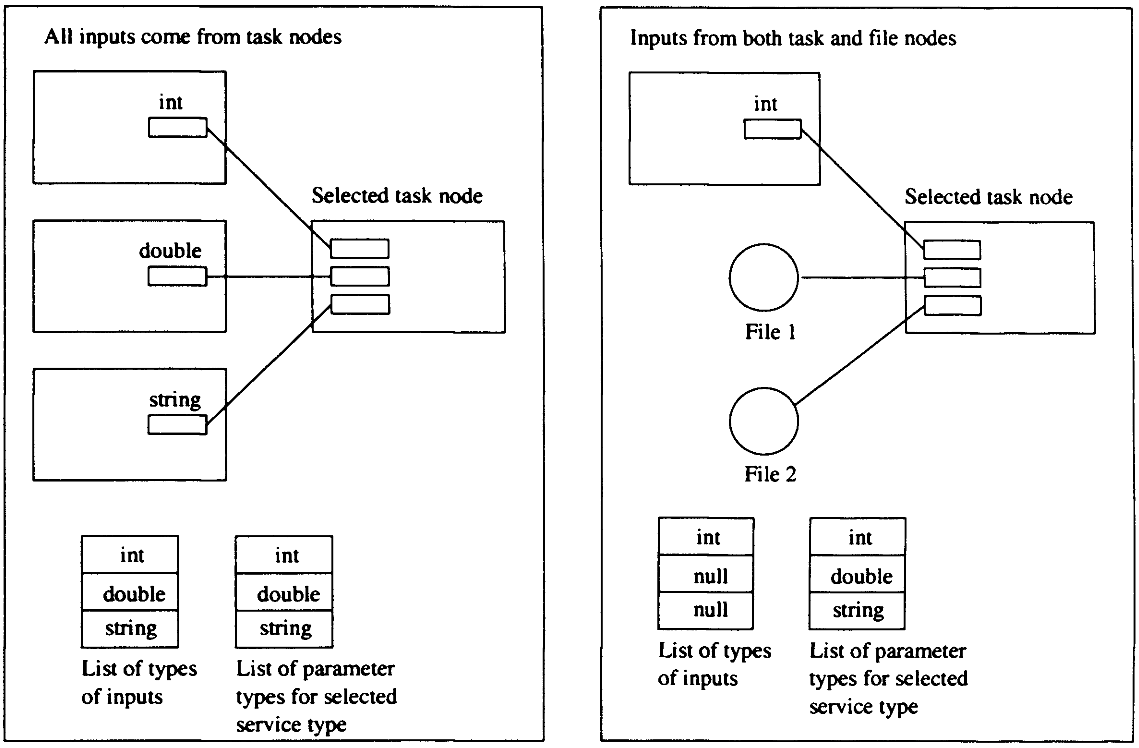


Figure 5.17: Two cases of matching. The lefthand figure showing a match when the inputs of the selected task node all come from other task nodes. The righthand figure shows a match when one input comes from a task node and the other two inputs come from file nodes.

The Label Tool also enables a user to reverse the labeling process by unlabeling a specific service type from a task node. This might be done if the user wants to replace a previously specified service type for a task node by another suitable service type. In the case of a task node with rank 0, once the labeling process is reversed, the Label Tool removes all information describing the service type from the Java Bean object representing the task node, and updates the status attribute to "unlabeled". Any file nodes connected to this task node are refreshed and listed as being available, with no file type associated with it. If labeling is reversed for a task node with rank greater than 0 then the Label Tool only refreshes this task node and all file nodes connected to it to be in the available state.

5.5 XML Schema for Abstract Workflow Description

The Label Tool marshals all Java Bean objects representing nodes labeled with service type or file type information into an XML document that describes the final abstract workflow. The XML schema that defines the description of an abstract workflow imports the XML schema for describing skeleton workflows, as given in Section 4.1.4, and adds definitions for new XML elements to describe the service type or file type information associated with a node. This information is used by the Build Tool to automatically generate the portal and a set of portlets to let the user select the Web service instances. The information describing the service type includes the following:

- Service type ID – the unique UUID registered in the UDDI Web service repository.
- Service type name – the name of the Web service type registered in the UDDI Web service repository
- Input list – the list of parameters for the service type.
- Output list(only one item) – the list of outputs for the service type.

The information describing the file type includes the following:

- File type ID – indicates what type the file is.

- Role – input or output.

- File type description – an optional description of the file type.

The output operation performed by the Label Tool is based on JDOM techniques, which converts the Java Bean objects into a JDOM tree model and then saves it into an XML document with the name specified by the user.

5.6 Chapter Review

This chapter has shown how to build a UDDI Web service repository based on JUDDI, and how to make UDDI queries with the UDDI4J API. This repository maps different elements of WSDL files describing Web services to the corresponding UDDI entities. Registering a Web service in this way allows support for advanced UDDI queries on based on Web services attributes. For example, the user can make a basic query to search for Web services with a specific name or built with the specific network protocol. The query that the Label Tool is most interested in is to find a specific portType with which a set of different Web services comply. The approach taken is quite similar to the object-oriented programming model – a portType is like a common interface, and a set of Web services correspond to different classes implementing this common interface in various ways. The Label Tool only labels the task nodes in a workflow with a specific service type. The particular Web service instance that actually performs the task is selected by the user in the portal application which is generated by the Build Tool, as described in the next chapter. With the Label Tool, a user is able to check whether a service type is matchable to a task node. When the file types required by the input ports of the task node are all the same as the parameter types of the selected service type, a match is achieved, and the service type is allowed to label the task node.

The current version of the Label Tool only allows users to associate service types for which the parameters are primitive data types. Since the Web services are implemented in Java, all the Java primitive data types are supported, such as int, double, long, string, etc. The WSDL schema can support these basic data types by default, and also supports arrays of primitive data types. However, the activeBPEL workflow execution engine used in evaluating the system (see Chapter 7) does not properly support arrays as defined by WSDL. Thus, for example, double arrays in the Web service WSDL files are defined as a customized complex data type with a specific XML schema. Enabling the Label Tool to support multiple customized complex data types can be achieved by expansion of the capacity of the file repository which saves all information about the files accessible by Web services in the service repository. The file repository is implemented as a MySQL relational database. For the current Label Tool, it only saves the file name and its corresponding file type. In order for the Label Tool to support multiple customized complex data types, this repository has to be expanded to save XML schema files for file types. The Label Tool can check whether the service type with complex data is matchable to a specific task node by retrieving the information describing the file type. Another improvement to the Label Tool is to make it possible to connect to multiple UDDI Web service repositories. In a distributed environment, the Web services are probably from different service providers, and each provider may use their own service repositories to save their Web services information. Accessing multiple Web service repositories over HTTP protocol can make the Label Tool more flexible and useful in a distributed environment.

Chapter 6

Build Tool for Automatic Portal Generation

6.1 Overview

The Build Tool is the core of the portal builder and is responsible for the automatic generation of the portal application based on the abstract workflow generated by the Label Tool. In the prototype implementation the portal application is deployed on the GridSphere[1] portal framework and hosted on the Apache Tomcat[2] servlet container. The portal application is a front-end interface accessed by a Web browser, which an end-user can use to create a concrete workflow description file by selecting the Web service instance or the file instance for each node in a specified abstract workflow. The concrete workflow description file is then passed to a Web service which converts it into a form suitable for deployment on a third-party workflow execution engine. In the prototype implementation the concrete workflow is converted

[1] http://www.gridsphere.org, accessed 12 January 2010.

[2] http://tomcat.apache.org/, accessed 12 January 2010.

to a compressed file with the "bpr" extension and this file is deployed onto an ActiveBPEL[3] workflow execution engine. The user can then execute the workflow on the ActiveBPEL workflow engine through the portal application. The concrete workflow generated by the portal application is an XML file that expands the abstract workflow description file with more details on the specific Web service instance for each service type in order for the workflow engine to be able to invoke these Web services.

GridSphere, Tomcat, and ActiveBPEL have been used in building the prototype implementation because they are open-source, freely-available, and are well-supported, stable software. However, the research contributions made in this dissertation do not depend on this software, and other software could be used with only a modest amount of effort. For example, to use a different portal framework would require the Build Tool to produce configuration files for that framework, instead of for GridSphere. Similarly, an alternative workflow execution engine could be used by having a different Web service convert the concrete workflow into the required format for deployment on that workflow engine.

The portal application consists of a set of Java codes, JSP pages, and several XML-based configuration files. The Build Tool uses XSLT [40] to convert the abstract workflow description file to these relevant files. The Web service that converts the concrete workflow to the bpr file also automatically deploys this file to the ActiveBPEL workflow execution engine through the HTTP protocol. The relationship between the Build Tool, conversion Web service and ActiveBPEL workflow engine is shown in Fig. 6.1.

[3] http://www.activebpel.org/, accessed 12 January 2010.

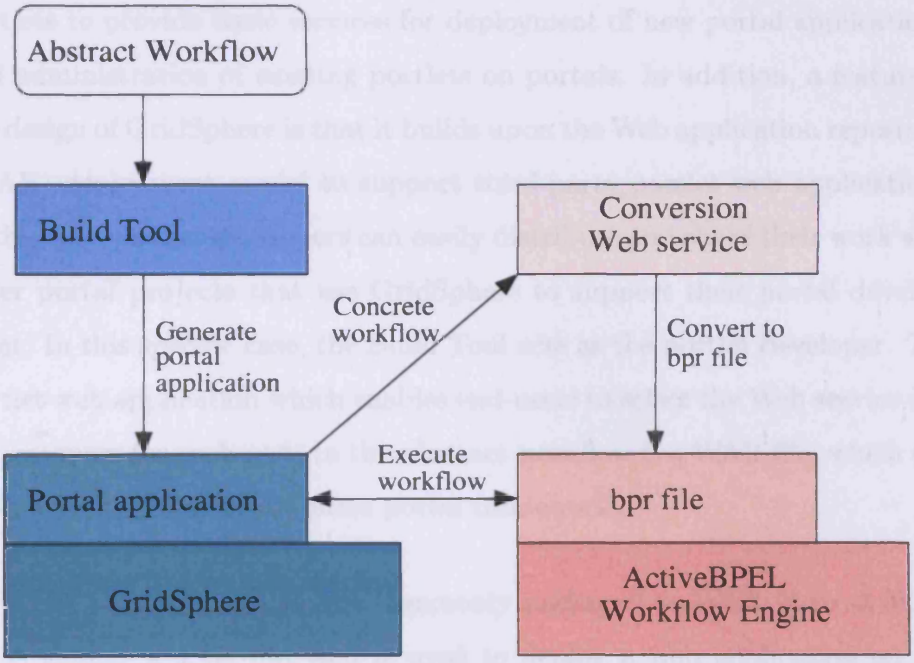


Figure 6.1: The relationship between the Build Tool and other software components.

6.2 The GridSphere Portal Framework

GridSphere is an open-source portal framework for building web-accessible portal applications, and also provides a set of portlet web applications that work seemlessly with the GridSphere framework. Support for the JSR 168 portlet API standard [41] is also provided by GridSphere, which enables it to support the development of reusable portlets. GridSphere includes a set of portlets to provide basic services for deployment of new portal applications, and administration of existing portlets on portals. In addition, a feature of the design of GridSphere is that it builds upon the Web application repository (WAR) deployment model to support third-party portlet web applications. In this way, portlet developers can easily distribute and share their work with other portal projects that use GridSphere to support their portal development. In this specific case, the Build Tool acts as the portlet developer. The portlet web application which enables end-users to select the Web service and file instances for each node in the abstract workflow is a WAR file, which can be deployed on any GridSphere portal framework.

Portlet web applications are commonly packaged as WAR files. A WAR file is similar a JAR file, and is used to deploy a web application on the GridSphere framework with Ant[4]. A WAR file for a portlet web application consists of a set of portlet classes, any required libraries or other resources, a portlet application deployment descriptor, and a web application deployment descriptor. Both of these descriptors are located in the portlet web application's WEB-INF directory. The directory structure of a portlet web application follows the standard web application structure. The root of this directory tree is a directory with the name of the portlet web application.

[4] http://ant.apache.org/, accessed 12 January 2010.

It contains four sub-directories: html and jsp directories contain the pages for each portlet; the META-INF directory has a file named MANIFEST.MF which stores the basic information about this portlet web application; and, the WEB-INF directory contains all the portlet classes, the required libraries, and the portlet configuration files.

6.3 The ActiveBPEL Workflow Execution Engine

ActiveBPEL is an open-source implementation of the Business Process Execution Language (BPEL) standard. In the prototype system, ActiveBPEL is the workflow execution engine that runs the workflow in BPEL format that is created from a concrete workflow by the conversion Web service. BPEL is an XML language for describing business processes based on Web services. BPEL can perform functional tasks by invoking Web services. Its notation includes flow control, variables, concurrent execution, input and output, and error handling. A process described by BPEL could be an abstract process or an executable process. An abstract process, like a library API, describes what the process does, and what are the input and output, but does not represent what is actually done. Abstract processes are useful for developing a business process for a third party who wants to use the process. The executable process is responsible for the workflow execution. It performs the invocation of each Web service and the IO operations, and enables all Web services to communicate with each other by passing data among the different Web services. BPEL is laid on top of other Web technologies, such as WSDL, XML schema, and XPath. A complete process is not described by only one file - a BPEL file describes a process, but it does not stand alone. A BPEL process refers to Web services, which are all described by WSDL files. It refers to partners described by partner files, which in turn reference WSDL files that describe the partner link types.

A typical BPEL process file is a compressed file with "bpr" as its extension. It contains three sub-directories: META-INF, wsdl and bpel. The META-INF directory has a catalog.xml file which is used to tell ActiveBPEL where to access the WSDL files and their XML schema files. All WSDL files referred to by the BPEL process are located in the wsdl directory. The bpel directory stores all the BPEL files describing the processes. The bpr file also contains a process deployment descriptor file with a pdd extension, that describes the relationship between the partner links defined in the BPEL files and the implementation required to interact with the actual partner endpoints.

6.4 XSLT: eXtensible Stylesheet Language Transformation

XSLT is designed for transforming XML files into any other text format, such as HTML, XML, and even Java source code. Although Java can do XML data transformation with SAX, DOM or JDOM without XSLT, in terms of complexity and sophistication, XSLT is easier to use than Java. In the Build Tool, XSLT is used to generate all files for the portlet web application. It has to generate JSP pages, Java code for each portlet, and a set of XML configuration files.

An XSLT process is an application that can be implemented by multiple programming languages. It applies an XSLT stylesheet to XML data. An

XSLT stylesheet works as a list of instructions to tell the processor how to transform the XML data source. The result generated by the XSLT processor is called a result tree, which could be a static file or an output stream. The input XML document and the associated XSLT stylesheet are separate files. Due to this feature of XSLT, it is possible to write several different stylesheets that convert the same XML into different output data formats.

6.5 The Structure of the Portlet Web Application

The portlet web application generated by the Build Tool is used to generate a concrete workflow by selecting a Web service or file instance for each node, deploying the workflow on the remote ActiveBPEL workflow execution engine, and running it. The Build Tool generates different portlet web applications based on the structure of the abstract workflow. The portlet web application provides the following four types of portlets for users to carry out the generation, deployment and execution of the workflow:

1. One or more file instance selection portlets. This type of portlet is used to associate a file node with a data file instance that complies with the file type that labels the node.

2. One or more Web service instance selection portlets. This type of portlet is used to associate a task node with a Web service instance that complies with the service type that labels the node.

3. A workflow generation and deployment portlet. This portlet submits the concrete workflow to the conversion Web service to produce the

deployment files (compressed as a bpr file), which are then used to deploy the workflow onto a third-party workflow execution engine.

4. A workflow execution portlet.

As an illustrative example, consider how the Build Tool generates the portal application for the abstract workflow shown in Fig. 6.2. This workflow has five input file nodes and one output file node. Thus, there are five file instance selection portlets in the corresponding portal application. Furthermore, the workflow has nine task nodes, so there are nine Web service instance selection portlets. There are also one generation and deployment portlet, and one execution portlet in every portlet web application. The information about file and service instances provided by the user through the file instance selection portlets and Web service instance selection portlets is transferred to the generation and deployment portlet by portlet session mechanisms[42]. This is implemented through the Portlet Session interface of the portlet API, and provides a simple way to identify a user across more than one request and to store transient information about that user. Thus, user information can be stored as s/he moves from one portlet to another in a session. The generation and deployment portlet invokes the conversion Web service and passes to it all the information gathered from the selection portlets. This Web service generates the concrete workflow, transforms it to a set of ActiveBPEL workflow script files, and then compresses these files into a bpr file. This bpr file is then deployed on a specified ActiveBPEL workflow execution engine. The generation and deployment portlet is notified of the response from the conversion Web service. If the deployment of the workflow on the ActiveBPEL workflow execution engine succeeds, the execution portlet will get a message from the generation and deployment portlet through the portlet session. The user can then execute the workflow by using the execution portlet, and the result data after the execution of the workflow is automatically stored in the specified output file location.

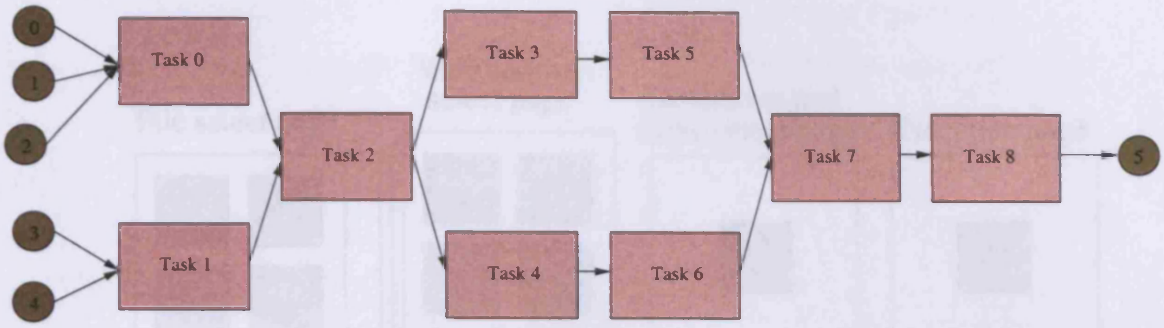


Figure 6.2: A more complicated workflow.

The GridSphere portal framework is a portlet container that manages the portlets at runtime, and complies with the JSR 168 portlet API. The container processes the portlets into fragments. Then it hands each fragment to the portal server that aggregates them into the portal pages. Figure 6.3 demonstrates the chain of events that occurs inside the GridSphere portal. As shown in Fig. 6.3, each workflow portal application has four types of portal pages: a file instance selection page, a Web service instance selection page, a generation and deployment page, and an execution page. The portlets which perform the file selection are aggregated onto the file selection page. There are one or more Web service instance selection pages, the number depending on the largest rank of the task nodes in the abstract workflow. The portlets for all the task nodes with the same rank are aggregated into a single Web service selection page. The last two pages contain one portlet for workflow generation and deployment, and one portlet for workflow execution. Each of these two pages can communicate with each other by sending and receiving the portlet session within the scope of the portal application.

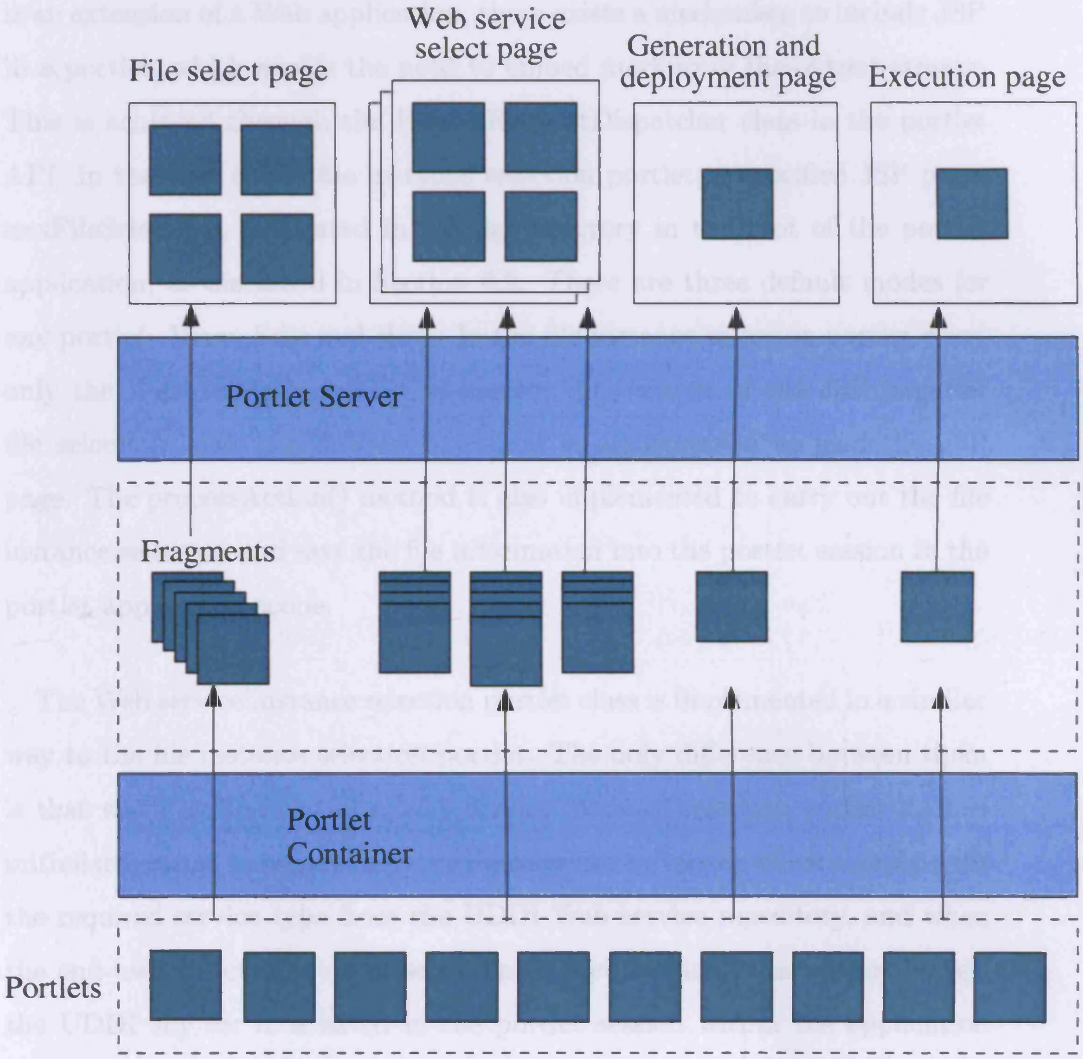


Figure 6.3: The structure of the portal application.

Each file instance selection portlet has a portlet class file and its related Java Server Pages (JSP) file for providing the user with an interface to select one file instance from the multiple candidates. Since a portlet web application is an extension of a Web application, there exists a mechanism to include JSP in a portlet, which avoids the need to embed markup in the output stream. This is achieved through the PortletRequestDispatcher class in the portlet API. In the case of the file instance selection portlet, a specified JSP page, xxxFileSelect.jsp, is located in the jsp directory in the root of the portlet application, as discussed in Section 6.2. There are three default modes for any portlet: View, Edit and Help. In the file instance selection portlet class, only the View mode is needed to present the content of the JSP page for file selection, thus the doView() method is implemented to load the JSP page. The processAction() method is also implemented to carry out the file instance selection and save the file information into the portlet session in the portlet application scope.

The Web service instance selection portlet class is implemented in a similar way to the file instance selection portlet. The only difference between them is that when an instance of a Web service instance selection portlet class is initialized, it has to retrieve a list of Web service instances which comply with the required service type from the UDDI Web service repository, and when the end-user selects one of these available Web service instance candidates, the UDDI key for it is saved in the portlet session within the application scope. The objects stored in a portlet session can be referenced by their attribute names across different portlets in the portal application.

The portlet performing the generation and deployment of the workflow on the workflow execution engine is different from the file and Web service selection portlets, and carries out two tasks. The first task is to retrieve all the information about the file and Web service instances specified by the selection portlets from the portlet session. The second task for this portlet is to invoke a conversion Web service to generate the workflow deployment files, and to deploy these onto a workflow execution engine. The portlet also includes a JSP page to display any necessary messages, such as information on the selected Web services and files, and the status of generation and deployment.

The final JSP page includes only one portlet for workflow execution. It is assumed that a workflow has been successfully deployed on the workflow engine and the generation and deployment portlet has passed the URL of the WSDL file of the workflow (ActiveBPEL considers any deployed workflows as Web services) to the execution portlet through the portlet session. The execution portlet enables users to invoke a workflow through its WSDL file by a mechanism known as *dynamic invocation.* There are many ways to invoke a Web service from a portlet, and most of them are much easier than using dynamic invocation of a Web service through its WSDL file. However, in this case, the Web service representing the workflow cannot be determined at design time as it is generated by the conversion Web service and deployed onto the workflow execution engine automatically. Therefore, the client code to invoke the Web service to execute the workflow is not generated at design time. Thus, after the WSDL representing the workflow is generated by the ActiveBPEL workflow execution engine, the execution portlet executes this workflow by dynamical invocation of its associated Web service at runtime.

6.6 Build Tool Design

6.6.1 The Structure of the Build Tool

The generation of a portal application based on a specified abstract workflow is divided into eight tasks, each of which is performed by a different component in the Build Tool. Figure 6.4 shows the whole procedure of the generation of a portlet application. The first task the Build Tool performs

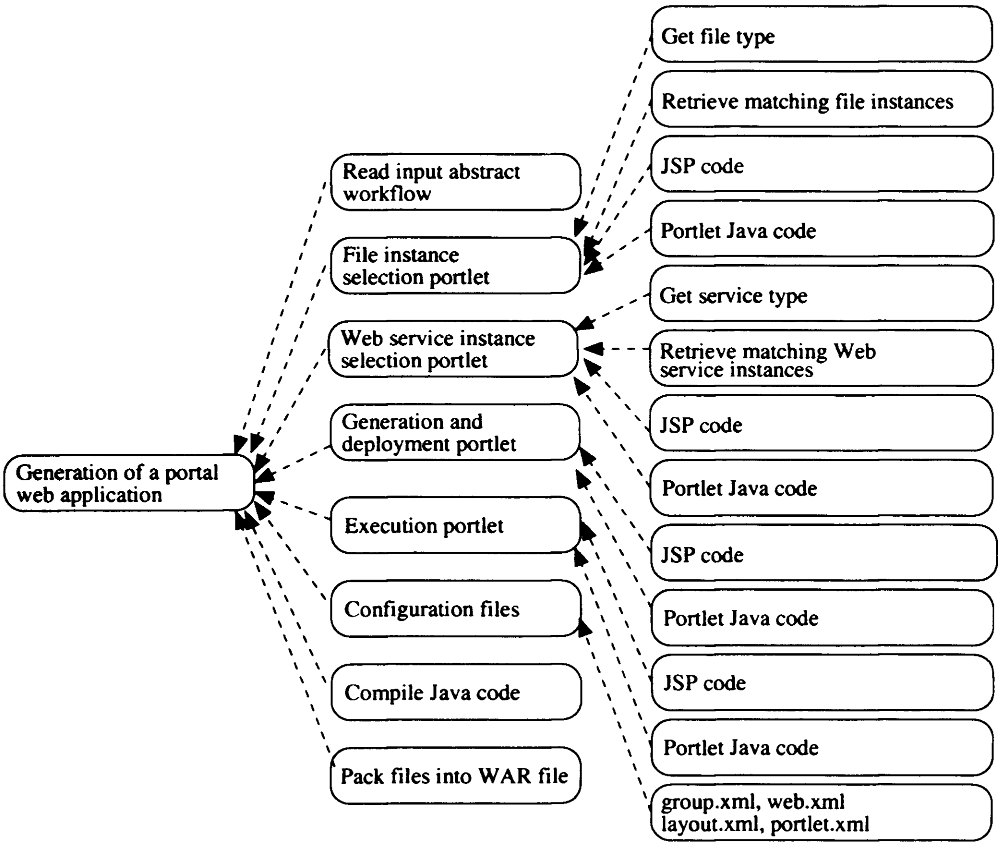


Figure 6.4: The Build Tool user case diagram.

is to read the XML document representing the abstract workflow. Then the XSLT processor transforms each task node and file node to create the corresponding portlet. In turn, each of these transformations is split into two steps: one to generate the portlet Java code, and the other to generate the JSP page. After all the portlets required by the portal application have been generated, the Build Tool then generates a set of configuration files: group.xml, layout.xml, web.xml and portlet.xml. Finally, all the files generated for the portlet application are compressed into a WAR file, which can be deployed on the Tomcat web server.

The Build Tool converts an abstract workflow description file, such as generated by the Label Tool, into a customized portal web application which provides a user-friendly web interface so that users can associate specific file and Web service instances with the nodes of the abstract workflow according to the file type and service type recorded in the abstract workflow description file. The Build Tool is made up of several components that work cooperatively to achieve the generation of the portal web application. Figure 6.5 illustrates the components of the Build Tool.

As illustrated in Fig. 6.5, the Build Tool has an embedded XSLT processor. Four types of XSLT stylesheets are attached to the XSLT processor, which provide a set of instructions to the processor to perform transformations to create the Java code and JSP pages for portlets, and portal configuration files. Since each portlet in the portal application is a Java class extending the "GenericPortlet" class, the XSLT processor embedded in the Build Tool applies an XSLT stylesheet to the abstract workflow description file to generate the Java portlet code. The number of Java files generated by the Build Tool depends on how many task and file nodes there are in the abstract workflow. Also for each portlet, the XSLT processor also applies an XSLT stylesheet to the abstract workflow description file to generate a set of JSP pages. The other XSLT stylesheet applied to the abstract workflow

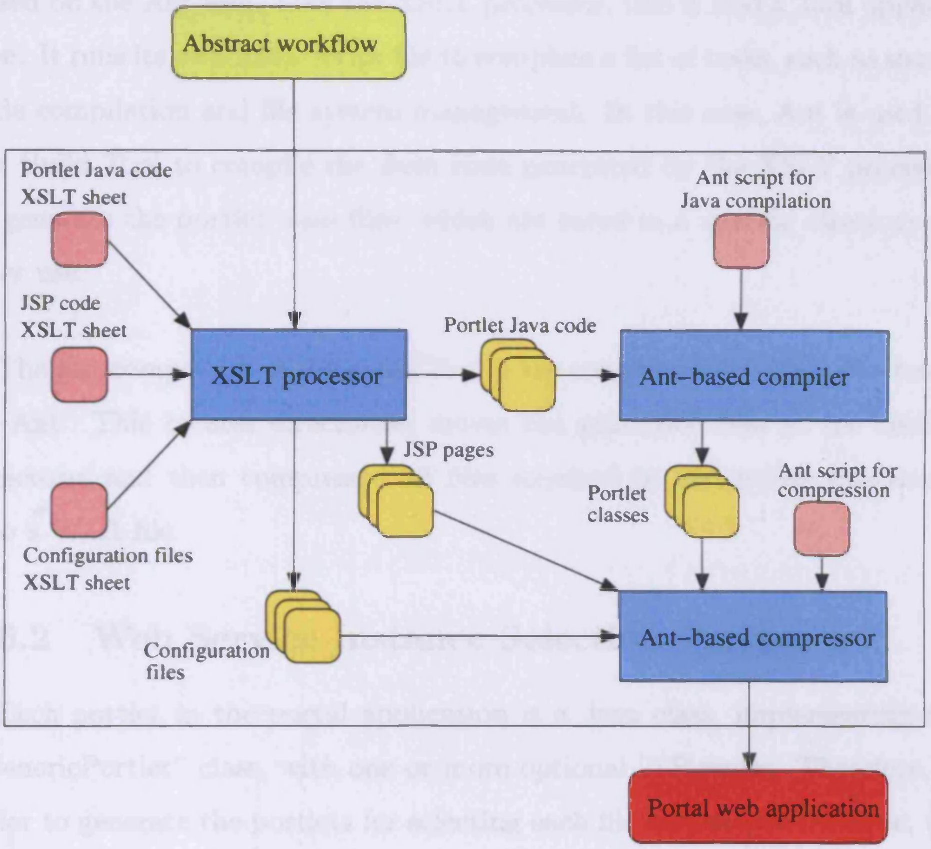


Figure 6.5: The components of the Build Tool.

description file generates the portal application configuration files, including the layout of portlets on the portal page and the list of portlets registered in the portal application.

Another important component in the Build Tool is the compiler script based on the Ant tool. Like the XSLT processor, this is also a Java application. It runs its own XML script file to complete a list of tasks, such as source code compilation and file system management. In this case, Ant is used by the Build Tool to compile the Java code generated by the XSLT processor to generate the portlet class files, which are saved in a specific directory for later use.

The last component of the Build Tool is the compressor which is also based on Ant. This creates directories, moves the generated files to the correct directory, and then compresses all files required by the portal framework into a WAR file.

6.6.2 Web Service Instance Selection Portlet

Each portlet in the portal application is a Java class, implementing the "GenericPortlet" class, with one or more optional JSP pages. Therefore, in order to generate the portlets for selecting each file and service instance, the Build Tool provides two separate XSLT stylesheets for generating Java code and JSP pages from the abstract workflow description file. Each task node element in the abstract workflow description file is transformed to a piece of Java code for the corresponding Web service instance selection portlet. A part of an abstract workflow description file is shown in Fig. 6.6. This element contains sufficient information for the XSLT process to transform it into the Java code for the corresponding portlet. The Java code generated by the embedded XSLT processor is shown in Fig. 6.7. In this Java code, all the required libraries are first imported. Since the portlet inherits from GenericPortlet, this code has to overwrite the "init" method. The information in the Java code includes the UUID for the service type, the namespace and access URL for the WSDL file of the service type, and the parameters required by the service type. The UUID of the service type is used to retrieve a list of Web service instances from the UDDI Web service repository in the "init" method. The name of the service type is used to name the corresponding Java code. The information about the inputs to this Web service type is used in the file instance selection portlet if this task has rank 0.

```
<task ID="0" rank="0" name="dustCloudPortType" type="normal">
   <ServiceType ServiceTypeID="uuid:45D265E0-4F65-11DC-A5E0-A5F21AD7226F"
             ServiceTypeName="dustCloudPortType">
      <ServiceTypeDescription>dustCloudPortType</ServiceTypeDescription>
      <namespace>http://dlu1.FFTWebservice1.dustCloud1/definitions</namespace>
      <urlLocation>
         http://chlorin.cs.cf.ac.uk:8080/axis/new1wsdl/dustCloudPortType1.wsdl
      </urlLocation>
      <portType name="DustCloudService">
         <operation id="0" name="yValues">
            <input type="dustCloudInput" name="dustCloudInput">
               <part name="in0" type="xs:string" />
               <part name="in1" type="xs:double" />
               <part name="in2" type="xs:int" />
            </input>
            <output name="output" type="dustCloudOutput">
               <part name="out" type="def:doubleArray" />
            </output>
         </operation>
      </portType>
   </ServiceType>
</task>
```

Figure 6.6: A task with rank 0 in an abstract workflow description file.

```
public class dustCloudPortType0Portlet extends GenericPortlet{
// too long to quote all content of the file
    public void init(PortletConfig config)throws UnavailableException, PortletException{
        super.init(config);
        System.setProperty(TransportFactory.PROPERTY_NAME,
                         "org.uddi4j.transport.ApacheAxisTransport");
        proxy=new UDDIProxy();
        try{
            proxy.setInquiryURL("http://chlorin.cs.cf.ac.uk:8080/juddi/inquiry");
            proxy.setPublishURL("http://chlorin.cs.cf.ac.uk:8080/juddi/publish");
            businessList=proxy.find_business("pupa",null,0);
            Vector businessInfoVector =
                         businessList.getBusinessInfos().getBusinessInfoVector();
            TModelBag tb=new TModelBag();
            uuid="uuid:45D265E0-4F65-11DC-A5E0-A5F21AD7226F";
            tmk=new TModelKey(uuid);
            tmn="dustCloudPortType0";
            tb.add(tmk);
            servicelist =
                proxy.find_service("03927110-4F02-11DC-B110-C30F49B92B9C",tb,null,10);
        }catch(MalformedURLException mue){
        }catch(UDDIException ue){
        }catch(TransportException te){
        }
    }
    public void doView(RenderRequest request,
                       RenderResponse response)throws PortletException, IOException{
        PortletURL url=response.createActionURL();
        response.setContentType("text/html");
        request.setAttribute("serviceType",uuid);
        request.setAttribute("servicelist",servicelist);
        request.setAttribute("url",url.toString());
        PortletContext portletContext=getPortletContext();
        PortletRequestDispatcher prd =
             portletContext.getRequestDispatcher(serviceSelecJsp);
        prd.include(request,response);
    }
    public void processAction(ActionRequest request, ActionResponse response){
        dustCloudPortType0Session=request.getPortletSession();
        dustCloudPortType0Session.setAttribute("dustCloudPortType0service",
            (String)request.getParameter("services"),PortletSession.APPLICATION_SCOPE);
    }
    public void destroy(){
        dustCloudPortType0Session=null;
    }
}
```

Figure 6.7: The Java code for the task element in Fig. 6.6.

The JSP page is responsible for providing a user-friendly interface to interact with the portlet application. Although its generation does not require much information about the service type from the abstract workflow description file, the XSLT processor does need to get the name of the service type from the file.

6.6.3 File instance selection portlet and JSP page

The file instance selection portlet is used to specify a file instance of the file type associated with a node. For example, the file containing the input data for the execution of the workflow, which is passed to a task node with rank 0. Thus, the embedded XSLT processor generates the Java code for a file instance selection portlet and its related JSP page by transforming task elements with rank 0. During generation of the Java code for selecting a file, the UUID in the source file is needed in the "init" method. This is needed in order to query the file repository based on the file type. The JSP page needs the name of the service type represented by the task element to indicate which Web service the file is linked to.

6.6.4 Generation and deployment portlet and JSP page

A complete portal application also includes a set of configuration files. These let the portal server know which portlets are registered in the portal application, and how they are laid out on the various portal pages. These configuration files will now be outlined.

group.xml

The group.xml file is a configuration file that defines a GridSphere group, and lists all portlet classes registered in this group. Grouping a set of portlet classes makes it easy to manage and maintain them. The embedded XSLT processor sets a name for this group according to the naming convention by which the name of this group depends on the name of the abstract workflow file. The next step is to register all portlet classes into this group.

portlet.xml

The portlet.xml deployment descriptor has placeholders for the custom portlet modes, custom windows states, user attributes, and security constraints. The XSLT stylesheet applied to the abstract workflow description file instructs the XSLT processor to create one placeholder for each task node in the file, and to add two more placeholders for the generation and deployment portlet, and the execution portlet.

layout.xml

The layout.xml file is used to tell the GridSphere portal framework how to lay out the portlets on the portal pages. In this case, all file instance selection portlets are aggregated onto the first portal page called the fileSelection page. The Web service instance selection portlets, representing task nodes with the same rank, are all aggregated onto the same page. The last two portal pages are for the generation and deployment portlet, and the execution portlet.

web.xml

The web.xml file contains basic information about the portal application, such as the name of the portal application. It also sets up several <servlet-

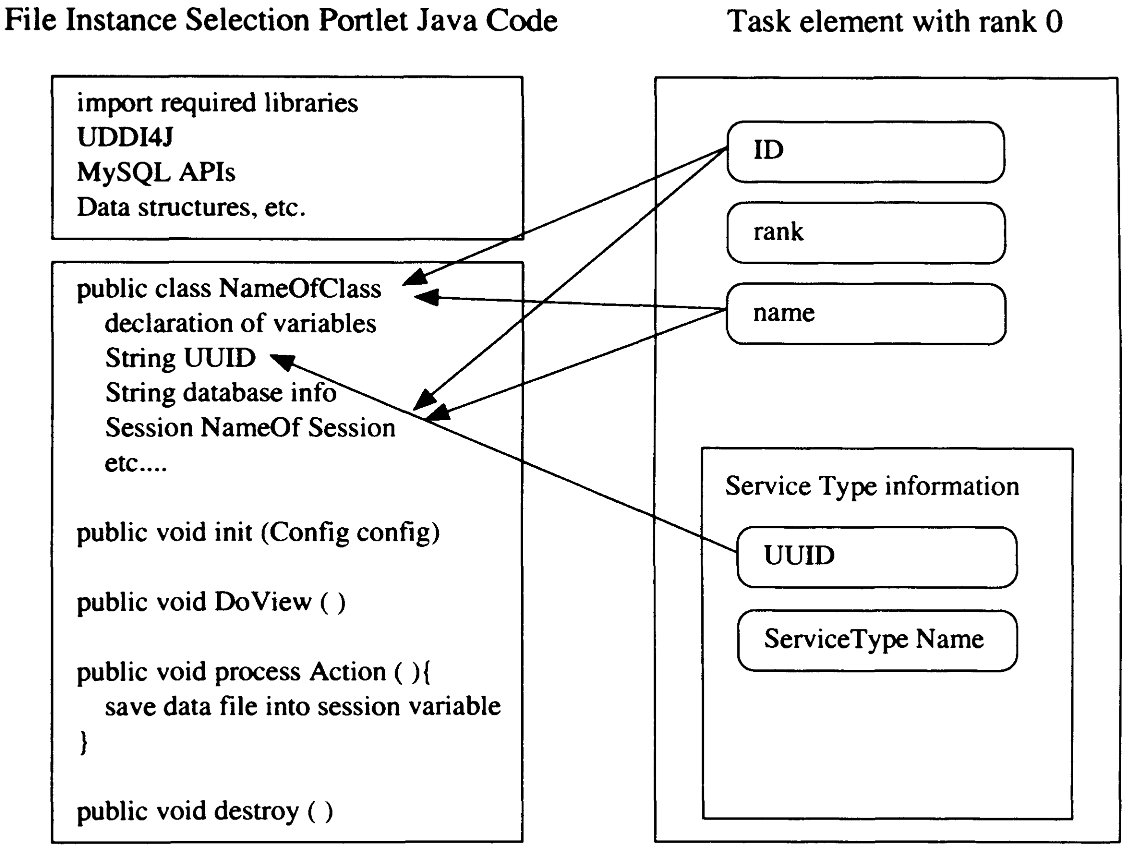
mapping> elements, whereby the GridSphere portal framework can communicate with the portal application. The XSLT stylesheet applied to the abstract workflow description file instructs the XSLT processor to create a web.xml file for the portal application. In web.xml, the name attribute for the portal application is the same as the name of the corresponding abstract workflow description file, followed by "Portal".

6.7 Build Tool Implementation

When the XSLT processor embedded in the Build Tool begins the XML document transformation, it looks in the XSLT stylesheets for templates to instruct it how to perform the transformation of the abstract workflow description file. As discussed previously, the Build Tool provides several XSLT stylesheets to the embedded XSLT processor to apply to the source data file to perform the transformation.

6.7.1 Implementation of file selection XSLT stylesheet

Before looking at the design of the file selection XSLT stylesheet, it is useful to examine the structure of the Java code for the file instance selection portlet, and to see which part of this code is determined by the information stored in the abstract workflow description file. This completely determines how to design the XSLT stylesheet to make the XML transformation simple and efficient. Figure 6.8 illustrates the relationship between the Java code for a fileSelect portlet and the related task node with rank 0. As illustrated in Fig. 6.8, the Java code for a file instance selection portlet Java code needs to import a set of Java libraries, including the portlet library and database APIs, to enable it to make queries to the relational database, and a data structure



```
import required libraries
UDDI4J
MySQL APIs
Data structures, etc.

public class NameOfClass
   declaration of variables
   String UUID
   String database info
   Session NameOf Session
   etc....

public void init (Config config)

public void DoView ( )

public void process Action ( ){
   save data file into session variable
}

public void destroy ( )
```

Figure 6.8: The mapping of task node to file instance selection portlet JAVA code.

for data storage. These Java libraries are independent of the content of abstract workflow description file,.so the Java code for every file instance selection portlet imports the same Java libraries. The name of the Java class must be unique for each portlet Java code and related with the corresponding task node in the abstract workflow description file, so a naming convention is adopted that sets the name of the Java class of a file selection portlet to be a merger of the name attribute of the task element and its ID attribute. This naming convention guarantees the uniqueness of the name of the Java portlet class. In the global variable declaration, the UUID value of the service type element nested in the task element is used to assign a String variable called UUID. Finally, there is a Session variable which is used to store information describing the file selected by the user and pass it to the portlet class. The name of the Session also needs to distinct from others. Therefore, it is named in the same way as the portlet, the only difference being that the name of the session variable is followed by "Session" to distinguish it from the name of the portlet class.

The portlet class also displays information to the user by means of a JSP page. The name of this JSP page also has to be distinct from any others. As in naming the session variable, the name of the JSP page for each portlet is a combination of the name of the task and its ID, ending with ".jsp". For example, suppose there is a task node with name attribute "task" and ID 0. According to the naming convention described above, the name of the portlet class corresponding to this task node is "Task0" and the name of its session variable is "task0Session". The name of the JSP page is "task0.jsp".

The rest of the content of the portlet class does not depend on the information stored in the abstract workflow description file. In the init(Config

config) method, the connection to the file repository hosted on a relational database system is set up with a fixed user name and password. A query to retrieve all files with the specified value of the UUID variable is made, and the result is stored in a Vector data structure. The next method overridden by this portlet class is called doView. The doView method is used in the view mode to display the current content of the portlet and allow interaction with users. In doView, the PortletRequestDispatcher is used to include the output of the corresponding JSP page. Before including the JSP page, the list of file candidates retrieved by the init() method and the URL to the processAction() method are set in the request object. The include() method of the PortletRequestDispatcher forwards both request and response objects to the JSP page included in the portlet. The processAction() method is overridden to save the selected file into the portlet's session by accessing the request object forwarded by the portlet's JSP page.

As discussed above, not all the content of the portlet class depends on the information stored in the abstract workflow description file, such as setting up the database connection. This requires a template which can be instantiated on demand, rather than only when a given pattern is encountered in the abstract workflow data tree, and these templates in the XSLT stylesheet for generating file selection portlets do not match any node pattern. These templates can be uniquely named through a name attribute. The call-template instruction element has one required attribute, also called name. This instruction element is used to call the named template whenever it is required with the name attribute. In the file selection portlet transformation, five named templates are defined to generate different parts of the portlet Java code. The first named template is the importLibraries template, and as its name implies, this template is responsible for generating the Java code for importing Java libraries.

```
<xsl:template name="importLibraries">
```

```
            <xsl:text>//import the package for the portlet</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.portlet.GenericPortlet;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.portlet.PortletException;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.portlet.RenderRequest;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.portlet.RenderResponse;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
</xsl:template>
```

The above piece of code is quoted from the XSLT stylesheet for generating the file selection portlet. In this code, the template named importLibraries contains several text elements. Although literal text output is also acceptable by the XSLT processor, in order to have more control over the resulting text output, the element "text" is used to write literal text to the output. Thus, the Java code for importing Java libraries is output using text elements. To make the portlet code more readable a new line character is added between lines of code. This is achieved by setting the disable-output-escaping attribute as "yes" and outputting the "&#10;" character using a text element.

The second named template declares the variable names and assigns an appropriate value to each variable. The list of variable names is as follows:

- UDDIProxy – represents a UDDI server and the actions that can be invoked on it.
- BusinessList – represents a list containing a set of business entities.
- ServiceList – represents a list containing a set of service entities.

- CategoryBag- - represents a categoryBag element which contains 0 or more keyed references.
- TModelDetail – represents a tModelDetail element.
- TModel – represents a tModel element.
- TModelKey – represents a tModelKey.

The information for the connection to the file repository is as follows:

- jdbc:mysql: the protocol to connect to the file repository.
- localhost: the name of the database server hosting the file repository.
- root: the username of file repository.
- 123456: the password to access the repository.

Like the importLibraries template, the template with name "declareVariable" also uses text elements to write literal text, including variable names and their values, to the output. In this template, a number of variable elements are specified through the XSLT stylesheet. Variable elements defined at the top level of the XSLT file are used to name the path of the JSP page and to store the UUID. The syntax of the variable element is as follow:

```
<xsl:variable name="variableName" select="expression"/>
```

Because the attribute "select" contains an expression the value of the variable can be specified by a location path to a node of the source document tree (i.e., the abstract workflow description file). Consider the following fragment of the source document:

```
<task ID="0" rank="0" name="dustCloudPortType" >
<ServiceType ServiceTypeID="uuid:45D265E0-4F65-11DC-A5E0-A5F21AD7226F"
ServiceTypeName="dustCloudPortType" >
```

The UUID and portlet name variables are then set by the following XSLT instructions:

```
<xsl:variable name="uuid" select="ServiceType/@ServiceTypeID"/>
<xsl:variable name="ID" select="@ID"/>
<xsl:variable name="nameOfPortlet" select="concat(@name,$ID)"/>
```

In the XPath expression in the select attribute, the root is the current task node, thus "@ID" means the path to the ID attribute of the current task element, and "ServiceType/@ServiceTypeID" means the path to the ServiceTypeID attribute of the ServiceType element, which is the child element of the current task element in the XML document tree structure. In the portlet name specification, the concat function is used in the selectattribute to concatenate the name attrribute and the ID variable. The rest of the named templates perform their processing in a similar way and so will not be described in detail. They are are responsible for generating the Java code for the init(), doView(), and processAction() methods. In the doView() method XSLT text elements are used to write literal Java code to include the specific JSP page with the PortletRequestDispatcher, and put a list of file candidates and a URL to invoke a processAction() method into the request object, which is forwarded to the included JSP page. The processAction() method puts the information about the file selected by the user into the corresponding JSP page in the portlet session.

6.7.2 Implementation of the file selection JSP page stylesheet

The JSP page included by the file selection portlet provides a user-friendly interface to select one of the file candidates as a workflow input. Information about the selected file is saved in the request object of the JSP page and forwarded to the portlet's processAction() method, which puts the information into the portlet session in the portlet application scope. Like the XSLT stylesheet for file portlet transformation, the file selection JSP XSLT stylesheet also defines two named templates for generating the header and body of the JSP page. The JSP page is an HTML file containing Java code. In the header of the JSP page, <xsl:text> is used to write the necessary Java libraries required in this JSP page, and to retrieve the request object from the portlet class and the URL to the processAction() method defined in the portlet class. In the body of the JSP page, the form tag is provided by <xsl:text> to enable the user to select one data file out of the list of file candidates. The value of the action attribute in the form tag is assigned to the URL retrieved from the portlet class in the header part. The selected file is sent to the processAction() method in the portlet class, which stores it in the portlet session.

6.7.3 Implementation of the service selection stylesheet

Like the file selection XSLT stylesheet, the Web service selection stylesheet has the same five named templates: importLibraries, declareVariable, init, doView, processAction. In addition, one more named template called destroy is also added to the XSLT stylesheet. This named template is used to generate Java code to delete the session object at the end of the portlet life cycle. In the Web service selection portlet, the init() method is used to connect to the Web service repository to retrieve the Web service instances with the UUID defined in the global variable. A connection is made with the JUDDI-based Web service repository in the init() method as follows:

```
proxy.setInquiryURL("http://chlorin.cs.cf.ac.uk:8080/juddi/inquiry");
proxy.setPublishURL("http://chlorin.cs.cf.ac.uk:8080/juddi/publish");
```

This is different from the file selection portlet code. It sets up the connection to the JUDDI server through a URL in order to make UDDI queries to the JUDDI-based server. Thus, in the service instance selection XSLT stylesheet, <xsl:text> elements is used to generate Java code for connection to the JUDDI server in the init() method. The rest of the methods in this portlet code are quite similar to those of the file selection portlet code.

6.7.4 Implementation of service selection JSP page stylesheet

The JSP page included by the service instance portlet provides a user-friendly interface to select one Web service instance to be associated with a task node in the abstract workflow. Information about the Web service instance selected by the user is saved in the request object of the JSP page and forwarded to the portlet's processAction() method. This method, like the one in the file selection portlet, saves this information into the portlet session. The JSP page also has two parts: header and body. <xsl:text> elements are used to write the Java code to import the necessary Java libraries and request object from the portlet class in the header, and provides a form to enable users to select a Web service instance out of the available Web service instance candidates with the same UUID as that defined in the UUID variable.

6.7.5 Implementation of the workflow generation and deployment stylesheet

Unlike the Web service selection portlets, which depend on the number of task nodes in the abstract workflow description file, there is only one generation and deployment portlet in the portal Web application. This portlet performs three tasks:

1. Retrieve the information stored in different sessions from file selection and Web service instance selection portlets.

2. Expand the abstract workflow description file to be a concrete workflow description file with information retrieved from sessions, and then convert the concrete workflow to ActiveBPEL scripts by invoking the conversion Web service.

3. The conversion Web service compresses the ActiveBEPL scripts into a file with bpr extension name and deploys it on the specified ActiveBPEL workflow execution engine.

Thus, in the "importLibraries" named template, <xsl:text> elements to write a piece of Java code to import the Web service client side classes, which are invoked by this portlet to generate the ActiveBPEL script.

```
<xsl:text>import dashanlu.a2bpel.A2BpelPT;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import dashanlu.a2bpel.A2BpelServiceLocator;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
```

These four <xsl:text> elements are used to import two Java classes to invoke a Web service. These classes are used in the processAction() method in the portlet implementation. Furthermore, since the names of sessions for the Web service selection portlet all comply with the naming convention described in Section 6.7.1, this XSLT stylesheet knows the names of sessions for different Web service selection portlets and can assign values stored in the sessions to other variables at design time[5]. The number of sessions depends on the number of task nodes in the abstract workflow description file. In the processAction() method implementation, <xsl:text> elements are used

[5] In this context, "design time" means the time at which the XSLT processor generates the portlet source code.

to write a try-catch structure to deal with any exceptions occurring in the Web service invocation. If an exception occurs during the generation of the ActiveBPEL workflow script or its deployment, the error message is saved into a string variable and displayed on the JSP page included by this portlet.

6.7.6 Implementation of the workflow generation and deployment JSP page stylesheet

The JSP page included by this portlet is responsible for displaying the selected file and Web service instances for each node in the abstract workflow saved in the corresponding session, and also provides a URL to the processAction() method defined in the portlet class. This is used to invoke the conversion Web service to generate and deploy the ActiveBPEL workflow script on the workflow execution engine with information retrieved from the sessions.

6.7.7 Implementation of the workflow execution stylesheet

The last portlet for a workflow's portal application is the execution portlet, which is used to execute the concrete ActiveBPEL workflow produced by the conversion Web service, and deployed on the ActiveBPEL workflow execution engine. The workflow deployed on the ActiveBPEL engine is considered to be a special Web service that consists of a set of sub-services. In the implementation of the execution portlet, the XSLT stylesheet uses <xsl:text> elements to output the Java code that invokes the Web service that executes the workflow deployed on the ActiveBPEL execution engine. A problem with the invocation of this execution Web service, which is going to be invoked by the execution portlet, is that it is not created and deployed when the portlet is generated by the Build Tool – it is created later by the conversion Web service which the user invokes through the generation and deployment portlet. This means that at the time that the execution portlet is generated it is not possible to determine the details of the execution Web service, and hence it cannot be invoked in the same way that the generation and deployment portlet invokes the conversion Web service. In this latter case all the details of the conversion Web service are known when the client code in the generation and deployment portlet is created. Therefore, the execution Web service is invoked dynamically using information in its WSDL definition. Although the execution portlet cannot get all the information needed to fully construct the client code for invoking the execution Web service, the WSDL filename and the deployment location on the workflow execution engine can be determined at the portlet design time. Thus, the portlet can get all the available operations, together with the number and types of the arguments that each operation requires, and the type of its return value, from the WSDL file at invocation time. The steps required to use the dynamic invocation interface are as follows:

1. The client code creates a Service object for the Web service.

2. Using the Service object, the client creates a Call object.

3. Methods in the Call object are used to specify which operation is to be called, and to discover and list the Java and XML types of the operation's arguments, and its return type.

4. The Web service is invoked by calling the invoke() method of the Call object, which provides the result of the operation as its return value.

The createCall() method is used to create a Call object that is not associated with any port or operation. The port and operation are configured by calling the setTargetEndpointAddress(), setPortTypeName(), and setOperationName() methods before the Call object is used. The WSDL representing the workflow deployed on the workflow execution engine has only one operation called runProcess, and the argument of this operation is the data streamed from the input files selected by users in the file selection portlets. The final result after execution of this workflow is saved into a specified directory and the file repository is updated with this information. This information is shown to the user by the JSP page of the execution portlet.

The instruction element <xsl:text> is used to write Java code to carry out the following tasks:

1. Import the Java libraries required to invoke the execution Web service.
2. Create the execution portlet and include the related JSP page in the doView() method.
3. Create a Call object and set its arguments, operation and access point.
4. Invoke the Web service representing the deployed workflow.
5. Update the file repository to reflect the creation of the result file in the processAction() method.
6. Provide a piece of Java code to deal with any exception that occurs during the invocation of the execution Web service. If an exception occurs, the information about this is forwarded to the included JSP page.

6.7.8 Implementation of the workflow execution JSP page stylesheet

The JSP page included by the execution portlet provides a user-friendly interface to enable users to execute the workflow through the link to the processAction() method of this portlet, and also shows the status of the execution of the workflow. In this JSP page, the session object only contains one object giving the name of the workflow which is deployed on the workflow execution engine by the generation and deployment portlet. This object is used to tell users which workflow they are about to execute. Another important String variable shows users the status of the execution of the workflow, or displays an error message during the execution.

6.7.9 Implementation of the configuration files stylesheet

portlet.xml

The portlet.xml file is the deployment descriptor for the portal application and contains information, such as portlet name, support mode, security, cache, and initial preference information for each portlet in the portal application. Consider the following example of portlet.xml for one portlet in the portal application.

```
<portlet>
    <description xml:lang="en">
        This Portlet is a telescopePorttype1FilePortlet
    </description>
    <portlet-name>telescopePorttype1FilePortlet</portlet-name>
    <display-name xml:lang="en">
        A telescopePorttype1File Portlet
    </display-name>
    <portlet-class>telescopePorttype1FilePortlet</portlet-class>
    <expiration-cache>60</expiration-cache>
    <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>config</portlet-mode>
    </supports>
    <supported-locale>en</supported-locale>
    <portlet-info>
        <title>A telescopePorttype1File Portlet</title>
        <short-title>telescopePorttype1FilePortlet</short-title>
        <keywords>telescopePorttypeFile</keywords>
    </portlet-info>
</portlet>
```

In the above example of an portlet.xml file the <portlet> element contains a <portlet-name> subelement that specifies the name of the portlet. The value of <portlet-name> complies with the naming convention described in Section 6.7.1. Thus, the name of portlet consists of:

- telescope – the name of the service type.
- Porttype – the name of the type.
- 1 - the rank number.
- FilePortlet – the type of the portlet.

The portal displays the value in the <display-name> element on each portlet window. The <supported-locale> element is used to support portals that enable the user to choose a favourite language. In this specific case, the portlet supports only English. The <portlet-info> element is used to specify basic information about the portlet. It contains three subelements, <title>, <short-title>, and <keywords>. The <title> element specifies a title that the portal application can use when it displays this portlet. The <short-title> element is used to specify a brief title. A keyword for this portlet is supplied through the <keywords> element. The <supports> element is used to define the modes that this portlet can support. Normally there are three modes that can be supported in a portlet, but in this case, the view mode is the only one required and this supported by default. The number of portlet elements in the portlet.xml file is identical to number of task and file instance selection portlets. The values of <portlet-name> and <title> are identical and both comply with the naming convention in Section 6.7.1.

layout.xml

The layout.xml file is another important configuration file that is used to specify the location of each portlet in the portlet Web application. Consider the following example fragment from an layout.xml file.

```
<portlet-tabbed-pane>
    <portlet-tab label="testPortal">
        <title lang="en">Services, Files Pick, Display, Invoke Portlets</title>
        <portlet-tabbed-pane style="sub-menu">
            <portlet-tab label="FILE Selection">
                <title lang="en">FILE-SELECTION</title>
                <table-layout>
                    <row-layout>
                        <portlet-frame label="dustCloudPortType0File">
                            <portlet-class>
                                dustCloudPortType0FilePortlet</portlet-class>
                        </portlet-frame>
                    </row-layout>
                    <row-layout>
                        <portlet-frame label="telescopePorttype1File">
                            <portlet-class>
                        telescopePorttype1FilePortlet</portlet-class>
                </portlet-frame>
            </row-layout>
        </table-layout>
    </portlet-tab>
```

The above fragment is part of an entire layout.xml file that describes the layout of the portlets located on a common portlet tab for the file instance selection portlets. This layout.xml file applies to the complicated workflow illustrated in Fig. 6.2, which consists of nine tasks, of which two tasks with rank 0 need inputs represented by circles on the lefthand side of the figure) from two data files. Therefore, in layout.xml all file instance selection portlets for files providing input to task nodes with rank 0 are aggregated onto the same portlet tab. In the portlet tab, each layout row is only allowed to contain one portlet, so there are two layout rows on this portlet tab. In each <row-layout> element the file instance selection portlet class is taken to the value of the <portlet-class> element, which is nested in the <portlet-frame> element. Task nodes having the same rank are grouped onto the same portlet tab, and last two portlet tabs are for the generation and deployment portlet and the execution portlet. All portlet tabs are subelements of a <portlet-tabbed-pane> element.

6.8 Implementation of the Conversion Web Service

The conversion Web service is invoked by the generation and deployment portlet with a list arguments which specify the file and Web service instances selected by the user in the file and sęrvice selection portlets, together with the abstract workflow description file. The conversion Web service converts the abstract workflow into a concrete workflow using information about the file and Web service instances, and returns a String value back to the user. This String value represents the name of the ActiveBPEL workflow file, if successfully deployed, or an error message. In this procedure, the information used to describe the file and Web service instances are merged into the abstract workflow description file, thereby producing a concrete workflow description that be readily converted to an ActiveBPEL workflow description. This approach also makes it easy to extend the prototype implementation to support other workflow execution engines by having a different conversion Web service for each execution engine. Before deploying the ActiveBPEL workflow to the specific workflow execution engine, the Web server hosting the ActiveBPEL workflow execution engine has to make sure of the existence of a servlet that is responsible for uploading the workflow file onto the workflow execution engine. Once the conversion Web service completes the conversion, it sends an HTTP request with the ActiveBPEL workflow file in the request body to the Web server hosting the ActiveBPEL workflow engine. When the Web server receives this request, it automatically uploads the workflow file to the specified file directory. This completes the deployment process.

As shown in Fig. 6.9, the process of generating the ActiveBPEL workflow script is divided into two steps. The first step generates a concrete workflow from the abstract workflow using the information about the file and Web service instances. In a concrete workflow description file, the details of the Web service instance are inserted as a child element into each task element of the abstract workflow description file. To illustrate this process consider the

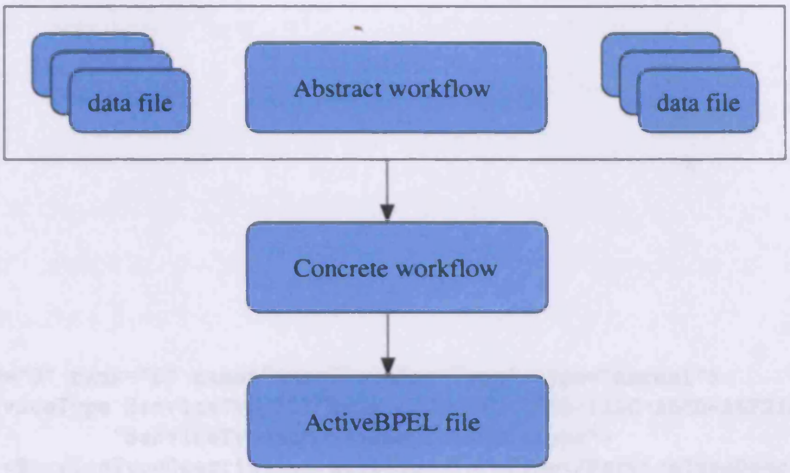


Figure 6.9: The conversion of an abstract Workflow to an ActiveBPEL workflow.

XML element in Fig 6.10, which is from an abstract workflow description file that contains all the information describing a service type, including name, UUID, operation name, input names, and output name.

When the conversion Web service is invoked with a list of arguments, it aggregates the information describing a Web service instances selected by the user, in the Web service instance selection portlets, with the corresponding task node element. The information added includes the service instance name and ID, the access point, and port name. It is inserted into the task node as a sibling of the ServiceType element, and takes the following form.

```
<serviceInstance instanceID="50960C80-4F6E-11DC-8C80-8F685355C7EF">
    <accesspoint>
        http://chlorin.cs.cf.ac.uk:8080/axis/new1wsdl/dustCloudService1.wsdl
    </accesspoint>
    <targetNamespace>
        http://dlu1.FFTWebservice1.dustCloud1/service
    </targetNamespace>
    <serviceName>DustCloudService1</serviceName>
```

```
<task ID="0" rank="0" name="dustCloudPortType" type="normal">
    <ServiceType ServiceTypeID="uuid:45D265E0-4F65-11DC-A5E0-A5F21AD7226F"
                 ServiceTypeName="dustCloudPortType">
        <ServiceTypeDescription>dustCloudPortType</ServiceTypeDescription>
        <namespace>http://dlu1.FFTWebservice1.dustCloud1/definitions</namespace>
        <urlLocation>
            http://chlorin.cs.cf.ac.uk:8080/axis/new1wsdl/dustCloudPortType1.wsdl
        </urlLocation>
        <portType name="DustCloudService">
            <operation id="0" name="yValues">
                <input type="dustCloudInput" name="dustCloudInput">
                    <part name="in0" type="xs:string" />
                    <part name="in1" type="xs:double" />
                    <part name="in2" type="xs:int" />
                </input>
                <output name="output" type="dustCloudOutput">
                    <part name="out" type="def:doubleArray" />
                </output>
            </operation>
        </portType>
    </ServiceType>
</task>
```

Figure 6.10: One element representing a task node in an abstract workflow.

```
    <portName>DustCloudService1</portName>
</serviceInstance>
```

After the information about the Web service instances has been integrated into the abstract workflow, the concrete workflow is still not complete as it is necessary to add input/output (IO) Web services at the head and tail of the concrete workflow to enable it to get initial arguments for the tasks with rank 0 from the input files. Figure 6.11 illustrates the relationship between the IO Web services and the task Web services. The role of the input Web services is to extract the parameters from the specified input files and transfer this to the corresponding Web services. Similarly the output Web services transfer workflow outputs to the appropriate output files. The IO Web services are necessary because all the Web services registered in the Web service repository only accept private data rather than extracting their data from files. Thus, the concrete workflow corresponding to the abstract workflow is embedded between IO Web services and is just a part of the complete concrete workflow. Each file contains a group of arguments for a particular service type. During the generation of the complete concrete workflow, the IO Web service for every Web service with rank 0 is automatically selected from the Web service repository and added in front of these Web services in order to extract the parameters from the specified files and transfer them to the correct Web service. Similarly an IO Web service from the Web service repository is also added to the end of the original concrete workflow in order to save the result data into a file.

After adding the IO Web services to the head and tail of the concrete workflow, the completed concrete workflow is then converted to a workflow description that is compatible with the ActiveBPEL workflow execution en-

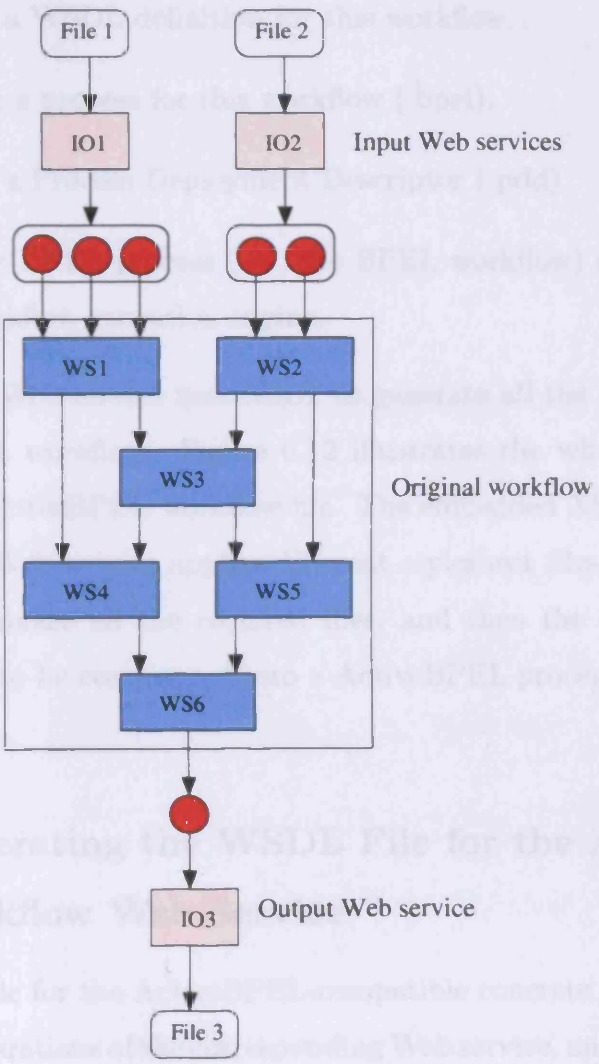


Figure 6.11: The concrete workflow with the added IO Web services.

gine. The ActiveBPEL workflow file comprises a set of files in different file directories. There are four steps to generate workflow for an ActiveBPEL execution engine:

1. Construct a WSDL definition for this workflow.
2. Implement a process for this workflow (.bpel).
3. Construct a Process Deployment Descriptor (.pdd)
4. Deploy the BPEL process (i.e., the BPEL workflow) as a Web service on the workflow execution engine.

The conversion Web service uses XSLT to generate all the required files for the ActiveBPEL workflow. Figure 6.12 illustrates the whole procedure of generating an ActiveBPEL workflow file. The embedded XSLT processor in the conversion Web service applies different stylesheet files to the concrete workflow to generate all the required files, and then the files are sent to the compressor to be compressed into a ActiveBPEL process file with a bpr extension name.

6.8.1 Generating the WSDL File for the ActiveBPEL Workflow Web Service

The WSDL file for the ActiveBPEL-compatible concrete workflow defines the available operations of the corresponding Web service, and the mechanism by which they are accessed. The interface of the Web service is defined in terms of its portType, operations and messages, and the access mechanism is defined in terms of partnerLinkType and binding, each of which has a section or sections defined by XML elements in the WSDL file. There are seven steps in creating the ActiveBPEL workflow, as follows:

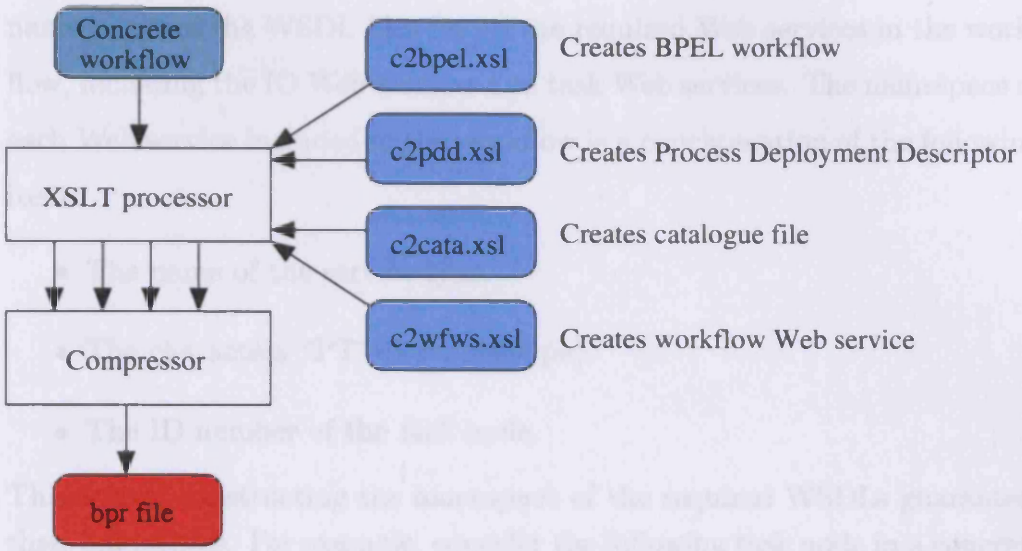


Figure 6.12: ActiveBPEL workflow generation.

1. Declare the targetNamespace of the workflow WSDL.

2. Declare the namespaces of the WSDLs required by the workflow.

3. Import all the namespaces of WSDLs through URLs.

4. Declare the message elements for the workflow WSDL.

5. Declare the operation element for the workflow WSDL.

6. Declare the partnerLinkType elements for each task node in the workflow.

7. Declare the service element of the workflow WSDL.

The first step is to define the targetNamespace of the WSDL file, which, according to the naming convention, is “http://concreteworkflow0/auto”, and is the same for all ActiveBPEL workflows. The next step is to import the namespaces of the WSDL files for all the required Web services in the workflow, including the IO Web services and task Web services. The namespace of each Web service included in the workflow is a concatenation of the following items:

- The name of the service type.
- The characters "PT" (for portType).
- The ID number of the task node.

This way of constructing the namespace of the required WSDLs guarantees their uniqueness. For example, consider the following task node in a concrete workflow:

```
<task ID="0" rank="0" name="dustCloudService" type="normal">
    <ServiceType ServiceTypeID="uuid:45D265E0-4F65-11DC-A5E0-A5F21AD7226F" ...>
...
    </ServiceType>
</task>
```

The namespace of the WSDL for this task node would be "dustCloudServicePT0".

The second step is to import all the WSDL files required by the workflow Web service by accessing them through their URLs. The URL of each WSDL file can be retrieved from the <urlLocation> element nested in the <ServiceType> element, which, in turn, is nested in the <task> element. In the workflow WSDL file, <xsl:text> elements are used to direct the XSLT processor to produce a group of <import namespace=".." location="..."> elements. The number of <import> elements equals the number of task nodes in the concrete workflow file. The next elements to be generated by <xsl:text> elements are two <message> elements that are used to represent inputs and output of the workflow WSDL. The <message> element representing the inputs contains a set of URLs to files that are the arguments of the IO Web services. The other <message> element that represents the output of the workflow Web service is a boolean value to indicate whether the execution of the workflow was successful or not. An example of the two <message> elements is shown below:

```
<message name="ClientRequest">
    <part name="in0" type="xs:string"/>
        ...
</message>
<message name="ClientResponse">
    <part name="out" type="xs:boolean"/>
</message>
```

Next, it is necessary to define the <partnerLinktype> for each WSDL file required by the workflow, including that of the workflow WSDL itself. The <partnerLinkType> element is used to define the types of the links between partners. Each <partnerLinkType> element defines a role, and each role references a portType. For example, the role referencing the portType of the workflow WSDL itself can be defined as follows:

```
<plnk:partnerLinkType name="ClientLink">
    <plnk:role name="ProcessProvider">
        <plnk:portType name="pns:ProcessClientPT"/>
    </plnk:role>
</plnk:partnerLinkType>
```

For the other roles referencing the portTypes in other included WSDL files, the name attribute of the <partnerLinkType> element must be unique. The convention for naming a <partnerLinkType> element is similar to naming the namespace of a WSDL file, and consists of the following:

- The name of the service type.

- The characters "PT".

- The ID number of the task node.

- The characters "Link".

For example, the <partnerLinkType> element referencing the portType of the Web service element in Fig. 6.10 is:

```
<plnk:partnerLinkType name="DustCloudServicePT0Link">
    <plnk:role name="DustCloudService10Provider">
        <plnk:portType name="DustCloudServicePT0:DustCloudService"/>
    </plnk:role>
</plnk:partnerLinkType>
```

The value of the name attribute of the <portType> element must be fully qualified with its namespace and local name. The last element in the workflow WSDL is the <service> element, and this just specifies the name of the Web service representing this workflow:

```
<service name="serviceProvider0"/>
```

6.8.2 Implementation of the Catalogue XSLT Stylesheet

The wsdlCatalog.xml file contains a catalogue of the WSDL and XML schema definitions needed to resolve all of the references throughout the ActiveBPEL workflow. This provides a way for the ActiveBPEL execution engine to find the WSDL files in a .bpr deployment archive. The following example shows an entry in wsdlCatalog.xml in the workflow in Fig. 6.11:

```
<wsdlEntry location="project:/wsdl/dustCloudPortType1.wsdl"
           classpath="wsdl/dustCloudPortType1.wsdl"/>
```

The location attribute maps to a WSDL file in one of two ways. Its value is either the location attribute of a <wsdl> element in the wsdlReference section of a .pdd file, or the location attribute of an <import> element in a WSDL file. When loading a WSDL file at deployment time, the ActiveBPEL workflow engine reads the WSDL references from the .pdd file and uses the location attribute of the <wsdl> element as a key into the WSDL catalogue. If the WSDL catalogue contains a matching location, the engine automatically loads the WSDL from the corresponding classpath. If no mapping existing in the catalogue, the ActiveBPEL engine assumes the location is an absolute URL and attempts to load the WSDL from that location. The classpath attribute indicates where the WSDL file resides in the .bpr file, relative to the root of the .bpr directory structure.

In the XSLT stylesheet for generating the catalogue file, the <xsl:text> instruction element is used to produce a group of <wsdlEntry> elements in the catalogue file. The number of <wsdlEntry> elements equals the number of task Web services and IO Web services in the workflow.

6.8.3 Implementation of the bpel stylesheet

BPEL not only provides a mechanism to orchestrate Web services to work cooperatively, but once deployed a BPEL process[6] is a Web service in its own right. The process definition is simpler and shorter than the WSDL file of the workflow Web service because most of the implementation details of a process, such as how to access it, what messages it uses, what operations it performs, and how incoming and outgoing data are formatted and marshaled, are defined in the WSDL files. A basic BPEL process file contains the following elements:

1. A list of namespace declarations.

2. A <partnerLinks> element with a list of <partnerLink> element declarations.

[6] A workflow is termed a "process" in BPEL.

3. A <variables> element with a list of <variable> element declarations.
4. A <sequence> element that contains:
   - a <receive> element to start the process;
   - a <sequence> element representing the workflow structure;
   - a <reply> element to end the process.

The concrete workflow generated by the portal web application through the user's selection of file and Web service instance differs from the ActiveBPEL process in vocabulary and structure. The conversion of a concrete workflow to an ActiveBPEL process is divided into several steps: namespace generation, partnerLinks generation, variables generation, and sequence generation.

The embedded XSLT processor uses the c2bpel.xsl XSLT stylesheet (see Fig. 6.12) to transform a concrete workflow into an ActiveBPEL process file. The first step of the transformation is to generate the namespaces, which follow the naming convention described in Section 6.8.1.

The next step is to import all the required WSDL files for every Web services involved into the bpel workflow by using <import> elements. The structure of an import element is like as follws:

```
<import importType="..." namespace="..." location="..."/>
```

The inportType attribute is used to indicate that the item imported by this element is a WSDL file. The value of namespace has to be identical to the relevant namespace value in the namespace declaration. The location attribute enables the ActiveBPEL workflow engine to find the WSDL file and load it during the deployment of the ActiveBPEL process.

After importing the namespace for each Web service involved in the workflow, it is necessary to declare the <partnerLinks> element. This contains several <partnerLink> subelements, which define the Web services used in the BPEL process. These Web services are described by WSDL files. The BPEL process refers to partners, which in turn reference WSDL files describing the partner link types. The syntax of a <partnerLink> element is:

```
<partnerLink name="..." partnerLinkType="..." partnerRole="..."/>
```

In each <partnerLink> element there are three attributes to specify. According to the naming convention, the name of a partnerLink is the concatenation of the name of the portType of the Web service, and rank of the task node represented by this Web service, and its ID. The partnerLinkType is a fully qualified name. All the partnerLinkTypes have been defined in the workflow Web service WSDL file already. The value of the partnerRole attribute must be identical to the value of the name attribute of the <role> element nested in the <partnerLinkType> element.

After declaring the <partnerLink> elements grouped in the <partnerLinks> element, it is necessary to declare all variables required by each Web service involved in the BPEL process. Each Web service instance involved in the BPEL process should have an input variable and an output variable. The syntax of the <variable> element is:

```
<variable name="..." messageType="..."/>
```

The value of the name attribute of the <variable> element for each Web service is the concatenation of the value of the name attribute of the <input> element (or <output> element) and the ID of its parent <task> element. This guarantees the uniqueness of each variable in the variable declaration. Without adding ID, two variables from two semantically equivalent Web services might have the same name, and cause a conflict in the deployment of the ActiveBPEL workflow. The value of the messageType attribute has to be a fully qualified name, so it has to contain the prefix for the corresponding namespace for the specific Web service and message types defined in its WSDL file. The prefix for the WSDL file of a Web service is declared according to the naming convention, and the value of the message type is also saved in the concrete workflow file. For example, consider the following task node fragmnent:

```
<task ID="0" rank="0" name="dustCloudPortType" type="normal">
    <ServiceType ServiceTypeID="..." ServiceTypeName="...">
        ...
        <portType name="DustCloudService">
            <operation id="0" name="yValues">
                <input type="dustCloudInput" name="dustCloudInput">
                    ...
                </input>
            </operation>
        </portType>
    </ServiceType>
</task>
```

The corresponding input variable for task 0 should be:

```
<variable name="dustCloudInput0" messageType="DustCloudServicePT0:dustCloudInput"/>
```

The number of variables declared in the bpel file is twice of the number of Web service instances in the BPEL process.

The last step is to transform the actual workflow structure described by the concrete workflow to an ActiveBPEL process. In the concrete workflow, a set of <activity> elements represent the source and destination of data for each task node. In the BPEL process, each <activity> element is represented as a pair of <assign> and <invoke> elements, and these pairs are grouped hierarchically into different <sequence> elements. Before doing the transformation, two elements have to be added as initialization and termination of the process. These two element are <receive> and <reply>, and serve as the interface between the bpel process and the client. The bpel process receives the request from the client and sends a response back through the ActiveBPEL engine.

All the elements describing the workflow structure, including the <receive> and <reply> elements, are enclosed in a <sequence> element. The <sequence> element is like a container which arranges and executes activities in an ordered list. This means that when the first activity (such as assignment) is finished, the second activity (such as invoke) is allowed to begin. Thus, each pair of <assign> and <invoke> elements representing an <activity> element in the concrete workflow is grouped into a <sequence> element to make sure they are executed in order. To illustrate this a sample <activity> element from a concrete workflow is shown below:

```
<activity targetID="3" targetRank="2">
    <source ID="0">
        <map port="1" fileTypeID="def:doubleArray" />
    </source>
    <source ID="2">
        <map port="0" fileTypeID="xs:int" />
    </source>
</activity>
```

The above <activity> represents two parameter assignments and a Web service invocation. The <assign> element is used to assign the value to the variable(s) required by the Web service invoked in the <invoke> element. The syntax of the <assign> element is as follows:

```
<assign>
    <copy>
        <from>...</from>
        <to>...</to>
    </copy>
```

```
</assign>
```

In the <assign> element, there is a subelement called <copy> that contains two elements: <from> and <to>. As their names imply, these are used to specify which variable the value is got from and which variable the value is assigned to. According to the source element in the concrete workflow, each assign element in an ActiveBPEL process represents a source element in the concrete workflow. The value of the ID attribute of the source element indicates which output variable that the data comes from. It is transformed to the relevant output variable of the Web service holding that ID in the <from> element. Similarly, the <map> element indicates the input variable that the data goes to. The value of the port attribute of the <map> element is used to specify the input variable.

A BPEL process not only supports primitive data transfer, but also the transfer of complex data, such as double arrays. However, unfortunately the <copy> element cannot support double arrays. The alternative way to copy a double array between variables in a BPEL process is to copy each item in the array to the destination variable by using an XPath expression. BPEL provides a *while* structure that can be used to assign values in a double array one-by-one. In order to use a *while* structure, two variables are needed, $count and $counter. The $count variable stores the number of doubles in the array. The $counter holds the current position in the double array. The indexing of the array starts at 1 rather than 0 as in other programming languages. The assignment of a double array in a BPEL process can be performed as follows:

```
<sequence name="copy-arraydustCloudOutputOTOpowerof2input2">
    <assign name="setupcounterandcount">
        <copy>
            <from>1</from>
            <to>$counter</to>
        </copy>
        <copy>
            <from>count($dustCloudOutput0.out/def:item)</from>
            <to>$count</to>
        </copy>
    </assign>
    <while>
        <condition>$counter&lt;=$count</condition>
        <sequence>
            <assign>
                <copy>
                    <from>$dustCloudOutput0.out/XPath expression</from>
                    <to>$powerof2input2.in0/XPath expression</to>
                </copy>
                <copy>
                    <from>$counter+1</from>
                    <to>$counter</to>
                </copy>
            </assign>
        </sequence>
    </while>
</sequence>
```

After assignment of all the variables required by the Web service, the next step is to invoke the Web service by calling the <invoke> element in the BPEL process. The syntax of the <invoke> element is as follows:

```
<invoke name="..." partnerLink="..."
        portType="prefix:portType"
        operation="..."
        inputVariable="inputName"
        outputVariable="outputName"/>
```

For each activity element, there should be a corresponding <invoke> element to invoke the corresponding Web service. The <invoke> element has six attributes: name, partnerLink, portType, operation, inputVariable and outputVariable. The portType has to be specified with as a fully qualified name.

6.8.4 Generating the Process Deployment Descriptor File

A process deployment descriptor (.pdd) file describes the service binding and, optionally, the process version details for a BPEL process. The service binding details describe the relationship between the partners defined in the bpel file and the implementation required to interact with actual partner endpoints (i.e., Web services). The descriptions include the binding information for each partner role defined in a partner link. For partner role endpoint references, the binding is described in the WS-Addressing standard format[7]. The WS-Addressing specification defines a standard for referencing a Web service, thereby allowing a Web service to be uniquely identified.

The basic WS-Addressing syntax is as follows:

```
<wsa:EndpointReference>
    <wsa:Address>prefix:anyURL</wsa:Address>
    <wsa:ServiceName PortName="portname">prefix:Service</wsa:ServiceName>
</wsa:EndpointReference>
```

In the above,

- Address is a mandatory element that identifies an endpoint, which could be specified as a network address or a logical address.

[7] http://www.w3.org/Submission/ws-addressing/, accessed 12 January 2010.

- ServiceName is a qualified name for the service being invoked and this service must be defined in a WSDL file that is deployed with the process or one that is available in the ActiveBPEL engine's WSDL catalogue.

- PortName identifies the service port.

In the bpel concrete workflow each task node has a corresponding <partnerLink> element that contains a <wsa:EndpointReference> element to reference the actual partner endpoint (i.e., the Web service). The <partnerLink> element relevant to the task with ID 0 is as follows:

```
<partnerLink name="DustCloudService10">
    <partnerRole endpointReference="static">
        <wsa:EndpointReference
                xmlns:DustCloudService10 =
                    "http://dlu1.FFTWebservice1.dustCloud1/service">
            <wsa:Address>DustCloudService10:anyURL</wsa:Address>
            <wsa:ServiceName PortName="DustCloudService1">
                DustCloudService10:DustCloudService1
            </wsa:ServiceName>
        </wsa:EndpointReference>
    </partnerRole>
</partnerLink>
```

The value of the name attribute of the <partnerLink> element also complies with the naming convention, and is the concatenation of the text value of the <serviceName> subelement nested in the <task> element and the ID of the task, and this combination is also the prefix of this Web service's namespace, as declared in the <wsa:EndpointReference> element. This prefix qualifies the text value of the <wsa:Address> element.

6.8.5 Deployment of the ActiveBEPL Workflow on the ActiveBPEL Engine

The files for an ActiveBPEL process, including the BPEL workflow file. the PDD file, wsdlCatalog.xml, and the WSDL files, are located in different file directories. The standard directory structure used in an ActiveBPEL process in the archive is as follow:

- Root - NameOfProcess
    - A bpel directory containing the .bpel file.
    - A META-INF directory containing the wsdlCatalog.xml file.
    - A wsdl directory containing all required WSDL files.

The root directory and its contents are compressed into a .bpr file by the conversion Web service, and this file is then sent by an HTTP request to a Java servlet on the Tomcat server that hosts the ActiveBPEL execution engine. This servlet uploads the compressed file and places it in a specific location that the ActiveBPEL engine can access. The ActiveBPEL process is now executable.

6.9 The Graphical User Interface for the Build Tool

The Build Tool and its related tasks can be run from the command line. However, for convenience a GUI is available through which users can interact with the Build Tool. The Build Tool GUI also provides users with extra advanced functions for monitoring the progress of the portal application generation and deployment. It can be used to select an abstract workflow from the local file system, and generate the corresponding Web portal application. During the progress of the generation of the portal application, the progress information is provided on the GUI. Moreover, the GUI can serve as a client to upload the Web portal application to a specific Tomcat Web server.

The GUI software is built on top of core functions of the Build Tool. The operations performed by the GUI are to get requests from the user, pass them to the Build Tool kernel, and return the response to the user once the kernel has completed the request. The GUI component is made up of five Java Swing components including three JButtons, one JTextArea, and one JFileChooser, as shown in Fig. 6.13. The "create" button is used to pop up the file chooser component for the selection of the abstract workflow description file. Once the abstract workflow description file has been specified by the user the GUI calls the kernel of the Build Tool, and display information on the status of the generation of the corresponding Web portal application on the text area component. Another button is responsible for uploading the portal application to the specified Tomcat Web server. The user can pick up the WAR file[8] and upload it to the specified Web server. The third button is used to exit the GUI.

In order to upload the compressed portal application to the Web server, a Java servlet must be deployed on the server side to receive upload request. Thus, it is necessary to develop a Java servlet that is responsible for receiving the data stream from the client side (i.e., the Build Tool), and which deploys the WAR file to the specified location.

[8] The Web portal application is compressed as a WAR file.

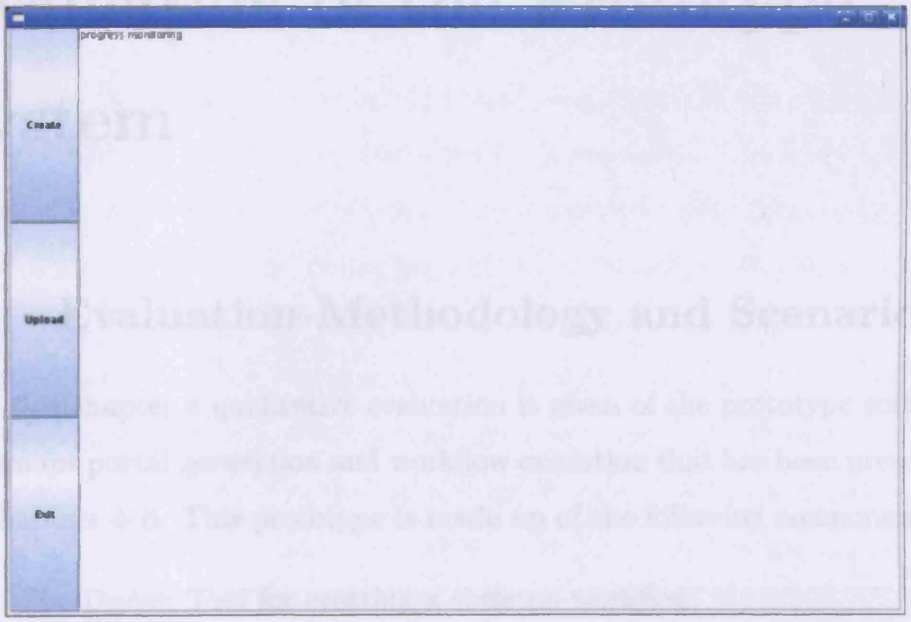


Figure 6.13: The GUI of the Build Tool.

Chapter 7

Evaluation of the Prototype System

7.1 Evaluation Methodology and Scenarios

In this chapter a qualitative evaluation is given of the prototype software system for portal generation and workflow execution that has been presented in Chapters 4–6. This prototype is made up of the following components:

1. The Design Tool for creating a skeleton workflow.
2. The Label Tool for converting a skeleton workflow into an abstract workflow.
3. The Design Tool for generating a portal application that can be used to convert an abstract workflow into a concrete workflow.
4. The portal application and its associated tools and services for converting an abstract workflow into a concrete workflow, and then executing it on a workflow engine.

The evaluation will particularly focus on the Build Tool that generates the portal application, and the use of this portal for creating and executing a concrete workflow, since it is these areas of the prototype system that are the most innovative, and yield the most significant research contributions. The evaluation uses six test cases to test the individual performance of each component and the interactions between them. These test cases range from very simple scenarios to workflow applications that are commercially and scientifically significant.

The first three test cases are quite simple, and represent workflow patterns that are commonly found in distributed applications. The last three test cases are more complex and represent real scientific and commercial workflows. The first test case is the simplest — the workflow contains only one task, which takes a string as input and echoes it on its output. This case demonstrates the basic function of the prototype system. The second test case is a little more complicated than the first case. The skeleton workflow consists of three tasks arranged in a linear sequence with each task taking one input, processing it, and outputting the result. In this test case all the inputs and outputs are strings. The main purpose of this scenario is to test the performance of communications between the different Web services, and the transportation of SOAP messages between them. The third test case also involves three tasks, however, it differs from the second test case as two of the tasks can execute concurrently, rather than in sequence. The third task receives the outputs from the two concurrent tasks, and generates the final output. Again, all tasks have strings as their input and output. This scenario is another common workflow pattern, and is used to test whether multiple Web services can work together in parallel. The remaining three test cases have workflows that involve more tasks and are more complex. The fourth test case involves a workflow that might be used in comparing astronomical observations with a computational model. The fifth and sixth test cases involve the processing of genealogical data by a commercial web site. Each of the six test cases will now be described in more detail, and evaluated.

Section 7.2 gives further details of the six test cases and their evaluation. Section 7.3 gives details of the computing environment that hosts the prototype system and the services and files for the test cases.

7.2 Workflow Test Cases

7.2.1 Test Case 1: the Echo Workflow

The first test case is very simple and contains one task, as shown in Fig. 7.1, and so the skeleton workflow is easy to draw with the Design Tool. The single task node has one input and one output port, each connected to a different file node. The XML file representing the skeleton workflow is saved by the Design Tool, and is subsequently opened by the Label Tool, which associates a service type with the task node, thereby creating the abstract workflow. A snapshot of the Label Tool, shown in Fig. 7.2, illustrates the interface for labeling a task node with a service type. The table on the righthand side of the Label Tool GUI list all the service types registered in the Web service repository to which the Label Tool is connected. The name of the service type for test case 1 is testechoPortType, and this is used to label the task node in the skeleton workflow. The table also shows the UUID of the service type, the number of inputs, and datatypes of the inputs and outputs. The table on the lefthand side of the Label Tool GUI shows the current status of the nodes. In Fig. 7.2 it can be seen that the single task node, and both the file nodes, have been labeled. The skeleton workflow is updated with the service and file type information to create the abstract workflow which is saved as an XML file.

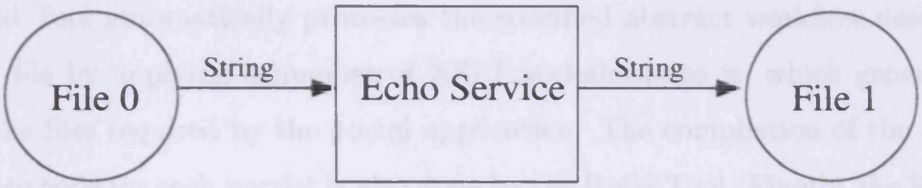
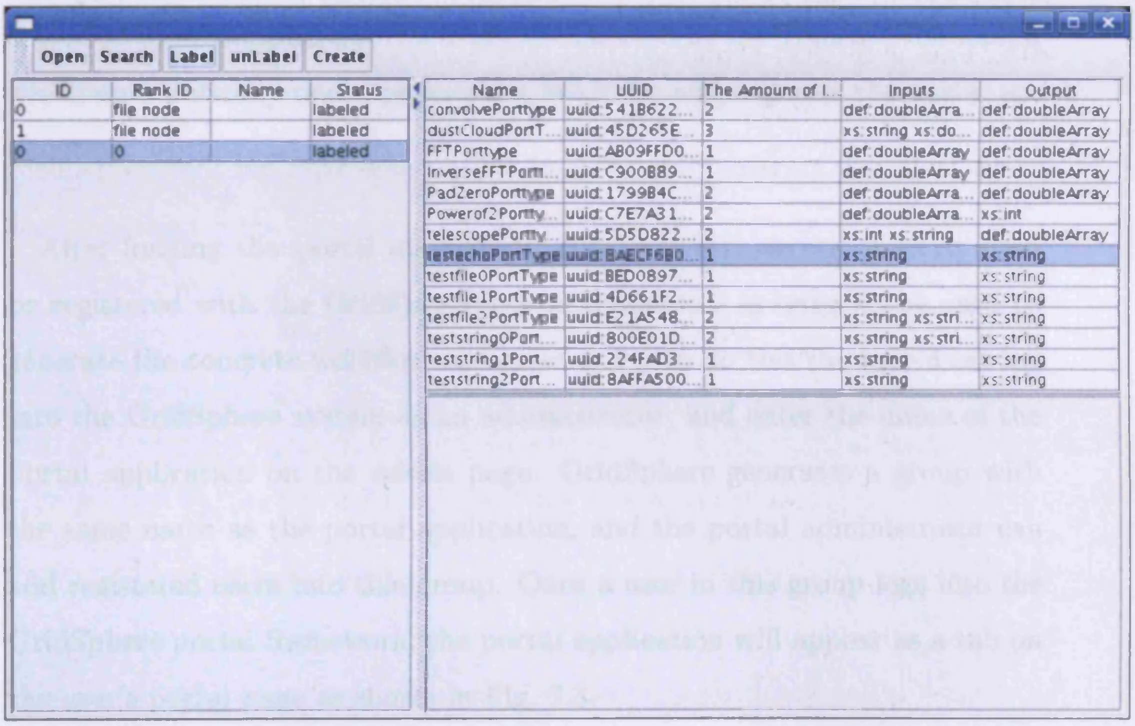


Figure 7.1: The echo abstract workflow.



ID	Rank ID	Name	Status
0	file node		labeled
1	file node		labeled
0	0		labeled

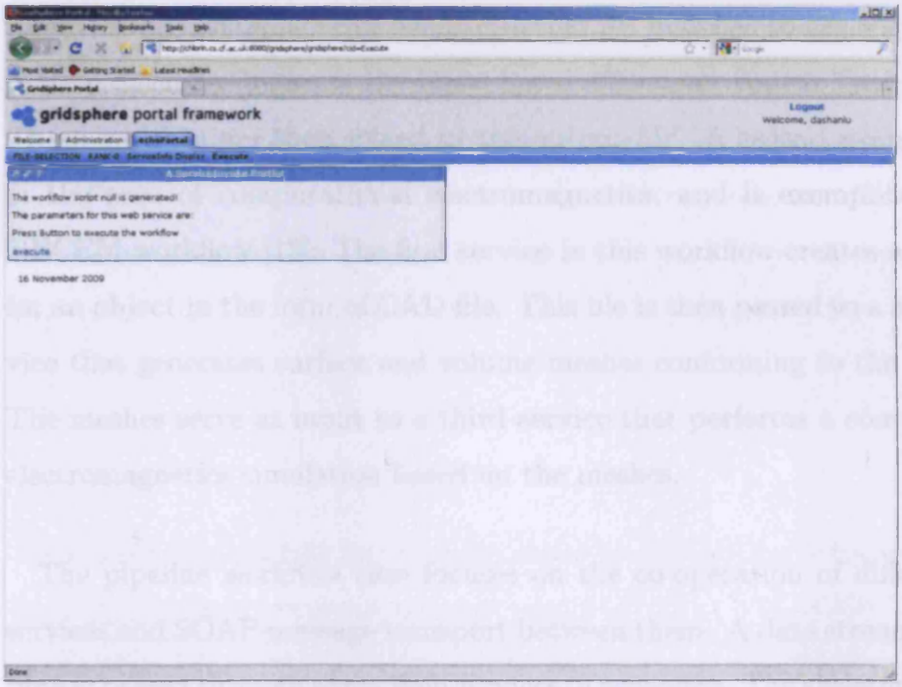
Name	UUID	The Amount of I...	Inputs	Output
convolvePorttype	uuid:541B622...	2	def:doubleArra...	def:doubleArray
dustCloudPortT...	uuid:45D265E...	3	xs:string xs:do...	def:doubleArray
FFTPorttype	uuid:AB09FFD0...	1	def:doubleArray	def:doubleArray
inverseFFTPort...	uuid:C900B89...	1	def:doubleArray	def:doubleArray
PadZeroPorttype	uuid:1799B4C...	2	def:doubleArra...	def:doubleArray
Powerof2Portty...	uuid:C7E7A31...	2	def:doubleArra...	xs:int
telescopePortty...	uuid:5D5D822...	2	xs:int xs:string	def:doubleArray
testechoPortType	uuid:BAECF6B0...	1	xs:string	xs:string
testfile0PortType	uuid:BED0897...	1	xs:string	xs:string
testfile1PortType	uuid:4D661F2...	1	xs:string	xs:string
testfile2PortType	uuid:E21A548...	2	xs:string xs:stri...	xs:string
teststring0Port...	uuid:800E01D...	2	xs:string xs:stri...	xs:string
teststring1Port...	uuid:224FAD3...	1	xs:string	xs:string
teststring2Port...	uuid:8AFFA500...	1	xs:string	xs:string

Figure 7.2: The Label Tool interface for the echo workflow.

In the next step the Build Tool generates the portal application based on the abstract workflow description file created by the Label Tool. As mentioned in Section 6.9, the Build Tool can be run either from the command line, or with a Java GUI application. In either case, the user simply has to supply the name and location of the abstract workflow description file. The Build Tool automatically processes the specified abstract workflow description file by applying a number of XSLT stylesheets to it, which generates all the files required by the portal application. The compilation of the Java source code for each portlet is also done by the Build Tool. Finally, the Build Tool bundles all the required files for the portal application into a WAR file in a specified directory. In the prototype software system the WAR file is copied to the GridSphere portal framework hosted by the Tomcat Web server, which automatically uncompresses the WAR file and registers the portal application.

After loading the portal application onto the Web server, it must then be registered with the GridSphere portal framework in order to be used to generate the concrete workflow and execute it. To do this the user must log into the GridSphere system as an administrator, and enter the name of the portal application on the *admin* page. GridSphere generates a group with the same name as the portal application, and the portal administrator can add registered users into this group. Once a user in this group logs into the GridSphere portal framework, the portal application will appear as a tab on the user's portal page as shown in Fig. 7.3.

The user can select file instances from the file instance selection pages and Web service instances from the service instance selection pages. All information about the file and Web service instances is transported by session variables to the specific portlet page which is responsible for the generation of the ActiveBPEL concrete workflow script. The user can invoke a conversion Web service to generate a concrete workflow and can deploy this onto a specific ActiveBPEL workflow execution engine. The last portlet page of this portal provide the user with a link to execute the concrete workflow generated by the previous portlet pages. The execution page of portal application is shown in Fig. 7.3. The workflow is executed by invoking the Web service



The workflow script null is generated!

The parameters for this web service are:

Press Button to execute the workflow

execute

16 November 2009

Figure 7.3: The execution page of the Echo Portal.

representing the workflow on the workflow execution engine. This page also provides basic feedback information to tell the user whether the execution has completed successfully, or failed due to some error. If the execution of the workflow is successfully completed, the final output file(s) is(are) saved to the specified location, and the file repository is updated with this information.

7.2.2 Test Case 2: the Pipeline Workflow

This scenario represents a pipeline pattern which is commonly-used in distributed applications, particularly in the biosciences. An example is the workflow discussed in [43] which makes use of three Web services. The first is the invokebrowser service that retrieves raw protein data from a URL stored in an input file. These data are then passed to the getproteinseq service which extracts a protein sequence from it. This protein sequence is next input to the blastall service which searches for matches to the sequence in a protein database by using the Basic Local Alignment Search Tool (BLAST). These matches are then stored in the output file. A second example arises in the area of computational electromagnetics, and is exemplified by the GECEM workflow [13]. The first service in this workflow creates a geometry for an object in the form of CAD file. This file is then passed to a second service that generates surface and volume meshes conforming to the geometry. The meshes serve as input to a third service that performs a computational electromagnetics simulation based on the meshes.

The pipeline workflow case focuses on the co-operation of different Web services and SOAP message transport between them. A data stream of String type is passed through three Web services in sequence. The skeleton workflow is illustrated in Fig. 7.4. Three task nodes are connected in sequence, and there is one input file and one output file. As in the echo test case, three service types registered in the Web service repository are used to label the three

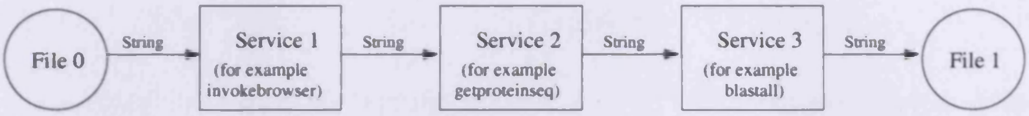


Figure 7.4: The pipeline skeleton workflow.

task nodes using the Label Tool. After this, the resulting abstract workflow description file is used by the Build Tool to generate the portal application, through which the user generates the ActiveBPEL concrete workflow script based on the abstract workflow. After deployment of this portal application onto the GridSphere portal framework, the portal is as shown in Fig. 7.5.

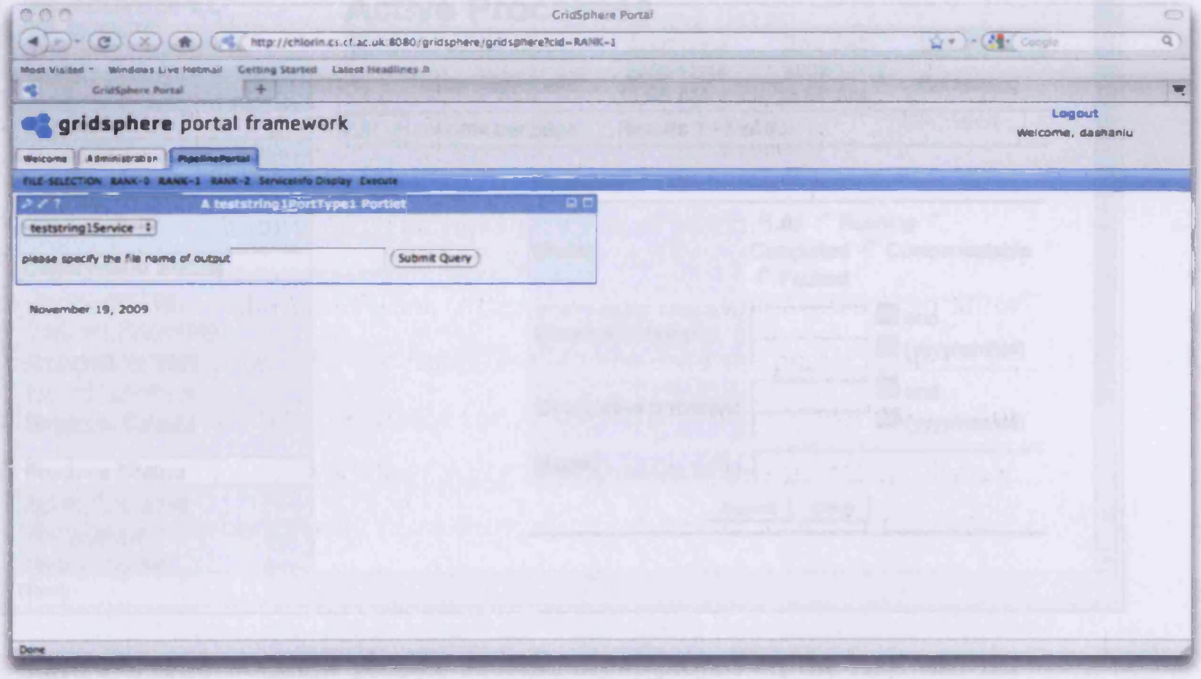
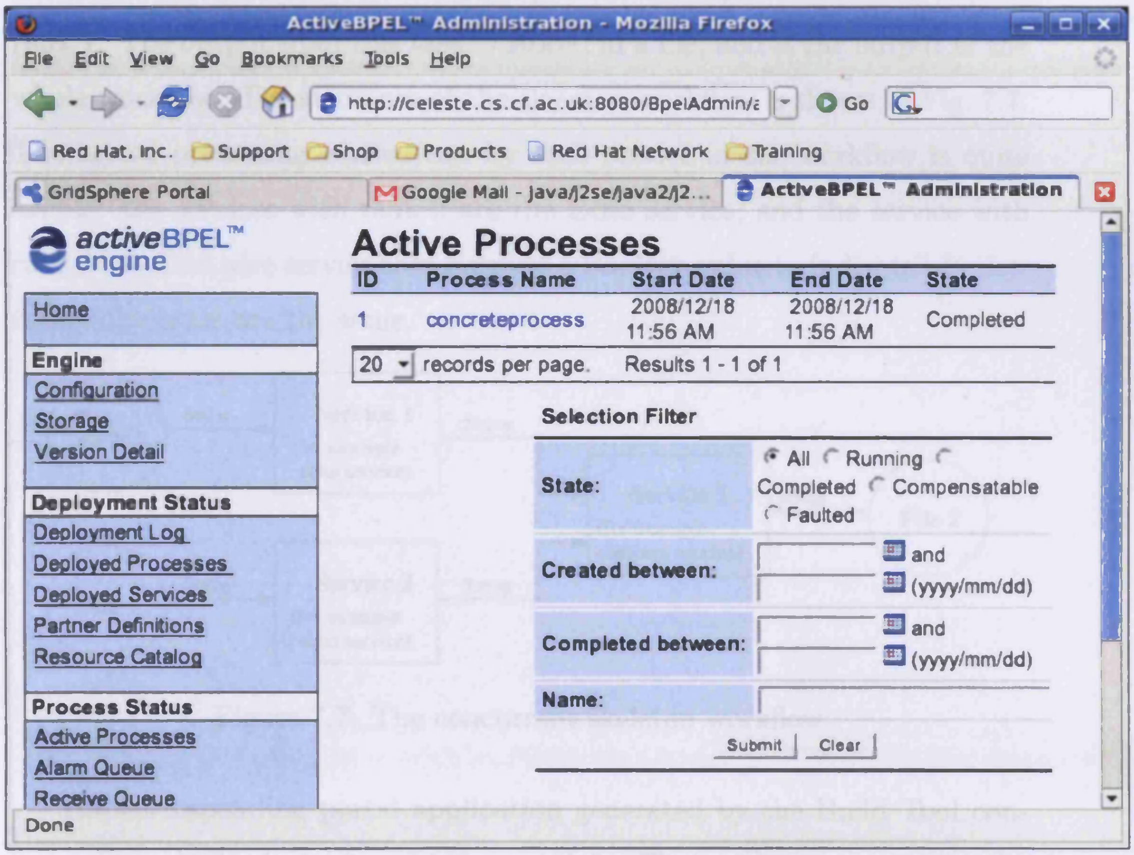


Figure 7.5: Selecting the Web service instance for the rank 1 service in the pipeline workflow. If a file name is supplied in the portlet then the output of the service is checkpointed.

The ActiveBPEL workflow script generated by the conversion Web service is then deployed on the workflow execution engine, as shown in Fig. 7.6.


ActiveBPEL™ Administration - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://celeste.cs.cf.ac.uk:8080/BpelAdmin/
Go
Red Hat, Inc. Support Shop Products Red Hat Network Training
GridSphere Portal
Google Mail - java/j2se/java2/j2...
ActiveBPEL™ Administration
activeBPEL™ engine
Home
Engine
Configuration
Storage
Version Detail
Deployment Status
Deployment Log
Deployed Processes
Deployed Services
Partner Definitions
Resource Catalog
Process Status
Active Processes
Alarm Queue
Receive Queue
Done

Active Processes

ID	Process Name	Start Date	End Date	State
1	concreteprocess	2008/12/18 11:56 AM	2008/12/18 11:56 AM	Completed

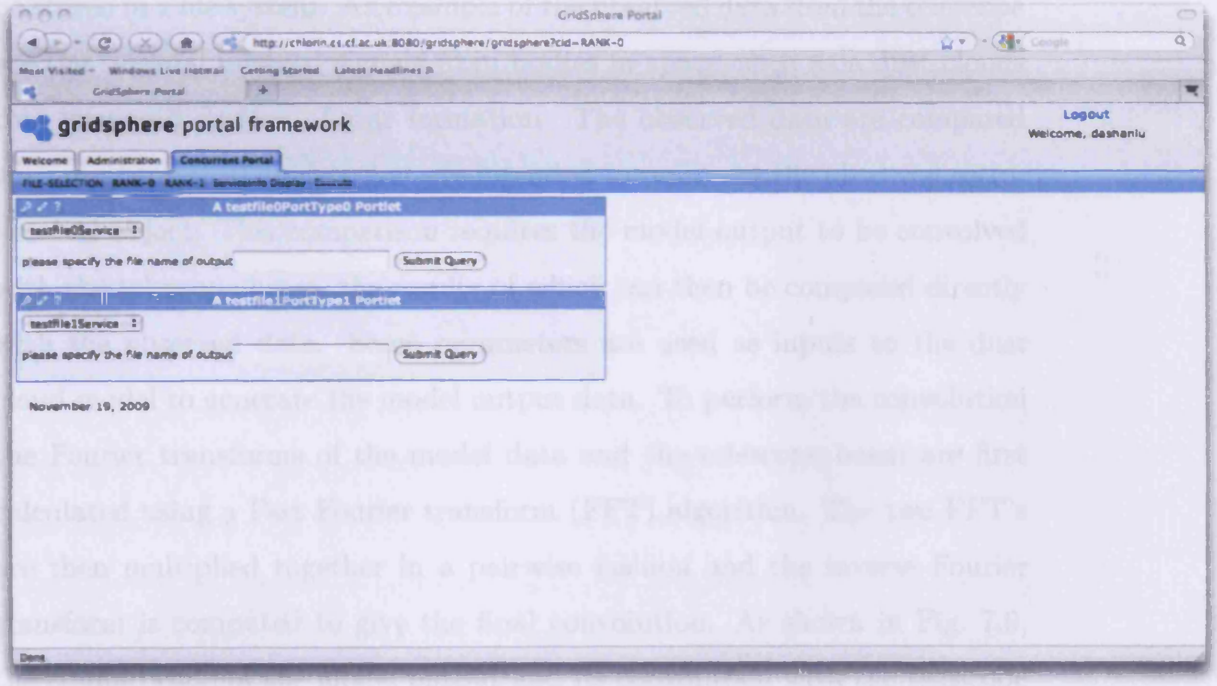
20 records per page. Results 1 - 1 of 1

Selection Filter

State:	● All ○ Running ○ Completed ○ Compensatable ○ Faulted
Created between:	and (yyyy/mm/dd)
Completed between:	and (yyyy/mm/dd)
Name:	

Submit Clear

Figure 7.6: The workflow process successfully deployed on the ActiveBPEL workflow execution engine.



please specify the file name of output

please specify the file name of output

November 19, 2009

Figure 7.8: The service instance selection portlets for the service with rank 1 in the concurrent workflow.

7.2.4 Test Case 4: the Astronomy Workflow

This test case demonstrates a common scenario in astronomy in which astronomers seek to gain a better understanding of astronomical objects and events by comparing observations with the output of advanced numerical models. Figure 7.9 presents a scenario in the astrophysics domain, where an astrophysicist analyzes sample data from a telescope, which would normally be stored in a file system. An example of the observed data from the telescope are the celestial infrared signals from bodies in space, such as a dust clouds that surround regions of star formation. The observed data are compared with mathematical models that are intended to represent the observed astronomical object. This comparison requires the model output to be convolved with the telescope beam, the results of which can then be compared directly with the observed data. Some parameters are used as inputs to the dust cloud model to generate the model output data. To perform the convolution the Fourier transforms of the model data and the telescope beam are first calculated using a Fast Fourier transform (FFT) algorithm. The two FFT's are then multiplied together in a pairwise fashion and the inverse Fourier transform is computed to give the final convolution. As shown in Fig. 7.9, the computation of the model output and its convolution with the telescope beam represent one processing step in the data analysis cycle carried out by an astronomer.

This test case is the most complicated one and contains nine tasks in total. Three pairs of tasks work in parallel, and the data passed between these Web services is a double array, rather than a simple string, requiring an advanced assignment structure to copy each item in the array to the destination variable. This was necessary because the ActiveBPEL workflow engine is unable

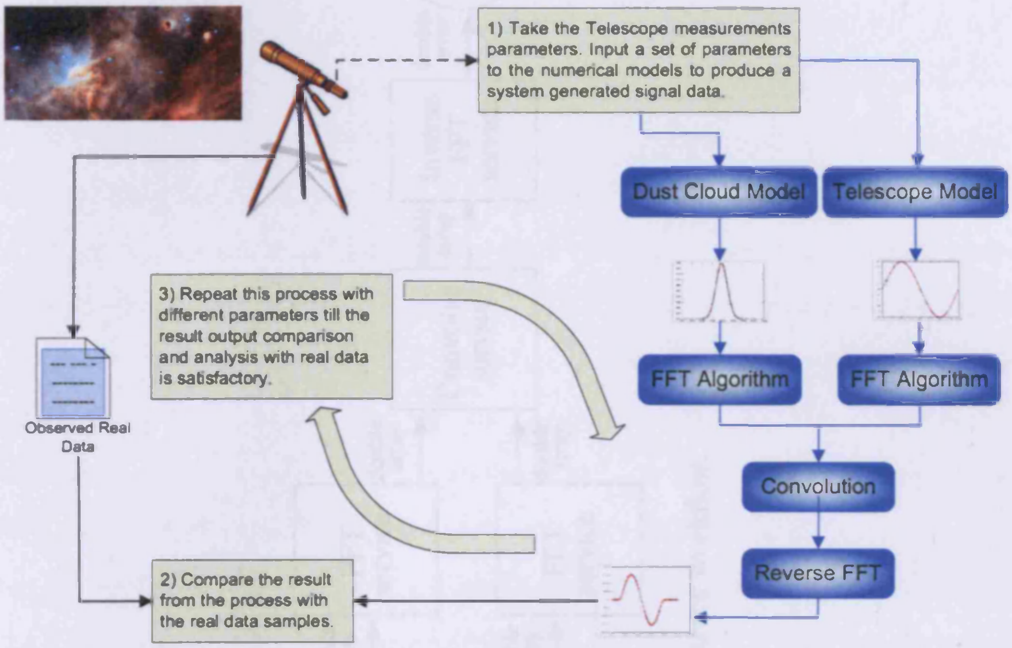

1) Take the Telescope measurements parameters. Input a set of parameters to the numerical models to produce a system generated signal data.
Dust Cloud Model
Telescope Model
FFT Algorithm
FFT Algorithm
Convolution
Reverse FFT
3) Repeat this process with different parameters till the result output comparison and analysis with real data is satisfactory.
Observed Real Data
2) Compare the result from the process with the real data samples.

Figure 7.9: Schematic representation of the astronomy scenario.

to handle the pre-defined double array XML schema. This means that the array cannot be passed as a single item, but has to be transferred one double value at a time. A similar technique could be applied to support other non-primitive data types. The test data is a double array with 10000 items, so it also evaluates the workflow engine's performance for large-scale data processing. The structure of the skeleton workflow for this test case is shown in Fig. 7.10.

In Fig. 7.10, File 0 provides the following input to the Dust Cloud service: an integer that is the number of points in the spherically symmetric dust cloud model; a double that represents the width of the dust cloud; and a string, which determines the type of dust cloud model to be used. File 1 provides the following input to the Telescope service: an integer that specifies the number of points in the telescope beam; and a string that determines the

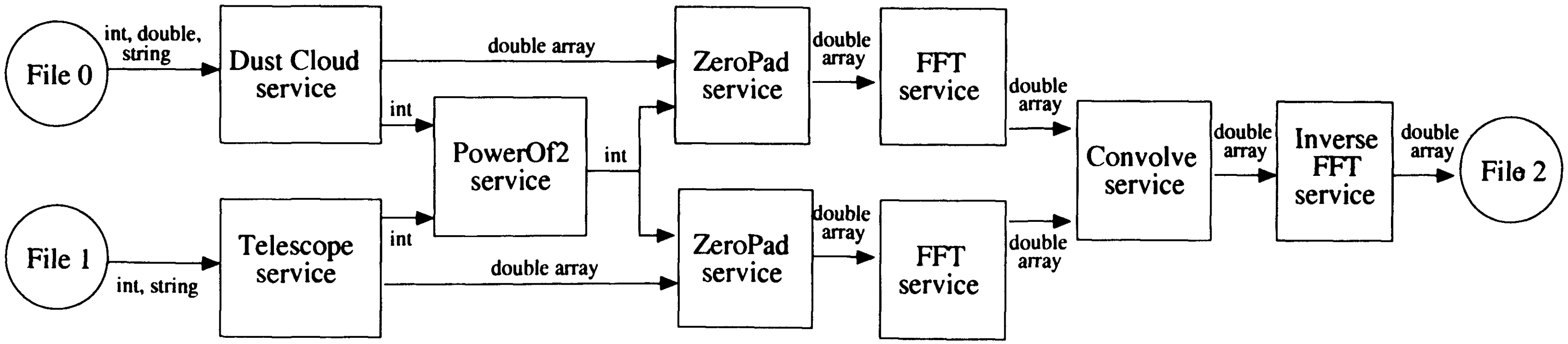


Figure 7.10: The astronomy abstract workflow.

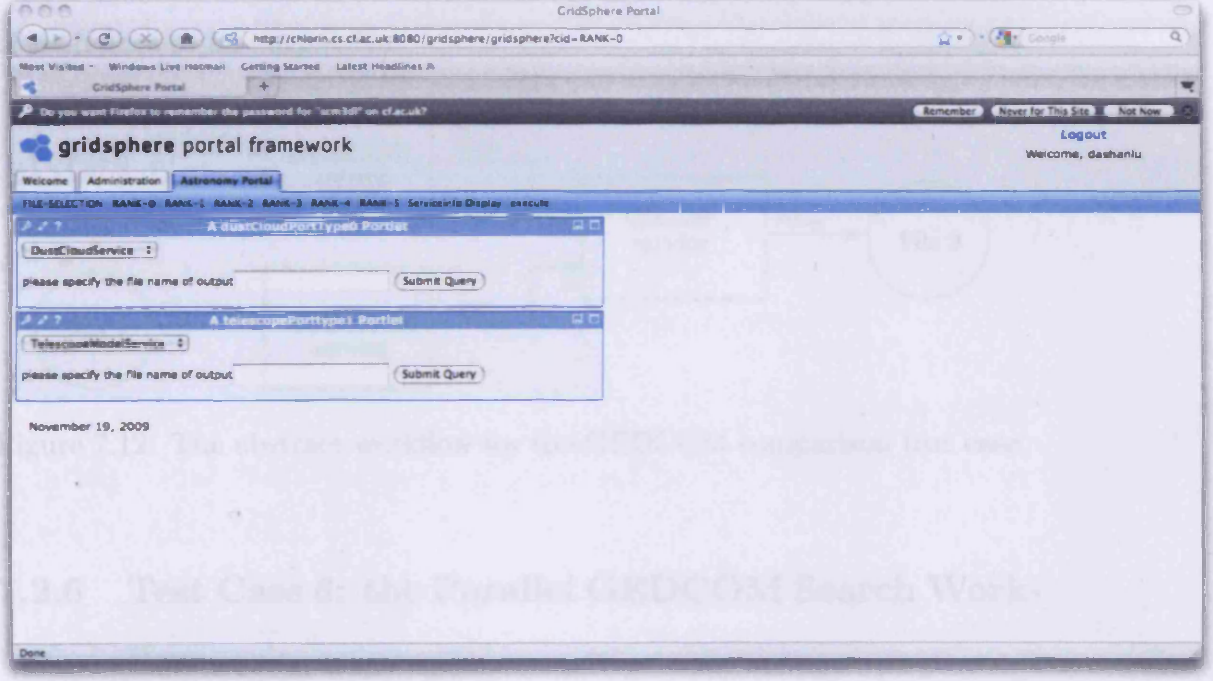
type of beam to use. Comparing the schematic illustration in Fig. 7.9 with the skeleton workflow in Fig. 7.10 it is apparent that the latter contains three extra services, namely, the PowerOf2 service and two ZeroPad services, which were omitted from the schematic for reasons of clarity. The PowerOf2 service takes as input the integers originally input to the Dust Cloud and telescope models, and outputs the smallest power of two that is greater than or equal to the larger of its inputs. This is necessary because the FFT algorithm requires its input array to be a power of two in length. The ZeroPad service then adds zeros to the arrays output by the Dust Cloud and Telescope services so that their lengths equal the value output by the PowerOf2 service. After this the FFT's of the extended Dust Cloud and Telescope arrays are evaluated by the FFT service, and the Convolve service convolves them in Fourier space. Finally, the Inverse FFT service calculates the convolution in physical space, and this is stored in File 3.

The Label Tool associates seven different service types with the nine task nodes in this more complex skeleton workflow. Some tasks are associated with the same service type and perform the same operations. The corresponding portal application generated by the Build Tool is shown in Fig. 7.11.

7.2.5 Test Case 5: the GEDCOM Comparison Workflow

The fifth test case involves a Web service, named GED2XML, made available by Genealogy Data Services Ltd[1]. The skeleton workflow in this case is actually the same as the concurrent workflow in test case 3 (see Fig. 7.7),

[1] http://www.ged2x.com, accessed 12 January 2010.



A dustCloudPortType0 Portlet

DustCloudService

please specify the file name of output

A telescopePorttype1 Portlet

TelescopeModelService

please specify the file name of output

November 19, 2009

Figure 7.11: The input file instance selection portlets for the astronomy workflow.

and the corresponding abstract workflow is shown in Fig. 7.12. In this case the two rank 0 GED2XML services both perform the same task. They each accept as input the data from a file stored in the GEDCOM format[2], which is the de facto standard for representing genealogical data. They then convert their input data to an XML format and output it. The third service, named Checker, takes these two data streams as input and compares them. The Checker service then writes the names of the individuals who appear in both of its inputs to an output file.

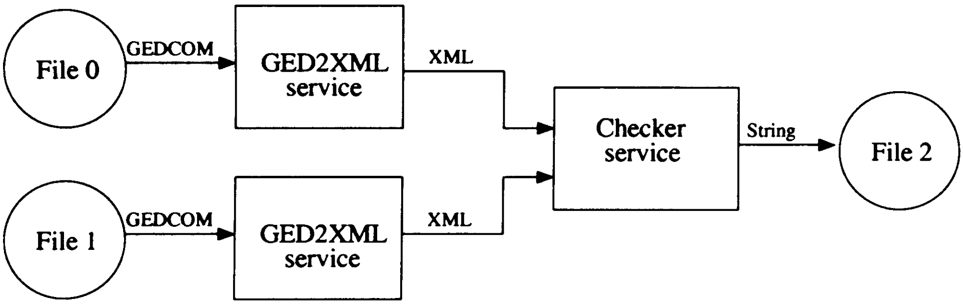


Figure 7.12: The abstract workflow for the GEDCOM comparison test case.

7.2.6 Test Case 6: the Parallel GEDCOM Search Workflow

The purpose of the sixth test case is to evaluate the stability of workflow execution with the large-scale data. This test case also involves the GED2XML Web service. The workflow application is used to check whether a person with the specific name exists in multiple GEDCOM files. The workflow contains 13 tasks in total, as shown in Fig. 7.13. The eight tasks with rank 0 are performed concurrently, and are of two different service types. As in test case 5, the services of type GED2XML take as input a GEDCOM file containing

[2] http://en.wikipedia.org/wiki/GEDCOM, accessed 12 January 2010.

genealogical data for a number of individuals, and convert it to XML which is then output. The services of type GetName take as input an XML file that includes the name to be searched for, and outputs this name as a plain string. The services of rank 1, which can also be executed concurrently, are all of the Search service type and accept as input the name string from a GetName service, and the XML data from a GED2XML service. Each Search service searches the XML data to see if it contains the specified name, and extracts from the XML data the information for any individual who matches the name. Finally, the rank 2 service of type Merge merges the lists that it receives as input and outputs the resulting merged list. Compared with the astronomy workflow case, the parameters passed between the Web services are are all strings – there are no double arrays, so a while loop in the ActiveBPEL workflow script for copying arrays is not necessary.

After Web serivce type association done by the Label Tool. The corresponding portal Web application is generated by the Building Tool is as shown in Figure 7.14.

7.3 Computing Environment for the Test Cases

The computing environment that the prototype system is built on, and that is used in the evaluation of the test cases, is made up of the following computers:

1. Chlorin, a 64-bit Linux computer running Fedora Core 5, with two 3.4GHz Pentium D processors and 2 Gbytes of memory.

2. Celeste, a 32-bit Linux computer running Fedora Core 9, with two 2.8GHz Pentium 4 processors and 1 Gbyte of memory.

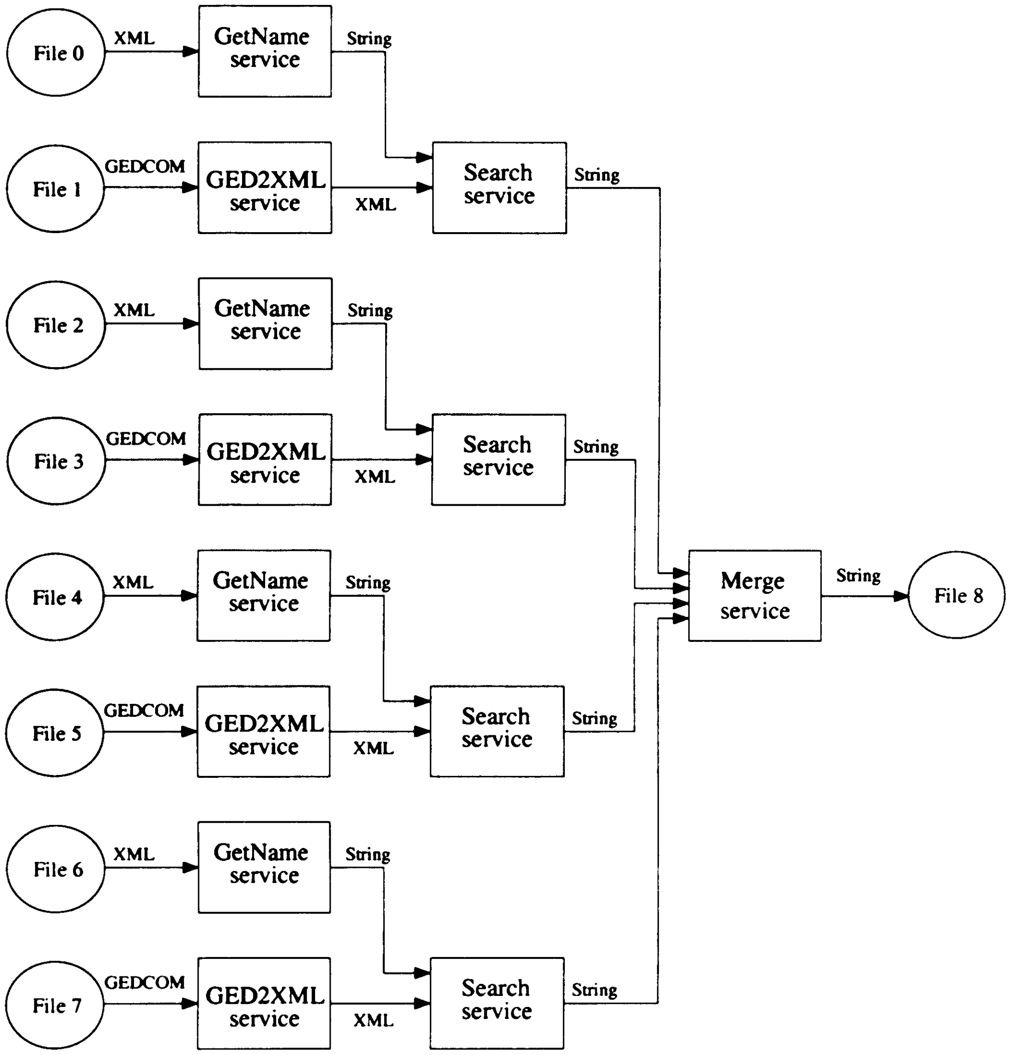


Figure 7.13: The abstract workflow for the parallel GEDCOM search test case.

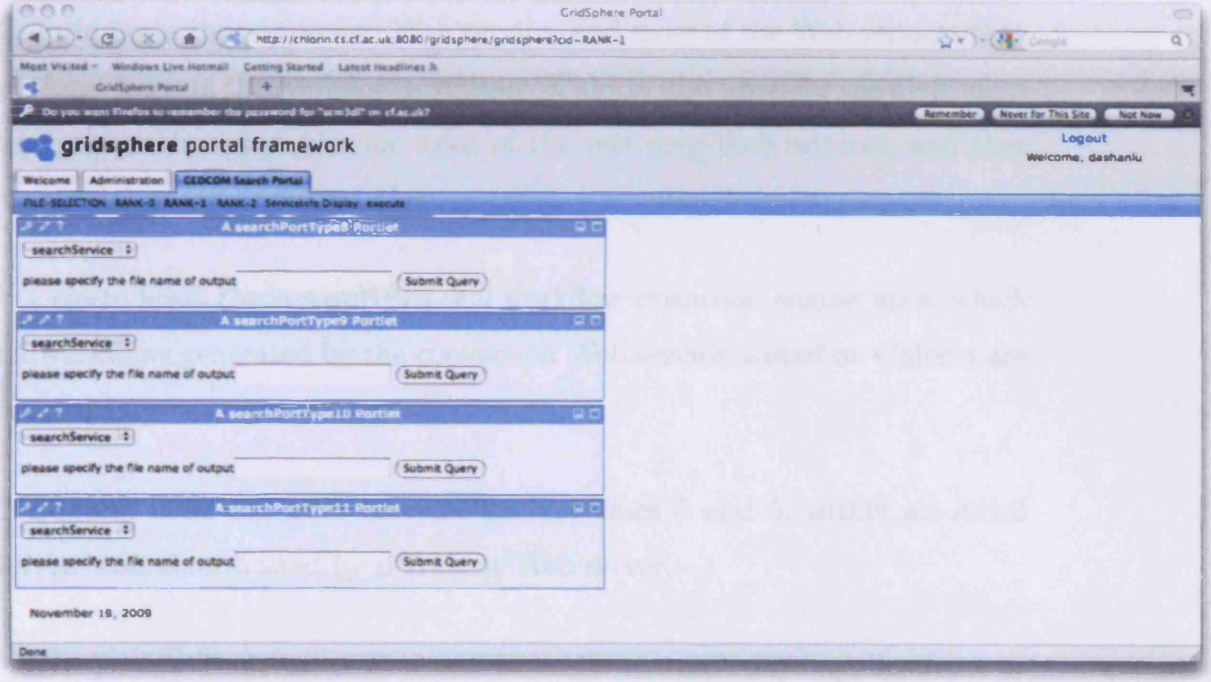

GridSphere Portal
http://chlorin.cs.cf.ac.uk:8080/gridsphere/gridsphere?cid=RANK-1
Most Visited
Windows Live Hotmail
Getting Started
Latest Headlines
GridSphere Portal
Do you want Firefox to remember the password for "scm3dl" on cf.ac.uk?
Remember
Never for This Site
Not Now
gridsphere portal framework
Logout
Welcome, dashanlu
Welcome
Administration
GEDCOM Search Portal
FILE-SELECTION
RANK-0
RANK-1
RANK-2
ServiceInfo Display
execute
A searchPortType8 Portlet
searchService
please specify the file name of output
Submit Query
A searchPortType9 Portlet
searchService
please specify the file name of output
Submit Query
A searchPortType10 Portlet
searchService
please specify the file name of output
Submit Query
A searchPortType11 Portlet
searchService
please specify the file name of output
Submit Query
November 18, 2009
Done

Figure 7.14: The service instance selection portlets for services with rank 1 in the parallel GEDCOM search workflow.

3. Bouscat, a 32-bit Linux computer running CentOS 5.4, with two 1GHz Pentium III processors and 1.5 Gbytes of memory.

Chlorin hosts the Tomcat 5.5.20 Web server, which in turn hosts the Axis and Axis2 Web service containers and the GridSphere 2.2.10 portal framework. Chlorin also hosts the MySQL 5.0.27 database system that serves as the file repository, and the UDDI Web service repository. In addition, Chlorin hosts the conversion Web service, and most of the Web services that perform tasks in the workflow test cases. Chlorin also provides the file system that stores the input files for most of the test case Web services, and that stores the workflow output files.

Celeste hosts the ActiveBPEL 4.0 workflow execution engine upon which all workflows generated by the conversion Web service hosted on Chlorin are executed.

Bouscat hosts the Web services for test cases 5 and 6, within an Axis2 service container hosted by a Tomcat Web server.

The Design Tool, Label Tool and Build Tool are all Java applications, so they can be run on any machine with a local Java Virtual Machine. The Label Tool and Build Tool need to access the Web service and file repositories to associate information with the workflow descriptions, generate portal applications, and remotely deploy portal applications, so the machines running these two tools have to have network connections.

Chapter 8

Conclusions and Future Work

This chapter presents the main conclusions and outcomes of the research described in this dissertation. Following this some reflections on the research are presented, and directions for future work are outlined.

8.1 Review

In the approach promulgated in this dissertation, a distributed application is viewed as a workflow, which is represented as a directed acyclic graph (DAG). A vertex node in this DAG is either a task or a file node. Arcs between nodes represent the flow of data from one task to another, from a task to a file, or from a file to a task.

This dissertation has described the design, implementation, and evaluation of a prototype software system that can automatically generate a portal based on the structure of an abstract workflow described in XML. The abstract workflow is composed of tasks and files, where each task node is a placeholder for a particular type of activity which is represented as a Web service, and each file node is an input or output file having a prescribed type of content. The generated portal can be used to select specific Web service and file instances to create a concrete workflow, which can be executed and monitored from within the portal on a third-party workflow execution engine.

The three different types of workflow considered in this dissertation separate out different aspects in the process of creating an executable workflow:

- A *skeleton workflow* specifies the structure of a workflow as a direct acyclic graph (DAG) made up of nodes representing files and services. The arcs in the graph represent the flow of data between a file and a service, or between one service and another. The content of the files, and the tasks carried out by a service, are not specified in a skeleton workflow.

- An *abstract workflow* specifies what the services that make up the workflow do, but does not specify particular implementations for these services.

- A *concrete workflow* specifies not only what the services in the workflow do, but also specifies their service implementations.

The process of composing the Web service based distributed application in terms of a workflow can be viewed as first creating a skeleton workflow, then converting this into an abstract workflow by associating a service or file type with each node in the skeleton workflow. Finally, each node in the abstract workflow must be associated with a specific service or file instance to become a concrete workflow, that can then be executed by a workflow engine. These three phases form a tool chain made up of the Design Tool, Label Tool and Build Tool. The Design Tool enables users to design a skeleton workflow visually. The Label Tool that accepts as input the skeleton workflow from the Design Tool and enables users to create an abstract workflow by associating service and file types with each node. The Build Tool accepts an abstract workflow as input from the Label Tool and automatically generates a portal application. This portal can then be used to create a corresponding concrete workflow, and to execute it on a workflow execution engine.

Based on the implementation and evaluation of the system described in Chapters 4–refchap:evaluation, the prototype system fully addresses all requirements discussed the Chapter 3. The user can use the tool chain to design, generate and execute their customized Web service-based distributed application through a Web browser.

8.2 Conclusions

The main conclusions and contributions of the research presented in this dissertation are as follows:

1. It has been shown that a portal application can be automatically generated based on an abstract workflow description, as described above. The portal generator, comprised of the Build Tool and related software, provides a simple and convenient way for users who have no experience in portal and Web service technologies to create and configure a portal application, thereby providing a large class of scientific applications with a user-friendly problem-solving environment for building, executing and managing distributed applications composed of Web services. This ease-of-use is one of the main motivations for the research described in this dissertation.

2. Existing workflows can be readily edited to change their structure and/or function. Having created a modified abstract workflow a user then just need to re-execute the Build Tool to generate a new Web-service-based workflow application.

3. The prototype system discussed in this dissertation is made up of generic lightweight tools that focus on performing small clearly-defined tasks, and that interoperate with third-party workflow tools. Thus, although the Design Tool and Label Tool have been used to create an abstract workflow, other third-party workflow design tools could also be used to do this. Provided that the workflow is described by an XML document, then an XSLT script can be used to convert it into a skeleton workflow compatible with the XML Schema used by the Design Tool.

4. Although the prototype system makes use of the GridSphere portal framework and the ActiveBPEL workflow execution engine, the dependencies on these components are restricted to small parts of the overall system. The use of other portal frameworks that comply with the JSR-168 portlet specification just requires that different portal configuration files be generated by the Build Tool. This would require writing new XSLT scripts to produce XML portal configuration files analogous to those described in Section 6.6.4. Similarly, a different third-party workflow execution engine could be used by creating a new conversion Web service. The conversion Web service, described in 6.5 for the ActiveBPEL workflow engine, takes the description of the concrete workflow specified by the user in the portal and generates the files required to deploy and execute the workflow on a particular workflow

engine.

8.3 Reflections on Research Challenges

This section makes some observations on some of the difficulties and challenges faced in carrying out the research in this dissertation.

An important design decision made quite early in the research was to use a third-party workflow engine to execute the concrete workflows specified in the portal. This was done to show how the portal, generated automatically by the Build Tool, could interoperate with such workflow engines, and the ActiveBPEL engine was chosen because it is open-source, freely available, with a large community of users. This approach is different from that used, for example, in the GECEM project [13] where the portal itself drives the execution of the workflow through a harness program that invokes the Web services in the correct order and manages the flow of data between workflow nodes. The latter approach would have been more self-contained, and would have avoided any dependencies on third-party software.

One issue that caused some difficulties in the implementation phase of the research was that of Web service timeouts. A Web service that is part of a workflow may be long running because it does a large amount of data processing or because a large amount of data must be transferred to/from it. As mentioned in Chapter 7, test cases 5 and 6 both contain a Web service (the GED2XML service) that does a large amount data processing (for sufficiently large input data), and hence can take a long time to run, dominating the execution time of the workflow. HTTP is used as the protocol for interacting with Web services and once the time taken by the workflow execution reaches the HTTP limit, the connection between the portal and third-party workflow execution engine is dropped, and the execution fails. There are two approaches to solve the lose of the HTTP connection. The simplest solution is to set the HTTP connection to have no time limit, so that the connection is always kept open until the Web service finishes its work. Alternatively, a more complicated approach is to make use of asynchronous Web service invocation. The details of asynchronous Web service invocation are introduced in the next section on future work.

Checkpointing is another aspect of the optimization of workflow execution. Checkpointing allows a workflow to be started from some intermediate point, rather than starting from the beginning by invoking the services with rank 0. This is particularly useful, and saves time, when a workflow has previously failed before completion. Checkpointing requires a consistent set of data files to be saved so that execution can restart from the corresponding intermediate point in the workflow. In the event of a failure in the execution of a workflow, without checkpointing the entire workflow would have to be re-executed after correcting the cause of the failure. It is obviously inconvenient and inefficient for users. Some support for checkpointing is provided in the portlets for service instance selection (see Fig. 7.5). Additional support for checkpointing is considered in the next section.

An expected problem was encountered when designing test case 6 (see Section 7.2.6). Originally it was intended to use a workflow as shown in Fig. 8.1. This workflow has a Splitter service that accepts an XML file as input, splits it into several parts, and outputs these parts on different ports. The output from the GetName service is replicated four times to pass the string input name to each of the Search services. This workflow was found to be problematical because of the way the ActiveBPEL engine operates. A basic Web service operation returns a single object — in the Splitter service the Split operation returns a part of the input XML file. Thus, to achieve the intention expressed in Fig. 8.1 the Splitter service would actually need to be invoked multiple times, with each invocation outputting a different part of the XML file. The ActiveBPEL engine is unable to internally buffer the large amounts of data needed to support multiple output ports in this workflow. Invoking the Splitter service multiple times means that the input data has to be transferred multiple times, which for large input data is very inefficient. Alternatively, if the input XML data is not too large, ActiveBPEL could be instructed to store a long string to represent the XML stream. The Splitter service would still need to be invoked multiple times to chop this string into smaller pieces, so the long string would need to be read multiple times, but the need for reads the input file multiple times is avoided.

One alternative approach is to not use the Splitter service at all and to pass the whole XML file to each of the Search services in Fig. 8.1, together with a sequence number that tells the service which part of the file it should operate on. Another approach is that actually adopted in test case 6, and shown in Fig. 7.13, in which the input file is divided manually before workflow execution.

Another challenge that arose in the course of the evaluation was the need to replicate the output of a service. For example, in the astronomy workflow shown in Fig. 7.10 the output of the PowerOf2 service is replicated and passed as input to each of the two ZeroPad services. Similarly, in Fig. 8.1 the output of the GetName service is replicated four times. This type of replication is achieved by storing the output as a variable in the ActiveBPEL engine, and

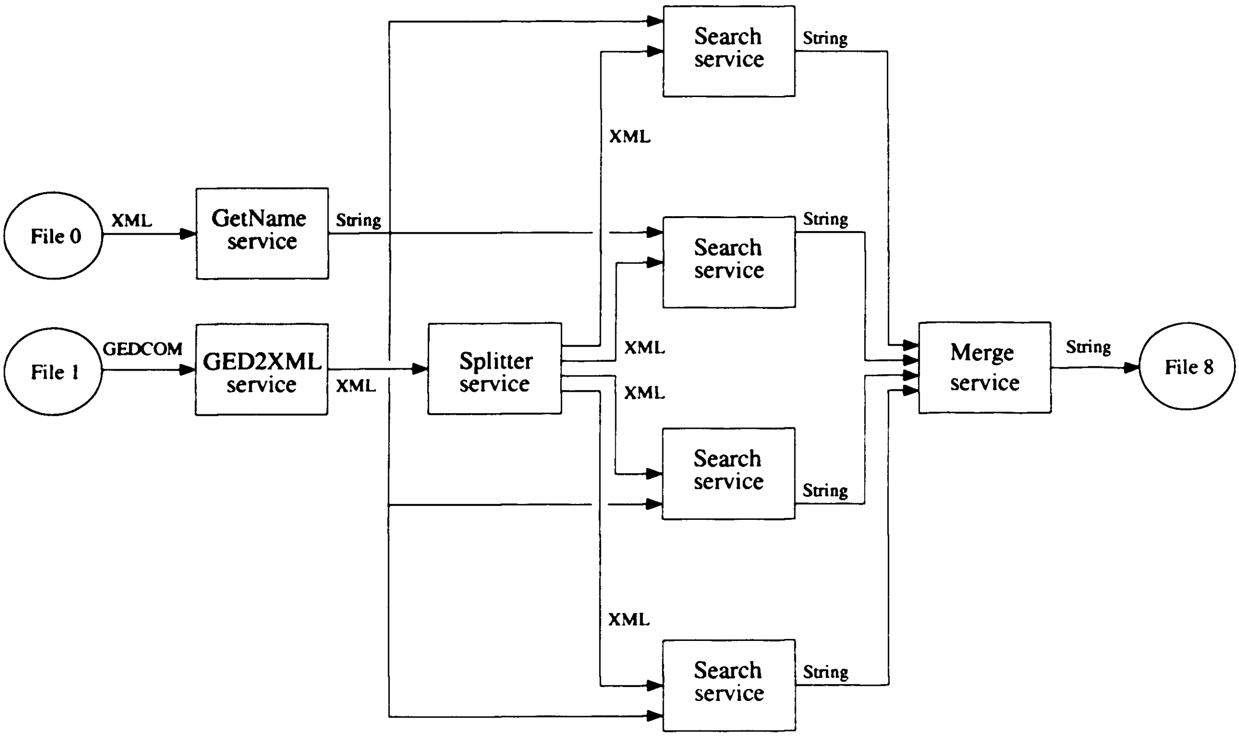


Figure 8.1: The abstract workflow for the parallel GEDCOM search test case incorporating the Splitter service.

then passing it to each of the downstream services.

It proved unexpectedly difficult to find workflows composed of Web services that could be executed remotely by members of the research community. Originally it was intended to perform the evaluation of the prototype system using workflows in the myExperiment workflow repository[1]. However, at the time the evaluation phase of the research was being planned none of the workflows in the repository could be run, either because they included Web services on inaccessible hosts, or because they included local code that was not exposed as a Web service.

8.4 Future Work

The evaluation in Chapter 7 has tested the individual operation of each component of the system. Each component was found to work properly and to address the basic requirements detailed in Chapter 3.

- Design Tool
  - Drawing the skeleton workflow by dragging and dropping GUI components around a canvas
  - Save the skeleton workflow as XML-format file with a specific XML schema
- Labeling Tool
  - Load a skeleton workflow file
  - Retrieve the information from the service repository
  - Associate service type with each task node

[1] http://www.myexperiment.org/workflows, accessed 12 January 2010.

  - Save the new workflow with more information to abstract workflow

- Building Tool
  - Convert the abstract workflow to a corresponding portal Web application
  - Compress all required files into a WAR file

- Portal Web application
  - Select file instances for initialization of workflow execution
  - Select Web Service instances according to Web service type information
  - Invoke a Web Service to generate and deploy the ActiveBPEL workflow script
  - Execute the workflow process

Currently the Label Tool is capable of connecting to only one service repository. The URL of the service repository is pre-defined at design time. A useful future extension of the Label Tool would be support for multiple service repositories, so that the service repository used by the Label Tool is selected by the user. The Label Tool could retrieve all available service repositories from a database, and list them to the user who chooses one or more service repositories that the Label Tool then connects with.

In this dissertation the focus has been on workflows for scientific computing. Thus, the prototype system does not handle complex[2] datatype parsing

[2] *Complex* here means non-primitive Java data types, and not just a data type for complex numbers.

and labeling. This is because most scientific computing is based on floating-point numbers. The datatypes supported by the Label Tool are all primitive datatypes defined by Java and arrays of these primitive datatypes. The Label Tool can be extended to support labeling with complex datatypes by adding XML schemas and advanced WSDL parsers.

The GUI for the Label Tool uses a table structure to display the information about the workflow and Web service types stored in the service repository. Sometimes it is not very easy for a user to label each task with a service type when the number of tasks is large. A future improvement would be make the interface of the Label Tool to be more like the Design Tool so that users can label task nodes with service types more conveniently.

The portal application generated by the Build Tool is only compatible with the GridSphere portal framework. Users should have a choice to put the workflow portal on other portal frameworks, such as JetSpeed[3] and Pluto[4]. This would require the development of several sets of XSLT stylesheets to support deployment on multiple portal frameworks. Other future work to improve the Build Tool includes extending the Build Tool to support access to abstract workflows through URLs and workflow repositories.

As mentioned in Section 8.3, workflows that contain long-running services may fail because of an HTTP timeout. The simplest solution to this problem is to set the HTTP connection to have no time limit, so that the workflow execution engine waits indefinitely for the response from a Web service. The advantage of this approach is that it is relatively easy to implement for most

[3] http://portals.apache.org/jetspeed-2/, accessed 12 January 2010.

[4] http://portals.apache.org/pluto/, accessed 12 January 2010.

workflow engines, and there is no need to make any changes to existing Web services, but there is a serious disadvantage — if a Web service invocation fails for any reason the workflow execution halts immediately, because the execution engine does not know that the Web service has failed and waits forever the Web service's response. An alternative solution is to use asynchronous Web service invocation based on dual HTTP channels. When a client invokes a Web service asynchronously, it passes the request to the Web service with one HTTP channel and receives the response with another one. When the Web service host receives the request from the client it creates and stores a message ID for the request. When the Web service finishes its work a response is sent back to the client with the message ID. The advantage of asynchronous invocation is that the workflow execution will not halt without any error message, because even if a Web service invocation fails, the the Web service host can still send an error report to the workflow execution engine, and performs operations to compensate for the failure of the Web service invocation. The disadvantage of this approach is that it is workflow engine dependent, and not all workflow engines support asynchronous invocation. Furthermore, and the Web services must also support asynchronous invocation mode.

Although the prototype system has some support for checkpointing, this is an area that if would be beneficial to develop in the future. Checkpointing permits a workflow to be started (or re-started) from an intermediate point. This is particularly useful if a long-running workflow fails. In a workflow, each Web service can be checkpointed by storing its output, together with details of the Web service, such as its location in the workflow. This information allows a user to select another Web service instead of the failed one, and resume the rest of the workflow. The execution of workflow carries on from the state at which the failure occurred. A simple algorithm can be used to determine how to restart a workflow for which some, or all, of the Web service outputs have been checkpointed — each checkpointed service has its input links removed from the workflow and is viewed as a file node, i.e., it is replaced by the file(s) that stores its output(s). To support checkpointing in this way in workflow execution, the workflow engine has to be customized, or the portal application has to drive the workflow execution directly through its own embedded workflow execution engine that also incorporates a checkpoint manager.

Bibliography

[1] Ian Foster and Carl Kesselman. *The Grid Blueprint for a New Computing Infrastructure (Second Edition).* Morgan Kaufmann, 2003.

[2] Computer bus, 2009. `http://en.wikipedia.org/wiki/Computer_bus`.

[3] Y. Huang. GSiB: PSE infrastructure for dynamic service-oriented grid applications. *Future Generation Computer Systems*, 18(6):868–877, 2005.

[4] Maozhen Li and Mark Baker. A review of grid portal technology. In Jose C. Cunha and Omer F. Rana, editors, *Grid Computing: Software Environments and Tools*, pages 126–156. Springer, 2006.

[5] Michael N. Huhns and Munindar P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

[6] Markus Aleksy, Axel Korthaus, and Martin Schader. *Implementing Distributed Systems with Java and CORBA*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[7] Fintan Bolton. *Pure Corba.* Sams Publishing, 2001.

[8] William Grosso. *Java RMI.* O'Reilly Media, 2001.

[9] Sean Baker. Web services and corba. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, Lecture Notes in Computer Science, pages 618–632. Springer, 2009.

[10] Steve Graham, Simeon Simeonov, Toufic Boubez, Doug Davis, Glen Daniels, Yuichi Nakamura, and Ryo Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI.* Sams Publishing, 2002.

[11] Steve Graham, Simeon Simeonov, Toufic Boubez, Doug Davis, Glen Daniels, Yuichi Nakamura, and Ryo Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI.* Sams Publishing, 2002.

[12] OASIS Web Services Business Process Execution Language Technical Committee. Web Services Business Process Execution Language Version 2.0. Available at `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`, April 2007.

[13] Maria Lin and David W.Walker. GECEM: A portal-based Grid application for computational electromagnetics. *Future Generation Computer System*, Vol. 24:66–72, 2008.

[14] D. W. Walker, J. P. Giddy, N. P. Weatherill, J. W. Jones, A. Gould, D. Rowse, and M.Turner. GECEM: Grid-enabled Computational Electromagnetics. *in Processings of the UK e-Science All Hands Meeting*, pages 436–443, 2003.

[15] Tomasz Haupt and Anand Kalyanasundaram. Cooperative grid vortals. *Concurrency and Computation: Practice and Experience*, 19(12):1671–1681, 2007.

[16] T. Delaitre, A. Goyenche, P. Kacsuk, T. Kiss, G.Z. Terstyanszky, and S.C. Winter. Gemlca*: Grid execution management for legacy code architecture design. In *Proceeding of the 30th EUROMICRO Conference*, 2004.

[17] T. Kiss, G. Terstyanszky, G. Kecskemeti, Sz. Illes, T. Delaittre, S. Winter, P. Kacsuk, and G. Sipos. Legacy code support for production grids. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 278–283. IEEE Computer Society, 2005.

[18] Csaba Nemeth, Gabor Dozsa, Robert Lovas, and Peter Kacsuk. The P-Grade grid portal. In *Computational Science and Its Applications ICCSA 2004*, volume 3044, pages 10–19. Springer Berlin / Heidelberg, 2004.

[19] Ruxandra Bondarescu, Gabrielle Allen, Gregory Daues, Ian Kelley, Michael Russel, Edward Seidel, John Shalf, and Malcolm Tobias. The astrophysics simulation collaboratory portal: a framework for effective distributed research. *Future Generation Computer Systems*, 21(2):259–270, February 2005.

[20] Yong Zhao, Michael Wilde, Ian Foster, Jens Voeckler, James Dobson, Eric.Gilbert, Thomas.Jordan, and Elizabeth.Quigg. Virtual data Grid middleware services for data-intersive science. *Concurrency and Computation: Practice and Experience*, Vol. 18:595–608, May 2006.

[21] I. Foster, J. Vockler, M. Wilde, and Y. Zhao. Chimera: a virtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, pages 37– 46, 2002.

[22] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, and Miron Livny. Pegasus: Mapping scientific workflows onto the grid. In *Grid Computing*, volume 3165, pages 11–20. Springer Verlag, Heidelberg, 2004.

[23] Rizos Sakellariou, Henan Zhao, and Ewa Deelam. Mapping workflows on grid resources: Experiments with the montage workflow. In *Proceedings of the CoreGRID ERCIM Working Group Workshop on Grids, P2P and Service Computing*, 2009. To be published by Springer in the LNCS series.

[24] Erol Akarsu, Geoffrey C. Fox, Wojtek Furmanski, and Tomasz Haupt. Webflow: high-level programming environment and visual authoring toolk it for high performance distributed computing. In *Proceedings of the ACM/IEEE Supercomputing Conference*, pages 1–7, Washington, DC, USA, 1998. IEEE Computer Society.

[25] Junwei Cao, Jarvis, S.A Saini, and S. Nudd. Gridflow: workflow management for grid computing. In *Proceedings of the Third IEEE/ACM International Symposium on Cluster Computing and the Grid2003.*, pages 198–205, 2003.

[26] Peter Kacsuk, Gabor Dozsa, Tibor Fadgyas, and Robert Lovas. The gred graphical editor for the grade parallel program development environment. In *High-Performance Computing and Networking*, volume 1401, pages 728–737. Springer Verlag, Heidelberg, 1998.

[27] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam. *Parallel Virtual Machine: A Users Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1994.

[28] Tom Oinn, Mark Greenwood, Matthew Addis, M.Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter Li, Phillip Lord, Matthew R.Pocock, Martin Senger, Robert Stevens, Anil Wipat, and Chris.Wroe. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. *Concurrency and Computation: Practice and Experience*, Vol. 18:1067–1100, 2005.

[29] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jeager, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience*, Vol. 18:1039–1065, 2006.

[30] I. Taylor, M. Shields, I. Wang, and A. Harrison. Visual Grid Workflow in Triana. *Journal of Grid Computing*, 3:153–169, September 2005.

[31] Lican Huang, Asif Akram, Rob Allan, David W. Walker, Omer F. Rana, and Yan Huang. A workflow portal supporting multi-language interoperation and optimization. *Concurrency and Computation: Practice and Experience*, 19:1583–1595, June 2007.

[32] David W. Walker and Dashan Lu. Automatic Portal Generation Based on Workflow Description, in Proceeding of the CoreGRID Integration Workshop. In preparation, 2009.

[33] David W. Walker. Lessons learned from the GECEM project. In *Grid-Based Problem Solving Environments*, volume 1401, pages 95–111. Springer Verlag, Heidelberg, 2007.

[34] Kim Topley. *Core SWING advanced programming.* Prentice Hall, 2000.

[35] Timothy Budd. *Understanding Object-Orented Programming With Java.* Addison Wesley, 1999.

Appendix A

XSLT Stylesheets for Automatic Portal Generation

A.1 Stylesheet to Generate Java Code for Service Instance Selection Portlets

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:variable name="uuid" select="ServiceType/@ServiceTypeID"/>
    <xsl:variable name="name" select="@name"/>
    <xsl:variable name="ID" select="@ID"/>
    <xsl:variable name="nameOfPortlet" select="concat($name,$ID)"/>
    <xsl:output method="text" indent="yes"/>
    <xsl:template match="task">
        <xsl:call-template name="importLibraries"/>
        <xsl:text>public class </xsl:text>
        <xsl:value-of select="$nameOfPortlet"/>
        <xsl:text>Portlet </xsl:text>
        <xsl:text>extends GenericPortlet{</xsl:text>

        <xsl:call-template name="declareVariable"/>
        <xsl:call-template name="init"/>
        <xsl:call-template name="doView"/>
        <xsl:call-template name="processAction"/>
        <xsl:call-template name="destroy"/>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>}</xsl:text>
    </xsl:template>

    <xsl:template name="importLibraries">
        <xsl:text>//import UDDI configuration</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import java.util.Properties;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import org.uddi4j.transport.TransportFactory;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>//import UDDI4J packages for the UDDI query</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import org.uddi4j.client.UDDIProxy;</xsl:text>
```

```
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.response.BusinessList;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.response.BusinessInfos;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.response.BusinessInfo;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.response.ServiceInfo;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.response.ServiceInfos;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.response.ServiceList;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.response.TModelList;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.response.TModelDetail;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.util.TModelBag;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.util.TModelKey;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.util.CategoryBag;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.util.KeyedReference;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.response.TModelInfo;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.datatype.tmodel.TModel;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.transport.TransportException;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import org.uddi4j.*;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>//the packages for the network facility</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import java.net.URL;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import java.net.URLConnection;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import java.net.MalformedURLException;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>//packages for the vector</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import java.util.*;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>//import the package for the portlet</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import javax.portlet.GenericPortlet;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import javax.portlet.PortletException;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import javax.portlet.RenderRequest;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import javax.portlet.RenderResponse;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import javax.portlet.PortletRequestDispatcher;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import javax.portlet.PortletContext;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import javax.portlet.PortletURL;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text>import javax.portlet.PortletConfig;</xsl:text>
```

```
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.ActionRequest;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.ActionResponse;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.UnavailableException;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletSession;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import java.io.*;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    </xsl:template>

    <xsl:template name="declareVariable">
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>private final String serviceSelecJsp="/jsp/</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$nameOfPortlet"/>
␣␣␣␣␣␣␣␣<xsl:text>Select.jsp";</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>private PortletSession </xsl:text>
        <xsl:value-of select="$nameOfPortlet"/>
        <xsl:text>Session;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>private UDDIProxy proxy;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>private BusinessList businessList;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>private ServiceList servicelist;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>private CategoryBag cb;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>private TModelDetail td;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>private TModel tm;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>private TModelKey tmk;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>private String tmn;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>private String uuid;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    </xsl:template>

    <xsl:template name="init">
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>public void init(PortletConfig config)throws
                UnavailableException, PortletException{</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>super.init(config);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text>//system setting</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text>System.setProperty(TransportFactory.PROPERTY_NAME,
                "org.uddi4j.transport.ApacheAxisTransport");</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text>//create an proxy instance</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text>proxy=new UDDIProxy();</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text>try{</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text>proxy.setInquiryURL(
                "http://chlorin.cs.cf.ac.uk:8080/juddi/inquiry");</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

        <xsl:text>proxy.setPublishURL(
                "http://chlorin.cs.cf.ac.uk:8080/juddi/publish");</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

        <xsl:text>businessList=proxy.find_business("pupa",null,0);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

        <xsl:text>Vector businessInfoVector=
                businessList.getBusinessInfos().getBusinessInfoVector();
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

        <xsl:text>TModelBag tb=new TModelBag();</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

        <xsl:text>uuid="</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$uuid"/>
␣␣␣␣␣␣␣␣<xsl:text>";</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

        <xsl:text>tmk=new TModelKey(</xsl:text>
            <xsl:text>uuid</xsl:text>
        <xsl:text>);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text>tmn="</xsl:text >
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$nameOfPortlet"/>
␣␣␣␣␣␣␣␣<xsl:text >";</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

        <xsl:text>tb.add(tmk);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

        <xsl:text>servicelist=proxy.find_service(
                 "03927110-4F02-11DC-B110-C30F49B92B9C",tb,null,10);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text>}catch(MalformedURLException mue){</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text>}catch(UDDIException ue){</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text>}catch(TransportException te){</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text>}</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>}</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    </xsl:template>

    <xsl:template name="doView">
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>public void doView(RenderRequest request,
                 RenderResponse response)throws PortletException, IOException{
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>PortletURL url=response.createActionURL();</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>response.setContentType("text/html");</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>request.setAttribute("serviceType",uuid);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>request.setAttribute("servicelist",servicelist);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>request.setAttribute("url",url.toString());</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>PortletContext portletContext=getPortletContext();</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>PortletRequestDispatcher prd=
                portletContext.getRequestDispatcher(serviceSelecJsp);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>prd.include(request, response);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>}</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    </xsl:template>

    <xsl:template name="processAction">
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>public void processAction(ActionRequest request,
                ActionResponse response){</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:value-of select="$nameOfPortlet"/>
        <xsl:text>Session=request.getPortletSession();</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>String fileName="null";</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>try{</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text>fileName=(String)request.getParameter("fileName");</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>}catch(Exception e){</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>}</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>String service=(String)request.getParameter("services")+
                "="+fileName;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:value-of select="$nameOfPortlet"/>
        <xsl:text>Session.setAttribute("</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$nameOfPortlet"/>
␣␣␣␣␣␣␣␣<xsl:text>service</xsl:text>
␣␣␣␣␣␣␣␣<xsl:text>",service,PortletSession.APPLICATION_SCOPE);</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>}</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    </xsl:template>

    <xsl:template name="destroy">
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>public void destroy(){</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:value-of select="$nameOfPortlet"/>
        <xsl:text>Session</xsl:text>
        <xsl:text>=null;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>}</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    </xsl:template>
</xsl:stylesheet>
```

A.2 Stylesheet to Generate JSP page for Service Instance Selection Portlet

```
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:variable name="uuid" select="ServiceType/@ServiceTypeID"/>
    <xsl:variable name="nameOfPortlet" select="@name"/>
    <xsl:output method="text" indent="yes"/>
    <xsl:template match="task">
        <xsl:call-template name="header"/>


        <xsl:variable name="uuid" select="./ServiceType/@ServiceTypeID"/>
        <xsl:variable name="id" select="./@ID"/>

        <xsl:text disable-output-escaping="yes">&lt;html&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">&lt;%String url=
                (String)request.getAttribute("url");</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>String serviceType=(String)request.getAttribute("serviceType");</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>ServiceList sl=(ServiceList)request.getAttribute("servicelist");</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>ServiceInfos sis=sl.getServiceInfos();</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">ServiceInfo si;%&gt;</xsl:text>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;form action="&lt;%=url%&gt;"
                method="post"&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;select name="services"&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;%for(int i=0;i&lt;sis.size();i++){
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#09;&#x09;</xsl:text>
        <xsl:text>si=sis.get(i);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">%&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;option value="&lt;%=serviceType+"="+si.getServiceKey()%>"
                &gt;&lt;%=si.getDefaultNameString()%&gt;&lt;/option&gt;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;%}%&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/select&gt;</xsl:text>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;p&gt;</xsl:text>
        <xsl:text>please specify the file name of output</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;input type="text"
                name="fileName"&gt;</xsl:text>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;input type="submit"/&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/form&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/html&gt;</xsl:text>
    </xsl:template>

    <xsl:template name="header">
            <xsl:text disable-output-escaping="yes">
                    &lt;%@page import="org.uddi4j.response.ServiceList"%&gt;
            </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;%@page import="org.uddi4j.response.ServiceInfo"%&gt;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;%@page import="org.uddi4j.response.ServiceInfos"%&gt;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    </xsl:template>
</xsl:stylesheet>
```

A.3 Stylesheet to Generate Java code for File Instance Selection Portlet

```
<xsl:stylesheet version=”1.0” xmlns:xsl=”http://www.w3.org/1999/XSL/Transform”>
    <xsl:output method=”text” indent=”yes”/>

    <xsl:template match=”/”>
        <xsl:call-template name=”importLibraries”/>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>public class ServicesDisPortlet extends GenericPortlet{</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:call-template name=”declareVariable”/>
        <xsl:call-template name=”init”/>
        <xsl:call-template name=”doView”/>
        <xsl:call-template name=”processAction”/>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>}</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
    </xsl:template>

    <xsl:template name=”importLibraries”>
        <xsl:text>import javax.portlet.GenericPortlet;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletException;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import javax.portlet.RenderRequest;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import javax.portlet.RenderResponse;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletRequestDispatcher;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletContext;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletURL;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletConfig;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletSession;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import javax.portlet.ActionRequest;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import javax.portlet.ActionResponse;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import javax.portlet.UnavailableException;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import java.io.IOException;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>//import self-defined library</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import dashanlu.a2bpel.A2BpelPT;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text>import dashanlu.a2bpel.A2BpelServiceLocator;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
    </xsl:template>

    <xsl:template name=”declareVariable”>
        <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
        <xsl:text disable-output-escaping=”yes”>&#x09;</xsl:text>
        <xsl:text>public final String serviceDisJsp=”/jsp/serviceDis.jsp”;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private PortletSession servicesdissession;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private String result="empty";</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
</xsl:template>

<xsl:template name="init">
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>public void init(PortletConfig config)throws UnavailableException,
            PortletException{</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>super.init(config);</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>}</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
</xsl:template>

<xsl:template name="doView">
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">protected void doView(RenderRequest request,
            RenderResponse response)throws PortletException, IOException{</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            PortletURL url=response.createActionURL();
    </xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            request.setAttribute("result",result);
    </xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            request.setAttribute("url",url.toString());
    </xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            response.setContentType("text/html");
    </xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            PortletContext portletContext=getPortletContext();
    </xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
```

```
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
              PortletRequestDispatcher prd=portletContext.getRequestDispatcher(serviceDisJsp);
        </xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">prd.include(request,response);</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">}</xsl:text>

    </xsl:template>

    <xsl:template name="processAction">
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>public void processAction(ActionRequest request,
                ActionResponse response)throws PortletException, IOException{</xsl:text>

        <xsl:for-each select="//task[@rank=0]">
            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
            <xsl:text>String </xsl:text>
            <xsl:value-of select="./@name"/>
            <xsl:value-of select="./@ID"/>
            <xsl:text>File</xsl:text>
            <xsl:text>=(String)request.getParameter("</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="@name"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="@ID"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>File</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>");</xsl:text>
        </xsl:for-each>

        <xsl:for-each select="//task">
            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
            <xsl:text>String </xsl:text>
            <xsl:value-of select="@name"/>
            <xsl:value-of select="@ID"/>
            <xsl:text>service</xsl:text>
            <xsl:text>=(String)request.getParameter("</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="@name"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="@ID"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>service");</xsl:text>
        </xsl:for-each>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>
                String abstractWorkflow=(String)request.getParameter("abstractWorkflow");
        </xsl:text>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>String[] initparam={</xsl:text>

        <xsl:for-each select="//task[@rank=0]">
            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
            <xsl:value-of select="@name"/>
            <xsl:value-of select="@ID"/>
            <xsl:text>File ,</xsl:text>
        </xsl:for-each>
        <xsl:text>};</xsl:text>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>String[] params={</xsl:text>

        <xsl:for-each select="//task">
            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
            <xsl:value-of select="@name"/>
            <xsl:value-of select="@ID"/>
            <xsl:text>service ,</xsl:text>
        </xsl:for-each>
        <xsl:text>};</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>String abstractWorkflowLocation=abstractWorkflow;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>try{</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text>A2BpelPT a2b=(new A2BpelServiceLocator()).geta2bpelService();</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text>result=a2b.creatBpelFiles(params,abstractWorkflowLocation);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>}catch(Exception e){</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text>response.setRenderParameter("errors", e.getMessage());</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text>result=e.toString();</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>}</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>servicesdissession=request.getPortletSession();</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>servicesdissession.setAttribute("generationResult",result,
                PortletSession.APPLICATION_SCOPE);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text>servicesdissession.setAttribute("initparam",initparam,
                PortletSession.APPLICATION_SCOPE);</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>}</xsl:text>
    </xsl:template>
</xsl:stylesheet>
```

A.4 Stylesheet to Generate JSP page for File Instance Selection Portlet

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text" indent="yes"/>
    <xsl:variable name="filelocation" select="//workflow/@location"/>
    <xsl:template match="/">
        <xsl:call-template name="header"/>
        <xsl:call-template name="body"/>
    </xsl:template>

    <xsl:template name="header">
        <xsl:text>&lt;%@page session="true" contentType="text/html"%&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>&lt;%String url=(String)request.getAttribute("url");%&gt;</xsl:text>
    </xsl:template>

    <xsl:template name="body">
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>&lt;html&gt;</xsl:text>

        <xsl:for-each select="//task[@rank=0]">
            <xsl:variable name="ID" select="@ID"/>
            <xsl:variable name="name" select="@name"/>
            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
            <xsl:text>&lt;%String </xsl:text>
            <xsl:value-of select="$name"/>
            <xsl:value-of select="$ID"/>
            <xsl:text>File</xsl:text>
            <xsl:text>=(String)session.getAttribute("</xsl:text>
____________<xsl:value-of_select="$name"/>
____________<xsl:value-of_select="$ID"/>
____________<xsl:text>File");%&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                  &lt;p&gt;This file LOCATION is &lt;%=
            </xsl:text>
            <xsl:value-of select="$name"/>
            <xsl:value-of select="$ID"/>
            <xsl:text>File</xsl:text>
            <xsl:text disable-output-escaping="yes">%&gt;&lt;/p&gt;</xsl:text>

        </xsl:for-each>
        <xsl:text disable-output-escaping="yes">
              &lt;p&gt;————————————————————————&lt;/p&gt;
        </xsl:text>

        <xsl:for-each select="//task">
            <xsl:variable name="ID" select="@ID"/>
            <xsl:variable name="name" select="@name"/>
            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

            <xsl:text disable-output-escaping="yes">&lt;%String </xsl:text>
            <xsl:value-of select="$name"/>
            <xsl:value-of select="$ID"/>
            <xsl:text>service</xsl:text>
            <xsl:text>=(String)session.getAttribute("</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$name"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$ID"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>service</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">");%&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                    &lt;p&gt;This service KEY is &lt;%=
            </xsl:text>
            <xsl:value-of select="$name"/>
            <xsl:value-of select="$ID"/>
            <xsl:text>service</xsl:text>
            <xsl:text disable-output-escaping="yes">%&gt;&lt;/p&gt;</xsl:text>


            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                &lt;p&gt;————————————————&lt;/p&gt;
        </xsl:text>

        </xsl:for-each>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;form action="&lt;%=url%&gt;" method="POST"&gt;
        </xsl:text>

        <xsl:for-each select="//task[@rank=0]">
            <xsl:variable name="ID" select="@ID"/>
            <xsl:variable name="name" select="@name"/>
            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;input type="hidden" name="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$name"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$ID"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>File</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">" value="&lt;%=</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$name"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$ID"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>File</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">%&gt;"/&gt;</xsl:text>
        </xsl:for-each>

        <xsl:for-each select="//task">
            <xsl:variable name="ID" select="@ID"/>
            <xsl:variable name="name" select="@name"/>
            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;input type="hidden" name="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$name"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$ID"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>service</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">" value="&lt;%=</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$name"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$ID"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>service</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">%&gt;"/&gt;</xsl:text>
        </xsl:for-each>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;input type="hidden" name="abstractWorkflow" value="
␣␣␣␣␣␣␣␣</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$filelocation"/>
␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;input type="submit"/&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/form&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/html&gt;</xsl:text>
    </xsl:template>
</xsl:stylesheet>
```

A.5 Stylesheet to Generate Java Code for Generation and Deployment Portlet

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text" indent="yes"/>
    <xsl:template match="/">
        <xsl:call-template name="importLibraries"/>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>public class ServiceInvokePortlet extends GenericPortlet{</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:call-template name="declareVariable"/>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:call-template name="init"/>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:call-template name="doView"/>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:call-template name="processAction"/>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>}</xsl:text>
    </xsl:template>

    <xsl:template name="importLibraries">
        <xsl:text>import javax.portlet.GenericPortlet;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletException;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.RenderRequest;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.RenderResponse;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletRequestDispatcher;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletContext;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletURL;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletConfig;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.PortletSession;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text>import javax.portlet.ActionRequest;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.portlet.ActionResponse;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.portlet.UnavailableException;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import java.io.IOException;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>//packages for the web service</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.xml.namespace.QName;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.xml.rpc.Call;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.xml.rpc.ServiceFactory;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.wsdl.factory.WSDLFactory;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.wsdl.xml.WSDLReader;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import com.ibm.wsdl.xml.WSDLReaderImpl;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.wsdl.Definition;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.wsdl.Port;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.wsdl.PortType;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.wsdl.Operation;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.wsdl.extensions.soap.SOAPAddress;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import javax.wsdl.extensions.ExtensibilityElement;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import java.util.Map;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import java.util.Iterator;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import java.util.StringTokenizer;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text>import java.util.Vector;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
</xsl:template>

<xsl:template name="declareVariable">
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>public final String servicesInvokeJsp="/jsp/serviceInvoke.jsp";</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>public final String serviceURL=
            "http://celeste.cs.cf.ac.uk:8080/active-bpel/services/serviceProvider0?wsdl";
    </xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private String accesspoint=null;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private WSDLReader wsdlReader=new WSDLReaderImpl();</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private String targetNamespace=null;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private Definition definition=null;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private Port port=null;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private PortType pt=null;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private Operation op=null;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private PortletSession servicesinvokesession=null;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private String status="ready";</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private javax.wsdl.Service wsdlservice=null;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private javax.xml.rpc.Service service=null;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>private Call call=null;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
</xsl:template>

<xsl:template name="init">
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    public void init(PortletConfig config)throws UnavailableException,
    PortletException{
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>super.init(config);</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>}</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
</xsl:template>

<xsl:template name="doView">
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>public void doView(RenderRequest request,RenderResponse response)throws
            PortletException, IOException{</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>PortletURL url=response.createActionURL();</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>response.setContentType("text/html");</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>request.setAttribute("url",url.toString());</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>request.setAttribute("status",status);</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>PortletContext portletContext=getPortletContext();</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>PortletRequestDispatcher
            prd=portletContext.getRequestDispatcher(servicesInvokeJsp);</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>prd.include(request, response);</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>}</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
</xsl:template>

<xsl:template name="processAction">
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text>public void processAction(ActionRequest request,
            ActionResponse response){</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>//invoke the web service with dynamic invoke</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>servicesinvokesession=request.getPortletSession();</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>
            String[] initparam=(String[])servicesinvokesession.getAttribute("initparam");
            </xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>Vector paraVector=new Vector();</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>status="successful";</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>


    <xsl:text>for(int i=0;i&lt;initparam.length;i++){</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
    <xsl:text>StringTokenizer st=new StringTokenizer(initparam[i],"=");</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
    <xsl:text>st.nextToken();</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
    <xsl:text>initparam[i]=st.nextToken();</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>}</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

    <xsl:for-each select="//task[@rank=0]/ServiceType/portType/operation/input/part">
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text>paraVector.add(initparam[</xsl:text>
    <xsl:value-of select="../../../../../@ID"/>
    <xsl:text>]);</xsl:text>
</xsl:for-each>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>


<xsl:text>try{</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>definition=wsdlReader.readWSDL(serviceURL);</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>targetNamespace=definition.getTargetNamespace();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>Map services=definition.getServices();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>int count=0;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>Iterator iterator=services.keySet().iterator();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>while(iterator.hasNext()){</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>Object key=iterator.next();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>wsdlservice=(javax.wsdl.Service)services.get(key);</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>}</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>QName serviceQN=wsdlservice.getQName();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>ServiceFactory factory=ServiceFactory.newInstance();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>service=factory.createService(serviceQN);</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>Map ports=wsdlservice.getPorts();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>count=0;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
```

```
<xsl:text>iterator=ports.keySet().iterator();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>while(iterator.hasNext()){</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>Object key=iterator.next();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>port=(Port)ports.get(key);</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
<xsl:text>}</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>
        SOAPAddress ext =(SOAPAddress)port.getExtensibilityElements().get(0);
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>accesspoint=ext.getLocationURI();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>pt=port.getBinding().getPortType();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>op=pt.getOperation("runProcess",null,null);</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>call=(Call)service.createCall();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>call.setProperty(Call.ENCODINGSTYLE_URL_PROPERTY,"");</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>call.setProperty(Call.OPERATION_STYLE_PROPERTY,"wrapped");</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>call.setTargetEndpointAddress(accesspoint);</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>call.removeAllParameters();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>call.setPortTypeName(pt.getQName());</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text>call.setOperationName(new QName(targetNamespace,op.getName()));</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
```

```
<xsl:text>
        if(call.isParameterAndReturnSpecRequired(call.getOperationName())){
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text>int numberOfPara=</xsl:text>
<xsl:value-of select=
        "count(//task[@rank=0]/ServiceType/portType/operation/input/part)"/>
<xsl:text>;</xsl:text>


<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text disable-output-escaping="yes">for(int i=0;i&lt;numberOfPara;i++){</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>call.addParameter("in"+Integer.toString(i),</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>
        new QName("http://www.w3.org/2001/XMLSchema","String"),
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>String.class,</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>javax.xml.rpc.ParameterMode.IN);</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>}</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>
    call.setReturnType(new QName("http://www.w3.org/2001/XMLSchema","string"));
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
<xsl:text>}</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
<xsl:text>call.invoke(paraVector.toArray());</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
<xsl:text>status="successful";</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
<xsl:text>}catch(Exception e){</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text>status="error_"+e.toString();</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
<xsl:text>}</xsl:text>
```

```
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>}</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    </xsl:template>
</xsl:stylesheet>
```

A.6 Stylesheet to Generate JSP Page for Generation and Deployment Portlet

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text" indent="yes"/>
    <xsl:template match="/">
        <xsl:call-template name="header"/>
        <xsl:call-template name="body"/>
    </xsl:template>

    <xsl:template name="header">
            <xsl:text disable-output-escaping="yes">&lt;%@page session="true"
                    contentType="text/html"%&gt;</xsl:text>
    </xsl:template>

    <xsl:template name="body">
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;html&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
            &lt;%String fileName=(String)session.getAttribute("generationResult")
        </xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>String url=(String)request.getAttribute("url");</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text>String status=(String)request.getAttribute("status");</xsl:text>
        <xsl:text disable-output-escaping="yes">%&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
            &lt;p&gt;The workflow script &lt;%=fileName%&gt; is generated!&lt;/p&gt;
        </xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;%try{%&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
            &lt;p&gt;The paramerters for this web service are: &lt;/p&gt;
        </xsl:text>
```

```
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
<xsl:text disable-output-escaping="yes">
       &lt;%String[] initparam=(String[])session.getAttribute("initparam");
</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
<xsl:text disable-output-escaping="yes">
       for(int i=0;i&lt;initparam.length;i++){%&gt;
</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
<xsl:text disable-output-escaping="yes">
       &lt;p&gt;parameter is &lt;%=initparam[i]%&gt;&lt;/p&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;br/&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
<xsl:text disable-output-escaping="yes">
       &lt;p&gt;————————————————————————&lt;/p&gt;
</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;%}</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
<xsl:text>}catch(NullPointerException npe){</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
<xsl:text disable-output-escaping="yes">}%&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
<xsl:text disable-output-escaping="yes">
       &lt;%if(status.equals("ready")){%&gt;
</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
<xsl:text disable-output-escaping="yes">
       &lt;p&gt;Press Button to execute the workflow&lt;/p&gt;
</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
<xsl:text disable-output-escaping="yes">
       &lt;%}else if(status.contains(".bpr")){%&gt;
</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
<xsl:text disable-output-escaping="yes">
       &lt;p&gt;The bpr file has been successfully generated!&lt;/p&gt;
</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
```

```
        <xsl:text disable-output-escaping="yes">&lt;%} else{%&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;p&gt;Some exception has happened, the details is: &lt;/p&gt;
        </xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;p&gt;&lt;%=status%&gt;&lt;/p&gt;
        </xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;%}%&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;a href="&lt;%=url%&gt;"&gt;execute&lt;/a&gt;
        </xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/html&gt;</xsl:text>
    </xsl:template>
</xsl:stylesheet>
```

A.7 Stylesheet to Generate Build File for ANT

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:output method="xml" indent="yes"/>
    <xsl:variable name="nameofWF" select="//workflow/@name"/>
    <xsl:variable name="nameofPortal" select="concat($nameofWF,'Portal')"/>

    <xsl:template match="/">

            <xsl:text disable-output-escaping="yes">&lt;project name="New_Project"
                default="new-project" basedir="."&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">
                &lt;target name="check-project-exists"&gt;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">
                &lt;condition property="new-project-exists"&gt;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">&lt;not&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
```

```
<xsl:text disable-output-escaping="yes">
        &lt;available file="${project.dir}/build.xml"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;/not&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;/condition&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;/target&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;!-- ============================================================ --&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;!-- Updates a portlet project in projects directory
--&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;!-- ============================================================ --&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;target name="update-project"
        description="Updates␣existing␣project␣with␣updated␣build.xml␣script"&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;input message=
        "Please␣enter␣the␣project␣web␣app␣to␣update␣inside␣projects/␣directory"
        addproperty="project.name"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;property name="newproject.dir" value="projects/${project.name}"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;copy file="${newproject.dir}/build.xml"
        tofile="${newproject.dir}/build.xml.bak"
        overwrite="true"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
```

```
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;copy file="config/build/build.project.xml"
         tofile="${newproject.dir}/build.xml"
         overwrite="true"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;replace file="${newproject.dir}/build.xml"
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text>token="@PROJECT_TITLE@"</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
         value="${project.title}"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;replace file="${newproject.dir}/build.xml"
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text>token="@PROJECT_NAME@"</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">value="${project.name}"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;/target&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;!-- ================================================================ --&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;!-- Creates a new portlet project  in projects directory
--&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;!-- ================================================================ --&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
```

```
        <xsl:text disable-output-escaping="yes">
                &lt;target name="new-project"
                description="Creates␣a␣new␣portlet␣project␣in␣projects␣directory"&gt;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">&lt;mkdir dir="projects"/&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">
                &lt;property name="project.title" value="
␣␣␣␣␣␣␣␣</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$nameofPortal"/>
␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>


        <xsl:text disable-output-escaping="yes">
                &lt;property name="project.name" value="
␣␣␣␣␣␣␣␣</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$nameofPortal"/>
␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>


        <xsl:text disable-output-escaping="yes">
                &lt;property name="project.dir" value="projects/${project.name}"/&gt;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">
                &lt;antcall target="check-project-exists"/&gt;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">
                &lt;fail if="${new-project-exists}" message=
                "This␣project␣already␣exists!␣Please␣delete␣the␣old␣one␣in␣${project.dir}"/&gt;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">&lt;mkdir dir="${project.dir}"/&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">
                &lt;property name="use.jsr" value="jsr"/&gt;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">&lt;copy file="config/build/build.project.xml"
                tofile="${project.dir}/build.xml"/&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
```

```
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;copy file="config/build/build.properties"
        tofile="${project.dir}/build.properties" filtering="true"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;replace file="${project.dir}/build.properties"
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text>token="@GRIDSPHERE_HOME@"</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">value="${GRIDSPHERE_HOME}"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;replace file="${project.dir}/build.properties"
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text>token="@PROJECT_TITLE@"</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">value="${project.title}"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;replace file="${project.dir}/build.properties"
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text>token="@PROJECT_NAME@"</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">value="${project.name}"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;replace file="${project.dir}/build.xml"
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text>token="@PROJECT_NAME@"</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">value="${project.name}"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
```

```
<xsl:text disable-output-escaping="yes">
        &lt;replace file="${project.dir}/build.properties"
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text>token="@USE_JSR@"</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">value="${use.jsr}"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;mkdir dir="${project.dir}/src"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;mkdir dir="${project.dir}/lib"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;mkdir dir="${project.dir}/webapp"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes"
        >&lt;mkdir dir="${project.dir}/webapp/WEB-INF"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;mkdir dir="${project.dir}/webapp/WEB-INF/classes"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;copy file="config/log4j.properties"
        tofile="${project.dir}/webapp/WEB-INF/classes/log4j.properties"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;mkdir dir="${project.dir}/webapp/WEB-INF/persistence"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;copy todir="${project.dir}/webapp/WEB-INF"&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
```

```
        &lt;fileset dir="config/template"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;mapper type="glob" from="*.xml.tpl" to="*.xml"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;/copy&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;copy file="config/template/hibernate.properties"
        todir="${project.dir}/webapp/WEB-INF/persistence"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;antcall target="checkjsr"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;antcall target="checkgs"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;replace file="${project.dir}/webapp/WEB-INF/web.xml"
        token="@PROJECT_TITLE@"
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">value="${project.title}"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;replace file="${project.dir}/webapp/WEB-INF/layout.xml"
        token="@PROJECT_TITLE@"
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">value="${project.title}"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;replace file="${project.dir}/webapp/WEB-INF/web.xml" token="@PROJECT_NAME@"
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">value="${project.name}"/&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
```

```
        &lt;mkdir dir="${project.dir}/webapp/html"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;mkdir dir="${project.dir}/webapp/jsp"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;/target&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;target name="checkjsr" if="use.jsr"&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;copy overwrite="true" file="config/template/gridsphere-portlet-jsr.xml"
         tofile="${project.dir}/webapp/WEB-INF/gridsphere-portlet.xml"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;copy overwrite="true" file="config/template/portlet-jsr.xml"
         tofile="${project.dir}/webapp/WEB-INF/portlet.xml"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;copy overwrite="true" file="config/template/layout-jsr.xml"
         tofile="${project.dir}/webapp/WEB-INF/layout.xml"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;copy overwrite="true" file="config/template/web-jsr.xml"
         tofile="${project.dir}/webapp/WEB-INF/web.xml"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &lt;copy overwrite="true" file="config/template/group.sample.xml.tpl"
         tofile="${project.dir}/webapp/WEB-INF/group.sample.xml"/&gt;
</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes">&lt;/target&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

<xsl:text disable-output-escaping="yes"
        >&lt;target name="checkgs" unless="use.jsr"&gt;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">
                &lt;delete file="${project.dir}/webapp/WEB-INF/portlet.xml"/&gt;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">&lt;/target&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>

        <xsl:text disable-output-escaping="yes">&lt;/project&gt;</xsl:text>

    </xsl:template>
</xsl:stylesheet>
```

A.8 Stylesheet to Generate the Configuration File for Grouping Portlets

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:output method="xml" indent="yes"/>
        <xsl:variable name="nameofWF" select="//workflow/@name"/>
        <xsl:variable name="nameofPortal" select="concat($nameofWF,'Portal')"/>

        <xsl:template match="/">
            <xsl:text disable-output-escaping="yes">&lt;portlet-group&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;group-name&gt;</xsl:text>
            <xsl:value-of select="$nameofPortal"/>
            <xsl:text disable-output-escaping="yes">&lt;/group-name&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;group-description&gt;</xsl:text>
            <xsl:text>A </xsl:text>
            <xsl:value-of select="$nameofWF"/>
            <xsl:text> group</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/group-description&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                    &lt;group-visibility&gt;PUBLIC&lt;/group-visibility&gt;
            </xsl:text>

            <xsl:call-template name="portletrole"/>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/portlet-group&gt;</xsl:text>
        </xsl:template>

        <xsl:template name="portletrole">
            <xsl:for-each select="//workflow/task[@rank=0]">
                <xsl:variable name="name" select="@name"/>
```

```
    <xsl:variable name="ID" select="@ID"/>
    <xsl:variable name="nameOfPortlet" select="concat($name,$ID)"/>

    <xsl:variable name="nameOfPortlet"
            select="concat($nameOfPortlet, 'FilePortlet')"/>
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt; portlet-role-info&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt; portlet-class&gt;</xsl:text>
    <xsl:value-of select="$nameofPortal"/>
    <xsl:text>#</xsl:text>
    <xsl:value-of select="$nameOfPortlet"/>
    <xsl:text disable-output-escaping="yes">&lt;/ portlet-class&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            &lt; required-role&gt;USER&lt;/ required-role&gt;
    </xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;/ portlet-role-info&gt;</xsl:text>
</xsl:for-each>

<xsl:for-each select="//workflow/task">
    <xsl:variable name="name" select="@name"/>
    <xsl:variable name="ID" select="@ID"/>
    <xsl:variable name="nameOfPortlet" select="concat($name,$ID)"/>
    <xsl:variable name="nameOfPortlet" select="concat($nameOfPortlet, 'Portlet')"/>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt; portlet-role-info&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt; portlet-class&gt;</xsl:text>
    <xsl:value-of select="$nameofPortal"/>
    <xsl:text>#</xsl:text>
    <xsl:value-of select="$nameOfPortlet"/>
    <xsl:text disable-output-escaping="yes">&lt;/ portlet-class&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            &lt; required-role&gt;USER&lt;/ required-role&gt;
    </xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;/ portlet-role-info&gt;</xsl:text>
</xsl:for-each>


<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
<xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt; portlet-role-info&gt;</xsl:text>
```

```
            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;portlet-class&gt;</xsl:text>
            <xsl:value-of select="$nameofPortal"/>
            <xsl:text disable-output-escaping="yes">
                    #ServicesDisPortlet&lt;/portlet-class&gt;
            </xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                    &lt;required-role&gt;USER&lt;/required-role&gt;
            </xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/portlet-role-info&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;portlet-role-info&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;portlet-class&gt;</xsl:text>
            <xsl:value-of select="$nameofPortal"/>
            <xsl:text disable-output-escaping="yes">
                    #ServiceInvokePortlet&lt;/portlet-class&gt;
            </xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                    &lt;required-role&gt;USER&lt;/required-role&gt;
            </xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/portlet-role-info&gt;</xsl:text>
        </xsl:template>
</xsl:stylesheet>
```

A.9 Stylesheet to Generate the Configuration File for Portlets Layout

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" indent="yes"/>
    <xsl:variable name="nameofWF" select="//workflow/@name"/>
    <xsl:variable name="nameofPortal" select="concat($nameofWF,'Portal')"/>
    <xsl:variable name="maxRank" select="//workflow/task[last()]/@rank"/>

    <xsl:template match="/">
        <xsl:text disable-output-escaping="yes">&lt;portlet-tabbed-pane&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;portlet-tab label="</xsl:text>
```

```
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$nameofPortal"/>
␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>

        <xsl:text disable-output-escaping="yes">&lt;title lang="en"&gt;</xsl:text>
        <xsl:value-of select="$nameofPortal"/>
        <xsl:text disable-output-escaping="yes">&lt;/title&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;portlet-tabbed-pane style="sub-menu"&gt;
        </xsl:text>

        <xsl:call-template name="FileSelection"/>

        <xsl:call-template name="serviceSelection">
            <xsl:with-param name="rank">
                <xsl:value-of select="0"/>
            </xsl:with-param>

            <xsl:with-param name="count">
                <xsl:value-of select="$maxRank"/>
            </xsl:with-param>
        </xsl:call-template>

        <xsl:call-template name="ServiceDis"/>

        <xsl:call-template name="ServicesInv"/>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/portlet-tabbed-pane&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/portlet-tab&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/portlet-tabbed-pane&gt;</xsl:text>
    </xsl:template>

    <xsl:template name="FileSelection">
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;portlet-tab lable="FILE␣Selection"&gt;
        </xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;title lang="en"&gt;FILE-SELECTION&lt;/title&gt;
        </xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;table-layout&gt;</xsl:text>

        <xsl:for-each select="//workflow/task[@rank=0]">
        <xsl:variable name="name" select="./@name"/>
        <xsl:variable name="namewithID" select="concat($name,./@ID)"/>
        <xsl:variable name="nameOfPortlet" select="concat($namewithID,'FilePortlet')"/>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &#x09;&#x09;&#x09;&#x09;&#x09;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;row-layout&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
     </xsl:text>
     <xsl:text disable-output-escaping="yes">
             &lt;column-layout width="50%"&gt;
     </xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;portlet-frame label="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$namewithID"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">File"&gt;</xsl:text>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;portlet-class&gt;</xsl:text>
        <xsl:value-of select="$nameOfPortlet"/>
        <xsl:text disable-output-escaping="yes">&lt;/portlet-class&gt;</xsl:text>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/portlet-frame&gt;</xsl:text>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/column-layout&gt;</xsl:text>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &#x09;&#x09;&#x09;&#x09;&#x09;
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/row-layout&gt;</xsl:text>
    </xsl:for-each>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            &#x09;&#x09;&#x09;&#x09;
    </xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;/table-layout&gt;</xsl:text>
```

```
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/portlet-tab&gt;</xsl:text>
    </xsl:template>

    <xsl:template name="serviceSelection">
        <xsl:param name="rank"/>
        <xsl:param name="count"/>

        <xsl:if test="$rank␣&lt;=␣$count">
            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;portlet-tab label="RANK-</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$rank"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;title lang="en"&gt;RANK-</xsl:text>
            <xsl:value-of select="$rank"/>
            <xsl:text disable-output-escaping="yes">&lt;/title&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;table-layout&gt;</xsl:text>

            <xsl:for-each select="//workflow/task[@rank=$rank]">
                <xsl:variable name="name" select="@name"/>
                <xsl:variable name="ID" select="@ID"/>
                <xsl:variable name="namewithID" select="concat($name,$ID)"/>
                <xsl:variable name="nameOfPortlet" select="concat($namewithID,'Portlet')"/>


                <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
                <xsl:text disable-output-escaping="yes">
                        &#x09;&#x09;&#x09;&#x09;&#x09;
                </xsl:text>
                <xsl:text disable-output-escaping="yes">&lt;row-layout&gt;</xsl:text>


                <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
                <xsl:text disable-output-escaping="yes">
                        &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
                </xsl:text>
                <xsl:text disable-output-escaping="yes">
                        &lt;column-layout width="50%"&gt;
                </xsl:text>

                <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
                <xsl:text disable-output-escaping="yes">
                        &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
                </xsl:text>
                <xsl:text disable-output-escaping="yes">&lt;portlet-frame label="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$namewithID"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"&gt;</xsl:text>


                <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
                <xsl:text disable-output-escaping="yes">
                        &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
                </xsl:text>
            <xsl:text disable-output-escaping="yes">&lt; portlet-class&gt;</xsl:text>
            <xsl:value-of select="$nameOfPortlet"/>
            <xsl:text disable-output-escaping="yes">&lt;/portlet-class&gt;</xsl:text>


            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                    &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/portlet-frame&gt;</xsl:text>


            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                    &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/column-layout&gt;</xsl:text>


            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                    &#x09;&#x09;&#x09;&#x09;&#x09;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/row-layout&gt;</xsl:text>



        </xsl:for-each>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/table-layout&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/portlet-tab&gt;</xsl:text>

        <!--increate the rank by 1-->
        <xsl:call-template name="serviceSelection">
            <xsl:with-param name="rank">
                <xsl:value-of select="$rank+1"/>
            </xsl:with-param>
            <xsl:with-param name="count">
                <xsl:value-of select="$count"/>
            </xsl:with-param>
        </xsl:call-template>
    </xsl:if>
</xsl:template>

<xsl:template name="ServiceDis">
    <xsl:variable name="disRank" select="$maxRank+1"/>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            &lt;portlet-tab label="Service_Information_Display"&gt;
    </xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>

    <xsl:text disable-output-escaping="yes">&lt; title lang="en"&gt;</xsl:text>
    <xsl:text disable-output-escaping="yes">ServiceInfo Display</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;/title&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;table-layout&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;row-layout&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
    </xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;column-layout width="50%"&gt;</xsl:text>


    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
    </xsl:text>
    <xsl:text disable-output-escaping="yes">
            &lt;portlet-frame label="ServicesDis"&gt;
    </xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
    </xsl:text>
    <xsl:text disable-output-escaping="yes"
            >&lt;portlet-class&gt;ServicesDisPortlet&lt;/portlet-class&gt;
    </xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">
            &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;
    </xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;/portlet-frame&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;/column-layout&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;/row-layout&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;/table-layout&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#x09;&#x09;&#x09;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;/portlet-tab&gt;</xsl:text>

</xsl:template>


<xsl:template name="ServicesInv">
    <xsl:variable name="invRank" select="$maxRank+2"/>
```

```
<xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&#x09;&#x09;&#x09 ;</xsl:text>

<xsl:text disable-output-escaping=”yes”>&lt ; portlet-tab label=”Execute”&gt ;</xsl:text>

<xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&#x09;&#x09;&#x09;&#x09 ;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&lt ; title lang=”en”&gt ;</xsl:text>
<xsl:text disable-output-escaping=”yes”>Execute</xsl:text>
<xsl:text disable-output-escaping=”yes”>&lt ;/ title&gt ;</xsl:text>

<xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&#x09;&#x09;&#x09;&#x09 ;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&lt ; table-layout&gt ;</xsl:text>

<xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&#x09;&#x09;&#x09;&#x09;&#x09 ;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&lt ; row-layout&gt ;</xsl:text>

<xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&#x09;&#x09;&#x09;&#x09;&#x09;&#x09 ;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&lt ; column-layout width=”50%”&gt ;</xsl:text>


<xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
<xsl:text disable-output-escaping=”yes”>
        &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09 ;
</xsl:text>
<xsl:text disable-output-escaping=”yes”>
        &lt ; portlet-frame label=”ServicesInvoke”&gt ;
</xsl:text>

<xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
<xsl:text disable-output-escaping=”yes”>
        &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09 ;
</xsl:text>
<xsl:text disable-output-escaping=”yes”>
        &lt ; portlet-class&gt ; ServiceInvokePortlet&lt ;/ portlet-class&gt ;
</xsl:text>

<xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
<xsl:text disable-output-escaping=”yes”>
        &#x09;&#x09;&#x09;&#x09;&#x09;&#x09;&#x09 ;
</xsl:text>
<xsl:text disable-output-escaping=”yes”>&lt ;/ portlet-frame&gt ;</xsl:text>

<xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&#x09;&#x09;&#x09;&#x09;&#x09;&#x09 ;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&lt ;/ column-layout&gt ;</xsl:text>

<xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&#x09;&#x09;&#x09;&#x09;&#x09 ;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&lt ;/ row-layout&gt ;</xsl:text>

<xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&#x09;&#x09;&#x09;&#x09 ;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&lt ;/ table-layout&gt ;</xsl:text>

<xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&#x09;&#x09;&#x09 ;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&lt ;/ portlet-tab&gt ;</xsl:text>
```

```
    </xsl:template>
</xsl:stylesheet>
```

Appendix B

XSLT Stylesheets for Automatic Workflow Execution Script Generation

B.1 Stylesheet to Generate the BPEL File

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:wfconverter="xalan://WFwsConverter"
         extension-element-prefixes="wfconverter">
    <xsl:output method="xml" indent="yes"/>
    <xsl:template match="/">
        <xsl:text disable-output-escaping="yes">&lt;process </xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
        <xsl:value-of select="wfconverter:getBpelNS()"/>

        <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
        <xsl:value-of select="wfconverter:getExtension()"/>

        <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
        <xsl:value-of select="wfconverter:getDoubleArrayNS()"/>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:call-template name="importNS"/>



        <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">name="concreteprocess"</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">suppressJoinFailure="yes"</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                targetNamespace="http://concrete/FlowAuto"
        </xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">ext:createTargetXPath="yes"</xsl:text>

        <xsl:text disable-output-escaping="yes">&gt;</xsl:text>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:call-template name="importWSDL"/>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:call-template name="importSchema"/>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:call-template name="importExtension"/>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:call-template name="importPLnk"/>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:call-template name="importVariables"/>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:call-template name="importProcess"/>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/process&gt;</xsl:text>
    </xsl:template>


    <xsl:template match="task" name="importNS">
        <xsl:for-each select="//workflow/task">
            <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
            <xsl:variable name="id" select="@ID"/>
            <xsl:variable name="portType" select="ServiceType/portType/@name"/>
            <xsl:text disable-output-escaping="yes">xmlns:</xsl:text>
            <xsl:value-of select="wfconverter:getNamespaceprefix($portType,$id)"/>
            <xsl:text disable-output-escaping="yes">="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="ServiceType/namespace/text()"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"</xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        </xsl:for-each>

        <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
        <xsl:text>xmlns:pns="</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="wfconverter:getProcessClientPTNS()"/>
␣␣␣␣␣␣␣␣<xsl:text>"</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
        <xsl:text>xmlns:def="http://dlu.FFTWebservice.dustCloud"</xsl:text>
   </xsl:template>


    <xsl:template match="task" name="importWSDL">
        <xsl:for-each select="//workflow/task[@type='normal']">
            <xsl:variable name="task" select="."/>
            <xsl:variable name="ptNS" select="$task/ServiceType/namespace/text()" />
            <xsl:variable name="ptURL" select="$task/ServiceType/urlLocation/text()"/>
            <xsl:variable name="taskID" select="@ID"/>
            <xsl:variable name="uuid" select="$task/ServiceType/@ServiceTypeID"/>
            <xsl:variable name="exist" select=
                 "count(//workflow/task[@ID&gt;$taskID]/ServiceType[@ServiceTypeID=$uuid])"/>

            <xsl:if test="$exist=0">
                <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">
                        &lt;import importType="http://schemas.xmlsoap.org/wsdl/"
                </xsl:text>

                <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">namespace="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$ptNS"/>

␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"&#10;&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">location="project:/wsdl/</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="wfconverter:getfileName($ptURL)"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"/&gt;&#10;</xsl:text>
            </xsl:if>
        </xsl:for-each>

        <!--the new part for the new IO web service WSDL files importing-->
        <xsl:for-each select="//workflow/task[@type='input'␣or␣@type='output']">
            <xsl:variable name="task" select="."/>
            <xsl:variable name="ptNS" select="$task/ServiceType/namespace/text()" />
            <xsl:variable name="ptURL" select="$task/ServiceType/urlLocation/text()"/>
            <xsl:variable name="taskID" select="@ID"/>

            <xsl:variable name="uuid" select="$task/@client"/>

            <xsl:variable name="exist"
                 select="count(//workflow/task[@ID&gt;$taskID␣and␣@client=$uuid])"/>

            <xsl:if test="$exist=0">
                <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">
                        &lt;import importType="http://schemas.xmlsoap.org/wsdl/"
                </xsl:text>

                <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">namespace="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$ptNS"/>

␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"&#10;&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">location="project:/wsdl/</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="wfconverter:getfileName($ptURL)"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"/&gt;&#10;</xsl:text>
            </xsl:if>
        </xsl:for-each>

        <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                &lt;import importType="http://schemas.xmlsoap.org/wsdl/"
        </xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">namespace="</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="wfconverter:getProcessClientPTNS()"/>

␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"&#10;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">
                location="project:/wsdl/concretewfws.wsdl"
        </xsl:text>
        <xsl:text disable-output-escaping="yes">/&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    </xsl:template>

    <xsl:template match="task" name="importSchema">
        <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
        <import importType="http://www.w3c.org/2001/XMLSchema"
                    location="project:/wsdl/doubleArray1.xsd"
                    namespace="http://dlu.FFTWebservice.dustCloud"/>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    </xsl:template>


    <xsl:template match="null" name="importExtension">
        <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;extensions&gt;</xsl:text>
                <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
                <extension mustUnderstand="yes"
                    namespace="http://www.activebpel.org/2006/09/bpel/extension/query_handling"/>
                <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/extensions&gt;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    </xsl:template>

    <xsl:template match="task" name="importPLnk">
            <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;partnerLinks&gt;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:for-each select="//workflow/task">
                <xsl:variable name="id" select="@ID"/>

                <xsl:variable name="serviceName" select="serviceInstance/serviceName/text()"/>
                <xsl:variable name="partnerLink" select="concat($serviceName,$id)"/>

                <xsl:variable name="ptName" select="ServiceType/portType/@name"/>
                <xsl:variable name="ptPrefix"
                  select="wfconverter:getNamespaceprefix($ptName,$id)"/>
                <xsl:variable name="partnerLinkType" select="concat($ptPrefix,'Link')"/>

                <xsl:variable name="role" select="concat($partnerLink,'Provider')"/>


                <xsl:text disable-output-escaping="yes">&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">&lt;partnerLink name="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$partnerLink"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"&#10;&#9;&#9;&#9;</xsl:text>

                <xsl:text disable-output-escaping="yes">partnerLinkType="pns:</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$partnerLinkType"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"&#10;&#9;&#9;&#9;</xsl:text>

                <xsl:text disable-output-escaping="yes">partnerRole="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$role"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"/&gt;&#10;</xsl:text>

            </xsl:for-each>
             <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
             <xsl:text disable-output-escaping="yes">&lt;partnerLink name="Client"</xsl:text>
             <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
             <xsl:text>partnerLinkType="pns:ClientLink"</xsl:text>
             <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
```

```
          <xsl:text>myRole="ProcessProvider"</xsl:text>
          <xsl:text disable-output-escaping="yes">/&gt;</xsl:text>
      <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/partnerLinks&gt;</xsl:text>
</xsl:template>

<xsl:template match="task" name="importVariables">
     <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
     <xsl:text disable-output-escaping="yes">&lt;variables&gt;</xsl:text>
     <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
     <xsl:for-each select="//workflow/task/ServiceType">
        <xsl:variable name="id" select="../@ID"/>
        <xsl:variable name="ptName" select="portType/@name"/>
        <xsl:variable name="ptPrefixName"
                select="wfconverter:getNamespaceprefix($ptName,$id)"/>

        <xsl:for-each select="portType/operation">
            <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
            <xsl:variable name="inputMessageType" select="input/@type"/>
            <xsl:variable name="inputMessageName" select="input/@name"/>
            <xsl:variable name="inputMessageName" select="concat($inputMessageType,$id)"/>

            <xsl:variable name="outputMessageType" select="output/@type"/>
            <xsl:variable name="outputMessageName" select="output/@name"/>
            <xsl:variable name="outputMessageName" select="concat($outputMessageType,$id)"/>

            <xsl:text disable-output-escaping="yes">&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;variable name="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$inputMessageName"/>

␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">" messageType="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$ptPrefixName"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">:</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$inputMessageType"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"/&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;variable name="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$outputMessageName"/>


␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">" messageType="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$ptPrefixName"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">:</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$outputMessageType"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"/&gt;&#10;</xsl:text>
        </xsl:for-each>
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
    </xsl:for-each>

            <xsl:text disable-output-escaping="yes">&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                &lt;variable name="userRequest" messageType="pns:ClientRequest"/&gt;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                &lt;variable name="userResponse" messageType="pns:ClientResponse"/&gt;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                &lt;variable name="count" type="xsd:integer"/&gt;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                  &lt;variable name="counter" type="xsd:integer"/&gt;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/variables&gt;</xsl:text>
</xsl:template>

<xsl:template match="task" name="importProcess">
    <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;sequence name="main"&gt;</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;receive name="receiveRequest"</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">partnerLink="Client"</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">portType="pns:ProcessClientPT"</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">operation="runProcess"</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">variable="userRequest"</xsl:text>
    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">createInstance="yes"/&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;sequence name="InputWS"&gt;</xsl:text>

    <xsl:variable name="requestInputID" select="wfconverter:resetID()"/>
    <xsl:for-each select="//workflow/task">
        <xsl:variable name="id" select="@ID"/>
        <xsl:variable name="pt" select="ServiceType/portType/@name"/>
        <xsl:variable name="ptPrefix" select="wfconverter:getNamespaceprefix($pt,$id)"/>
        <xsl:variable name="serviceName" select="serviceInstance/serviceName/text()"/>

        <xsl:if test="@type='input'">
            <xsl:for-each select="ServiceType/portType/operation/input/part">
                <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">&lt;assign&gt;</xsl:text>


                <xsl:variable name="operation" select="../../@name"/>
                <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">&lt;copy&gt;</xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">&lt;from&gt;</xsl:text>
                <xsl:text disable-output-escaping="yes">$userRequest.in</xsl:text>
                <xsl:value-of select="wfconverter:getID()"/>
                <xsl:text disable-output-escaping="yes">&lt;/from&gt;</xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">&lt;to&gt;</xsl:text>
                <xsl:variable name="inputMessageName" select="../@type"/>
                <xsl:variable name="inputMessageName"
                        select="concat($inputMessageName,$id)"/>
                <xsl:value-of select="concat('$',$inputMessageName)"/>
                <xsl:text>.</xsl:text>
                <xsl:value-of select="@name"/>

                <xsl:text disable-output-escaping="yes">&lt;/to&gt;</xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">&lt;/copy&gt;</xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
```

```
                    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
                    <xsl:text disable-output-escaping="yes">&lt;/assign&gt;</xsl:text>

                    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
                    <xsl:text disable-output-escaping="yes">&lt;invoke name="</xsl:text>
                    <xsl:value-of select="concat($pt,$operation)"/>
                    <xsl:text>"</xsl:text>

                    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
                    <xsl:text>partnerLink="</xsl:text>
                    <xsl:value-of select="concat($serviceName,$id)"/>
                    <xsl:text>"</xsl:text>

                    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
                    <xsl:text>portType="</xsl:text>
                    <xsl:value-of select="$ptPrefix"/>
                            <xsl:text>:</xsl:text>
                            <xsl:value-of select="$pt"/>
                    <xsl:text>"</xsl:text>

                    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
                    <xsl:text>operation="</xsl:text>
                    <xsl:value-of select="$operation"/>
                    <xsl:text>"</xsl:text>

                    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
                    <xsl:text>inputVariable="</xsl:text>
                    <xsl:variable name="inputMessageName" select="../@type"/>
                    <xsl:variable name="inputMessageName"
                            select="concat($inputMessageName,$id)"/>
                    <xsl:value-of select="$inputMessageName"/>
                    <xsl:text>"</xsl:text>

                    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
                    <xsl:text>outputVariable="</xsl:text>
                    <xsl:variable name="outputMessageType"
                            select="../../output/@type"/>
                    <xsl:variable name="outputMessageName"
                            select="concat($outputMessageType,$id)"/>
                    <xsl:value-of select="$outputMessageName"/>
                    <xsl:text>"</xsl:text>
                    <xsl:text disable-output-escaping="yes">/&gt;</xsl:text>
                </xsl:for-each>
            </xsl:if>
        </xsl:for-each>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/sequence&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:variable name="targetRank">0</xsl:variable>

        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;sequence name="Rank0WS"&gt;</xsl:text>
        <xsl:for-each select="//workflow/activity[@targetRank=0]">
            <xsl:variable name="taskID" select="@targetID"/>
            <xsl:variable name="targetTask" select="//workflow/task[@ID=$taskID]"/>
            <xsl:variable name="uuid"
                    select="//workflow/task[@ID=$taskID]/ServiceType/@ServiceTypeID"/>

            <xsl:variable name="inputMessageType"
                    select="$targetTask/ServiceType/portType/operation/input/@type"/>
```

```
<xsl:variable name="inputMessageName"
        select="$targetTask/ServiceType/portType/operation/input/@name"/>
<xsl:variable name="inputMessageName"
        select="concat($inputMessageType,$taskID)"/>


<xsl:variable name="sourceTask"
        select="//workflow/task[@type='input'␣and␣@clientID=$taskID]"/>
<xsl:variable name="sourceID" select="$sourceTask/@ID"/>

<xsl:variable name="partID" select="wfconverter:resetID()"/>

<!--assign-->
<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;assign name="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="concat($inputMessageName,$partID)"/>

␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">parameterassignment"&gt;</xsl:text>
<xsl:for-each select="$sourceTask/ServiceType/portType/operation">
    <xsl:variable name="partID" select="wfconverter:getID()"/>
    <xsl:variable name="outputMessageType" select="output/@type"/>
    <xsl:variable name="outputMessageName" select="output/@name"/>
    <xsl:variable name="outputMessageName"
            select="concat($outputMessageType,$sourceID)"/>


    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;copy&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;from&gt;</xsl:text>
    <xsl:value-of select="concat('$',$outputMessageName)"/>
    <xsl:text>.output</xsl:text>
    <xsl:value-of select="$partID"/>
    <xsl:text disable-output-escaping="yes">&lt;/from&gt;</xsl:text>


    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;to&gt;</xsl:text>
    <xsl:value-of select="concat('$',$inputMessageName)"/>
    <xsl:text>.in</xsl:text>
    <xsl:value-of select="$partID"/>
    <xsl:text disable-output-escaping="yes">&lt;/to&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;/copy&gt;</xsl:text>
</xsl:for-each>

<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/assign&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>

<!--invoke part-->
<xsl:variable name="plnk"
        select="concat($targetTask/serviceInstance/serviceName/text(),$taskID)"/>
<xsl:variable name="taskPortType"
        select="$targetTask/ServiceType/portType/@name"/>
<xsl:variable name="ptprefix"
        select="wfconverter:getNamespaceprefix($taskPortType,$taskID)"/>
<xsl:variable name="operation"
        select="$targetTask/ServiceType/portType/operation/@name"/>
<xsl:variable name="outputMessageType"
```

```
                    select="$targetTask/ServiceType/portType/operation/output/@type"/>
            <xsl:variable name="outputMessageName"
                    select="$targetTask/ServiceType/portType/operation/output/@name"/>
            <xsl:variable name="outputMessageName"
                    select="concat($outputMessageType,$taskID)"/>

            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>

            <xsl:text disable-output-escaping="yes">&lt;invoke name="invoke</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$targetTask/@name"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"</xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
            <xsl:text>partnerLink="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$plnk"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>"</xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
            <xsl:text>portType="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$ptprefix"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>:</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$taskPortType"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>"</xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
            <xsl:text>operation="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$operation"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>"</xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
            <xsl:text>inputVariable="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$inputMessageName"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>"</xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
            <xsl:text>outputVariable="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$outputMessageName"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"/&gt;</xsl:text>
        </xsl:for-each>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/sequence&gt;</xsl:text>


        <!--the rest of WS-->
        <xsl:for-each select="//workflow/activity[@targetRank&gt;0]">
            <xsl:variable name="taskID" select="@targetID"/>
            <xsl:variable name="targetTask" select="//workflow/task[@ID=$taskID]"/>
            <xsl:variable name="taskPortType"
                    select="$targetTask/ServiceType/portType/@name"/>
            <xsl:variable name="inputMessageType"
                    select="$targetTask/ServiceType/portType/operation/input/@type"/>
            <xsl:variable name="inputMessageName"
                    select="$targetTask/ServiceType/portType/operation/input/@name"/>
            <xsl:variable name="inputMessageName"
                    select="concat($inputMessageType,$taskID)"/>


            <xsl:variable name="partID" select="wfconverter:resetID()"/>


            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;sequence name="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$targetTask/@name"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"&gt;</xsl:text>
            <xsl:for-each select="source">
                <xsl:variable name="sourceID" select="@ID"/>
                <xsl:variable name="sourceTask" select="//workflow/task[@ID=$sourceID]"/>
        <xsl:variable name=”outputMessageType”
                select=”$sourceTask/ServiceType/portType/operation/output/@type”/>
        <xsl:variable name=”outputMessageName”
                select=”$sourceTask/ServiceType/portType/operation/output/@name”/>
        <xsl:variable name=”outputMessageName”
                select=”concat($outputMessageType,$sourceID)”/>
        <xsl:variable name=”dataType”
                select=”$sourceTask/ServiceType/portType/operation/output/part/@type”/>

        <xsl:for-each select=”map”>
            <xsl:variable name=”partID” select=”wfconverter:getID()”/>
            <xsl:call-template name=”array”>
                <xsl:with-param name=”source” select=”$outputMessageName”/>
                <xsl:with-param name=”dest” select=”$inputMessageName”/>
                <xsl:with-param name=”type” select=”$dataType”/>
                <xsl:with-param name=”partID” select=”$partID”/>
            </xsl:call-template>
        </xsl:for-each>
    </xsl:for-each>
    <!--invoke -->
    <xsl:variable name=”outputMessageType”
            select=”$targetTask/ServiceType/portType/operation/output/@type”/>
    <xsl:variable name=”outputMessageName”
            select=”$targetTask/ServiceType/portType/operation/output/@name”/>
    <xsl:variable name=”outputMessageName”
            select=”concat($outputMessageType,$taskID)”/>

    <xsl:variable name=”targetTaskServiceName”
            select=”$targetTask/serviceInstance/serviceName/text()”/>


    <xsl:call-template name=”invoke”>
        <xsl:with-param name=”name” select=”$targetTask/@name”/>
        <xsl:with-param name=”plnk” select=”concat($targetTaskServiceName,$taskID)”/>
        <xsl:with-param name=”taskPortType”
                select=”$targetTask/ServiceType/portType/@name”/>
        <xsl:with-param name=”ptprefix”
                select=”wfconverter:getNamespaceprefix($taskPortType,$taskID)”/>
        <xsl:with-param name=”operation”
                select=”$targetTask/ServiceType/portType/operation/@name”/>
        <xsl:with-param name=”inputMessageName” select=”$inputMessageName”/>
        <xsl:with-param name=”outputMessageName” select=”$outputMessageName”/>
    </xsl:call-template>

    <xsl:text disable-output-escaping=”yes”>&#10;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping=”yes”>&lt;/sequence&gt;</xsl:text>
    <xsl:text disable-output-escaping=”yes”>&#10;</xsl:text>
</xsl:for-each>

<!--OutputWS-->
<xsl:text disable-output-escaping=”yes”>&#10;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping=”yes”>&lt;sequence name=”OutputWS”&gt;</xsl:text>
<xsl:for-each select=”//workflow/activity[@isFile=’true’]”>
    <xsl:variable name=”fileName” select=”@fileName”/>
    <xsl:variable name=”sourceID” select=”source/@ID”/>
    <xsl:variable name=”sourceTask” select=”//workflow/task[@ID=$sourceID]”/>
    <xsl:variable name=”uuid” select=”$sourceTask/ServiceType/@ServiceTypeID”/>

    <xsl:variable name=”outputMessageType”
            select=”$sourceTask/ServiceType/portType/operation/output/@type”/>
```

```
<xsl:variable name="outputMessageName"
        select="$sourceTask/ServiceType/portType/operation/output/@name"/>
<xsl:variable name="outputMessageName"
        select="concat($outputMessageType,$sourceID)"/>
<xsl:variable name="dataType"
        select="$sourceTask/ServiceType/portType/operation/output/part/@type"/>

<xsl:variable name="outputWS"
        select="//workflow/task[@type='output'␣and␣@client=$uuid␣and
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣@clientID=$sourceID]"/>
<xsl:variable name="outputID" select="$outputWS/@ID"/>
<xsl:variable name="outputWSID" select="$outputWS/@ID"/>
<xsl:variable name="outputWSServiceName"
        select="$outputWS/serviceInstance/serviceName/text()"/>
<xsl:variable name="inputMessageType"
        select="$outputWS/ServiceType/portType/operation/input/@type"/>
<xsl:variable name="inputMessageName"
        select="$outputWS/ServiceType/portType/operation/input/@name"/>
<xsl:variable name="inputMessageName"
        select="concat($inputMessageType,$outputID)"/>

<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">
        &lt;assign name="parametersassignment"&gt;
</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;copy&gt;</xsl:text>


<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;from&gt;</xsl:text>
<xsl:text>'</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<!--
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="wfconverter:getOutputFileName($outputWS/@name)"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-->
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$fileName"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>'</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/from&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;to&gt;</xsl:text>
<xsl:value-of select="concat('$',$inputMessageName)"/>
<xsl:text>.in0</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/to&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/copy&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/assign&gt;</xsl:text>

<xsl:call-template name="array">
    <xsl:with-param name="source" select="$outputMessageName"/>
    <xsl:with-param name="dest" select="$inputMessageName"/>
    <xsl:with-param name="type" select="$dataType"/>
    <xsl:with-param name="partID">1</xsl:with-param>
</xsl:call-template>
```

```
    <xsl:variable name="outputMessageType"
            select="$outputWS/ServiceType/portType/operation/output/@type"/>
    <xsl:variable name="outputMessageName"
            select="$outputWS/ServiceType/portType/operation/output/@name"/>
    <xsl:variable name="outputMessageName"
            select="concat($outputMessageType,$outputWSID)"/>


    <xsl:call-template name="invoke">
        <xsl:with-param name="name" select="$outputWS/@name"/>
        <xsl:with-param name="plnk"
                select="concat($outputWSServiceName,$outputWSID)"/>
        <xsl:with-param name="taskPortType"
                select="$outputWS/ServiceType/portType/@name"/>
        <xsl:with-param name="ptprefix" select=
          "wfconverter:getNamespaceprefix($outputWS/ServiceType/portType/@name,$outputWSI
        <xsl:with-param name="operation"
                select="$outputWS/ServiceType/portType/operation/@name"/>
        <xsl:with-param name="inputMessageName"
                select="$inputMessageName"/>
        <xsl:with-param name="outputMessageName"
                select="$outputMessageName"/>
    </xsl:call-template>


</xsl:for-each>
<!--output 2 response-->
<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;assign&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;copy&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;from&gt;</xsl:text>
<xsl:text>'These␣output␣files␣are␣'</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/from&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;&#9;</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;to&gt;</xsl:text>
<xsl:text>$userResponse.out</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/to&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/copy&gt;</xsl:text>

<xsl:for-each select="//workflow/task[@type='output']">

    <xsl:variable name="outputWS" select="."/>
    <xsl:variable name="outputWSID" select="$outputWS/@ID"/>
    <xsl:variable name="outputMessageType"
            select="$outputWS/ServiceType/portType/operation/output/@type"/>
    <xsl:variable name="outputMessageName"
            select="$outputWS/ServiceType/portType/operation/output/@name"/>
    <xsl:variable name="outputMessageName"
            select="concat($outputMessageType,$outputWSID)"/>

    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
    <xsl:text disable-output-escaping="yes">&lt;copy&gt;</xsl:text>

    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;&#9;</xsl:text>
```

```
            <xsl:text disable-output-escaping="yes">&lt;from&gt;</xsl:text>
            <xsl:text>concat($userResponse.out,</xsl:text>
            <xsl:value-of select="concat('$',$outputMessageName)"/>
            <xsl:text>.out</xsl:text>
            <xsl:text>)</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/from&gt;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;&#9;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;to&gt;</xsl:text>
            <xsl:text>$userResponse.out</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/to&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/copy&gt;</xsl:text>

        </xsl:for-each>

        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/assign&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/sequence&gt;</xsl:text>


        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;reply name="replyResponse"</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">partnerLink="Client"</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes"> portType="pns:ProcessClientPT"</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">operation="runProcess"</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">variable="userResponse"/&gt;</xsl:text>

        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;/sequence&gt;</xsl:text>
    </xsl:template>

    <xsl:template name="array">
        <xsl:param name="source"/>
        <xsl:param name="dest"/>
        <xsl:param name="type"/>
        <xsl:param name="partID"/>
        <xsl:choose>
        <xsl:when test="contains($type,'Array')">
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;sequence name="copy-array</xsl:text>
____________<xsl:value-of_select="$source"/>
____________<xsl:text_disable-output-escaping="yes">TO</xsl:text>
____________<xsl:value-of_select="$dest"/>
____________<xsl:text_disable-output-escaping="yes">"&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                &lt;assign name="setupcounterandcount"&gt;
            </xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;&#9;</xsl:text>
```

```
<xsl:text disable-output-escaping="yes">&lt;copy&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;from&gt;1&lt;/from&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;to&gt;$counter&lt;/to&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;&#9;</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/copy&gt;</xsl:text>


<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;copy&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;from&gt;count(</xsl:text>
<xsl:value-of select="concat('$',$source)"/>
<xsl:text disable-output-escaping="yes">.out/def:item)&lt;/from&gt;</xsl:text>
<xsl:text disable-output-escaping="yes"
        >&#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;to&gt;$count&lt;/to&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/copy&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/assign&gt;</xsl:text>



<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;while&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;condition&gt;</xsl:text>
<xsl:text>$counter&lt;=$count</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/condition&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;sequence&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;assign&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;copy&gt;</xsl:text>
```

```
<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;from&gt;</xsl:text>
<xsl:value-of select="concat('$',$source)"/>
<xsl:text disable-output-escaping="yes">
        .out/def:item[$counter]&lt;/from&gt;
</xsl:text>

<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;to&gt;</xsl:text>
<xsl:value-of select="concat('$',$dest)"/>
<xsl:text disable-output-escaping="yes">.in</xsl:text>
<xsl:value-of select="$partID"/>
<xsl:text disable-output-escaping="yes">/def:item[$counter]&lt;/to&gt;</xsl:text>


<xsl:text disable-output-escaping="yes"
        >&#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/copy&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;copy&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">
        &lt;from&gt;$counter+1&lt;/from&gt;
</xsl:text>

<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;to&gt;$counter&lt;/to&gt;</xsl:text>


<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/copy&gt;</xsl:text>


<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/assign&gt;</xsl:text>

<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/sequence&gt;</xsl:text>
<xsl:text disable-output-escaping="yes">
        &#10;&#9;&#9;&#9;&#9;&#9;
</xsl:text>
<xsl:text disable-output-escaping="yes">&lt;/while&gt;</xsl:text>
```

```
            <xsl:text disable-output-escaping="yes">
                    &#10;&#9;&#9;&#9;&#9;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/sequence&gt;</xsl:text>
        </xsl:when>
        <xsl:otherwise>

                <xsl:text disable-output-escaping="yes">
                        &#10;&#9;&#9;&#9;&#9;
                </xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;assign&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">
                    &#10;&#9;&#9;&#9;&#9;&#9;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;copy&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">
                    &#10;&#9;&#9;&#9;&#9;&#9;&#9;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;from&gt;</xsl:text>
            <xsl:value-of select="concat('$',$source)"/>
            <xsl:text>.out</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/from&gt;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                    &#10;&#9;&#9;&#9;&#9;&#9;&#9;
            </xsl:text>


            <xsl:text disable-output-escaping="yes">
                    &#10;&#9;&#9;&#9;&#9;&#9;&#9;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;to&gt;</xsl:text>
            <xsl:value-of select="concat('$',$dest)"/>
            <xsl:text>.in</xsl:text>
            <xsl:value-of select="$partID"/>
            <xsl:text disable-output-escaping="yes">&lt;/to&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">
                    &#10;&#9;&#9;&#9;&#9;&#9;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/copy&gt;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                    &#10;&#9;&#9;&#9;&#9;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/assign&gt;</xsl:text>
        </xsl:otherwise>
        </xsl:choose>
    </xsl:template>

    <xsl:template name="invoke">
        <xsl:param name="name"/>
        <xsl:param name="plnk"/>
        <xsl:param name="taskPortType"/>
        <xsl:param name="ptprefix"/>
        <xsl:param name="operation"/>
        <xsl:param name="inputMessageName"/>
        <xsl:param name="outputMessageName"/>

        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
        <xsl:text disable-output-escaping="yes">&lt;invoke name="invoke</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$name"/>
```

```
␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
        <xsl:text>partnerLink="</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$plnk"/>
␣␣␣␣␣␣␣␣<xsl:text>"</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
        <xsl:text>portType="</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$ptprefix"/>
␣␣␣␣␣␣␣␣<xsl:text>:</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$taskPortType"/>
␣␣␣␣␣␣␣␣<xsl:text>"</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
        <xsl:text>operation="</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$operation"/>
␣␣␣␣␣␣␣␣<xsl:text>"</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
        <xsl:text>inputVariable="</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$inputMessageName"/>
␣␣␣␣␣␣␣␣<xsl:text>"</xsl:text>
        <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
        <xsl:text>outputVariable="</xsl:text>
␣␣␣␣␣␣␣␣<xsl:value-of␣select="$outputMessageName"/>
␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"/&gt;</xsl:text>
    </xsl:template>
</xsl:stylesheet>
```

B.2 Stylesheet to Generate the PDD File

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:wfconverter="xalan://WFwsConverter"
         extension-element-prefixes="wfconverter">
        <xsl:output method="xml" indent="yes"/>
        <xsl:template match="/">
            <xsl:text disable-output-escaping="yes">&lt;process</xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">
                        xmlns="http://schemas.active-endpoints.com/pdd/2005/09/pdd.xsd"
                </xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">
                        xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
                </xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">
                        xmlns:bpelns="http://concrete/FlowAuto"
                </xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">
                        name="bpelns:concreteprocess"
                </xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">location="bpel/concrete.bpel"</xsl:text>
                <xsl:text disable-output-escaping="yes">&gt;</xsl:text>

                <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">&lt;partnerLinks&gt;</xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">
                        &lt;partnerLink name="Client"&gt;
                </xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">&lt;myRole service="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">serviceProvider</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="wfconverter:getid()"/>

␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" allowedRoles="" binding="RPC"/&gt;
                </xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">&lt;/partnerLink&gt;</xsl:text>

                <xsl:apply-templates select="workflow"/>

                <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
                <xsl:text disable-output-escaping="yes">&lt;/partnerLinks&gt;</xsl:text>

                <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
                <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
                <xsl:call-template name="addwsdlReference"/>

            <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/process&gt;</xsl:text>
        </xsl:template>

        <xsl:template match="workflow" >
            <xsl:apply-templates select="task"/>
        </xsl:template>



        <xsl:template match="task">
            <xsl:variable name="id" select="@ID"/>
            <xsl:variable name="serviceName" select="serviceInstance/serviceName/text()"/>
            <xsl:variable name="servicePortName" select="serviceInstance/portName/text()"/>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;partnerLink name="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="concat($serviceName,$id)"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"&gt;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">
                &lt;partnerRole endpointReference="static"&gt;
            </xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;wsa:EndpointReference xmlns:</xsl:text>
            <xsl:value-of select="concat($serviceName,$id)"/>
            <xsl:text>="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="serviceInstance/targetNamespace/text()"/>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;wsa:Address&gt;</xsl:text>
            <xsl:value-of select="$serviceName"/>
            <xsl:text disable-output-escaping="yes">:anyURL&lt;/wsa:Address&gt;</xsl:text>

            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;wsa:ServiceName PortName="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$servicePortName"/>

␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text␣disable-output-escaping="yes">"&gt;</xsl:text>
            <xsl:value-of select="concat($serviceName,$id)"/>
            <xsl:text disable-output-escaping="yes">:</xsl:text>
            <xsl:value-of select="$serviceName"/>

            <xsl:text disable-output-escaping="yes">&lt;/wsa:ServiceName&gt;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>

            <xsl:text disable-output-escaping="yes">&lt;/wsa:EndpointReference&gt;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/partnerRole&gt;</xsl:text>
            <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;</xsl:text>
            <xsl:text disable-output-escaping="yes">&lt;/partnerLink&gt;</xsl:text>

        </xsl:template>

        <xsl:template name="addwsdlReference">
             <wsdlReferences>
             <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
             <xsl:for-each select="//workflow/task[@type='normal']">
                 <xsl:variable name="taskID" select="@ID"/>
                 <xsl:variable name="ptNS" select="ServiceType/namespace/text()"/>
                 <xsl:variable name="ptURL" select="ServiceType/urlLocation/text()"/>
                 <xsl:variable name="serviceNS"
                         select="serviceInstance/targetNamespace/text()"/>
                 <xsl:variable name="serviceURL"
                         select="serviceInstance/accesspoint/text()"/>


                 <xsl:variable name="uuid" select="ServiceType/@ServiceTypeID"/>
                 <xsl:variable name="exist" select="wfconverter:isExist()"/>

                 <xsl:for-each
                     select="//workflow/task[@ID&gt;$taskID]/ServiceType[@ServiceTypeID=$uuid]">
                     <xsl:variable name="exist" select="wfconverter:setExist()"/>
                </xsl:for-each>

                <xsl:if test="wfconverter:isExist()=false">
                    <xsl:text disable-output-escaping="yes">&#9;&#9;&#9;</xsl:text>
                    <xsl:text disable-output-escaping="yes">&lt;wsdl </xsl:text>

                    <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
                    <xsl:text disable-output-escaping="yes">namespace="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$ptNS"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>"</xsl:text>

                    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;&#9;</xsl:text
                    <xsl:text disable-output-escaping="yes">location="project:/wsdl/</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="wfconverter:getfileName($ptURL)"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>"</xsl:text>
                    <xsl:text disable-output-escaping="yes">/&gt;</xsl:text>
                    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>



                    <xsl:text disable-output-escaping="yes">&#9;&#9;&#9;</xsl:text>
                    <xsl:text disable-output-escaping="yes">&lt;wsdl </xsl:text>

                    <xsl:text disable-output-escaping="yes">&#9;</xsl:text>
                    <xsl:text disable-output-escaping="yes">namespace="</xsl:text>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:value-of␣select="$serviceNS"/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<xsl:text>"</xsl:text>

                    <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
                    <xsl:text disable-output-escaping="yes">location="project:/wsdl/</xsl:text>
```

```
                     <xsl:value-of select="wfconverter:getfileName($serviceURL)"/>
                     <xsl:text>"</xsl:text>
                     <xsl:text disable-output-escaping="yes">/&gt;</xsl:text>
                     <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
                 </xsl:if>
                 <xsl:variable name="exist" select="wfconverter:resetExist()"/>

              </xsl:for-each>

              <xsl:for-each
                      select="//workflow/task[@type='input' or @type='output']/serviceInstance">
                 <xsl:variable name="serviceURL" select="accesspoint/text()"/>
                 <xsl:variable name="taskID" select="../@ID"/>
                 <xsl:variable name="uuid" select="../@client"/>

                 <xsl:if test="count(//workflow/task[@ID&gt;$taskID and @client=$uuid])=0">
                     <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>

                     <xsl:text disable-output-escaping="yes">&lt;wsdl namespace="</xsl:text>
                     <xsl:value-of select="targetNamespace/text()"/>
                     <xsl:text disable-output-escaping="yes">"</xsl:text>

                     <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
                     <xsl:text disable-output-escaping="yes">location="project:/wsdl/</xsl:text>
                     <xsl:value-of select="wfconverter:getfileName($serviceURL)"/>
                     <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
                 </xsl:if>
              </xsl:for-each>

              <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;</xsl:text>
              <xsl:text disable-output-escaping="yes">
                      &lt;wsdl namespace="http://concreteworkflow0/auto"
              </xsl:text>

              <xsl:text disable-output-escaping="yes">&#10;&#9;&#9;&#9;&#9;</xsl:text>
              <xsl:text disable-output-escaping="yes">
                      location="project:/wsdl/concretewfws.wsdl"/&gt;
              </xsl:text>

              <xsl:text disable-output-escaping="yes">&#10;&#9;</xsl:text>
              </wsdlReferences>
        </xsl:template>

        <xsl:template name="addExtension">
           <extensions>
              <extension mustUnderstand="yes"
                 namespace="http://www.activebpel.org/2006/09/bpel/extension/query_handling"/>
             </extensions>
        </xsl:template>
</xsl:stylesheet>
```

B.3 Stylesheet to Generate the Catalog File

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:wfconverter="xalan://WFwsConverter"
                extension-element-prefixes="wfconverter">
    <xsl:variable name="object" select="wfconverter:new()"/>
    <xsl:output method="xml" indent="yes"/>
          <xsl:template match="/">
```

```
            <xsl:element name="catalog"
                xmlns="http://schemas.active-endpoints.com/catalog/2006/07/catalog.xsd">
            <xsl:apply-templates select="workflow"/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="workflow">
        <xsl:apply-templates select="task"/>
        <xsl:element name="wsdlEntry"
                xmlns="http://schemas.active-endpoints.com/catalog/2006/07/catalog.xsd">
                <xsl:attribute name="location">
                        project:/wsdl/concretewfws.wsdl
                </xsl:attribute>
            <xsl:attribute name="classpath">wsdl/concretewfws.wsdl</xsl:attribute>
        </xsl:element>

        <xsl:element name="schemaEntry"
                xmlns="http://schemas.active-endpoints.com/catalog/2006/07/catalog.xsd">
                <xsl:attribute name="location">
                        project:/wsdl/doubleArray1.xsd</xsl:attribute>
                <xsl:attribute name="classpath">
                        wsdl/doubleArray1.xsd</xsl:attribute>
        </xsl:element>
    </xsl:template>

    <xsl:template match="task">
        <xsl:if test="@type='normal'">
            <xsl:variable name="taskID" select="@ID"/>
            <xsl:variable name="uuid"
                    select="ServiceType/@ServiceTypeID"/>

            <xsl:if test=
             "count(//workflow/task[@ID&gt;$taskID]/ServiceType[@ServiceTypeID=$uuid])=(
                <xsl:apply-templates select="ServiceType"/>
                <xsl:apply-templates select="serviceInstance"/>
            </xsl:if>

        </xsl:if>

        <xsl:if test="@type='input'␣or␣@type='output'">
            <xsl:variable name="taskID" select="@ID"/>
            <xsl:variable name="uuid" select="ServiceType/@ServiceTypeID"/>
            <xsl:if test="count(//workflow/task[@ID&gt;$taskID␣and␣@client=$uuid])=0">
                <xsl:apply-templates select="serviceInstance"/>
            </xsl:if>
        </xsl:if>

    </xsl:template>

    <xsl:template match="ServiceType">
            <xsl:element name="wsdlEntry"
                xmlns="http://schemas.active-endpoints.com/catalog/2006/07/catalog.xsd">
            <xsl:variable name="pturl" select="urlLocation/text()"/>
            <xsl:variable name="ptFileName" select="wfconverter:getfileName($pturl)"/>
            <xsl:attribute name="location">
                <xsl:value-of select="concat('project:/wsdl/',$ptFileName)"/>
            </xsl:attribute>
            <xsl:attribute name="classpath">
                <xsl:value-of select="concat('wsdl/',$ptFileName)"/>
            </xsl:attribute>
            </xsl:element>
    </xsl:template>
```

```
        <xsl:template match="serviceInstance">
                <xsl:element name="wsdlEntry"
                    xmlns="http://schemas.active-endpoints.com/catalog/2006/07/catalog.xsd">
                        <xsl:variable name="serviceurl"
                                select="accesspoint/text()"/>
                        <xsl:variable name="serviceFileName"
                                select="wfconverter:getfileName($serviceurl)"/>
                    <xsl:attribute name="location">
                        <xsl:value-of select="concat('project:/wsdl/',$serviceFileName)"/>
                    </xsl:attribute>
                    <xsl:attribute name="classpath">
                        <xsl:value-of select="concat('wsdl/',$serviceFileName)"/>
                    </xsl:attribute>
                </xsl:element>
        </xsl:template>
</xsl:stylesheet>
```