Publish/Subscribe Scientific Workflow Interoperability Framework (PS-SWIF)

Ahmed Alqaoud

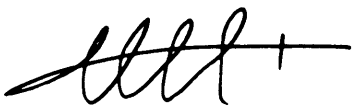
School of Computer Science & Informatics
Cardiff University

January 2010

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

DECLARATION

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

Signed  Date 03/03/2010

STATEMENT 1

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

Signed  Date 03/03/2010

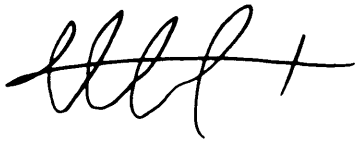
STATEMENT 2

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

Signed  Date 03/03/2010

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 organisations.

Signed  Date 03/03/2010

Acknowledgements

Praise to Allah (God) Almighty for providing me with faith, patience and the commitment to complete this research.

My special admiration and gratitude to my parents, sister, brothers, aunts and uncles whose prayers, love, care, patience, support and encouragement have always enabled me to perform to the best of my abilities.

I must thank my supervisors, Dr Ian Taylor and Dr Andrew Jones, for their expert guidance and support throughout this research. I am grateful for their careful reading and constructive comments on this thesis and our joint papers; without their constant advice and help this thesis could not have been completed.

I acknowledge, with grateful thanks, the Saudi government, represented by the Higher Education Ministry, for sponsoring me throughout the research period.

Special thanks are due to the members of the school for their help, especially Helen Williams for her help on administrative issues, Robert Evans for his technical assistance, and Andrew Harrison for help with coding at the beginning of this project.

To my special friends Dr. Badr Aldaihani, Dr Hassan Khayt Dr. Abdullah Althyab, Dr. Fahad Alwasl, Dr. Mohammad Alkarawi, Raied Alamri, Dr. Mohammad Alymi, Homoud Aldossari, Ahmed Alazzawi, Sultan Alyahya, Waleed Alnuwaiser, and Yasser Alosefer, my thanks for their support, encouragement and, most importantly, for their friendship.

My sincere gratitude to my friends in Saudi Arabia, and especially to my brother Faisal Alqaoud for his unconditional support, encouragement and help, Osama Aldosary for his support and advice, and Abdularhman AlMunief for taking care of my personal issues in Saudi Arabia, for his true friendship and keeping in touch at all times.

Last, but certainly not least, I am indebted to my wife Maram for her endurance and unconditional love which provided vital encouragement during my PhD study. Finally, my love to my adored child Saad; born while I completed this thesis.

Dedication

For my aunts; Noura and Aljohara Alqaoud

I dedicate this thesis to my aunts, Noura Alqaoud and Aljohara Alqaoud for their love, affection, prayers and support; not only during this research period but throughout my life. Without them I could never have reached this stage in my life.

Abstract

Scientific workflow is a special type of workflow for scientists to formalize and structure complex e-Science applications. Many workflow systems have been released to solve problems in special domains. In large collaborative projects, it is often necessary to recognize the heterogeneous workflow systems already in use by various partners and any potential collaboration between these systems requires workflow interoperability. Workflow interoperability has received much interest from the distributed computing community and many workshops and meetings have been organized to discuss, from different perspectives, how interoperability can be achieved among scientific Workflow Systems.

In this thesis, a general approach to achieving interoperability among workflow systems, based on a WS-based notification messaging system, is proposed. This approach presents a Publish/Subscribe Scientific Workflow Interoperability Framework (PS-SWIF) and for validation, it is implemented in multiple workflow systems to demonstrate run-time interoperability. The Publish/Subscribe paradigm provides a loosely-coupled communication pattern for large scale distributed computing and the resulting asynchronous messaging exchange between workflow systems promises scalability and flexibility for distributed applications.

The PS-SWIF system is based on Web Services that enable scientists to use a Publish/Subscribe mechanism to publish a topic, and enables different workflow systems to subscribe to this topic and receive notification messages when an event is executed in the first workflow. The second workflow system can further process results and send to further systems, if applicable, in a similar way.

Different or similar workflow systems, hosted anywhere on a network, written in any language and running on different operating systems, can easily use the full range of PS-SWIF tools to interoperate with each other. The PS-SWIF approach provides interoperability among a wide range of scientific workflow systems.

Contents

Tables

Figures

Glossary

ASAP	Asynchronous Service Access Protocol
BPEL4WS	Business Process Execution Language for Web Services
CFD	Computational Fluid Dynamics
CIPRES	Cyberinfrastructure for Phylogenetic Research
CORBA	Common Object Requesting Broker Architecture
DAG	Directed Acyclic Graph
DART	Distributed Audio Retrieval Using Triana
DDBJ	DNA Data Bank of Japan
DIPSO	Environment for Industrial Design Optimization
EDGeS	Enabling Desktop Grids for e-Science
EMBL-EBI	European Molecular Biology Laboratory-European Bioinformatics Institute
FAEHIM	Federated Analysis Environment for Heterogeneous Intelligent Mining
GAP	Grid Application Prototype
GAT	Grid Application Tool
GEMLCA	Grid Execution Management for Legacy Code Architecture
GEMSS	Grid-Enabled Medical Simulation Services
GENIUS	Grid Enabled Web eNvironment for site Independent User job Submission
GEO600	Gravitational Wave Project
GEON	Geosciences Network
GGF	Global Grid Forum
GRAM	Grid Resource Allocation Management
GRMS	GridLab Resource Management System
ICENI	Imperial College e-Science Network Infrastructure
IWR	Intermediate Workflow Representation
JMS	Java Message Service
MOTEUR	hoMe-made OpTimisEd scUfl enactoR
NATs	Network Address Translations
NCBI	National Center for Biotechnology Information
OGF	Open Grid Forum
OGSI	Open Grid Services Infrastructure
OMG	Object Management Group

PS-SWIF	Publish/Subscribe Scientific Workflow Interoperability Framework
QoS	Quality of Services
ROADNet	Real-time Observatories, Applications, and Data Management Network
Scufl	Simple conceptual unified flow language
SEEK	Science Environment for Ecological Knowledge
SOAP	Simple Object Access Protocol
SODIUM	Service Oriented Development In a Unified framework
SPA	Scientific Process Automation
TRIACS	Triana Advanced Clinical Support
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
USCL	Unified Service Composition Language
UUID	Universally Unique Identifier
WDLs	Workflow Definition Languages
WFEE	Workflow Enactment Engine
WfMC	Workflow Management Coalition
WHIP	Workflows Hosted In Portals
WS	Web Services
WSDL	Web Service Description Language
WS-RF	Web Services Resource Framework
XML	eXtensible Markup Language
XPDL	XML Process Definition Language

CHAPTER 1

Introduction

1.1 Overview

This Chapter presents an outline of the research thesis and opens with a background to provide an overview of the project and the motivation behind the research. The aim, objectives, research questions and hypothesis are then described, which define the scope of this project. The research methodology for the project is described, and the contributions and achievement section lists papers emerging from the project. Finally, the organisation of the thesis is presented.

1.2 Background

Workflow systems have become attractive for scientific computing projects, especially for their ability to describe experimental processes in a way that makes it easy to create, manage and execute such projects over a distributed set of resources. Workflow in the business community is defined by the Workflow Management Coalition (WfMC) as '*The automation of a business process, in whole or parts, where documents, information or tasks are passed from one participant to another to be processed, according to a set of procedural rules*' [2] . This definition unfortunately does not accurately capture current needs for scientific applications in Grid environments. In Grid environments, e-Science applications become more complex in the context of managing and processing large data sets and executing scientific experiments on widely spread

computing resources. Fox and Gannon [3] define workflow in a grid context as '*The automation of the processes, which involves the orchestration of a set of Grid services, agents and actors that must be combined together to solve a problem or to define a new service*'.

There are various workflow systems to resolve problems in special domains, such as gravitational-wave physics, geophysics, bioinformatics and astronomy. In each of these domains, a variety of tools and functions are available to scientists. In large collaborative projects, it is often necessary to recognize the heterogeneous workflow systems already in use by various partners and any potential collaboration between these systems requires workflow interoperability.

Workflow interoperability was officially addressed for the first time in 1996 by the Workflow Management Coalition (WfMC) [4], with the WfMC defining Workflow interoperability as: '*the ability of two or more workflow engines to communicate and interoperate in order to coordinate and execute workflow process instances across those engines*'. The WfMC has published different standards and specifications [4-6] to achieve workflow interoperability at different levels and using various models. Since that time there has been little research in this area.

Over the last five years, a variety of workflow systems have been released and a diversity of e-Science projects have made scientific workflow interoperability an important subject for researchers. Many workshops and meetings [7-10] have been organized by distributed computing committees to discuss, from different perspectives, how interoperability can be achieved among scientific Workflow Systems. Workflow interoperability can be classified at different levels according to the workflow lifecycle presented by Deelman [11]

In this thesis, a general approach to achieving interoperability among workflow systems, based on a WS-based notification messaging system, is proposed. This

approach presents a Publish/Subscribe Scientific Workflow Interoperability Framework (PS-SWIF) and for validation, it is implemented in multiple workflow systems to provide run-time interoperability. The Publish/Subscribe paradigm provides a loosely-coupled communication pattern for large scale distributed computing and the resulting asynchronous messaging exchange between workflow systems promises scalability and flexibility for distributed applications.

Communication using an asynchronous messaging paradigm is characterized by a decoupling of participants in both time and space [12]. The PS-SWIF system is based on Web Services that enable scientists to use a Publish/Subscribe mechanism to publish a topic, and enables different workflow systems to subscribe to this topic and receive notification messages when an event is executed in the first workflow. The second workflow system can further process results and send to further systems, if applicable, in the similar way.

1.3 Significant of the Interoperability

For collaboration between different systems and tools, interoperability is essential. Within large collaborative projects combinations of workflow systems are already in use. Workflow interoperability is a significant problem that can determine if collaboration between e-Science projects, using heterogeneous workflow systems, can be successfully conducted. This section discusses the significance of interoperability when it is applied to real world projects in the science and industry field.

1.3.1 SIMDAT Project

SIMDAT [13] Data grids for process and product development, using numerical simulation and knowledge discovery, started in 2004. The SIMDAT project is funded by the European Commission under the Information Society Technologies Programme (IST). There are several applications with SIMDAT, four of which address interoperability challenges in the Grid for industrial

applications, namely: the aerospace industry [14], automotive application [15], meteorology [16], and pharmaceuticals [17]. The author will discuss the aerospace industry project and more about other projects can be found in these literatures.

1.3.1.1 Interoperability between 5 Workflow Engines in the Aerospace

This is a joint programme between the SIMDAT partners BAE Systems, EADS Innovation Works, IT Innovation and the School of Engineering Sciences at the University of Southampton.

Within the aerospace industry highly complex products that have data creation, management and curation requirements are expanded over hundreds of collaborating organizations. The aerospace activity project focuses on demonstrating how to deploy Grid technologies in order to aid complex collaborative engineering tasks. The majority of product developments within the aerospace industry take place in a collaborative fashion and these collaborations span organization and international boundaries. '*The goal of this phase of the project was to demonstrate interoperability in the Grid system and produce a prototype showing this interoperability in action*' [14].

The system was built from different workflows, from a top level view to the individual analysis Service workflows. The responsibility for each workflow belonged to a different company which used different workflows system to construct these workflows.

'*The original Aerospace scenario was based just on the Taverna workflow tool. For the interoperability phase of the project, the aim of the aerospace scenario was to demonstrate interoperability between the workflow tools. This is necessary as each organisation will has its own favoured processes and workflow tools and as outlined in the SIMDAT Workflow Interoperability Framework, organisations should not have to be tied to migrate to another workflow system. Within the aerospace sector, 4 problem solving environments*

were identified to be integrated into the scenario; ModelCenter, FIPER, MATLAB and Optimus' [18].

ModelCenter [19] is a graphical problem solving environment provided by Phoenix Integration. It is used heavily within the aerospace industry for product optimisation and integration within BAE Systems. ModelCenter was integrated with GRIA (Grid Resources for Industrial Applications). GRIA provides the core job Services, the ability to upload and download data to data stagers. GRIA Service is provided by SIMDAT to allow engineers to access distributed Services and workflows from applications using a workflow system they are familiar with.

FIPER [20] is a system engineering framework that allows organizations to deploy their integrated applications to more users and run them on a powerful computing framework. FIPER is integrated with GRIA to provide more flexible way to integrate distributed business Services.

MATLAB [21] is a high-level language and environment that enables a user to perform computational tasks faster than with traditional programming languages. MATLAB is used widely in the aerospace industry.

The OPTIMUS [22] workflow tool is provided by LMS/NOESIS as a multi-disciplinary program that automates simulation tasks across multiple engineering disciplines.

In this scenario, interoperability is said to be achieved when a workflow published by a workflow system can be accessed from another workflow system through the GRIA 5 job Service interface [18]. Therefore, the level of interoperability is through a third party job Service interface and therefore differs to that provided by PS-SWIF. In the aerodynamics analysis Service the interaction is more complex and requires different workflow tools interacting. Optimus provides the response surface modelling (RSM) techniques used in the

aerodynamic workflows as surrogate data models. These tools are accessible through GRIA Services, and also the Optimus workflow engine is available as a Service. This means that the RSM workflows can be constructed in the Optimus engine and then published as a GRIA job Service. In a similar way this can be achieved with Taverna. The ModelCenter workflow tool is also used to tie the published workflows together into a third workflow and enact this published workflow using both the Optimus and Freefluo Services. Figure 1.1 shows these interactions.

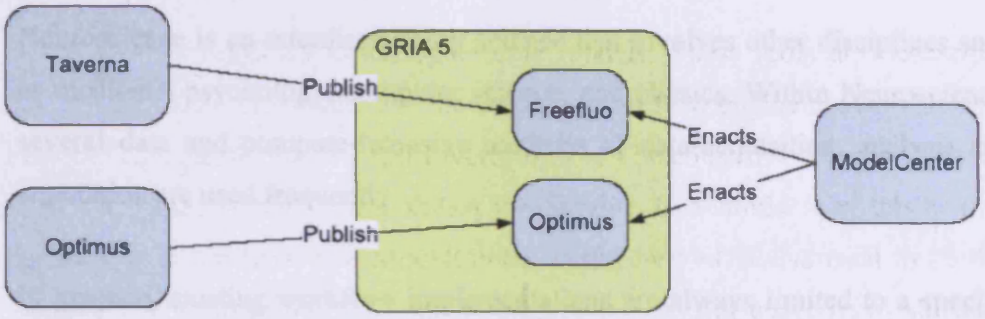


Figure 1.1 Interactions between Taverna/Freefluo, Optimus and ModelCenter

The prototypes successfully demonstrated runtime workflow interoperability, showing how experts in a specific domain can publish their workflows and be available to be called from any of the other workflow environments.

1.3.2 SHIWA Project

SHIWA [23] **SH**aring **I**nteroperable **W**orkflows for large-scale scientific simulations on **A**vailable DCIs (Distributed Computing Infrastructures) is a project submitted to the EU; the 7th Research Framework Programme (FP7). The primary goal of the project is to design Services that provide workflow interoperability between five leading scientific workflow systems: ASKALON [24], MOTEUR [25], P-GRADE [26], Pegasus [11] and Triana [27], using coarse-grained and fine-grained interoperability approaches.

Coarse-grained interoperability is defined as one workflow system that can be invoked or embedded by another workflow system as a local or distributed Service. The fine-grained interoperability acts at the workflow language level and enables the execution of an arbitrary workflow through the invocation of any workflow engine into which the workflow description can be converted.

Three use cases have been proposed to be supported by the Services operated by SHIWA.

1.3.2.1 Interoperable Workflows for Neuroimaging

Neuroscience is an interdisciplinary science that involves other disciplines such as medicine, psychology, computer science, and physics. Within Neuroscience, several data and compute-intensive methods of data-acquisition, analysis and simulation are used frequently.

In practice, existing workflow implementations are always limited to a specific workflow system. For example, the AMC (Academic Medical Center of the University of Amsterdam) and Charité (University of Berlin) have both developed a variety of workflows for analysis of functional MRI data.

These implementations/workflows could be exchanged between various groups and implemented by different workflow systems in different organizations. Unfortunately the workflows available at Charité and AMC (and the other groups) are not compatible because they were built for specific workflow systems and Grid infrastructures. As a result, great effort is needed to interoperate between different workflows and combine them.

The primary goal of this scenario is to facilitate researchers in neuroscience to access and integrate the different methods into their data processing. Depending on the requirements, coarse-grained or fine-grained interoperability can be applied to provide the most effective and appropriate solution.

1.3.2.2 Creating and running meta-workflows – Quantitative evaluation of medical imaging algorithms

The quantitative evaluation of medical image algorithms is an important part of the scientific process. It is a difficult process due to the variety of considered problems and the variability among images.

Evaluation methods such as image segmentation and registration are typical image processing problems. The goal of this project is to enable the sharing of quantitative evaluation methods among users of various workflows with various workflow systems.

Interoperability approaches provided by the SHIWA project will provide a solution to integrate statistical evaluation methods developed in various workflow systems into a single evaluation pipeline by enabling the integration of multiple workflows hosted on different systems into a single master workflow [23].

1.3.2.3 Running workflows on multiple DCIs – Access to computing resources for medical simulation

In the medical sector, simulation has become a major tool and deals with a variety of phenomena (for instance, image acquisition in different modalities, cancer therapy, etc.) and is multi-scale regarding the simulated objects and the corresponding computing challenges. Several simulators have been integrated with a Virtual Imaging Platform and provide access to DCIs. This platform is dependent on workflows, described with the Scufl language and enacted on the MOTEUR engine through the GASW backend (Generic Application Service Wrapper).

However, MOTEUR can only access the computing resources of the EGEE infrastructure (Enabling Grids for e-Science), which not only suffers from periodical resource shortage but also is suitable only for some simulation

experiments. For example, wide-scale Monte-Carlo simulations require the provision of a few thousand computing nodes over a very limited time-period, which is difficult to achieve on a single DCI. Besides, experiments requiring capacity infrastructures (e.g. MPI 'Message Passing Interface' applications) are known to suffer from limitations on capability DCIs such as EGEE.

Interoperability solutions provided by SHIWA will enable access to the target DCIs and technical issues such as data sharing between DCIs, the management of credentials, and error handling could be addressed. Also, it will be used to benefit from the capabilities of the various workflow engines on the targeted DCIs.

1.4 Definition and Scope of Interoperability

Interoperability can be defined and classified at three different levels: workflow design, workflow mapping and execution, and data provenance. Workflow design describes a selection of application components and defines their dependencies. The application components could be tasks, jobs, services or any executable units. Workflow mapping and execution refers to turning the components application in the workflow design into an executable state. Workflow mapping refers to the process that maps an abstract workflow to appropriate resources. Mapping a workflow onto resources requires certain issues to be considered, namely information retrieval, workflow QoS constraints, workflow scheduling, fault tolerance and data provenance. Data provenance refers to the ability to obtain the history of data products.

The author presents an interoperability solution that will be applied at workflow mapping and execution level. Specifically, the scope of interoperability presented in this thesis is concerned with a mechanism for transferring and managing data to and from workflow systems. However, error and exception handling and transactions are out of the scope of this thesis. The author defines managing data as the ability of the user to issue different types of requests. These requests are: (1) subscribe request to allow a user to receive data for a

period of time; (2) unsubscribe request to show that a user has no interest in receiving data; and (3) renew the subscription which allows the user to receive the data for another time period. Further, the author provides additional flexibility by providing multiple communications mechanisms, "asynchronous and synchronous", which are required to support different models used within the different workflow systems. The ability to perform both blocking and non-blocking communication provides multiple critical configurable and programmatical options for workflow systems, including the ability to parallel process third party workflow systems and expose a workflow system itself as a third party service. The author therefore defines interoperability as:

'*The ability to exchange and manage data and control the communication between two or more workflow engines at run time during their execution*'.

1.5 Motivation

The thesis is motivated by the desire and need to achieve interoperability between workflow systems; particularly for scientific applications. Scientists need to repeat their experiments across different workflow systems to get consistent results. Scientists also need to invoke and use different tools from other systems which are not available on their own workflow system to complete their experiments or improve performance results. For collaboration between different systems and tools, interoperability is essential. Within large collaborative projects [28-30] combinations of workflow systems are already in use. Workflow interoperability is a significant problem that can determine if collaboration between e-Science projects, using heterogeneous workflow systems, can be successfully conducted.

This thesis proposes a system to be utilized in such architectures to provide interoperability between workflow systems at message passing levels, using asynchronous notification messaging methods.

1.6 Hypothesis, Aims and Objectives

Research Hypothesis

> *'It is possible to achieve interoperability between workflow systems, by using a generic approach that leverages asynchronous notification messaging and Web Services standards. Such an approach will enable the movement and management of data and will control the communication between different workflow systems to provide maximum flexibility and simplicity for scientists, in terms of the variety of workflow systems supported and their execution in both local and remote environments '.*

The aim of the thesis is thus to investigate the best approaches for achieving interoperability among scientific workflow systems and to develop a system to extend and transition abstract theories to solve a practical problem. To guide the development of such a system, a number of research objectives are defined:

1) Identify Workflow Interoperability Levels and Models

This objective intends to answer the questions:

- How interoperability levels and models are classified and identified?
- What level and models for workflow interoperability will be achieved by conducting this research?
- How workflow interoperability is achieved in current projects?

2) Identify the Appropriate Publish/Subscribe Model for Workflow Systems

Investigate existing publish/subscribe models and determine the appropriate candidate for workflow systems. This objective intends to answer the questions:

- What type of functions from the publish/subscribe paradigm should be investigated?
- How should a notification message be delivered to subscribers?
- What requirements and specifications must be supported to adapt a publish/subscribe system?
- What should the architecture for a workflow interoperability framework look like?

3) Design and Develop a System that Uses the Publish/Subscribe Paradigm to Achieve Workflow Interoperability.

To achieve interoperability among workflow systems, the possible architecture for a system that facilitates passing asynchronous notification messages among the workflow systems must be investigated. This objective addresses the questions:

- How can a publish/subscribe model be utilised by the different workflow systems?
- How a workflow system sends and receives notification messages?
- How can different interoperability models among different workflow systems using a publish/subscribe model be achieved?

4) Develop a Framework that Allows Scientists to Run their Experiments among Different Workflow Systems

This objective intends to answer the questions:

- What are the requirements needed for a framework to provide tools and services for workflow interoperability, without the need for programming or a deep technical computer background?
- How can the framework provide run-time interoperability among different workflow systems in different environments?
- How can workflow interoperability be achieved remotely?
- How can systems expose experiments for re-use without major modifications to their workflow?

1.7 Research Methodology

The scientific workflow system is chosen as a research domain because of a high demand by e-Science applications. The literature review determines the current state of existing workflow systems and defines the most important problems that need be addressed. Workflow interoperability is a significant problem that can even affect whether if collaboration between e-Science projects that use heterogeneous workflow systems can be successfully conducted.

The research methodology adopted for this research is based on a process iteration model, especially the spiral development model, as defined by Sommerville [31] :

> *'Rather than represent the software process as a sequence of activities with some backtracking from one activity to another, the process is represented as a spiral. Each loop in the spiral represents a phase of the software processes.*

Figure 1.2 overleaf shows the spiral model with each loop split into four sectors:

1. **Objective.** Setting of the research objectives and hypothesis are defined at this stage. These research objectives and the hypothesis are revised at each loop of the spiral. At an early stage the hypothesis normally becomes clearer and less changeable. Only the objectives evolve throughout the research.
2. **Risk Assessment and Reduction.** For each identified objective a risk analysis is carried out. The time allocated to each objective was considered to be a major risk in this project.
3. **Development and Validation.** The design and development of the prototype is based on a set of requirements and specifications to achieve the objectives.
4. **Planning.** The project is reviewed to determine whether to proceed with a further loop of the spiral.

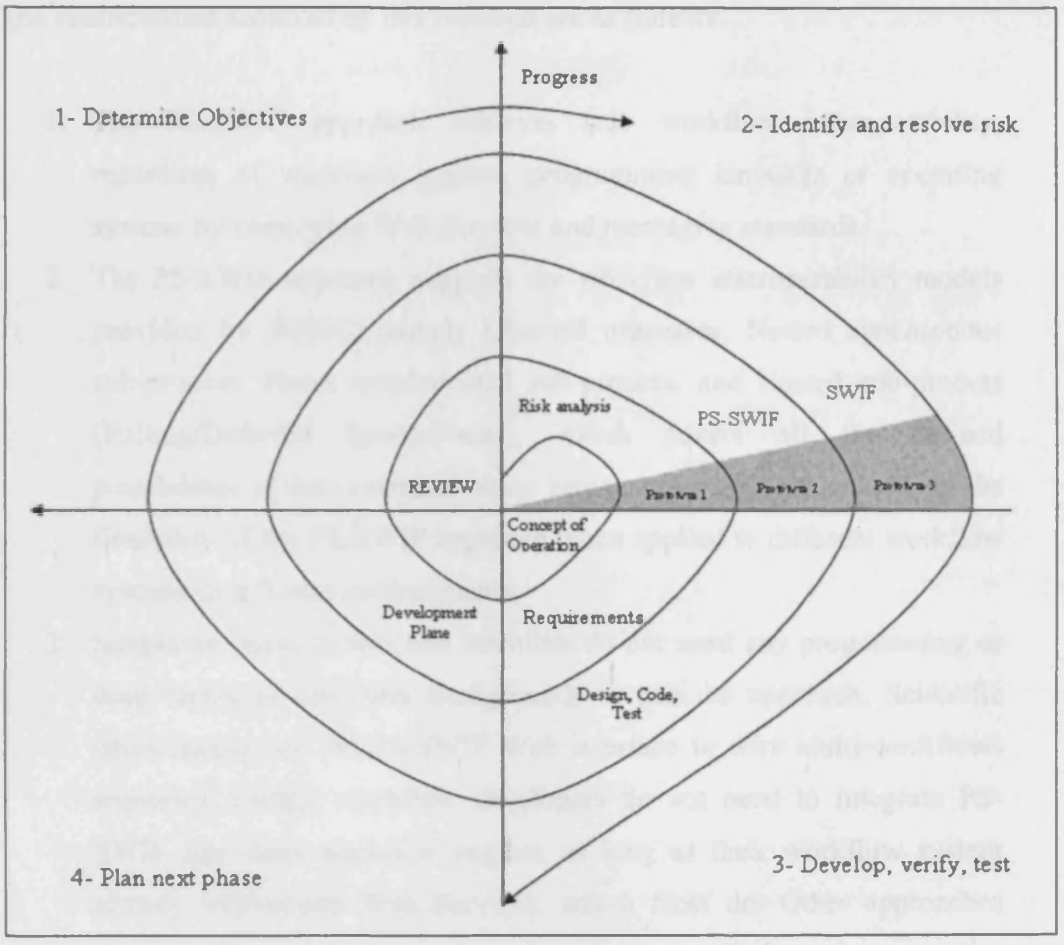


Figure 1.2: Boehm's Spiral Model

1.8 Contributions and Achievements

The novel aspect of this thesis is the research proposal and subsequent implementation of workflow interoperability tools, called PS-SWIF, to provide interoperability among different scientific workflow systems. The PS-SWIF prototype is designed and implemented in the context of a publish/subscribe model as Web Services based on WS-Eventing specifications [32]. PS-SWIF achieves true workflow interoperability, regardless of workflow systems, written in any language and running on different operating systems, by employing Web Services and asynchronous standards. PS-SWIF is validated using performance and scalability measurements that show PS-SWIF is scalable and reliable when used with a variety of workflow systems in various numbers.

The contributions achieved by this research are as follows:

1. The PS-SWIF approach achieves true workflow interoperability, regardless of workflow system, programming language or operating system, by employing Web Services and messaging standards.
2. The PS-SWIF approach supports the workflow interoperability models provided by WfMC, namely Chained processes, Nested synchronous sub-process, Event synchronized sub-process, and Nested sub-process (Polling/Deferred Synchronous), which covers all the defined possibilities of data communication between two systems and proves the flexibility of the PS-SWIF approach when applied to different workflow systems in different environments.
3. Simplicity: Easy to use, and scientists do not need any programming or deep technical computer backgrounds to use the approach. Scientific users simply use the PS-SWIF Web interface to wire multi-workflows scenarios. Further workflow developers do not need to integrate PS-SWIF into their workflow engines as long as their workflow system already implements Web Services, which most do. Other approaches discussed in this thesis require an expert to install the software and to first set up the environments to enable their use by scientists.
4. Deployment Platform: PS-SWIF can provide run-time interoperability among different workflow systems in both local and remote environments, providing flexibility on the distributed execution of multi-workflow systems.
5. Reusability: Experiments can be repeated with the same topic and subscription but use different data. Since the wiring is applied at a topic level, the multi-workflow PS-SWIF experiment can be saved for later use and ran multiple times using different data sets.

Achievements on this research written by the author include:

1. ***Interoperability between Scientific Workflows***

A.Alqaoud, I. Taylor, A. Jones, "Interoperability between Scientific Workflows", UK e-Science Programme All Hands Meeting 2008 (AHM2008), Edinburgh, UK.

In this paper, a generic approach was proposed for achieving interoperability among workflow systems, based on asynchronous notification. The basic concept of PS-SWIF was presented using a direct interaction between Triana [27] and Taverna [33] workflow systems.

2. ***Workflow Interoperability among Different Scientific Workflow Systems, focusing on Triana, Taverna and Kepler***
   (2008 not published).

A full implementation of WS-Eventing was integrated with the Triana workflow system that allows Triana to easily send and receive notification messages. A mechanism was developed using a WSPeer Framework to allow Taverna and Kepler [34] to act as Web Services to send and receive notification messages. The current design of Taverna and Kepler workflows do not have the capability of deploying a workflow as a Web Service. Appendix *A* provides more detail on this approach. (The novelty of this work was not published as the author found an improved method to provide a generic approach.)

3. ***Publish/Subscribe Scientific Workflow Interoperability Framework***
   A.Alqaoud, I. Taylor, A. Jones, "Publish/Subscribe Scientific Workflow Interoperability Framework", Workflows in Support of Large-Scale Science (WORK 09), Portland Oregon, USA: ACM, 2009

In this paper a comprehensive approach to achieving interoperability among workflow systems, based on a WS-Eventing standard, is proposed. An API was developed that can be installed and easily configured by any workflow system that supports the invocation of Web Services. Different models of interoperability provided by WfMC are supported. More about the PS-SWIF API is in Appendix *B*.

4. ***Scientific Workflow Interoperability Framework***

A. Alqaoud, I. Taylor, A. Jones, "Scientific Workflow Interoperability Framework", International Journal of Business Process Integration and Management, Special Issue on Scientific Workflows 2010.

In this journal paper a comprehensive approach to achieving interoperability among workflow systems, based on a WS-Eventing standard, is presented. In addition, a new framework through Web interfaces was designed and developed to provide Workflow Interoperability tools and functions. The new application provides run-time interoperability among different workflow systems in remote environments. The reusability of experiments is supported by the system and a qualitative analysis was performed.

1.9 Organization of Thesis

Chapter 1 ~ Introduction presents the background to the research undertaken, the hypothesis to be tested and highlights the aims and objectives of the research, the research methodology and the contributions of the thesis.

Chapter 2 ~ Background surveys the background of research related to workflow systems and the scientific workflow systems presented in the thesis. The levels and models of workflow system interoperability provided by the Workflow Management Coalition (WfMC) are discussed in some detail in this Chapter.

Chapter 3 ~ Messaging Systems this Chapter provides an overview of publish/subscribe models in general and focuses on the WS-based Publish/Subscribe systems in distributed applications.

Chapter 4 ~ Scientific Workflow Systems discusses different workflow systems, such as Triana, Taverna and Kepler, in more detail. A brief summary of other scientific workflow systems that support Web Services is also presented. Workflow interoperability projects in this area are presented with strengths and weaknesses highlighted.

Chapter 5 ~ PS-SWIF Architecture, Requirements and Design describes the core architecture design for the system. This Chapter also provides the main concept proposed by the thesis: to employ the use of asynchronous notification systems to achieve interoperability between different workflow systems. The notification system to be used is identified and the level of interoperability which can be achieved by adapting the proposed system is considered. The workflow interoperability model is presented and the main components of the model are discussed.

Chapter 6 ~ PS-SWIF Implementation covers these issues for the proposed system. It presents the implementation of the WS-Eventing specification and discusses how this can be integrated with the WSPeer framework and how the system can be applied to different workflow systems thereby conducting a qualitative study of this work. The PS-SWIF Web application with tools and function for workflow interoperability is presented in this Chapter.

Chapter 7 ~ Case Study in this case study we use Triana, Taverna, and Kepler workflow systems to shows how interoperability can be achieved among different workflow systems.

Chapter 8 ~ Evaluation presents a quantitative and further qualitative evaluation of the PS-SWIF system. Several experiments are conducted to determine scalability of the system when applied with different numbers of machines and a large volume of data; another experiment is conducted which proves the workflow interoperability models and also proves the flexibility of the SWIF system.

Chapter 9 ~ Conclusion and future work concludes the thesis with summary comparison between the PS-SWIF approach and other approaches in the same area. Extensions possible to the PS-SWIF system and future research directions are discussed.

CHAPTER 2

Background

2.1 Overview

This Chapter presents the background of research related to scientific workflow systems. In addition a brief summary of Workflow Management System classification is presented. The levels, models and standards of workflow system interoperability provided by the Workflow Management Coalition (WfMC) are discussed in some detail in this Chapter.

2.1.1 Scientific Workflow Definition

Scientific workflow is a new evolution of workflow that emerged for scientists to formalize and structure complex e-Science applications, such as gravitational-wave physics, geophysics, bioinformatics, astronomy, climate modelling, structural biology and chemistry. A scientific workflow management system is a system that supports the modelling, composition, execution, failure recovery, and data provenance of a scientific workflow using the workflow logic to execute tasks within scientific workflows.

2.1.2 Workflow Classifications

There are several workflow classifications proposed in the literature. The workflow classification defined by Deelman and Yu [35, 36] focuses on the scientist's and developer's view to describe scientific applications within the Workflow Management System. The workflow classification presented in this

thesis is based on these publications. Workflow classification is divided as two categories: Workflow design and definition; and Workflow mapping and execution.

2.1.3 Workflow Design and Definition

Workflow design involves three key factors, namely: workflow structure, workflow model and workflow composition system.

2.1.3.1 Workflow Structure

In general, a workflow representation can be achieved in a number of formats and the most common ones are the Directed Acyclic Graph (DAG) [37] or non DAG.

The workflow representation structure based on DAG is classified to sequence, parallelism, and choice representations. In the Sequence representation, tasks are defined as an ordered series. The next tasks will not start until the first one is fully completed. In the Parallelism representation, multiple tasks can be performed concurrently. In choice representation a task can only be performed if its related conditions are true. Condor [38], DAGMan [39], Pegasus [40], and Taverna are examples of the workflow systems that use directed Acyclic Graph (DAG) as their representation format.

A non-DAG workflow supports all the representation structure provided by a DAG-based workflow. In addition, an iteration representation known as 'loop or cycle' is added. The iteration representation structure allows tasks to be repeated several times based on associated conditions. Triana, ICENI (Imperial College e-Science Network Infrastructure) [41], and Unicore (Uniform Interface to Computing Resources) [42] are example of the workflow system that use non DAG as the representation format.

2.1.3.2 Workflow Model

Workflow Model defines a workflow in terms of task definition and structure definition. In general, two types of workflow models are defined, namely abstract and concrete (executable) models. The abstract workflow model describes workflow activities without referring to particular Grid resources for task execution. Describing a workflow in the abstract model provides users a flexible way to map their workflow onto suitable resources within different workflow systems. This level of abstraction provides an easy way for sharing workflow descriptions between VO (virtual organisation) participants [40].

In the concrete workflow model workflow tasks are mapped to specific Grid resources. The concrete workflow may include data movement to stage data in and out of the computations [36].

2.1.3.3 Workflow composition system

Workflow composition systems allow workflow users to specify the steps and dependencies to build components into workflows. They provide a high level representation for construction of Grid resource and hide the complexity of the Grid environment. The composition of workflow is categorised into two broad categories: User-directed and automatic compositions. In the user directed method users directly indicate computational steps and the data that flows through them. In the automatic composition method workflows are automatically generated for users.

Within the User-directed model, users either use language-based modelling or graph-based modelling to compose workflows. In language-based modelling, many workflow systems can be represented using a specific language, such as GridAnt [43], BPEL4WS [44], Gridbus Workflow [45], and Condor DAGMan [46], which have a specific syntax that can be written by hand using a text editor. However, whereas Language-based modelling works well with highly skilled users for some systems, it is extremely difficult for users to express a complex and large workflow by scripting workflow components by hand.

In Graph-based modelling users use a graphical tool for composing workflows. Most e-Science workflow systems are designed to support Graph-based modelling such as Triana, Kepler, and Vistrails [47]. In those systems, users can easily compose a workflow by dragging and dropping the components of interest. The most popular approaches to represent the Graph-based modelling are Petri Nets [48] and UML (Unified Modeling Language) [49] .

Petri Nets are a specialized class of directed graphs that can represent sequence, parallel, loops and condition models for the execution of tasks [50]. They have been adapted by different workflow systems, such as Grid Workflow Description Language (GWorkflowDL) [51], and FlowManager [52].

The Unified Modeling Language (UML) is a standardized language which has emerged for the modelling language, particularly for object-oriented software. The activity diagram provided by UML is used to represent the dependencies between different tasks within workflow applications. Askalon [24] is an example of a workflow system that uses UML activity diagrams.

2.1.4 Workflow Mapping and Execution

Workflow mapping refers to the process that maps an abstract workflow to appropriate resources. When mapping workflow tasks onto suitable resources, information about the resources has to be obtained from appropriate sources [53]. Mapping workflow onto resources requires certain issues to be considered, namely information retrieval, Workflow QoS Constraints, workflow scheduling, fault tolerance and data provenance.

2.1.4.1 Information Retrieval

The information retrieved about computational resources can be static, dynamic or historical. The static information is information that can not change with time, such as operating system and number of processors. The dynamic information is related to status of the Grid resource such as availability of the resource and

CPU usage. Historical information refers to previous events that have occurred, such as performance history [36].

2.1.4.2 Workflow QoS Constraints

There are a large number of similar resources and so mapping workflow to appropriate resource requires finding out, not only the functionality of resources, but also how the resources respond. Quality of services (QoS) requirements are important issues that must be considered when mapping workflow onto resources. According to Cardoso [54], QoS requirements for workflow and Web Services include five dimensions: time, cost, fidelity, reliability and security. Time represents the total time taken to finish the execution of a workflow. Cost refers to the charge associated with Grid resources for execution of workflow tasks. Fidelity represents the quality of the results of workflow execution. Reliability refers to the measurement related to the number of failures for execution of workflows. Security relates to trust worthiness of resources and confidentiality of the execution of workflows.

2.1.4.3 Workflow Scheduling

Scheduling tasks within a workflow is related to managing the execution of dependent tasks on shared resources that are not directly under one's control [36]. The workflow schedule needs to be defined according to scheduling decisions applied to all tasks in the workflow or tasks to be scheduled by multiple schedulers. The best decision is based on numbers of tasks managed by schedulers, whereas one central scheduler is more scalable with a limited number of workflow tasks and multiple schedulers are more suitable for large numbers of tasks. More about Workflow scheduling can be found in [36].

2.1.4.4 Fault Tolerance

Failure within workflow execution can be caused by several factors: non-availability of resources, running out of memory and faults with network connections. Workflow management systems should be able to identify and handle failures and support recovery techniques for execution. Fault tolerance

can be defined by the types of workflows being executed, namely job level and service level. At the job level, there are a number of mechanisms, such as level check-pointing which can be applied at the operating system level for saving the state of an execution and resuming after a failure. At service-based level, fault tolerance might involve re-executing the same service on the same resource or trying to execute the same service on another equivalent resource that may be running elsewhere [35].

2.1.4.5 Data Provenance

Data Provenance refers to using the history of the creation of data objects to get the original or source data to initialize the workflow. Provenance is defined by Moreau [55] as *The provenance of a piece of data is the process that led to that piece of data.* The broadness of this definition does not only include the data derivation, libraries, hardware and run-time information but also considers the provenance of services and workflow [55]. There are various ways to record the history of the creation data. Some workflow systems, such as Triana, adopt an internal structure to handle provenance data while other workflow systems, such as The Karma [56], apply external services to manage provenance data [35].

2.2 Workflow Interoperability Standards

Workflow interoperability has received much interest from the distributed computing community, as can be seen from a number of current workshops [7-10]. Different workflow interoperability specifications have emerged and the most important ones are discussed here.

2.2.1 Workflow Standard-Interoperability Abstract Specification

In 1996 WfMC defined the Workflow Interoperability Abstract Specification, which presents an abstract specification defining the functionality required to support interoperability between two or more workflow engines [4].

Different strategies can be used to achieve workflow interoperability:

1. Direct Interaction: workflow systems use a common API to allow direct interaction.
2. Message Passing: workflow systems exchange information by sending packets of data messages through a communication network.
3. Bridging Strategy: workflow systems apply a bridging mechanism using a gateway technique to move data and tasks between workflow systems via protocol converters.
4. Shared Data Store: the transfer of data and tasks between workflow products is achieved through a shared database.

The Workflow Management Coalition classifies workflow interoperability into four models:

1. Chained Processes: Assumes a process instance enacted on workflow Engine *A* initiates a sub-process instance on workflow Engine *B*. Engine *A* may proceed with its own process instance or be terminated. Engine *A* shows no interest in any result from Engine *B*.
2. Nested Synchronous Sub-process: Assumes a process instance enacted on Engine *A* triggers a sub-process on Engine *B* as in the Chained Processes model. In addition Engine *A* in this model waits until it gets the result back from Engine *B*. Engine *B* carries on with its own process instance until completed and then forwards the result back to Engine *A* .
3. Event Synchronized Sub-process: Assumes the sub-process on Engine *B*, initiated by Engine *A*, may fire a trigger to activate an instance process in workflow Engine *A* due to a sub-process having aborted or as part of a defined check-point logic between two process instances enacted on separate workflow engines [4].
4. Nested Sub-process (Polling/Deferred Synchronous): Assumes workflow Engine *A* initiates a sub-process instance on Engine *B*. Engine *A* carries on with its own process instance until it reaches a stage which needs feed-back from Engine *B*. At this stage, it polls the enacting workflow engine to find if the sub-process has finished. If the sub-process instance

has finished before the invoking process is ready to deal with the event, the termination of the workflow on Engine *B* is queued until the event is required by Engine *A*. If the process instance in the first workflow engine requests the outcome of the enacted sub-process before it has finished, the request is queued by Engine *B* until the sub-process is completed.

The Workflow Management Coalition classifies workflow interoperability to eight levels:

1. No Interoperability Level: No communication between workflow products and interoperability cannot be applied at this level.
2. Coexistence Level: No standard approach to interoperability between workflow products at this level. Workflow products share the same run time environment such as operating system and network. There is no direct interaction between different workflow products. Interoperability can be achieved at this level when an application implements different parts of a complete process, using different workflow products.
3. Unique Gateways Level: Workflow products at this level use a bridging mechanism to work together to perform routing operations among engines. One possibility is to use a common Gateway API among workflow products.
4. Limited Common API Subset Levels: Workflow products can interoperate directly using a common standard API. A multiple API may be needed for a given workflow product to interoperate with different workflow products.
5. Complete Workflow API Level: A single standard API is shared by all workflow products to allow access to the entire range of potential functions.
6. Shared Definition Formats Level: This level requires a shared format for process definitions implemented by workflow products. Each process supported on the workflow system must have a single definition by an organization and guarantee the behaviour of the process regardless of the workflow system used.

7. Protocol Compatibility Level: This level requires that all API client and server communication must be standardized.
8. Common Look and Feel Utilities Level: In addition to the earlier levels, this level requires that all workflow products maintain the same, or at least a similar, interface. This level may not be achieved for commercial and practical reasons [4].

Since the introduction of Workflow Standard-Interoperability Abstract Specification in 1996, a number of workflow interoperability proposals and standards have emerged, which are discussed in the following sections.

2.2.2 Standard-Interoperability Internet e-mail MIME Binding

This standard maps to the Workflow Management Coalition Workflow Standard-Interoperability Abstract Specification. It provides a concrete definition for a message that transfers between two workflow engines to achieve interoperability as defined in WfMC. Abstract messages defined in WfMC are mapped to a simple text interface that uses Internet e-mail with MIME (Multipurpose Internet Mail Extension) encoding as the transport method [5]. Two models of interoperability defined by this standard are supported, namely chained processes and nested sub-process models.

2.2.3 Workflow Standard-Interoperability Wf- XML Binding

The goal of this standard is to produce a specification based on an XML language to model the data transfer requirements set in the Workflow Management Coalition Interoperability Abstract specification [6]. This language is the implementation language used as a basis for the functionality provided in the Interoperability Abstract. Three models of interoperability defined by WfMC are supported by Wf-XML, namely chained workflows, nested workflows and parallel synchronized workflows.

2.2.4 ASAP/Wf-XML 2.0

ASAP/Wf-XML [57] is an updated version of the Wf-XML that aims to link workflow engines together to achieve interoperability. It is based on the Asynchronous Service Access Protocol (ASAP) that integrates asynchronous services across the Internet and transfers structured information encoded in XML using the Simple Object Access Protocol (SOAP) [58]. Wf-XML extends the ASAP protocol and adds some additional capabilities between business process management systems [59]. Using the ASAP/Wf-XML standard, one can achieve interoperability at protocol compatibility level.

2.2.5 XML Process Definition Language

XML Process Definition Language (XPDL) [60] is the language proposed by the Workflow Management Coalition to interchange process models between different workflow systems. XPDL is implemented in most traditional workflow systems. XPDL language is classified as a graph-structured language. Defining processes in a workflow using XPDL language can achieve interoperability at shared definition format level regardless of the workflow system used.

2.2.6 Workshop on Scientific and Scholarly Workflow

In October 2007, a workshop focusing on scientific workflow and improving interoperability took place in Baltimore [8]. Different organizations and committees participated in this workshop, and a technical report, with recommendations, discusses workflow interoperability levels and provides different opportunities to achieve workflow interoperability. These levels include workflow design, workflow mapping and execution, and workflow and data provenance.

In this model, designing a workflow involves first creating a description of the workflow at an abstract level. Abstract Workflow describes a selection of application components and defines their dependencies. The application components could be tasks, jobs, services or any executable units. Dependencies between these components define the order in which components can be

executed [8]. It recommends the use of a common high level specification to describe what the workflow does to achieve the interoperability at workflow design level, regardless of the workflow language used. Using the common high level specification can lead to several advantages:

1. If a workflow system no longer exists, it is possible to re-render workflows that used the old workflow system to a different language.
2. Using such specification enhances the ability of existing workflows to be published, discovered and be more understandable.
3. The specification could be used as a standard metadata language or annotation language for describing workflows.

Workflow execution refers to turning the components application in the abstract workflow into an executable state. Workflow execution interoperability is essential when an instance in a workflow system needs to invoke an instance in another system.

In general, the data provenance refers to the ability to obtain the history of data products. In scientific workflow systems this not only includes reproducing the data, but also includes troubleshooting and optimizing efficiency. For example, when data product *X* generated from a workflow system *A* is then used by another workflow system *B* to generate new data *Y*. The significant advantages of workflow interoperability and data provenance here is when provenance record *Y* is used to trace back to original data *X*

2.2.7 Open Grid Forum (OGF)

Workflow interoperability has recently received much interest from the distributed-computing community, for example, in the Open Grid Forum (OGF) [7, 9, 10]. In the OGF [9], three levels for interoperability are identified: (1) workflow embedding (allowing workflows to run within their own environment, but invoked from another); (2) the development of a meta-language (allowing different proprietary languages to be mapped to a single standard one); and (3)

semantic annotation /description/ classification, (particularly important for sharing information).

2.3 Summary

In this Chapter, the author has presented a classification for workflow system. Workflow classification is divided into two categories: Workflow Design and Definition; and Workflow mapping and execution.

Workflow Design and Definition involves workflow structure, workflow model/specification and workflow composition systems. A workflow structure can be presented as directed Acyclic Graph or non DAG. Workflow Model is defined in terms of abstract workflow or concrete workflow. Workflow composition systems allow workflow users to specify the steps and dependencies to build components into workflows.

Workflow mapping and execution refers to the process that maps an abstract workflow to appropriate resources. Several steps are required to execute a workflow and these involve information retrieval, Workflow QoS Constraints, and workflow scheduling, fault tolerance and data provenance.

In this Chapter, workflow interoperability standards presented by the Workflow Management Coalition (WfMC) and numerous workshops and research group focusing on workflow interoperability are discussed in some detail.

CHAPTER 3

Messaging Systems

In this Chapter, an overview of the publish/subscribe approach is provided. The main WS-based publish/subscribe models are discussed in more detail, including the Java Message Service, CORBA Event and Notification service specifications, WS-Notification and WS-Eventing.

3.1 Publish/Subscribe Approach

The Publish/Subscribe paradigm has emerged to provide a loosely coupled communication pattern in large scale distributed computing. It is an asynchronous messaging system that provides more scalability and flexibility for distributed applications. Communication using an asynchronous messaging paradigm is characterized by a decoupling of participants in both time and space [12]. In the Publish/Subscribe paradigm, an application can send event notifications to other applications about its executing status, monitoring and complete results. Such events are called Event Notifications or Notification Messages. A notification consumer registers interest in a specific event as a subscriber, and a notification producer sends notification messages to one or more notification consumers, depending on the previous registration.

The emergence of Web Service technologies, such as Web Service Description Language (WSDL) [61] and SOAP, allow distributed heterogeneous applications written in several languages and running on different operating

system to be integrated.

WS-based notification messaging systems are becoming widely supported by Web Service based applications and especially by Grid projects. WS-based notification messaging systems provide a combination of the characteristics of both the event notification model and Web Service technologies. In WS-based notification messaging systems, notification messages are in XML format and interaction operations, such as subscribe, publish and message transfer, are performed using Web Service technologies. Most present event notification models depend on a particular vendor implementation. In contrast, WS-based notification messaging systems provide interoperability even when implemented by different vendors. This is of significant benefit as it is difficult to force numerous vendors to use a specific, or similar, notification messaging system in their applications. WS-Notification specification [62] and WS-Eventing specification are the most common examples of WS-based notification messaging systems. WS-based notification messaging systems are used by different scientific workflow systems, for example, WS-Notification is implemented by the Triana workflow system [63] .

3.2 The Java Message Service

The Java Message Service (JMS) [64] is an API messaging specification that allows an application program based on J2EE enterprise applications to create, send, receive and read messages. The JMS provides reliable, loosely coupled, asynchronous interactions between enterprise applications in a distributed computing environment. The JMS API defines two messaging models; (1) point-to-point or queuing model; senders posts messages to a specific receiver through a message queue. The point-to-point messaging model is equivalent to the e-mail messaging system. Senders post messages to a particular mailbox (queue), and the owner of the mailbox (queue) receives the messages in the same order they were sent. In the point-to-point messaging model, only one receiver gets the message. (2) The publish/subscribe messaging model allows a producer (also called a publisher) to publish messages to single or more consumers (called

subscribers) through a particular topic. A topic is a destination where publishers can publish messages, and subscribers can consume them. Publishers and subscribers are independent and they do not have to know anything about each other. Multiple subscribers can receive a message produced by one publisher. The disadvantage of using the JMS messaging system is that it just applies to applications using Java platforms.

3.3 CORBA Event Service and Notification Service Specification

The Common Object Requesting Broker Architecture (CORBA) [65] is a number of specifications for middleware defined by the Object Management Group (OMG) that allow a large range of application programs written in different languages and running on different operating systems to integrate together through an Object Request Broker (ORB). Event services specification [66] and Notification services specification [67] have been defined to support communication between CORBA objects. Both specifications are based on the publish/subscribe model.

In the Event services specification, two main objects are defined: the Event supplier and the Event consumer. Event suppliers generate event data and event consumers receive event data. Event consumers issue a standard CORBA request to an Event supplier. A Push model and a Pull model are two approaches defined to establish communication between event suppliers and event consumers. In the push model, an event supplier initiates the delivery of the data to event consumers. In the pull model, the event consumer initiates the request for the data from an event supplier. The communication between the event suppliers and event consumers occur through event channels by issuing standard CORBA requests. An event channel is a standard CORBA object that enables multiple event suppliers and numerous event consumers to communicate with each other asynchronously. However, the CORBA Event Service Specification comes with two main limitations. It does not address event filtering capability and does not support configuration for Quality of Service (QoS) [67].

The CORBA Notification service specification is considered to be an extension to the CORBA Event service specification. The primary goal for the CORBA Notification service specification is to overcome the limitations of the CORBA Event Service Specification. It supports filtering type and capability of configuring various QoS properties. This specification also presents a new type of message "Structured Events" which defines a well-known data structure to optimize event filtering. However, CORBA has been implemented on different platforms and different language, but it depends on a single vendor implementation. CORBA does not provide interoperability when it is implemented by different vendors, especially when one considers extending it into higher-level services, such as security or transaction management [68].

3.4 Open Grid Services Infrastructure (OGSI) specification

Open Grid Services Infrastructure (OGSI) specification [69] was released by Global Grid Forum (GGF). It defines approaches for creating, representing and managing information among grid services. A Grid Service is a Web Service with additional properties such as stateful resources and lifetime management. A Notification messages system is an important part of OGSI specification. The notification message in OGSI is also implemented using a publish/subscribe model. In OGSI a notification producer or publisher is called a notification source and a notification consumer is called a notification sink. The notification source is a Grid services instance that implements *NotificationSource portType* to send notification messages to multiple notification sinks. The notification sink is a Grid service instance that implements *NotificationSink portType* to receive notification messages from notification sources. The subscriber is also a Grid service instance that is implemented separately from Notification Sink. The subscriber sends a subscription expression request to notification source. The subscription expression request includes XML elements that describe when and where Notification messages should be sent to any Notification Sink. Designing of Notification roles; Notification Source, Notification Sink and Subscriber in the OGSI as Grid Service, allow them to be managed like any other Grid

Service. OGSI Notification was the earliest attempt toward a WS-based notification messaging system. The Web Services Resource Framework (WS-RF) [70] standard was produced to re-factor and evolve the OGSI. WS-RF divides the OGSI functionality to different specifications and the OGSI Notification was replaced by a new specification called WS-Notification.

3.5 WS-Notification

WS-Notification includes a number of specifications produced in January 2004 by IBM and Globus Alliance. It is based on a publish/subscribe model. In March 2004 the WS-Notification specification was re-factored into three separate specifications: WS-BaseNotification [71], WS-BrokeredNotification [72], and WS-Topics [73]. WS-BaseNotification specification provides the essential functionality and fundamental interactions between notification producers and notification consumers. WS-BrokeredNotification specification describes intermediary services to decouple Notification Consumers from Notification Producers. WS-Topic specifications define a number of topic expression used in a subscribe request as a subscription expression. WS-Notification has been implemented by various vendors and projects such as Triana Workflow system and OGSA-DAI [74].

The following description describes the key entities in the WS-Notification specification: A Notification Producer is a Web Service entity that supports one or more topics. It maintains a listing of subscription requests and distributes Notification Messages to Notification Consumers. A Publisher is an entity (may be a Web Service) that generates Notification messages. The publisher Web Service is different from the Notification Producer Web Service. A Notification Consumer is a Web Service entity that subscribed with a Notification Producer through a Subscriber Web Service in order to receive Notification Messages. A Subscriber is an entity (usually a Web Service) that used to send a subscription request message to Notification Producer. The Subscriber Web Service might be different from Notification Consumers. A Subscription is a resource created when a message request is sent by a subscriber to Notification Producer, which

follow the implied resource pattern WS-Resource and a Subscription Manager is a Web Service entity to manage subscription resources. Although WS-Notification provides a wide variety of publish/subscribe functionality, an application needs to implement different standards to adopt WS-Notification specifications.

3.6 WS-Eventing

The WS-Eventing specification was produced in January 2004 by Microsoft [32]. It defines a Source Web Service responsible for accepting requests to create subscriptions and send notification messages. The WS-Eventing specification defines a Sink Web Service that receives notification messages. A Subscriber Web Service is also used by the WS-Eventing specification to send subscribe requests to Source Web Services, and delegates other requests, such as *renew*, *getStatus* and *unsubscribe*, to the Subscription Manager Web Service. The WS-Eventing specification uses WS-Addressing [75] to address Web Services and messages. WS-Addressing defines two constructs, namely endpoint references and message information headers, to convey addressing information between Web Services. The subscribe request contains a number of properties in the request message that follow the WS-Addressing standard. There are various elements that must be defined in the message header:

1. Subscription action URL must be specified to a value that conforms to WS-Eventing syntax.
2. Hypothetical event source that specifies where message should be sent.
3. Reply element that specifies where response message should be sent.
4. Message ID that uses a unique identifier for the request message.

In the message body of the subscribe request, the following elements should be defined:

1. SubscriptionEnd address: specifies where to send a subscription end notification if the subscription is unexpectedly terminated by the source Web Service.
2. Delivery Mode: specifies how and where to send notification messages.

By default, WS-Eventing uses the push mode as the delivery mode which provides simple asynchronous messaging and provides delivery extension elements in the subscription request to support other types of delivery modes such as the synchronous mode.

3. Subscription Expiry: This expiration time specifies the Sink Web Service is not interested in receiving any notification messages after the time has expired.

When the subscribe request is successfully handled by the Source Web Service, it will generate a response message that states the subscription request has been successful. This response includes the *EndpointReference* of a subscription manager which may be used later by the Subscriber to manage the subscription. Requests by a Subscriber Web Service, such as renew or getStatus must be targeted to the Subscription Manager by including the subscription ID.

WS-Notification standard and WS-Eventing specifications are the main competing standards in this area. WS-Eventing is simpler, used by several software vendors to provide basic functions for the Publish/Subscribe paradigm.

3.7 Summary

In this Chapter an overview of the publish/subscribe approach is presented and discussed, with the most popular publish/subscribe systems, such as JMS, CORBA Event, CORBA Notification, WS-Notification, and WS-Eventing.

JMS defines two models to deliver messages to consumers: Point-to-Point that posts messages to a specific receiver through a message queue and publish/subscribe models that allow a producer to publish messages to single or more consumers through a particular topic.

CORBA Event specification is based on the publish/subscribe approach. Event suppliers generate event data and event consumers receive event data. The specification support push and pull models to establish communication between

event suppliers and event consumers. The CORBA Notification specification is similar to the CORBA Event specification but with more functions and features. It supports filtering type and capability of configuring various QoS properties.

WS-Notification is based on the publish/subscribe model. The main entities, such as Notification Consumers, Notification Producers, and Subscriber are implemented as Web Services. Although, WS-Notification provides more functionality than other systems, it needs to use different specifications, such as WS-ResourceProperties, WS-ResourceLifetime to be implemented.

The WS-Eventing specification uses a similar concept as the WS-Notification specifications where all the entities are implemented in Web Service pattern. WS-Eventing implement the basic functionality of a publish/subscribe model such as subscribe and renew and does not therefore need to implement other specifications, such as WS-Notification and therefore WS-Eventing is used in the PS-SWIF approach.

CHAPTER 4

Scientific Workflow Systems

In this Chapter, scientific workflow systems, such as Triana, Taverna and Kepler, are presented in more detail. A brief summary of other scientific workflow systems that support Web Services are presented followed by an overview of workflow interoperability projects in this area; with strengths and weaknesses highlighted.

4.1 Scientific Workflow Systems

There is wide range of scientific workflow systems today, each one designed to resolve problems at a specific level. In this section the author focuses on Triana, Taverna, and Kepler workflow systems because these systems are good representatives of scientific workflow systems at the service level where the approach in this thesis is applied. Different experiments to construct various workflows using Triana, Taverna, and Kepler workflow systems are conducted for proof of the concept and hypothesis.

4.1.1 Triana Workflow

Triana [27] is an open source problem-solving environment designed at Cardiff University in 1991. Triana combines a number of different libraries of pre-defined analysis tools developed for a wide variety of application scenarios and environments. Triana can be used for many applications, including use as a dataflow system, a distributed workflow system, choreographing Web or WS-RF

services, a workflow management system and an automated scripting tool [76]. Triana has the ability to work in Grid Computing and P2P environments and can dynamically discover and run distributed resources, such as Web Services. An advantageous feature of Triana is its graphical user interface. The current version of Triana has been developed using Java technology and utilises lessons learned from its previous C++ counterpart.

Triana provides a wide variety of built-in tools, which can be used for numerical data, audio data, images and text files. Most of the tools in Triana display a parameter window so users can add, or delete, parameters as required. If an appropriate tool is not available Triana provides a wizard that allows one to create a suitable tool with a parameter window. If other tools are available outside of Triana, Triana allows one to use these directly over the Internet without downloading to one's computer.

Tools in Triana are called units and are found in the Toolbox Window. Units are represented on the Main Triana Window by boxes with input and output nodes and connected by 'cables'. Each unit specifies the data type for the input and output nodes and a cable between two units represents the data type being passed between these units. To connect two units, a user drags a cable from the output node of the first unit to the input node of the second unit. Triana's underlying type-checking system inspects whether the data type sent by the first unit is compatible with the data type of the receiving unit. If so, the cable is connected between these units indicating that the data type matches; if not, an error message is displayed stating the data type between these units is not compatible.

Triana was originally designed to support collaboration between scientists for the GEO600 Gravitational Wave Project [77]. The GEO600 will normally produce numerous terabytes of numerical data every year, and Triana was used to automatically examine and run data analysis algorithms on this data at its source. Although Triana was designed for use by scientists in GE0600, it can be

used in many other ways and some 500 tools exist to cover a large range of applications.

Triana has been used in many different projects and domains, such as Triana Advanced Clinical Support (TRIACS) [78], Gravitational Wave Project (GEO600), Enabling Desktop Grids for e-Science (EDGeS) [79], Workflows Hosted In Portals (WHIP) [80], Distributed Audio Retrieval Using Triana (DART) [81], Biodiversity World [82], Environment for Industrial Design Optimization (DIPSO) [83], The Data Mining Tools and Services for Grid Computing Environments (DataMiningGrid) [84], Grid Enabled Web eNvironment for site Independent User job Submission (GENIUS) [85], Grid-Enabled Medical Simulation Services (GEMSS) [86], and Federated Analysis Environment for Heterogeneous Intelligent Mining (FAEHIM) [87].

In Triana two types of distributed components are used to interact with Grid components via the Grid Application Tool kit (GAT) [88] and with Service components via the Grid Application Prototype (GAP) [89, 90] interface. Figure 4.1 shows the distributed components within Triana.

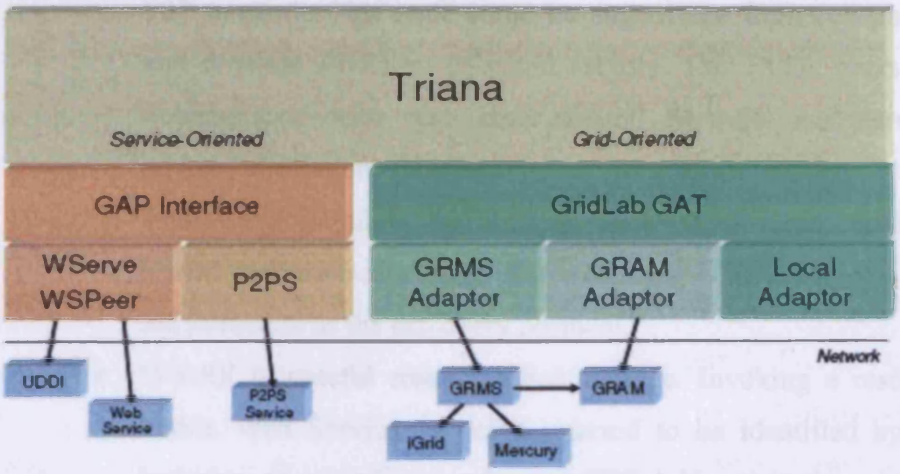


Figure 4.1: Distributed Components within Triana [91]

- GAT was developed through the GridLab project [92], and aims to

insulate Grid workflows from the underlying Grid middleware allowing Triana components to execute applications on the Grid via a resource manager, such as GRAM (Grid Resource Allocation Management) [93] and GRMS (GridLab Resource Management System) [94], and to perform Grid operations, such as file transfer and job submission [89].

- The GAP interface provides generic service functionalities for publishing and discovering services within dynamic service-oriented networks. GAP uses a P2P-based distributed mechanism [95] for communication. This mechanism allows different protocols to be implemented for binding to the GAP Interface as they follow a service-oriented approach. Four different infrastructures bindings are provided by the GAP Interface:
    1. P2PS [96] (Peer-to-Peer Simplified) is a lightweight infrastructure for P2P technology with the ability for advertisement, discovery and pipe-based communication. P2PS was designed to provide a simple platform to develop P2P style applications, to overcome the complexity of other similar designs such as JXTA [97] and JINI [98].
    2. JXTA is a set of open protocols that allows any connected node on the network to be discovered and communicated through a P2P network. This node could be any device from cell phones and wireless PDAs to PCs and servers. The JXTA peers can communicate with each other behind firewalls and network address translations (NATs).
    3. Web Services allow applications to be discovered, invoked, hosted and exposed as Web Services using UDDI [99] or a P2P infrastructure as the discovery protocol.
    4. WS-RF is stateful resource Web Service. Invoking a resource within Web Service, is being referred to be identified by the inclusion of an endpoint reference (WS-Address) in the message header [100]. Within Triana, an endpoint reference is defined as a context, and this context can be applied to multiple Web Service interactions. WS-RF resource is invoked into Triana in the same

way as a standard Web Services.

Triana uses WSPeer to implement the GAP Web Service binding, discussed in detail in Chapter 6.

In Triana, user workflows can be deployed as fully functional Web Services. The workflow in Triana can be either one or more tasks (a group). If a workflow consisting of more than one task is deployed in Triana, then the tasks must first be created in a group before the workflow is deployed. Group tasks in Triana act as an individual task which, receives data from input nodes, processes this data and the result is then passed to their output nodes. A user then selects the Web Service binding as the service host to run the Group task, and the GAP Interface automatically launches the workflow as a Web Service. Once Web Services have been deployed, they replace the equivalent group tasks in the user's workflow to a new Web Service task with a different colour. Data is delivered to, and from, the new Web Services through their input and output pipes.

A key advantage is the capability to advertise Web Services launched as remote Triana services using the Universal Description, Discovery and Integration (UDDI) for the GAP Web Service bindings. Once the Web Service Workflow is advertised, other instances of Triana and non-Triana related applications can discover and invoke these Web Services. This is a powerful feature that allows users to quickly create and launch a wide range of composed Triana algorithms and tools as Web Services.

Whenever Triana is started, the GAP interface automatically searches for existing published Web Services, which, when located, are inserted into the Triana toolbox window with existing local tools. Triana uses a Discover Services option to issue a GAP discover services call. When a remote service is found, it is loaded into the Triana Toolbox Window, and the remote services can be dragged and connected into the Main Triana Window in the same way as

local Triana tools, although Triana uses an input/output GAP pipe to connect the remote services instead of a local cable.

Triana workflow supports a notification message model using the WS-Notification specification. Since WS-Notification is based on Web Services and Triana supports deployment of a workflow as a Web Service, the workflow in Triana can be driven as a WS-Notification component. Each instance of this component can be invoked with the context of a WS-Resource and bind with one of the topics available in this resource. When the value of this topic is changed a notification message is sent to workflow components and the workflow then continues with further processing based on the new result received. This is not straightforward, Triana needs to expose a list of topics for each resource, which is modelled as a resource property (WS-ResouceProperty [98]). Each of these properties is defined in the WSDL description and Triana needs to construct a separate component for each in the Triana tool tree. When the Web Service that involves resources is invoked, users can drag the appropriate property resource from a Triana tool and construct a workflow [63].

In terms of interoperability, the ability of a workflow in Triana to be deployed as a Web Service makes it easy to interoperate Triana workflows at run time with other engines that support the invoking of the Web Service. Interoperability with the Triana Workflow has been addressed by different projects, including: VLE-WFBus [101] and P-GRADE/GEMLCA [102], and later sections give more detail.

4.1.2 Taverna Workflow

The Taverna Workflow system [33] is an open source that aims to provide a workflow language and graphical interface tools to enable scientific users to build, run and edit workflows via distributed computer technology. Taverna has been developed by the myGrid [103] project for designing and executing workflows for e-Science experiments. Taverna allows scientists to build complex analysis workflows from different components, located on local and remote machines, and then execute these workflows and visualize the results.

Taverna has been used in a wide range of different domains, from music to chemistry to biology [104]. Many different software tools, including Web Services, have been integrated into the Taverna Workflow. These include example, services provided by the European Bioinformatics Institute (EMBL-EBI) [105], which is part of the European Molecular Biology Laboratory(EMBL) [106] by the National Center for Biotechnology Information (NCBI) [107], and the DNA Data Bank of Japan (DDBJ) [108].

The design architecture of the Taverna in Figure 4.2 has three major layers, namely The Application Data Flow layer, The Execution Flow layer and The Processor Invocation layer.

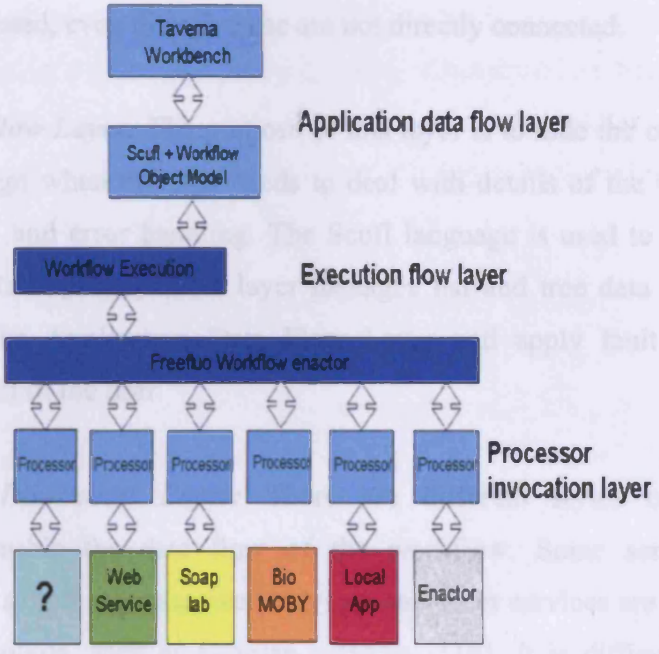


Figure 4.2: Taverna Layers [109]

Application Data Flow Layer: a new language called the Simple conceptual unified flow language (Scufl) is used to construct a workflow as a graph of processors with input and output nodes. The aim is to allow scientists to be

familiar with the service-orientated architecture concepts and hide the complexity of the implementation styles of the services.

The components of a Scufl workflow consist of:

- A set of inputs, which act as entry points for a processor.
- A set of outputs, which act as exit points from a processor.
- A set of processors; each one of which represents a logical service. From the user's perspective a processor is an entity that receives data, processes the data and then produces the data on its output ports.
- A set of data links, which mediate the flow of data between two processors.
- A set of coordination constraints, which apply constraints to control the execution of processors. For example, if two coordination links are applied, one processor will not process its data until the second processor has completed, even though these are not directly connected.

The Execution Flow Layer: The purpose of this layer is to hide the complexity of workflow design when the user needs to deal with details of the workflow, such as iterations and error handling. The Scufl language is used to deal with these requirements implicitly. This layer manages list and tree data structures that pass from the Application Data Flow Layer and apply fault recovery strategies on behalf of the user.

The Processor Invocation Layer: There are different styles of service interactions to enable the data flow of the workflow. Some services are presented using a simple query/answer interface and other services are presented using standard toolkits, such as Soaplab services [110]. It is difficult for the Scufl language to describe the interaction for all these kinds of services. Instead, the Scufl language can be extended to add different types of processor. This is accomplished through a set of processor plug-ins, provided by the Taverna workflow, such as a WSDL Scufl processor, a local Java function processor, a Soaplab processor, a nested workflow processor, and a Styx processor.

A major advantage of using Taverna is the ability to access a wide range of services, over 3,500, accessible to a myGrid user [35]. These services are not in Taverna by default, but can easily be added to the Taverna workbench. However the current state of the Taverna workflow does not support the deployment of a workflow as a Web Service. Moreover the notification messages model is not supported by Taverna.

In terms of interoperability, Taverna has been used with Triana and Kepler through different projects such as P-GRADE/GEMLCA and VLE-WFBus project; more details are given in later sections. In addition, Taverna is the main workflow for the myExperiment social Web site, which is a Virtual Research Environment where scientists can find, and share, workflows. The MyExperiment Taverna Plugin created by WHIP (Workflow Hosted In Portal) allows Taverna users to seamlessly move between their locally installed version of Taverna and the MyExperiment Website [111].

4.1.3 Kepler Workflow

The Kepler Workflow system [34] is an open source that aims to provide analysis and modelling of scientific data for research scientists in different domains. Kepler is supported by the National Science Foundation Kepler/ CORE team [112], which includes a number of institutions that created the Kepler project: UC Davis, UC Santa Barbara and UC San Diego.

The Kepler Workflow has been used in different scientific projects; in Chemistry with the RESearch sURGe ENabled by CyberinfrastructurE (RESURGENCE) project [113], in Ecology with the Science Environment for Ecological Knowledge (SEEK) project [114], in Geology with the Geosciences Network (GEON) Project [115], in Molecular Biology with the Scientific Process Automation (SPA) project [116] , in Oceanography with the Real-time Observatories, Applications, and Data Management Network (ROADNet) project [117] and in Phylogeny with the Cyberinfrastructure for Phylogenetic Research (CIPRES) project [118].

Kepler provides a graphical interface for scientists with no experience in computer science to create and execute the scientific data, such as streaming sensor data, medical and satellite images, simulation output and observational data using a visual representation.

The Kepler design is based on Ptolemy II framework [119] developed by the University of California, Berkeley. Ptolemy II is a Java-based component assembly framework with a visual interface called Vergil. The Ptolemy II framework supports modelling, designing and simulation of real-time embedded systems. Ptolemy provides a new feature called directors to control the execution model of a workflow. Workflow components are represented using reusable units called actors that represent the scientific data. An actor can take several input and output ports.

The Kepler workflow adapts, and extends, features provided by the Ptolemy II framework, adding more advanced features 'through specific actors' needed by scientific workflows, such as access to remote data, Database Access and Querying, Distributed Execution (Web and Grid-Services) and provenance tracking.

Like Triana and Taverna version 2.0 [104], a user can easily construct, or prototype, a workflow in Kepler by dragging and dropping the components onto a Workflow canvas. Prototyping a workflow requires the identification of a number of steps, from reading data to transforming and processing it, and visualizing or saving the output results in a specific format. The first step is to identify and select a director that controls the execution of the workflow. There are numerous directors provided by the Kepler workflow and a user can select one of these, depending on the types of processes the workflow will perform. For example, there are directors that specify whether the workflow can be executed synchronously in parallel. The user can then select the appropriate actors from the Kepler library and connect them on the workflow canvas to

create a special data workflow. The workflow can then be saved in XML format and executed.

A Web Service is invoked through a Web Service actor, by specifying the URL of a WSDL file. Normally a WSDL file provides a description for methods and data types the service can execute. If the service uses complex types, a *WSWithComplexType* actor must be used instead of the Web Service actor. When a user specifies the URL of a WSDL file in the Web Service *WSWithComplexType* actor, available methods will be automatically populated in a drop-down menu. Once the user selects and commits to a desired method, the necessary input and output ports are automatically created.

A major advantage of using a Kepler workflow is that it provides more tools and functions than provided by other workflow systems that cover different scientific domains. The Kepler library provides a wide range of useful actors that can easily be used to access the powerful statistical and data processing of R and/or MATLAB for image processing functionality [120]. R [121] is free software which is part of the GNU project [122] used for statistical computing, data manipulation and graphics. However the current state of the Kepler workflow does not support the deployment of a workflow as a Web Service. Moreover the notification messages model is not supported by Kepler; instead Email actors are used to send email notifications from a workflow to a specified address.

In terms of interoperability Kepler has been integrated with different systems such as Pegasus, Triana and Taverna in the P-GRADE/GEMLCA and VLE-WFBus projects.

4.2 Other Workflow Projects Supporting Web Services

Here, the author outlines some of the workflow projects that use Web Services standards, using a Web Service to compose a workflow. Any potential collaboration between these systems requires workflow interoperability. The

approach presented in this thesis is based on Web Service standards and can be used with these systems to achieve workflow interoperability.

4.2.1 The GEODISE System

GEODISE [123] is designed to support engineers and provide a seamless access to an intelligent knowledge repository in the Grid environment. GEODISE is integrated to provide Design Optimization Tools for Computational Fluid Dynamics (CFD) for industrial application. The GEODISE project supports a number of toolboxes used to integrate Grid client functionality into problem-solving environments. A Web Service toolbox is provided through the .Net Web Service-enabled interface to the Condor system.

4.2.2 The OMII-BPEL Workflow

OMI-BPEL [124] aims to provide process modelling and process execution for scientific workflows expressed in Business Process Execution Language (BPEL); officially a Web Service standard for business process modelling and execution.

4.2.3 The SODIUM Workflow

SODIUM (Service Oriented Development In a Unified framework) [125], consists of a set of languages and tools, with corresponding middleware, used for modelling and execution of scientific workflows composed of heterogeneous services. The SODIUM 'execute' engine is written in Unified Service Composition Language (USCL) [126]. SODIUM provides a Web Service Plug-in to invoke Web Services by dynamically assembling a SOAP message and sending it to the service provider [127].

4.2.4 The VisTrails Workflow

VisTrails [128] is a scientific workflow and provenance management system developed at the University of Utah, and provides a useful tool to manage data and the data exploration process [129]. The VisTrails workflow is written in

Python. A Web Service is supported in the VisTrails through a Web Service Package.

4.2.5 The Discovery Net System

Discovery Net [130] is middleware that allows scientists to create and manage complex data analysis on Grid technologies, such as Web Services, without needing much background on computer science. Discovery Net provides a different range of bio-informatic tools to compose a workflow using grid services. The Web Service is managed through a component service layer provided by Discovery Net.

4.2.6 MOTEUR

MOTEUR (hoMe-made OpTimisEd scUfl enactoR) [25] is a service-based workflow which provides maximal flexibility to make the description and enactment of data-intensive easier from the application point of view. MOTEUR provides enactment of a workflow of Web- Services described with the Scufl language used by Taverna workbench [131] .

4.2.7 The Workflow Enactment Engine

The Workflow Enactment Engine (WFEE) [132] developed at the University of Melbourne, Australia, uses a just in-time scheduling system and decentralized event-driven scheduling architecture which provides a more flexible and loosely-coupled control . The WFEE defines a XML-based workflow language (xWFL) to allow describing tasks and dependencies. WFEE uses a dispatcher to interact with Web Service resources.

4.3 Related Work

To date, only limited or *ad-hoc* solutions have been attempted to achieve scientific workflow interoperability between e-Science workflow systems. In this section the author presents the most popular approaches that have tried to achieve interoperability within Scientific Workflow Systems.

4.3.1 P-GRADE/GEMLCA

This approach is based on integration of GEMLCA [133] with P-GRADE [26] to achieve workflow interoperability [102]. GEMLCA (Grid Execution Management for Legacy Code Architecture) is produced by Westminster University as an application repository for deploying legacy code applications as Grid services. The P-GRADE Portal (Parallel Grid Run-time and Application Development Environment) is a workflow-oriented grid portal which supports applications at all stages of grid workflow development and manages how Web-based Grid portals can be executed on top of Globus middleware [134]. GEMLCA is integrated with the P-GRADE Grid portal [135] to introduce a user-friendly Web interface that deploys legacy code as Grid services, and to build, execute and visualize the execution of complex Grid workflows built from legacy code and Grid services. To achieve workflow interoperability in the GEMLCA/P-GRADE approach, it is required that a *home* workflow must first integrate with GEMLCA. The home workflow can be any workflow, such as Triana, Taverna or Kepler, while other workflow systems can execute as nodes. Because the P-GRADE was integrated with GEMLCA, the P-GRADE is used as the default home workflow. Three workflow systems, Triana, Taverna and Kepler, have been installed in a cluster at Westminster University. The P-GRADE portal is used to enable scientists to select one of these workflow systems, upload the input parameters and input files and then execute using the GEMLCA. Although this approach introduces a type of workflow interoperability, it is a requirement to integrate one's own workflow system with GEMLCA if one needs to use a different workflow as the home workflow because the default home workflow is the P-GRADE one. So the P-GRADE/GEMLCA approach is limited to specific types of workflow systems. Also data is not transferred between workflow systems directly, but is transferred through a stored file.

4.3.2 VLE-WFBus

VLE-WFBus [101] is a workflow bus based solution developed by Dutch Virtual Laboratory for the e-Science (VL-e) project [136] to achieve workflow

interoperability. The VLE-WFBus combines different scientific workflow systems as federated components on one workflow system using a software bus. This workflow is called a workflow bus and any workflow wrapped to a workflow bus is called a sub-workflow. The VLE-WFBus wraps four popular workflow systems, namely Triana, Taverna, Kepler and VLAM-G workflow systems [137] as the sub-workflow. In a workflow bus, the data and control dependencies among sub-workflows is achieved through data objects. The data object can be any type of data entities such as input/output files and a set of parameters. However, the VLE-WFBus approach only provides workflow interoperability for certain scientific workflow systems, and is limited to these workflow systems. It is, moreover, required that the workflow systems should provide flexible engine level API for integration.

Beside the disadvantages mentioned in this section for workflow interoperability approaches, the P-GRADE/GEMLCA and VLE-WFBus approaches need to install an API on a user machine which requires an expert or developer to set up the environment and apply the configuration for the system. These approaches do not support interoperability among workflow systems running remotely, as does the PS-SWIF approach presented in this thesis.

4.3.3 Intermediate Workflow Representation

The Intermediate Workflow Representation (IWR) [138] framework proposal by Oxford University Computing Laboratory is for build time interoperability of workflow definition languages. Since there are varieties of different workflow definition languages (WDLs) emerging from different workflow systems, IWR aims to provide interoperability among workflow system by creating a mediator to carry out language-to-language conversions. IWR [138] classifies the different workflows that exist across different systems to three categories: an atomic workflow that allows users to use an interface to algorithm or tool with input data; a partial workflow that does some of the desired functionality within the workflow itself. Users can incorporate the partial workflow to build the whole workflow. The partial workflow can be used as a 'sub-process' which

allows a calling workflow to invoke it and when it is finished returns a control to the caller; and a composite workflow which combines atomic workflows and/or partial workflows. Users can download a composite workflow and directly use it without any modification.

The IWR framework composes two logical units: Workflow Library Services and Workflow Translation Services. The user is enabled to use their workbench to search for existing workflows and services. The Workflow Library Services represents a repository of the three types of the workflow classifications defined earlier and the user may use them to create a workflow. Users would use the Workflow Translation Services with a specific WDL to transform workflow scripts to the IWR format. The transformation is performed through a syntax analyzer which uses some algorithm to parse the original workflow and produce the IWR format.

The IWR format is going to support a superset of control flow constructs such as sequence, parallelism, conditional routing and structured iteration. In terms of data flow, it is going to support wide ranges of data types such application specific data, and process control data.

The solution proposed by the IWR Framework tried to achieve interoperability at an abstract workflow level, which is undoubtedly useful, but interoperability at run-time or execution level is still needed. Moreover the IWR framework is required to install a plug-in to the user's existing Workflow Management System which requires an expert or developer to set up the environment and apply the configuration for the system.

4.3.4 SIMDAT

SIMDAT [13] Data grids for process and product development using numerical simulation and knowledge discovery, started in 2004. The SIMDAT is funded by the European Commission under the Information Society Technologies Programme (IST). There are numerous publications on the SIMDAT project and

the one most related to this thesis is Design Document for Workflow Interoperability and Management [139, 140]. SIMDAT provides two models to achieve workflow interoperability: Web Service model and Grid Service model.

In the Web Service model three types of workflow interoperability are supported: Deployed Server; Deployed Service and Language Translation.

The deployed server approach involves deploying a workflow engine as a Web Service through its API. Because the workflow engine is represented here as a Web Service, this requires any workflow description or data to be submitted as inputs. Then users use an interface to construct a workflow visually using a workflow language that the server understands to describe the composition of each component. Each component is mapped onto a concrete Web Service. Information of the Web Service is then called from the server side (the workflow engine).

The deployed service approach involves only deploying an actual workflow as a Web Service. This approach requires that all the workflow systems within SIMDAT must have the capability to deploy their workflows as a Web Service.

The disadvantages of both the deploying service and deploying server approach are that it is required to use a special API to deploy the workflow engine or actual workflow as a Web Service; the user must construct a workflow using a language that the workflow engine understands. Moreover it is limited to workflow systems supported by SIMDAT (OPTIMUS, InforSense's KDE, and Freefluo/Taverna) [141]. These approaches present a solution to achieve interoperability at Limited Common API Subset level according to the WfMC classification.

The language translation approach has tried to achieve interoperability when the workflow engine is able to support other workflow languages. This approach has been attempted by InforSense's KDE (Knowledge Discovery Environment)

[142] where XPDL (XML Process Definition Language) and BPEL (Business Process Execution Language for Web Services) languages were used to describe workflows and then executed using the InforSense engine. This approach does not provide interoperability between workflow systems. This approach is a simple workflow language, which mimics the behaviour and uses the syntax of other workflow languages. The proposed approach tries to present a solution for workflow interoperability at Shared Definition Formats level according to the WfMC classification.

In the Grid Service model two types of workflow interoperability are supported: Deployed Server and Deployed Service. This model wraps workflow engines or workflows as a Grid Service using the GRIA (Grid Resources for Industrial Applications) system. Although the Grid Service model GRIA provides support for security, authentication and quality of service, it has similar disadvantages to those of the Web Service model.

4.3.5 Kepler/Pegasus Integration

Pegasus [40] is a workflow that provides mapping and planning for resources in distributed systems at an abstract level. In Pegasus a workflow is described in resource-independent ways and Pegasus finds appropriate resource to execute them. Kepler/Pegasus integration [143] provides an integration of Kepler workflow with Pegasus that aims to allow Kepler users to construct a workflow in resource-independent ways and benefits of the advantage provided by Pegasus. In the other direction Pegasus users can benefit from the Kepler workflow to visualize a workflow composition and monitoring. Composition of a workflow in Pegasus based on Transformation Catalog contains all information about the workflow components and also their resource requirements. The Transformation Catalog is integrated as a director for Kepler. To support Pegasus users, a capability of importing existing DAX workflows into Kepler is developed for modification and visualization of Pegasus workflow. However integrating the Kepler workflow with Pegasus does not

provide full interoperability between them; instead of only a partial integration between these systems that might be useful for some scientists.

4.4 Comparison of Workflow Interoperability Approaches

The other approaches discussed in this section (GEMLCA/P-GRADE, VLE-WFBus, IWR, SIMDAT, and Kepler/Pegasus Integration) are limited to specific types of workflow systems, whereas the PS-SWIF approach presented in thesis is general and can be applied to different workflow systems.

In general, the other approaches discussed in this section require the installation of an environment exposing an API on a user's machine, which requires an expert or developer to set up the environment and apply the configuration for the system. In contrast, the PS-SWIF approach presented by the author is simple and based on standardized messaging Web Services standards and as long as a workflow environment already provides access to Web Services, it does not require any further installation or configuration on a user's machine or modification of the workflow engine.

The other approaches also require more modifications in the constructed workflow in order to repeat the experiment and support reusability. Within the PS-SWIF approach the reusability of the same experiment or a similar experiment with different data input and parameters can be achieved without major modifications to the system.

The other approaches, except the SIMDAT approach, do not support interoperability among workflow systems running remotely, unlike the PS-SWIF approach presented in this thesis.

4.5 Summary

In this Chapter, a number of different examples of scientific workflow systems are discussed, focusing specifically on Triana, Taverna and Kepler, being both popular and representative of the area. The author discusses how Web Services

are discovered and configured by these systems within the context of this work and how they can be used, depending on the Web Services standard. These workflow systems are used as the main workflow systems to test and evaluate the PS-SWIF approach and system. Other scientific workflow systems that support Web Services, such as GEODISE, OMII-BPEL SODIUM, VisTrails, Discovery Net, Moteur, WFEE Workflow systems, are also presented in this Chapter. These workflows are not used or tested with the PS-SWIF system. However, the Web Services of the PS-SWIF system are simple and can easily be configured by any workflow systems supporting Web Services, while no special toolkit or API is needed.

Other related approaches that tried to achieve workflow interoperability are discussed. The disadvantages of these approaches are that they are limited to specific workflow systems, require hard coding or implementation of specific APIs and are complex.

CHAPTER 5

PS-SWIF, Requirements, Architecture and Design

This Chapter describes the PS-SWIF architecture and its components that collectively provide interoperability between heterogeneous scientific workflow systems. Requirements to achieve interoperability are identified. This Chapter also provides a detailed investigation and design of models and solutions for system requirements, and considers how workflow interoperability models provided in Chapter 2 can be achieved using the PS-SWIF system.

5.1 PS-SWIF Approach

The PS-SWIF model (Publish/Subscribe for Scientific Workflow Interoperability Framework) presented in this thesis aims to achieve interoperability among scientific workflow systems using the publish/subscribe model. PS-SWIF uses asynchronous messaging exchanges between participants which provides a loosely coupled communication pattern in large scale distributed computing. In some situations, it may be more convenient for a notification consumer to use synchronous messaging in order to control the flow and timing of message arrival and therefore the PS-SWIF approach also supports synchronous communication for this reason.

5.2 PS-SWIF Requirements

During the initial stage of this research, looking for a solution to present interoperability among different workflow systems, a number of requirements

that should be included in the design and implementations were identified. Although, as is the case with most research projects, it was difficult to identify a complete set of requirements at the beginning, so a number of the requirements were changed or modified throughout the research.

1. There are various workflow systems developed by different partners to resolve problems in special domains using specific workflow languages. The system should provide interoperability among a wide range of systems and not be limited to any special type of workflow system.
2. There are different types of workflow systems in terms of invoking and deploying a workflow as a Web Service. Some of the existing workflow systems have the ability to deploy a workflow as a Web Service, such as Triana workflow. Others do not support the deployment of a workflow as a Web Service, such as the Taverna and Kepler workflow systems. The system presented in this thesis should provide interoperability for both these types of workflow systems.
3. The system should achieve interoperability among different workflow systems without the need for integration or modification of the source code of the workflow systems. The system should be a stand-alone framework and not be dependent on other systems. Advantages of this requirement are:
    - It allows users to extend the system without affecting other workflow systems.
    - It avoids the integration of the system with each workflow system, which is time and cost effective.
    - It avoids the modification of a workflow system once a new version of a workflow system is released (future proof).
    - Some source codes for the workflow system can be private. This requirement ensures that modifications to the implementation of individual Services of the system will not disrupt the workflow system infrastructure as a whole. This is essential for developing and the usage of the system and allows non-public workflow

implementations to be used without compromising an internal private licence of the source code.

4. The system should achieve workflow interoperability that covers different types of communication between workflow systems. The advantage here is that different scenarios of interoperability can be applied among e-science projects.
5. The system must provide a user-friendly interface that allows users to easily interact with the system components. This requirement provides the advantage of allowing scientists to focus on their experiments and not spend time on learning technical computer skills or APIs.
6. The system should provide reusability of experiments without major modifications to the experiment. Scientists might need to repeat their experiments to get consistent results.
7. The system should be able to protect the data published by a user, such as topic and subscription details, and should not allow unauthorized users to remove and update this information.
8. The system should be able to recover the data when the system unexpectedly shuts down or Services are not available for any reason.
9. The system should be able to run on different platforms and in different environments. This allows different workflow systems running remotely to interoperate with each other.

5.3 Alternate Designs

In this section, other alternate designs are discussed; namely, direct communication and distributed storage, such as replica location service (RLS) [144].

5.3.1 Direct Communication Design

Direct communication design requires that all participants (producer and receiver) involved in the communication are present at the same time, which is why it is called a synchronous communication or real time communication. On the other hand, communication using an asynchronous messaging approach

provides a decoupling of participants in both space and time domains. In the space domain, the notification messages are delivered between participants without the need to know about each other. The producer sends notification messages through a mediator and the receiver receives these notifications indirectly through the mediator. In the time domain, the asynchronous communication allows the producer and receiver to communicate with each other, even if they are not active at the same time. Specifically, the producer can send notification messages while the receivers are disconnected, and, in the opposite direction, the receiver can receive notification messages while the producer that generates these messages is disconnected.

For these reasons, it is clear that providing a system using a direct communication approach is not fulfilling the requirements of the system presented in this thesis.

5.3.1 Distributed Storages Design

The replica location service (RLS) maintains and provides access to mapping information from logical names for data items to target names [144]. It allows data to be put onto a distributed storage system and then retrieved later. This is overkill and heavyweight, especially for passing data between two workflow systems. Moreover, there are similar systems that provide similar functionality such as Freenet [145], and Ocean Store [146]. There are multiple storage systems that could also be used but they have several disadvantages:

- They require administration on setting up, hosting, and port permissions, etc.
- Management: how do you manage the data when it populates the system? i.e. do you delete when data is consumed? Or after a period? Or when it is told to by the workflow. I think inevitably such an approach would be hard to manage transparently so you would end up having to incorporate garbage collection routines into the Client API, which would result is a far more complicated approach.

For these reasons, it is clear that providing a system using a distributed storage is not fulfilling the requirements of the system presented in this thesis.

5.4 PS-SWIF Architecture

Figure 5.1 shows the high level architecture for the PS-SWIF approach. The system consists of three layers: Workflow layer, Web Service layer and publish/subscribe layer.

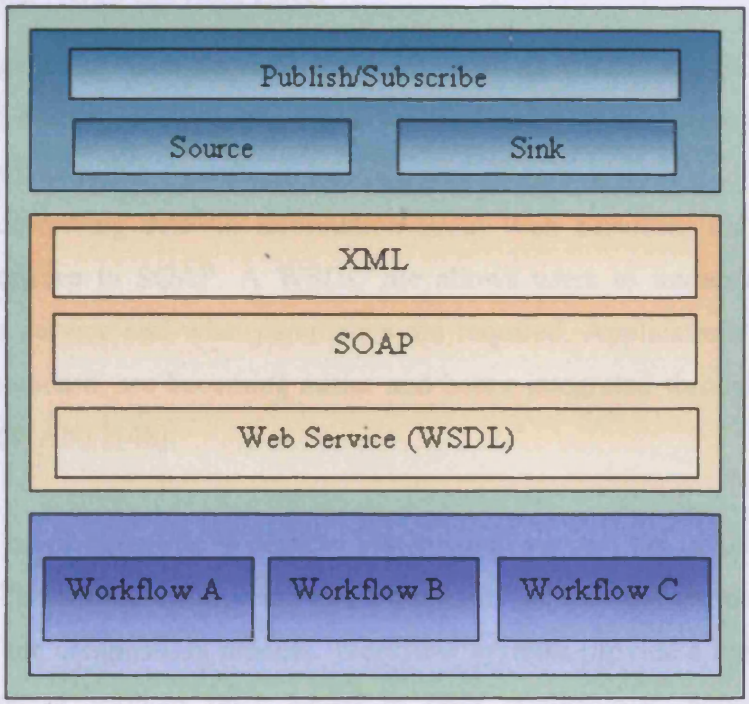


Figure 5.1: PS-SWIF High Level Architecture

5.4.1 Workflow Layer

The Workflow Layer represents different workflow systems. Each of these systems uses their proprietary language, which allows users to construct workflows, and has its own enactment engine to execute these workflows. Each programming language and engine thus has implicit assumptions and constraints about what types of applications should be developed and also about what types of workflows can be constructed. The only assumption made in using the PS-

SWIF approach is that the workflow engines must have a capability to invoke the Web Services standard. The PS-SWIF approach supports workflow systems written in any language and running on different operating systems. Each of the available workflow systems used for the various domains can leverage this architecture. The workflow system can act as a workflow publisher and/or a workflow subscriber. The workflow publisher represents an entity that generates notification messages, whereas the workflow subscriber represents an entity that receives and consumes the notification messages.

5.4.2 Web Services Layer

The emergence of Web Services standards, such as WSDL and SOAP, allows distributed heterogeneous applications, written in several languages and running on different operating systems, to be integrated. WSDL is an XML [147] format used for describing detailed information about Web Services, and provides a simple interface to SOAP. A WSDL file allows users to understand how to invoke the service and what parameters are required. Applications using Web Services standard, are becoming easier and better integrated through wide-area networks (WAN) [148].

There are many Scientific Workflow systems that support the invoking of Web Services. Publishing, discovering and availability of services are considered to be part of the composition process. Workflow systems provide a registry to add descriptions of services once added to their applications. Some workflow systems use a search mechanism to find a special service or provide direct invocation for WSDL files. Some Workflow systems use Web Services standards to invoke a remote service or resource or to send jobs to be executed on remote resources.

The PS-SWIF components are implemented as Web Services components. The PS-SWIF Web Services operate in a workflow system environment.

5.4.3 Publish/Subscribe Layer

Within Notification messaging systems, there are two main distinct roles involved in a notification: source and sink. The notification is a message that encapsulates information about a situation (topic) that might be of interest to other entities. The source is an entity responsible for generating the notification message. The sink is an entity that wishes to receive the notification message. The sink first needs to register its interest with the source entity, which is normally to referred as a subscription request. The source then checks to find whether the message fulfils the constraints specified in the previous request; if so, the source sends the message to the registered sink, normally referred as a notification.

5.5 Publish/Subscribe Model with PS-SWIF

In Chapter 3, several publish/subscribe paradigms are presented. The WS-Notification standard and WS-Eventing specifications are the main competing standards in this area. They use Web Services standard to allow distributed heterogeneous applications, such as workflow systems, written in several languages and running on different operating system, to be integrated.

WS-Notification relies on the WS-Resource Framework (WS-RF). WS-RF defines the relationship between a Web Service and a stateful resource; pairing of a Web Service with a resource is called a WS-Resource or just resource. WS-RF is a set of five separate specification documents that provide the standard definition of the framework: WS-ResourceProperties, WS-ResourceLifetime, WS-RenewableReferences, WS ServiceGroup and WS-BaseFaults.

WS-Notification must use WS-ResourceProperties and WSResourceLifetime to provide asynchronous notification, as well as third party specification, such as WS-Addressing. WS-ResourceProperties provides a set of interfaces that allow user to access, modify, and query resource properties. Topics are modelled as resource properties with the WS-Notification specification. WS-

ResourceLifetime provides mechanisms to manage the lifecycle of the resource, such as destroy, to finish the resource. Any application or service that wishes to communicate with a service that implements WS-Notification must be presented as a stateful service (WS-RF). In contrast, WS-Eventing does not depend on many specifications and the only specification required is WS-Addressing. In addition, the WS-Eventing service is represented as stateless service. Most workflow systems that support the invoking of Web Services only deal with standard Web Services (stateless) and most existing services are in the standard Web Service form. Workflow systems that support deploying a workflow as a Web Service also expose it as a standard Web Service, such as InforSense's KDE and Triana workflow systems. Even the Triana workflow system which supports the WS-RF and WS-Notification standard, the Web Service deployment is accomplished using standard Web Services.

In terms of delivery of notification messages to subscribers, WS-Notification does not support a synchronous notification mode (only supporting asynchronous notifications), whereas WS-Eventing supports both asynchronous and synchronous modes to deliver the message. The synchronous notification mode is especially needed with workflow systems that do not have the ability to deploy a workflow as a Web Service.

WS-Eventing provides the required functions for the Publish/Subscribe paradigm, such as subscribe, renew, unsubscribe and getStatus. For the above reasons and its simplicity and features, the WS-Eventing specification is used with some modification in the PS-SWIF approach. Since the WS-Eventing specification uses a Web Service standard to implement their entities, the PS-SWIF components also implement a Web Services standard.

5.6 Proposed PS-SWIF Framework

This section provides a deeper investigation into the design of the general architecture. Figure 5.2 shows the components of the system architecture that interact and work together to achieve its design aims. The system operation, the

role of its components and the information exchanged between components are described here.

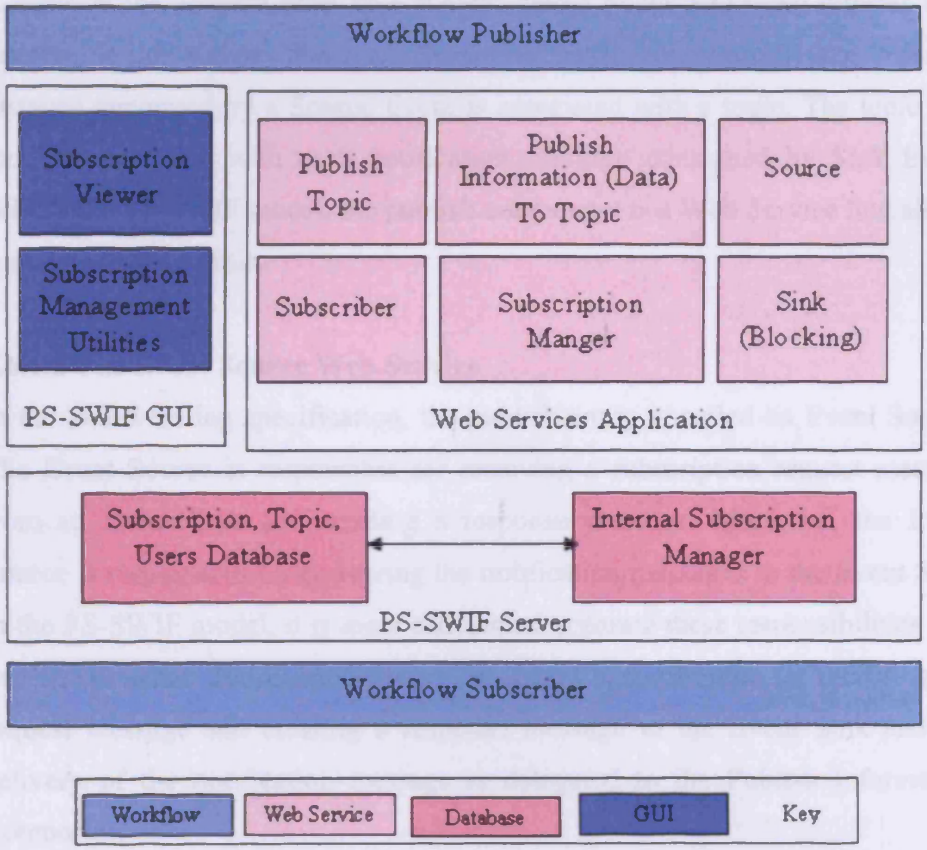


Figure 5.2: PS-SWIF Architecture Components

5.6.1 Application Web Services

The Web Service Application represents all the interfaces needed by users to interact with the system. Users may interact with the service from their application without needing to use PS-SWIF GUI. Full descriptions for each of these services are presented here.

In WS-Eventing, there is only one source Web Service called Event Source, which is responsible for receiving requests from subscribers, generating the notification messages, and sending the notification message to the subscriber. In

the PS-SWIF design, the Event Source responsibility is divided into three Web Services: Publish Topic, Source, and Publish Information Web Service.

5.6.1.1 The Publish Topic Web Service

A topic is the concept used as a subject where Event Sinks can register their interests in this activity through a Subscriber entity. Every notification message instance generated by a Source Event is associated with a topic. The topic also must be specified with each notification message consumed by Sink Event. Within the PS-SWIF model, the publish component is a Web Service that allows a user to create a topic.

5.6.1.2 The Event Source Web Service

In the WS-Eventing specification, the publish entity is called an Event Source. The Event Source is responsible for receiving a subscription request message from an Event Sink and creating a response message. Moreover, the Event Source is responsible for delivering the notification messages to the Event Sink. In the PS-SWIF model, it is more efficient to separate these responsibilities into two components. The Event Source Component is responsible for receiving the request message and creating a response message to the Event Sink and the delivery of the notification message is delegated to the Publish Information Component.

5.6.1.3 The Publish Information Web Service

The Publish Information Web Service is responsible for delivering notification messages to the Event Sinks. This service is created to act at the workflow system level. In this service, a notification message is associated with a specific topic and then sent to a subscriber client's 'Event Sink'.

Figure 5.3 illustrates how the source side services interact with each other within the PS-SWIF framework. The Publisher component represents the workflow system user who publishes a specific topic for the first time and, at the later stage, as a publisher for data for this topic, which represents a notification message that will be sent to event sink. The Publish Topic Component is

responsible for generating the Event Source Web Service. The Topic and Web Service which represent the topic are registered with the Subscription Manager component. The Source Component represents the both Source Web Service and the URL that represents this Web Service, made available to users through a PS-SWIF GUI. Anyone who wants to subscribe to this topic should use this URL. At the final stage, the Publisher sends the data for processing by other workflow engines through the Publish Information component. The Publish Information component ties the data with the specific topic and then sends it to interested sinks, representing a workflow subscriber.

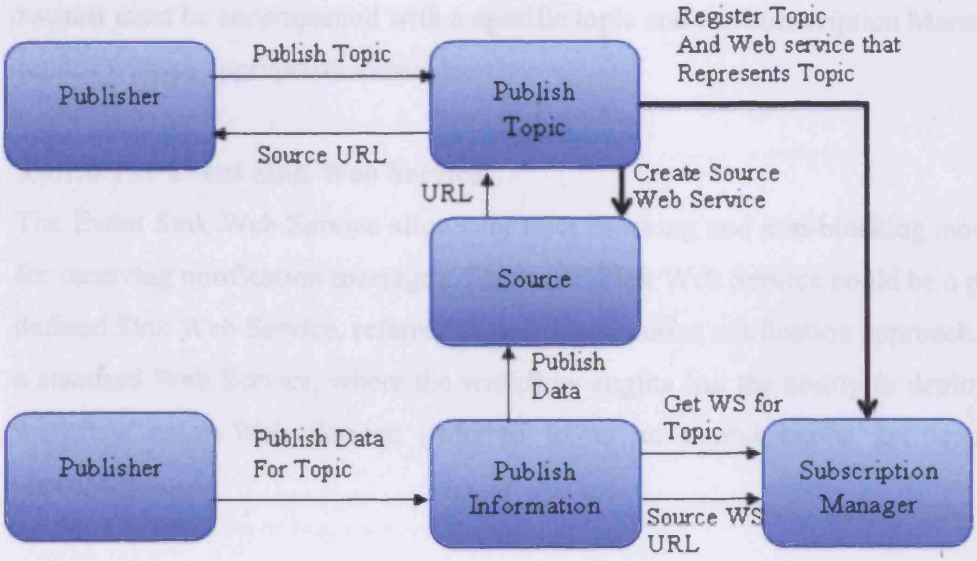


Figure 5.3: The Source Side Services Interaction

5.6.1.4 The Subscriber Web Service

The Subscriber Web Service is used to allow a client to issue different message requests to the Event Source Web Service and Subscription Manager Web Service. The message requests in the PS-SWIF model are similar to the message requests defined in the WS-Eventing. A subscription request is sent to Event Source and other requests, such as renew, unsubscribe, and getStatus, are sent to the Subscription Manager Component. Each request must be associated with a specific topic and the recipients of the messages, either Event Source or

Subscription Manager Web Service, will reply to the subscriber with an appropriate message.

5.6.1.5 The Subscription Manager Web Service

In the PS-SWIF model, the Subscription Manager Web Service is used to manage the subscription created by the Subscriber Web Service. The Subscription Manager Web Service therefore handles requests, such as renew, unsubscribe, and getStatus. These requests are supported by WS-Eventing and give flexible options to users to manage their subscriptions. The renew request is used to extend an existing subscription with new data. The unsubscribe request is used to unsubscribe an existing subscription for a specific topic. The getStatus request is used to obtain the current status of an existing subscription. Each request must be accompanied with a specific topic and the Subscription Manager replies with an appropriate message.

5.6.1.6 The Event Sink Web Service

The Event Sink Web Service allows for both blocking and non-blocking modes for receiving notification messages. The Event Sink Web Service could be a pre-defined Sink Web Service, referred to as a synchronous notification approach, or a standard Web Service, where the workflow engine has the ability to deploy a workflow as a Web Service (referred to as an asynchronous notification approach).

Within a synchronous approach, users issue a request on a service which blocks their processing while they wait for a response. The synchronous approach is preferred when the service can perform the request within a short time. The synchronous approach is also preferred when the calling application requires a fast response to a request.

The predefined Sink Web Service (synchronous approach) supports workflow systems that do not have the ability to deploy a workflow as a Web Service, such as the Taverna and Kepler workflows. The predefined Sink Web Service

methods are invoked by the workflow systems to allow them to receive notification messages synchronously.

Figure 5.4 shows how a subscription is made in the case of the synchronous approach. The Subscriber Workflow subscribes a sink service with a specific topic through the Subscriber Web Service. Then, the Subscription Manager looks up the topic and returns the Web Service that represents this topic. If the subscription is handled successfully, a subscription ID with UUID (Universally Unique Identifier) format that represents a unique ID, will be registered with the topic and also returned to the workflow subscriber. The sink service is blocked and waits for notification data. When the event is sent, the Source will notify the sink component and data will be returned to the subscriber workflow.

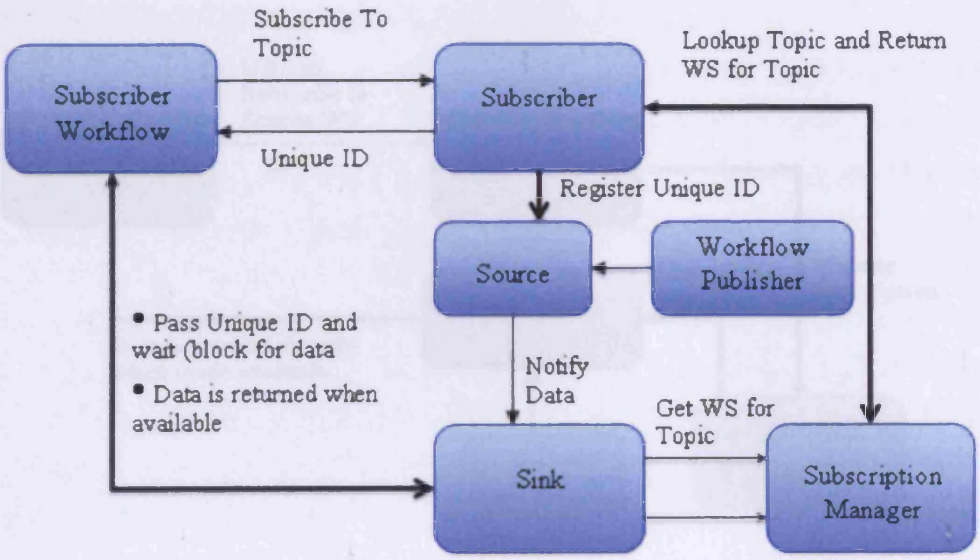


Figure 5.4: Synchronous Subscription

Within the asynchronous approach, users issue a request on a service and then continue their processing without waiting for an immediate response. The service receives the user request and returns a response at some later stage, at which time the users retrieves the response and continues with their processing.

If the workflow systems support deployment of a workflow as a Web Service, such as the Triana workflow, then the workflow Web Service (asynchronous approach) represents the Sink Web Service and receives notification messages instead of using the predefined Sink Web Services.

Figure 5.5 shows how the subscription is made using the asynchronous approach. This approach is valid only for a workflow system that has the ability to deploy a workflow as a Web Service. The subscriber workflow here acts as a sink service and does not need to use the predefined sink as in the synchronous approach. The subscriber workflow subscribes with a specific topic through the Subscriber Web Service, using a URL that represents the subscriber workflow Web Service. The subscriber workflow is notified with data when available through a direct invocation of its deployment Web Service.

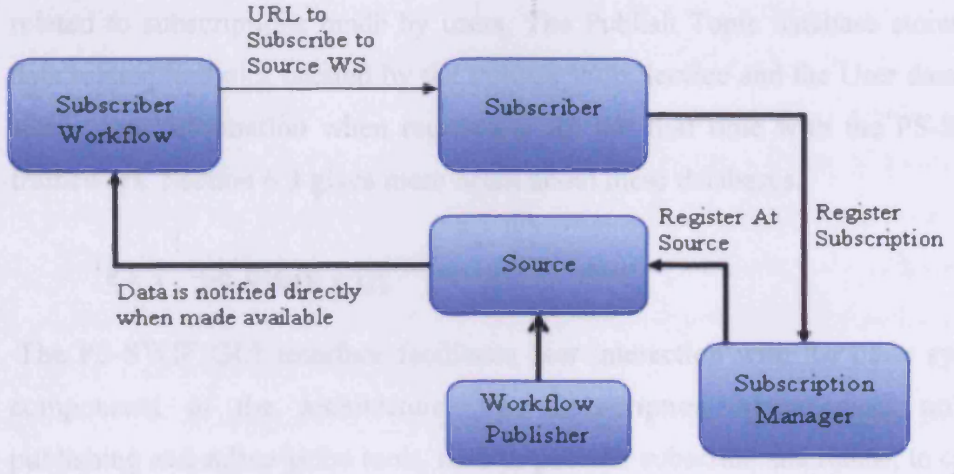


Figure 5.5: Asynchronous Subscription

5.6.2 PS-SWIF Server

The PS-SWIF Server represents a back end implementation for PS-SWIF Web Services and is responsible to save the topics and subscriptions created in the PS-SWIF GUI in a database.

5.6.2.1 Internal Subscription Management Component

The Internal Subscription Management Component presented by the PS-SWIF model provides management for subscriptions, and delivery for notification messages. Such a Subscription Management component represents a neutral mediator and controls all the components in the PS-SWIF model. The component provides a backend housekeeping implementation for all the features and operations provided by other components in the PS-SWIF.

5.6.2.2 PS-SWIF Databases

The system should store the information on topics, subscriptions and user information in a database. The author has designed three databases to handle the different information sets: the Subscription database; Published Topic database; and Users database. The Subscription database is used to store all information related to subscriptions made by users. The Publish Topic database stores the data related to topics created by the Publish Web Service and the User database stores user information when registering for the first time with the PS-SWIF framework. Section 6.3 gives more detail about these databases.

5.6.3 PS-SWIF GUI

The PS-SWIF GUI interface facilitates user interaction with the other system components in the architecture. The Subscription Management utilities publishing and subscription tools, such as publish, subscribe and renew, to create topics and manage the subscriptions. Once the subscription is successful the Subscription Viewer allows users to view detail, such as topic, subscription ID and sink. Section 6.7 gives more detail about PS-SWIF GUI.

5.6.4 Workflow Publisher and Workflow Subscriber

The workflow publisher represents an entity that generates notification messages. The workflow publisher must invoke the Publish Information Web Service to send data/notification messages to Workflow Subscribers. The workflow subscriber represents an entity that subscribes and receives the notification messages. The subscription is made through the Subscriber Web

Service using PS-SWIF GUI. The Workflow Subscriber needs only to invoke the Sink Web Service for synchronous notification or use its own workflow Web Service for asynchronous notification to receive the notification message.

5.7 Interaction between Components in the Architecture

The UML sequence diagram in Figure 5.6 on the following page, illustrates the sequence of activities undertaken by the PS-SWIF system. The sequence diagram simplifies all the interactions with the system from the point of view of the user and does not give full details of how the system works.

1) The scientist creates a topic using the Publish Topic Web Service.
2) A source Web Service with the topic name will be generated automatically to receive a subscription request and create the response message.
3) The scientist creates a subscription request through the subscriber Web Service specifying the sink value which is interested to receive a notification message on this topic and also to identifying the expiry date for this subscription. If the sink value is inserted as URL, the system automatically recognizes this as an asynchronous subscription. If the sink value represented a string, the system considers this as a synchronous subscription and will use the predefined sink Web Service.
4) If the subscription is handled successfully by the system, the reply message will contain a subscription ID and reference point of the Subscription Manager for further interaction regarding to this subscription.
5) The scientist can create other requests such as renew, unsubscribe, and getStatus and send them to Subscription Manager Web Service using the subscription ID.
6) The Subscribe Manager will replies with an appropriate message if the request is handled successfully by the system.
7) When the event is fired, all sink Web Services subscribed to this topic will receive a notification message.

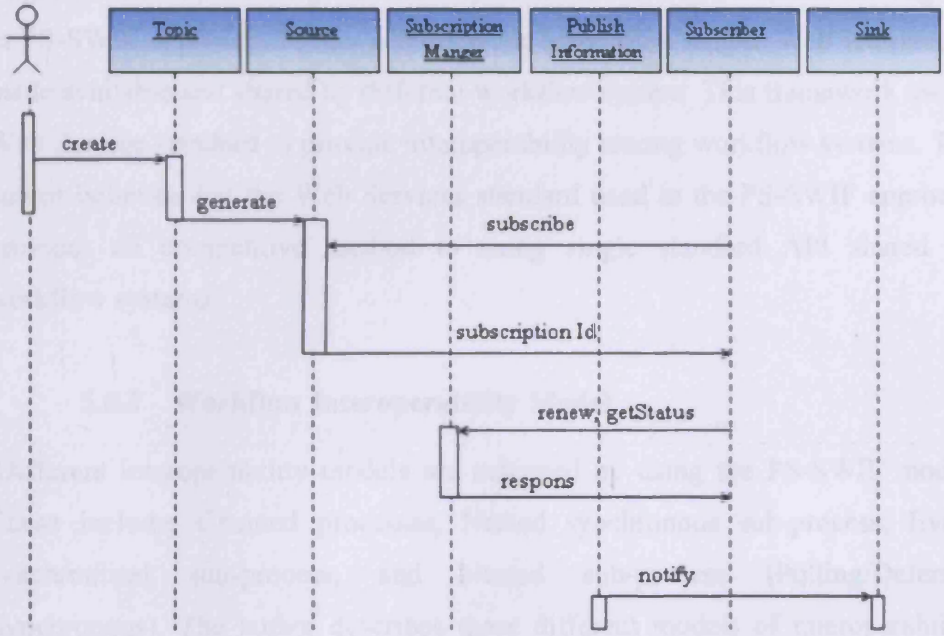


Figure 5.6: Interaction between Components

5.8 Workflow Interoperability

In Chapter 2, interoperability strategies, levels and models are discussed based on WfMC specifications. In this section we identify the PS-SWIF approach according to these classifications.

5.8.1 Workflow Interoperability Strategies

According to the WfMC specification, interoperability within PS-SWIF can be classified as a Message Passing strategy where Workflow Systems exchange information by sending packets of data messages through a communication network. This is represented by notification messages sent by one workflow system to another using PS-SWIF Web Services.

5.8.2 Workflow Interoperability Level

Workflow Interoperability can be achieved at different levels according to WfMC. One of these levels is a complete workflow API Level which shares a

single standard API among workflow systems. There is no API used directly the in PS-SWIF approach. Rather a framework based on a simple Web interface is made available and shared by different workflow system. This framework uses a Web Service standard to provide interoperability among workflow systems. The author believes that the Web Services standard used in the PS-SWIF approach provides an competitive method to using single standard API shared by workflow systems.

5.8.3 Workflow Interoperability Model

Different interoperability models are achieved by using the PS-SWIF model. These include: Chained processes, Nested synchronous sub-process, Event synchronized sub-process, and Nested sub-process (Polling/Deferred Synchronous). The author describes these different models of interoperability between heterogeneous workflow systems and how they can be supported in a generic way.

5.8.3.1 Chained Process Model

In the following descriptions the term Workflow System *A* is used to present the workflow system that uses a process instance to send separate data to a process instance on another workflow system, called Workflow Consumer *B*. The process instance on Workflow System *A* is represented by a Publish Information Web Service, which is responsible to send the data or messages that need to be processed on other workflow systems. The process instance on Workflow System *B* is represented by the Sink Web Services. The Sink Web Service receives the messages and forwards them to another process in the workflow to do further processing. Figure 5.7 shows the model.

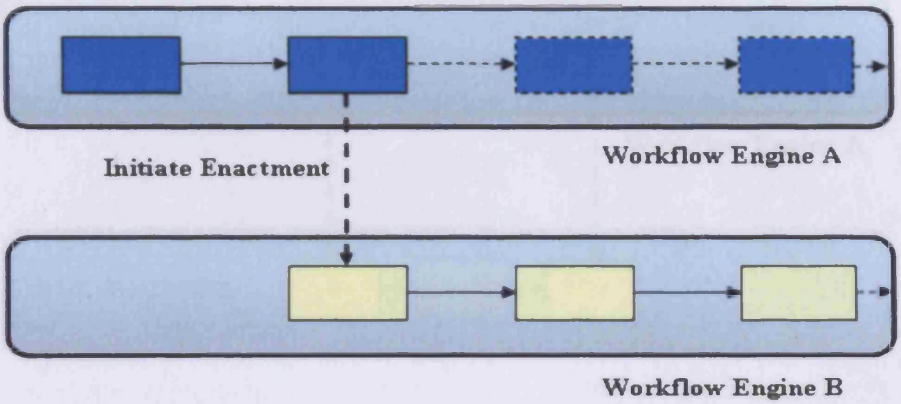


Figure 5.7: Chained Process Model

5.8.3.2 Nested Synchronous Sub-Process

This model assumes a process instance in Workflow System *A* sends data to Workflow System *B* and Workflow System *A* waits at the same stage until the data is processed by System *B* and a result is sent back to Workflow System *A* for another process with the data. The PS-SWIF model satisfies the requirements of this model by allowing Workflow System *A* to construct a workflow that includes two process instances next to each other. The first process uses the Publish Information Web Services to send data to System *B*. The second process uses the Sink Web Service to receive the result back from Workflow System *B*. In addition, Workflow System *B* constructs a workflow using a process instance that invokes the Sink Web Service to receive the data from Workflow System *A* and uses other process instances to do more processing with data, and, at some stage, the Sink Web Service is invoked by one process instance to send the final result to Workflow System *A*. This model is showed in Figure 5.8.

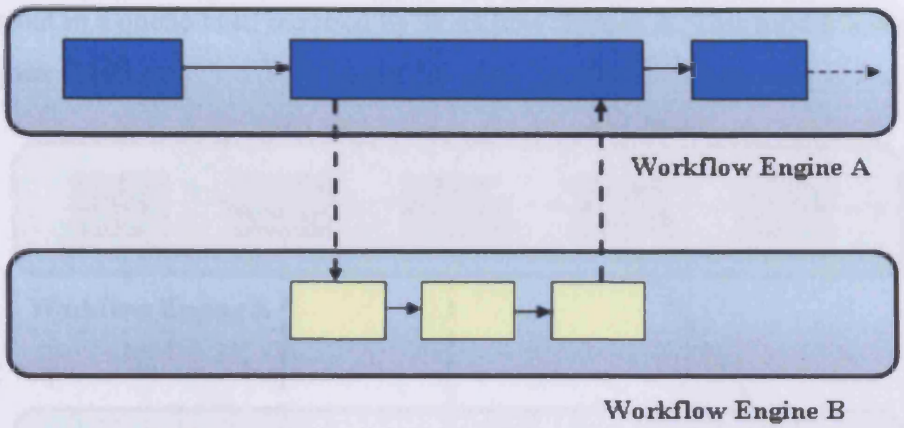


Figure 5.8: Nested Synchronous Sub-Process

5.8.3.3 Event Synchronized Sub-Process

This model is similar to the Nested synchronous sub-process except that the result received from the workflow system *B* should be received at a different stage of processing. The PS-SWIF model achieves this type of interoperability by adopting the similar scenario in the previous model with a little modification. The invocation of the Sink Web Service would be constructed at a later stage of the workflow by an instance process. This model is shown in Figure 5.9.

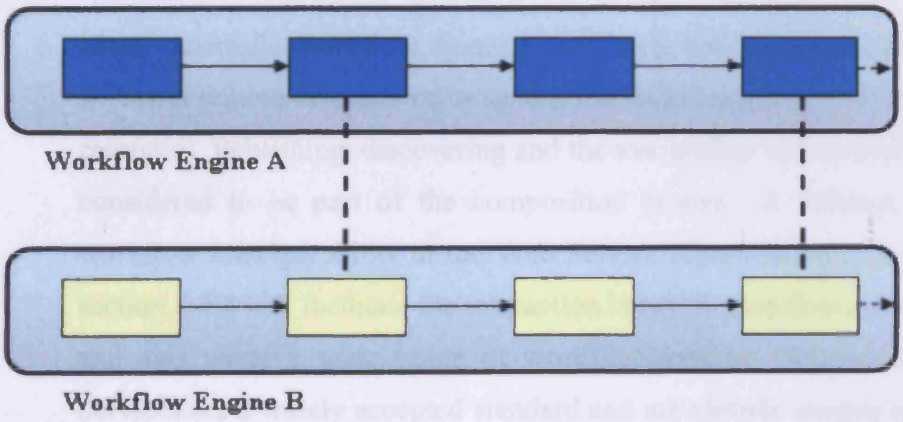


Figure 5.9: Event Synchronized Sub-Process

5.8.3.4 Nested Sub-Process (Polling/Deferred Synchronous)

This model is similar to the Event synchronized sub-process model except that Workflow System *B* might complete its process before the invoking process in Workflow System *A* is ready to deal with the event. The PS-SWIF model saves

the event in a queue until required by Workflow System A. This model is shown in Figure 5.10.

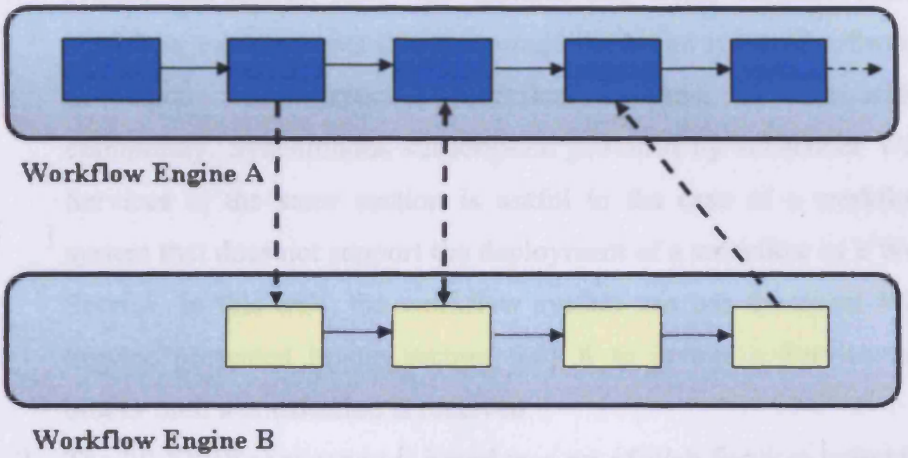


Figure 5.10: Nested Sub-Process (Polling/Deferred Synchronous)

5.9 Design Discussion

This section presents the justification for design against requirements presented in the section in the same numerical order.

1. Many scientific workflow systems use Web Service standards to invoke a remote resource or to send a job to be executed on remote resources. Publishing, discovering and the availability of Services are considered to be part of the composition process. A solution for workflow interoperability at the Web Service layer discussed in the section 5.4.2 will facilitate the interaction between workflow systems and also cover a wide range of workflow systems because Web Services are a widely accepted standard and are already integrated in most workflow systems.
2. Workflow interoperability is achieved through two types of subscriptions: asynchronous and synchronous subscription. The asynchronous subscription provided by a subscriber Web Service presented in the section 5.6.1.4 maintains a level of decoupling that can allow the coexistence of multiple workflows without the

necessity of tight integration or dependency. The advantage of this approach is that a workflow user can extract workflows for each system that exhibit their best features and create enriched multi-workflow environments that encourage the broad reuse of software, developed within specific workflow systems, by the wider community. Synchronous subscription provided by subscriber Web Services in the same section is useful in the case of a workflow system that does not support the deployment of a workflow as a Web Service. In this case, the workflow system can use the event Web Service presented in the section 5.6.1.6 to invoke a Service that blocks until a notification is received.

3. The PS-SWIF application is based on a set of Web Services presented on the section 5.6.1 and those Web Services are available online, making the application easy to use, and scientists do not need to integrate or modify the source code for any workflow systems. The communication between the workflow systems occurs through these Web Services.
4. Workflow interoperability is achieved using different scenarios of communications between workflow systems supported by different interoperability models presented in the section 5.8.3.
5. The PS-SWIF GUI interface presented in the section 5.6.3 facilitates user interaction with the other system components in the architecture.
6. The design that separates the responsibilities of the event source component to Publish Information Web Service and Event Source Web Service presented in the sections 5.6.1.2 and 5.6.1.1 respectively allows scientists to repeat their experiments with the same topics and subscriptions.
7. The user database presented in the section 5.6.2.2 allows scientists to store user details when registering for the first time on the PS-SWIF system. This information is used to authenticate the users when they want to remove a topic or subscription.

8. The PS-SWIF subscription and published topic database presented in the section 5.6.2.2 preserves the data used for interoperability, such as subscription and published topic.
9. The PS-SWIF components presented in the section 5.6.1 are implemented as Web Services and these Services are available online and access to these Services can be carried out remotely.

5.10 Summary

In this Chapter, the PS-SWIF architecture has been presented as a means to achieve workflow interoperability. The architecture is based on service-oriented architecture (SOA) technologies such as Web Services, WSDL and SOAP, which can be used with Workflow Systems to achieve workflow interoperability. The publish/subscribe system WS-Eventing is used to provide a synchronous and an asynchronous notification among various workflow systems. PS-SWIF provides a mechanism to transfer data as notification messages among remote workflow system for further processing. The PS-SWIF approach can fit with any workflow system written in any language and running on different operating systems that have the capability to invoke Web Services. The flexibility of the PS-SWIF approach allows construction of a workflow with a number of PS-SWIF Web Services leading to different interoperability models provided by WfMC.

CHAPTER 6

PS-SWIF Implementation

This Chapter presents an overview of the implementation of the PS-SWIF system. PS-SWIF Web Services are discussed in terms of creation and deployment using a WSPeer framework. How these services are implemented by different workflow system is also discussed. A PS-SWIF framework user interface is presented with functions and tools that can be used to achieve workflow interoperability. This Chapter does not give full details of implementation or a user guide of the system, but rather highlights the system's functionality and implementation.

6.1 Implementation Overview

The implementation of the PS-SWIF system uses Java as programming language. The WSPeer framework is used to host PS-SWIF Web Services, and provides a hosting and invocation environment for such services. WSPeer has several advantages, with a key advantage that it allows the creation, deployment and handling of a Web Service without using a container. WSPeer acts as a Web server engine that uses Apache's Axis 1.2 [149] as a SOAP processor. In addition to the main capabilities provided by Axis, such as deployment and message handling, WSPeer supports publishing and discovery mechanisms. The WSPeer deployment interface is used to deploy all PS-SWIF services. Once the deployment is successful, WSDL files are generated to describe the PS-SWIF services. Appendix C shows the WSDL file for all PS-SWIF services. Figure 6.1 presents an overview of the implementation architecture, showing how the PS-SWIF system components and application are implemented.

6.2 PS-SWIF Web Services

Because PS-SWIF is based on WS-Eventing, most of the important entities are implemented as Web Services, and the system implements all the Web Services provided by WS-Eventing, including a Source Web Service, Subscriber Web Service, Subscription Manager Web Service and Sink Web Service. In addition two Web Services, namely Publish Topic Web Service and Publish Information Web Service, which are not covered by WS-Eventing, are implemented by the PS-SWIF system to add additional flexibility to the system.

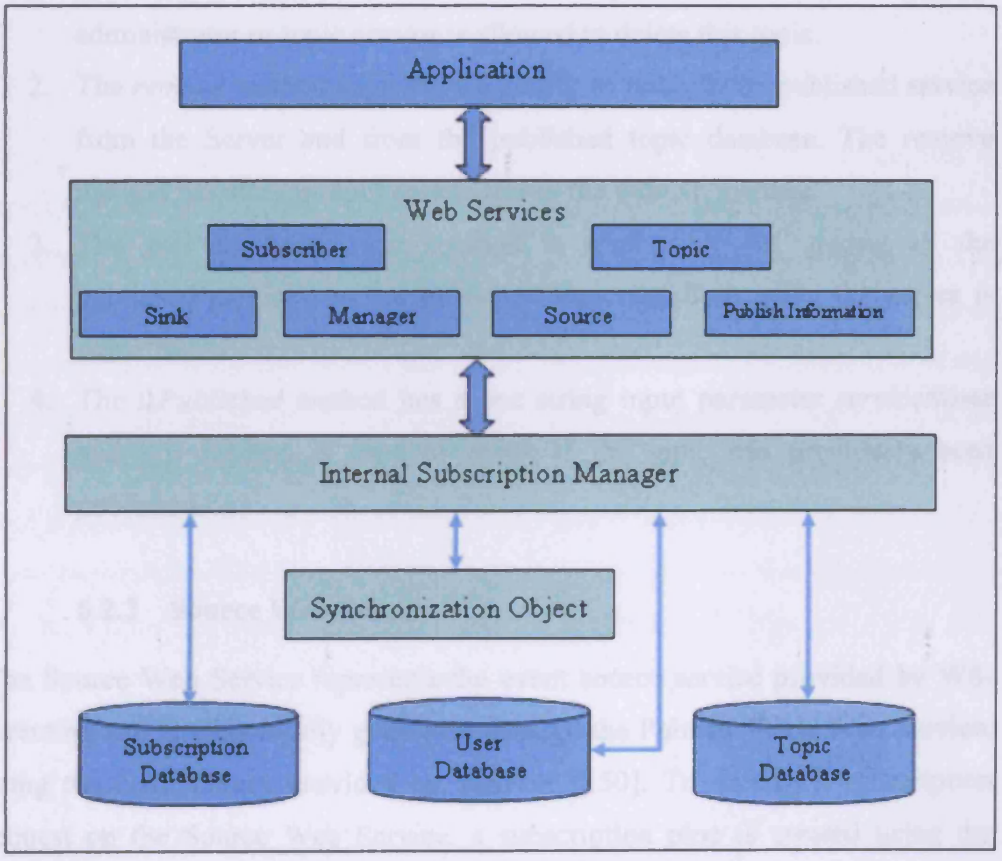


Figure 6.1: Implementation Architecture

6.2.1 Publish Topic Web Services

WS-Eventing does not define how a topic should be created, and so the author has created a Web Service called Publish Topic Web Service to allow users to

create and manage topics. When a topic is created, a Source Web Service is dynamically generated with the topic name. Topics are created through a *publish* operation and leave the management of the topic to other operations, such as *remove*, *getPublishedServices* and *isPublished.*

The Publish Topic Web Service has the following operations:

1. The *publish* method has three string input parameters; *serviceName*, *user* and *password.* The *serviceName* allows a user to define a title for a published topic (service) to be used as a Source Web Service. The *user* specifies who published the topic and the *password.* Only the administrator or topic creator is allowed to delete this topic.
2. The *remove* method supports the ability to remove the published service from the Server and from the published topic database. The remove method has similar input parameters to the publish method.
3. The *getPublishedServices* method is responsible for getting all the published services on the published topic database when the server is restarted.
4. The *isPublished* method has a one string input parameter *serviceName* and this method is used to check if the topic has previously been published.

6.2.2 Source Web Service

The Source Web Service represents the event source service provided by WS-Eventing and is dynamically generated through the Publish Topic Web Service, using the GAP library provided by WSPeer [150]. To receive a subscription request on the Source Web Service, a subscription pipe is created using the *InputPipes* to receive messages. The *createControlPipe* method creates a service control pipe with the subscriber name, and attaches the specified message listener to listen for messages received. When the control pipe receives a message, the input objects are passed to the *messageReceived* method as an Object, which results in the creation of a subscription response. Control pipes

and subscription responses are only guaranteed to be active when the Source Web Service has been called.

The Source Web Service is responsible for the following operations to:

- Receive the subscription requests from the subscriber, and check if the request is compatible with WS-Eventing syntax;
- Create a response message to the subscription request, including a unique subscription ID;
- Delegate the adding of subscription information to the internal subscription manager service;
- Delegate the Subscription Manager to handle future requests.

6.2.3 Publish Information Web Service

A Publish Information Web Service is used to send notification messages from a workflow client, such as Triana, Kepler and Taverna, on behalf of the Source Web Service, to Sink Web Services. The workflow client invokes the Publish Information Web Service, then constructs a workflow instance, using one of the operations provided by this service, and then executes the workflow. The Publish Information Web Service has the following operations: (1) *sendNotification*, and (2) *sendNotificationValues*. Both operations have two input parameters. The first parameter is used to enable the workflow client to choose a topic previously published. The second parameter represents data that should be sent as a notification message to the registered Sink Web Service as a single value in the *sendNotification* operation and as an array of values attached in the *sendNotificationValues*.

The Publish Information Web Service uses an asynchronous notification mechanism because the *sendNotification* operation returns immediately and the server then notifies the subscribers, which means that the notifier does not need to wait until all subscribers are notified, as the server is responsible to deliver the notification. In a workflow publisher that invokes the *sendNotification* method, a

process that comes after the *sendNotification* process will execute immediately and not affect whether the subscribers receives the notification.

The delivery of notification messages is delegated to the internal subscription manager, more fully discussed in Section 6.4.

6.2.4 Subscriber Web Service

A Subscriber Web Service acts as a service requester, sending request messages to a Source Web Service, on behalf of the Sink Web Service, to perform a particular operation. There are four operations provided by this service: *subscribe*, *renew*, *unsubscribe* and *getStatus*.

The *subscribe* operation is used to send a subscribe request to register a Sink Web Service with the Source Web Service. The Sink Web Service represents a workflow consumer and the Source Web Service represents a workflow producer. The subscriber request is created through a *SubscriberClient* Class. The *SubscriberClient* sends a SOAP message to a Source Web Service, and if the subscription is successfully generated a SOAP response message will be sent to the subscriber.

A valid subscribe request should include the following elements in the header of the SOAP message:

1. Event Source Endpoint references: This specifies a Source Web Service that the Event Sink wishes to subscribe to and from which it also gets notification messages. The Endpoint reference here is provided by the WS-Addressing which is supported by WS-Eventing. The Endpoint reference must have a *To* element in the header of the SOAP message that identifies the receiver of the message.
2. Message ID: This is presented by a Universally Unique Identifier (UUID) which is a unique identifier for a message. Any response to this message must include the *messageId* of the request in the *RelatesTo* element.

Figure 6.2 shows the SOAP message for the subscribe request.

```
<?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"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">

<soapenv:Header>
<wsa:MessageID soapenv:mustUnderstand="0">
uuid:a01c5bb0-8293-11de-be03-bcc58378b077
</wsa:MessageID>
<wsa:To soapenv:mustUnderstand="0">
http://alqaoud:4804/wspeer/PublishTwoTest
</wsa:To>
<wsa:From soapenv:mustUnderstand="0">
<wsa:Address>
http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
</wsa:Address>
</wsa:From>
</soapenv:Header>

<soapenv:Body>
<wsewf:Subscribe_PublishTwoTest xmlns:wsewf=
"http://alqaoud:4804/wspeer/PublishTwoTest">
<ns3:Subscribe
xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:ns3="http://schemas.xmlsoap.org/ws/2004/08/eventing"
xmlns:ns4="http://www.wseventing.workflow.com"
ns4:UserName="alqaoud" ns4:Password="*****">

<ns3:Delivery>
<ns3:NotifyTo>
<ns2:Address
xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/08/addressing">
http://alqaoud:4802/wspeer/TrianaImage
</ns2:Address>
<ns4:operationName>TrianaImage</ns4:operationName>
</ns3:NotifyTo>
</ns3:Delivery>

<ns3:Expires>2009-10-22T15:15:00.000+01:00</ns3:Expires>
</ns3:Subscribe>
</wsewf:Subscribe_PublishTwoTest>
</soapenv:Body>
</soapenv:Envelope>
```

Figure 6.2: Subscribe Request SOAP Message

In the body of the SOAP message the following elements should be defined:

1. Request Type Element: This specifies the action type; subscribe, unsubscribe, renew or *getStatus* request. Within this element, two attributes are defined, namely username and password to identify the user identity and combine these with each subscription.

2. Event Sink Endpoint references: This enables the Event source to send notification messages that match the subscription constraint to the event sink. The address of the event sink is defined with *Delivery* and *NotifyTo* elements supported by the WS-Eventing specification.
3. Expires At: The expiry element provides date and time which define when this subscription should no longer be valid.

The Source Web Service then creates a Subscribe Response and sends it back to the subscriber through *SubscriberClient*. This response includes an Endpoint reference (EPR) for a subscription manager which the subscriber may interact with to manage the subscription. The EPR contains a subscription identifier generated by the Source Web Service to uniquely represent the subscription. This subscription identifier is essential for all other communications (*renew*, *getStatus* and *unsubscribe*) pertaining to the subscription. Figure 6.3 overleaf shows a subscription response received by the subscriber from the Source Web Service. The Subscription ID contained within this response message is '5ba94273-f449-4fc4-8ca4-6a300b93aa6a'. The response message also includes the expiry element that specifies the expiry time for this subscription.

The *getStatus* operation performs a check request for the status of an existing subscription. The SOAP encoded *getStatus* request will be sent to the Subscription Manager address with the subscribe ID generated by the Source Web Service in the response message. The Subscription Manager replies with the status of the subscription; whether the subscription is still valid or has expired.

The *renew* operation allows the subscriber to extend a valid subscription with a new date and time. The *renew* request SOAP message also contains the subscription ID generated by the Source Web Service in the response message. Once the request is successfully updated, a new response message is sent to the subscriber with a new date and time.

The *unsubscribe* operation allows the users to cancel the registered subscription with the Source Web Service so the Sink Web Service will not receive any further notification messages. The request is sent to the Subscription Manager Web Service with subscription ID. If the request is handled successfully by the system, there is no reply to this message sent to the subscriber, as stated by the WS-Eventing standard; the system will instead delete the subscription from the subscription database.

```
<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"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">

<soapenv:Header>
<wsa:MessageID soapenv:mustUnderstand="0">
uuid:a081ad30-8293-11de-be03-bcc58378b077
</wsa:MessageID>
<wsa:To soapenv:mustUnderstand="0">
http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
</wsa:To>
</soapenv:Header>

<soapenv:Body>
<ns1:subscribeResponse soapenv:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://alqaoud:4804/wspeer/WseRpcSubscriberService">
<ns3:SubscribeResponse xsi:type="ns3:SubscribeResponse"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:ns3="http://schemas.xmlsoap.org/ws/2004/08/eventing">

<ns3:SubscriptionManager>
<ns2:Address>
http://alqaoud:4804/wspeer/WseRpcSubscriptionManagerService
</ns2:Address>
<ns2:ReferenceParameters>
<ns3:Identifier>
uuid:5ba94273-f449-4fc4-8ca4-6a300b93aa6a
</ns3:Identifier>
</ns2:ReferenceParameters>
</ns3:SubscriptionManager>

<ns3:Expires>2009-10-22T15:15:00.000+01:00</ns3:Expires>
</ns3:SubscribeResponse>
</ns1:subscribeResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Figure 6.3: The Subscribe Response SOAP Message

6.2.5 Subscription Manager Web Service

The Subscription Manager Web Service manages the subscriptions created. The WS-Eventing standard notes it is more flexible to delegate the subscription management to a separate Web Service. The *EndpointReference* of the Subscription Manager Web Service is included in the response message to a subscription request to the Source Web Service. Interactions, such as *renew*, *unsubscribe* and *getStatus* requests are handled by the Subscription Manager Web Service. It also implements a new operation *getAllSubscriptions*, not provided for in the WS-Eventing standard. The *getAllSubscriptions* operation is used to populate the list of subscriptions in the PS-SWIF application.

6.2.6 Sink Web Services

The Sink Web Service represents the workflow needed to receive notification messages. The Sink Web Service has only one operation *receiveNotification*. The *receiveNotification* is a blocking operation that blocks until a notification is received for the specified source ID. The *receiveNotification* has two input parameters both Strings: *source ID*, and *workflow subscriber*. The *source ID* represents the Source Web Service. The *workflow subscriber* is specified by the user when the subscription is made. The user who makes this subscription can use the *receiveNotification* operation to receive notification messages. In the Triana Workflow, or any standard Web Service that needs to receive notification messages, using the Sink Web Service is not essential. Because the workflow instance in Triana can be deployed as a Web Service, once the instance in Triana makes a subscription to a Source Web Service, it is notified when the event occurs. The Sink Web Service provides support to workflow products that do not have the capability to deploy a workflow as a Web Service. The Sink Web Services act as a listener for a notification message and when a notification message is received, the dependent processes will carry on with their execution and if the workflow supports parallel execution; other processes will not be affected by the Sink Web Service while waiting for a notification message.

6.3 PS-SWIF Databases

The system uses 'Apache Derby' as a relational database to implement the PS-SWIF database. The PS-SWIF database includes three tables which represent the Subscription Database, Published Topic Database and Users Database. The connections to these databases are made through JDBC, and follow the same pattern for query construction and result set processing.

The Subscription database is used to store all information related to subscriptions, and Table 6.1 gives brief details on information saved in this database.

Column Name	Description
SUBSCRIPTION_ID	A unique subscription ID for each subscription is created
SUBSCRIPTION_TS	A timestamp is registered when the subscription is created
SOURCE_ID	Represents a workflow publisher
EXPIRATION	The expiry data specified by user
SUBSCRIBE	Represent a workflow subscriber
OWNER	The user who creates the subscription

Table 6.1: Description of the Subscription Database

Different SQL statements are used to manipulate the data in the Subscription database and Appendix D gives more detail of all the SQL statements used to manipulate this database.

The Publish Topic database stores the data related to topics created by the Publish Web Service. There are only two columns with this database: *topic* and *owner*. The topic represents the Source Web Service and the owner represents the user who creates the topic. The topic combines with the owner in this database to allow the owner to remove the topic, if needed.

The User database stores user detail when registering for the first time on the PS-SWIF Web application. This information is used to authenticate the users when logging in to the system, for example, to remove a topic or subscription.

6.4 The Internal Subscription Manager

The internal subscription manager plays a major role in the PS-SWIF system. It implements all the functions provided by PS-SWIF Web Services, and manages everything related to a subscription and the persistence of the published services that allows for receiving subscriptions and notifications from many sources. There is only a single instance of the internal service, which holds some information in memory, such as published services and subscriptions, to avoid reading the database every time.

The internal subscription manager uses a synchronous method to protect the published topic, just in case two concurrent invocations of the publish Service method of the Publish Topic Web Service are received.

The internal subscription manager uses a *sendNotification* operation called by the Publish Information Web Service to protect the publish topic by a synchronized method and calls a *SendNotification* synchronized method implemented by a Synchronization object. Figure 6.4 describes the main functions supported by the Internal Subscription Manager.

```
public void setPublishListener(PublishListener listener) [1]
public synchronized AddSubscriptionResult addSubscription [1]
public synchronized void publishService [1]
public synchronized void removePublishedService [1]
public PublishedServiceEntry[] getPublishedServices [1]
public boolean isPublishedService(String serviceName) [1]
public List<SubscriptionInfo> getAllSubscriptions [1]
public boolean isSubscribed [1]
public void removeSubscription [1]
private void expireEntry(SubscriptionInfo entry) [1]
public String renewSubscription [1]
private boolean authenticate [1]
public String getSubscriptionExpiration [1]
public static EndpointReferenceType
createSubscriptionManagerEPR [1]
public void notifySubscribers [1]
public Object[] receiveNotification[1]
```

Figure 6.4: Internal Subscription Manager Methods

- The *setPublishListener* method sets the publish listeners that will receive notifications when a service is published or removed.
- The *addSubscription* method adds a new subscription for the given source and returns the assigned subscription ID. The source ID represents the Source Web Service which must be previously published using the publish Service operation.
- The *publishService* method uses a synchronous clause to protect the published topic, just in case two concurrent invocations of the publish Service method are received. This method is used to implement the publish operation used by Publish Web Services.
- The *removePublishedService* method implements the remove operation provided by Publish Web Service to remove the published service with a specific username from the publish topic database.
- The *getPublishedServices* method retrieves all published topics from the topic database and makes them available on the PS-SWIF application.
- The *isPublishedService* method checks if a new topic has been published before, to prevent the duplication of the same topic.
- The *getAllSubscriptions* method retrieves all current subscription from the subscription database and makes them available on the PS-SWIF application.
- The *isSubscribed* method checks if the same workflow subscriber is subscribed with the same producer to prevent duplication of the subscription.
- The *removeSubscription* method allows authorized users to delete a subscription from the subscription database.
- The *expireEntry* method automatically checks whether the subscription has expired and, if so, the subscription is deleted from the subscription database using the *removeSubscription* method.
- The *renewSubscription* method is used to renew an expired subscription

or extend a valid subscription with new data.

- The *authenticate* method authenticates the received user name and password and returns 'true' if the authentication succeeds.
- The *getSubscriptionExpiration* obtains subscription expiry data and time.
- The *createSubscriptionManagerEPR* creates the EPR for the subscription manager which are the returns to the subscriber in the response message of the subscribe request.
- The *notifySubscribers* method implements the *sendNotification* operation used by Publish Information Web Services to send notification messages to the subscribers.
- The *receiveNotification* method implements the *receiveNotification* operation used by Sink Web Services for notification messages by a consumer.

6.5 Synchronization Object

This object manages the synchronization associated with the *sendNotfication* and *receiveNotification* methods invoked by the Publish Information Web Service and the Sink Web Service. This object's class has four variables; *data*: the data that should be sent in the notification message, *sourceId*: represents the name of the published service, *waitingThreads*: the number of threads waiting for a notification messages and *queue*; the queue with pending notification messages.

The *SendNotification* method is indirectly invoked by the Publish Information Web Service through the Internal Subscription Manager and checks if there are any subscribers (workflow subscribers) waiting for this notification, by implementing a thread. If so all subscribers are notified through a *notifyall* operation that wakes up all waiting threads, otherwise the notification is held in a queue until required by a subscriber.

The *receiveNotification* method is also invoked by the Sink Web Service through the Internal Subscription Manager and checks if there are any pending messages in the queue. If so the subscribers (workflow subscribers) will be notified, otherwise it waits until the thread is released by the *sendNotification* method and a notification is received for the specified source ID.

The Internal Subscription Manager and Synchronization Object provide a decoupling mechanism between workflow provider and workflow subscribers in three different domains:

- ***Space decoupling domain:*** The notification messages are delivered between the workflow publisher and workflow subscriber without the need to know about each other. The workflow publisher sends notification messages through the Internal Subscription Manager and the workflow subscriber receives these notifications indirectly through the Internal Subscription Manager. The workflow publisher does not normally have references to workflow subscribers and also does not know how many of these workflow subscribers are subscribed to this topic. In a similar manner workflow subscribers do not normally have references to the workflow publisher. Both workflow publisher and workflow subscriber are connecting through the Internal Subscription Manager.

- ***Time decoupling domain:*** The methods provided by the Synchronization Object allows the workflow publisher and workflow subscriber to communicate with each other even if they are not active at the same time. Specifically the workflow publisher can send notification messages while the workflow subscribers are disconnected, and, in the opposite direction, the workflow subscribers can receive notification messages while the workflow publisher that generates these messages is disconnected.

- **Synchronization decoupling domain**: The workflow publisher is not blocked while generating notification messages and workflow subscribers

can asynchronously receive notification messages while performing some concurrent activity.

6.6 Fault Exceptions

The main fault messages provided by the WS-Eventing specification are supported by the PS-SWIF system. The description of the fault uses the following parameters:

1. Code: to describe the fault code,
2. Reason: to describe the fault reason in English, and
3. Detail: to provide more detail about the fault.

Other fault messages supported by the PS-SWIF system:

1. *InvalidMessage*: Generated when a request message, such as *subscribe*, *unsubscribe* or *renew*, is not valid or does not comply with the WS-Eventing specification.
2. *InvalidExpirationTime*: Generated when a Subscribe request specifies an expiration time that is in the past or is invalid.
3. *WseAuthenticationException*: Generated when the authentication information in the message provided by user is invalid.
4. *WsePermissionException*: Generated when a user does not have sufficient permissions to perform the required operation.

6.7 PS-SWIF Framework Interface

The PS-SWIF Web interface is developed using Java Server Pages (JSP) [151] that pass requests to the Servlet container, i.e. Tomcat. This server accepts the incoming Servlet as JSP requests and processes handles and responds to these requests. The JSP pages are simply an interface between the user and the background system. A JSP page lets the user enter topics and subscription detail and then passes this information on to the PS-SWIF system.

The PS-SWIF GUI interface facilitates user interaction with the other system components in the architecture. Users access the system through a user interface;

Web Services interfaces. The user must register first to be allowed to use the PS-SWIF system. Once the user logs in to the system, the available topics and subscriptions are uploaded.

Figure 6.5 shows the PS-SWIF framework interface which provides publishing and subscription tools to achieve Workflow Interoperability. A set of graphical components is included in the system to make access to interoperability functions easier for non-technical users. The application interface is divided into three panels: The **Publish Topic panel,** the **Create Subscription panel** and the **Manage Subscriptions** panel.

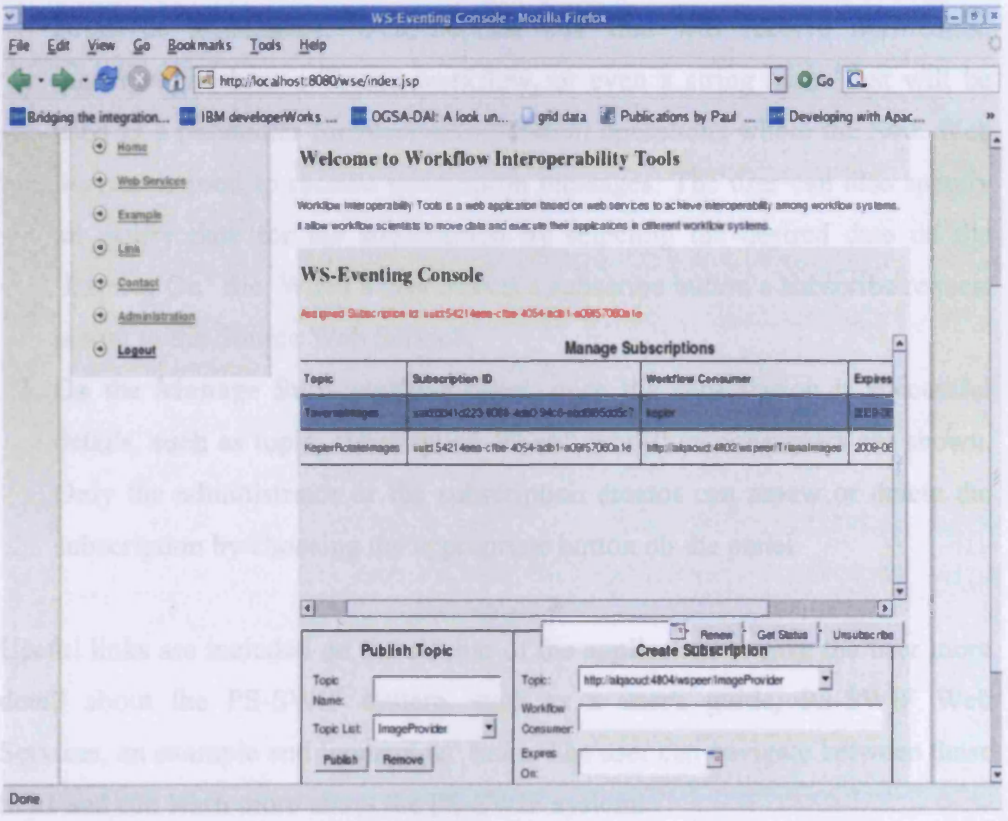


Figure 6.5: The PS-SWIF Framework Interface

1. The **Publish Topic** panel allows the user to publish a new topic to be used as a Source Web Service. Users need only enter a title for this topic. The

system checks whether the topic exists in the topic database to prevent any duplication of the topic. Once published the topic is saved in the published topic database, a WSDL file is generated that represents the Source Web Service and the WSDL file can be access through http://swif.cs.cf.ac.uk:8080/ . The topic will be available for use in the Topic combo box in the **create subscription** panel. Only the administrator or the topic publisher can remove this topic by selecting the remove button in the publish panel. The system also then deletes the topic from the topic published database.

2. The **Create Subscription** panel allows the user to select a topic from the Topic combo box and enter a value for the workflow consumer. The value could be a standard Web Service link that will receive notification messages, such as a Triana workflow, or even a string name that will be used as a parameter for *receiveNotification* operations where the Sink Web Service is used to receive notification messages. The user can also specify an expiry date for the subscription by selecting the desired date on the 'Expires On' file. When a user selects a subscribe button a subscribe request is sent to the Source Web Service.
3. On the **Manage Subscriptions** panel; once the subscription is successful details, such as topic, subscription ID and workflow consumers are shown. Only the administrator or the subscription creator can renew or delete the subscription by choosing the appropriate button on the panel.

Useful links are included on the sidebar of the application to give the user more detail about the PS-SWIF system, such as a user's guide, PS-SWIF Web Services, an example and 'contact us' links. The user can navigate between these links and can learn more about the PS-SWIF system.

6.8 Availability

PS-SWIF Web Services are hosted at Cardiff University at http://swif.cs.cf.ac.uk:8080/. Information on how to use and invoke these services is available at [152]. Users should have their own workflow systems

installed on their machines and should be familiar with how to invoke a Web Service on their workflow systems.

6.9 Summary

In this Chapter, the key PS-SWIF services implemented by WSPeer, to provide workflow interoperability among different workflow systems are discussed. These include:

- Publish Topic Web Services, which is used by a system to create a topic for workflow publishers.
- The Source Web Service that represents the event source service provided by WS-Eventing to receive subscription requests, to create response messages and, to delegate the delivery of the notification of publish Information.
- A Publish Information Web Service is used to send notification messages from a workflow publisher to workflow subscribers.
- The Subscriber Web Service, which acts as a service requester, sends messages to a Source Web Service on behalf of the Sink Web Service, with *subscribe*, *renew*, *unsubscribe* and *getStatus* requests.
- The Subscription Manager Web Service manages the subscription requests, such as *renew*, *unsubscribe* and *getStatus* requests.
- The Sink Web Service is used to receive notification message for workflow products that do not have the capability of deploying a workflow as a Web Service.

Also in this Chapter descriptions for databases used by the PS-SWIF system are discussed. The system uses the Publish Topic database to store publish topics generated by the Publish Web Services. The system also uses the subscription database to store subscription details made by users for specific topics. The user database is created to store user details when first logged into the system to specify the user identity wanting to remove topics or subscriptions.

Lastly the PS-SWIF framework interface and the user-friendly interface are presented in this Chapter. The application provides functions and tools that allow users to publish topics and subscribe for receiving notification messages in their workflow system.

CHAPTER 7

Case Study

In this Chapter, the author applied some scenarios that used real workflow examples to validate the PS-SWIF approach and system. In this Example we use Triana, Taverna, and Kepler workflow systems to show how interoperability can be achieved among different workflow systems.

7.1 Overview

A Taverna workflow is used as a Source Web Service generating notification messages. The Workflow example used in Taverna is taken from the myExperiment Web site [153]. This example was originally designed for Taverna workflow to show the use of The European Molecular Biology Open Software Suite (EMBOSS) [154] based Soaplab services [110]. This workflow involves a set of services provided by Soaplab and all these services take a special string as input values and then process to produce a result. We modified this workflow to adapt it to a multi-workflow example that linked Kepler to Taverna, then finally to Triana. At a certain stage of the Taverna workflow we linked in our sendNotification service to notify (send) the output data of one of the soaplab services to another subscribed workflow (Kepler workflow in this case). Then, the remainder of the Taverna workflow is executed, which results in the generation of a PNG file. This file is then passed to a listening Triana workflow, using asynchronous notification, and then further post-processed before being presented to the user.

This example, although fictitious, is highly representative of the type of environment that a multi-workflow scenario can create, taking pre-existing Workflows and connecting these together to make the most of the features of each system and encouraging their reuse, whilst decoupling processes and promoting collaboration. In our example, the Kepler workflow can act as Sink Web Services and Source Web Services. It acts as a Sink Web Service when the Kepler workflow is subscribed to the Taverna workflow and invokes a Sink Web Service. The Notification message is received by the Sink Web Service in the Kepler workflow and then passes it to other Soaplab services invoked by the Kepler workflow to produce a PNG Image. Moreover, the Kepler workflow acts as a Source Web Service when invoking the Publish Information Web Service and uses the SendNotification operation to send the PNG images to any subscribed workflow (Triana workflow in this case). The Triana workflow is used as a Sink Web Service that subscribes to the Kepler workflow, and then when an event has occurred in the Kepler workflow, the PNG image is received by the Triana workflow, which performs image post-processing and displays the results to the user. The following Sections describe how to use the system in more detail.

7.2 Publish Topics

A new service for Taverna workflow, called *SequenceProvider*, is published using the Publish Topic panel. A new Source Web Service is then automatically generated and will be available in the Topic list in the Create Subscription panel. In a similar way another service is created for the Kepler workflow and called *Kepler*.

7.3 Create Subscription

The first subscription is to subscribe the Kepler workflow to the Taverna workflow. The Create Subscription panel is used to create the subscriptions. To make the first subscription, the *SequenceProvider* topic is selected as the source services to represent the Taverna workflow and a new value, called *kepler*, is

inserted as the Workflow Consumer field to represents the Kepler workflow. This value (*kepler*) must be used as the second parameter for the receiveNotification operation of the Sink Web Service to receive notification messages. The second subscription is to subscribe the Triana workflow to the Kepler workflow. The *Kepler* topic is selected as source service to represent the Kepler workflow and a new value, *http://localhost:4802/wspeer/trianaImage* is inserted as the Workflow Consumer field to represent the constructed workflow in Triana. See Triana Workflow in Section 7.6.

7.4 Taverna Workflow

Figure 7.1 shows the Taverna workflow which involved invocation of the Publish Information Web Service available at [155] and uses a number of services provided by Soaplab (*seqret*, *emma*, and *prophet*). This Figure shows a sequence set being fetched using the *seqret* tool, then simultaneously scanned for predicted transmembrane regions and subjected to a multiple alignment using the *emma* service (calculates the multiple alignment of nucleic acid or protein sequences). This alignment is then plotted to a set of PNG images and also used to build a profile using *prophet* tools.

The SendNotification operation of the Publish Information Web Service is used to send the notification messages to any subscribed workflow. The sendNotification operation takes two parameters and *SequenceProvider* is specified for the first parameter that represents the Taverna workflow, with the second parameter representing the notification message that must be sent to all subscribers. Any services subscribed (Kepler workflow in this case) to *SequenceProvider* will receive the PNG image once the Taverna workflow is executed.

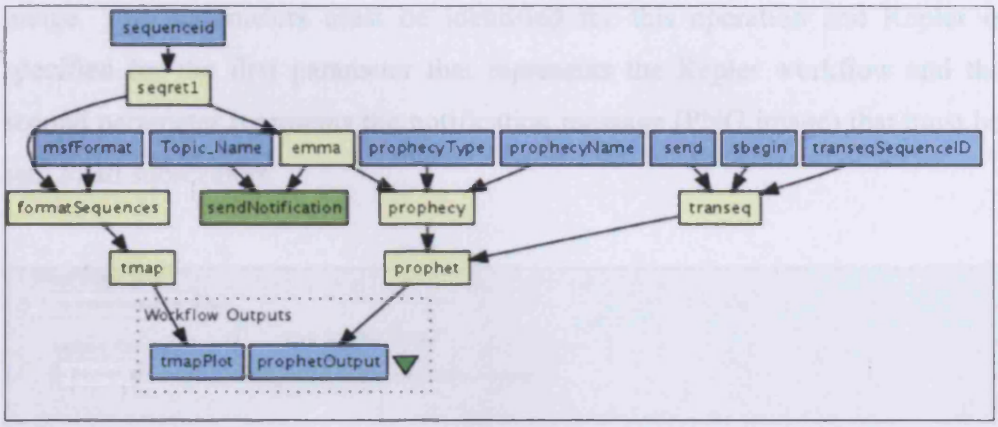


Figure 7.1: Taverna Workflow

7.5 Kepler Workflow

Figure 7.2 shows the Kepler workflow for the invocation of the Sink Web Service and Publish Information Web Service. The receiveNotification operation of the Sink Web Service receives notification messages from the Taverna Workflow. The message is the output of the *emma* service in the Taverna Workflow.

Two parameters for this operation must be specified. The first parameter represents the Source Web Service that messages come from, and, in this case, *SequenceProvider* is specified. The second parameter represents the workflow consumer that was specified when the subscription was made, and, in this case, *kepler* was specified. Moreover, this workflow involves invocation of a number of Soaplab service namely: (1) *SoaplabChooseOperation*; (2) Soaplab-*ServiceStarter*; (3) *SoaplabAnalysis*; and (4) *SoaplabChooseResultType*. The *SoaplabChooseOperation* takes the output result from the receiveNotification operation and passes it to other soaplab services, which use the *Prettyplot* service to draw a plot of the input sequence alignment. The sequences are rendered and formatted for the specified graphics device and a PNG image is created and presented to the user.

The sendNotification operation of the Publish Information Web Service is used to send the PNG image to the Triana workflow for another process with the

image. Two parameters must be identified for this operation and Kepler is specified for the first parameter that represents the Kepler workflow and the second parameter represents the notification message (PNG image) that must be sent to all subscribers.

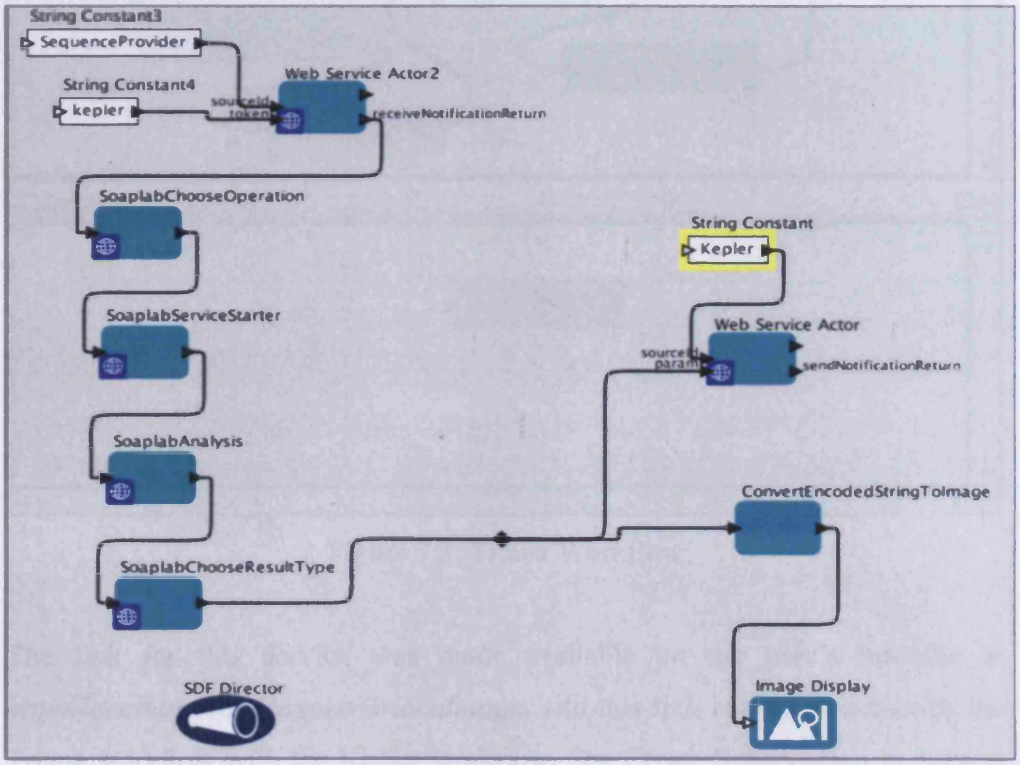


Figure 7.2: Kepler Workflow

7.6 Triana Workflow

Figure 7.3 shows the Triana workflow which includes four tasks used to construct the workflow: 1. *ImageDecoder*, to allow a Web Service to receive encoding images and convert to a standard image; 2. *RotateLeft*, to rotate the received image by 90° to the left; 3. *Snb*, to set the value of a pixel to the value of the smallest 8 way connected pixel; and 4. *ImageListView*, to display any images in a separate window. Because the Triana workflow system provides a facility to deploy a user workflow as a fully-functional Web Service, one does not need to invoke the predefined Sink Web Service here. Instead, these tasks

are grouped by a Triana tool, called *trianaImages* and then deployed as a Sink Web Service, which is subsequently invoked directly by Kepler.

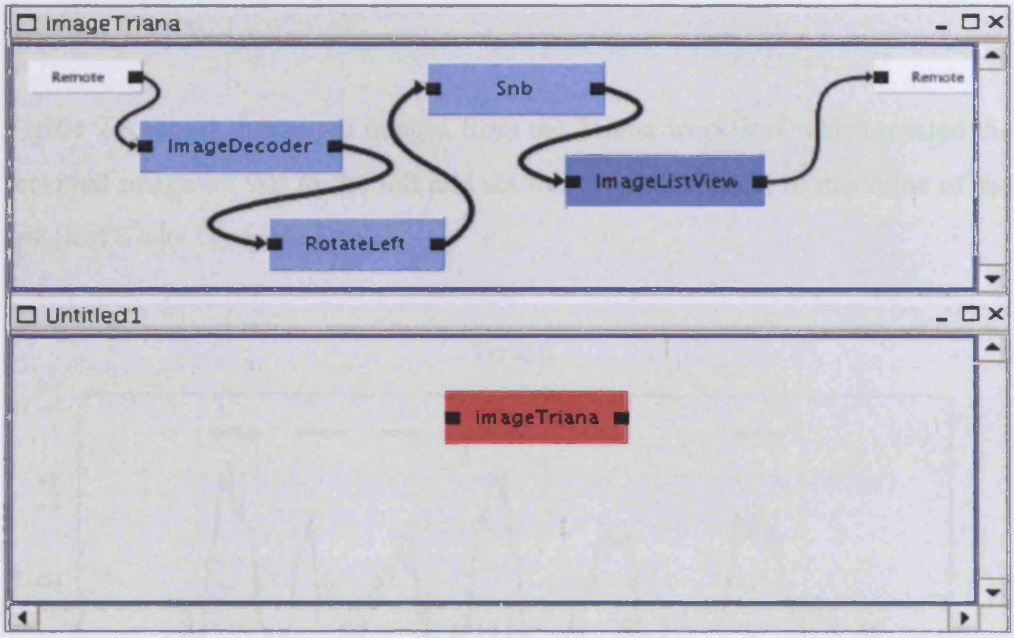


Figure 7.3: Triana Workflow

The link for this service was made available on the user's machine at *http://localhost:4802/wspeer/trianaImages* and this link is used to subscribe the Triana workflow with the Kepler workflow. See Create Subscription in Section 7.3.

7.7 Output Result

Finally, the Kepler and Taverna workflows must be executed to obtain the result. Figure 7.4 shows the output results from the execution of the Taverna workflow especially from the tmap processor which predicts transmembrane segments for an aligned set of protein sequences. A plot of propensities to form the middle (solid line) and the end (dashed line) of transmembrane regions is given as output. Bars are displayed in the plot above the regions predicted as being most likely to form transmembrane regions.

Figure 7.5 shows the result images from the Kepler workflow. The result is the output of the prettyplot service which draws a plot of the input sequence alignment. The sequences are rendered in pretty formatting on the specified graphics device.

Figure 7.6 shows the output images from the Triana workflow which rotated the received image by 90° to the left and set the value of a pixel to the value of the smallest 8 way connected pixel.

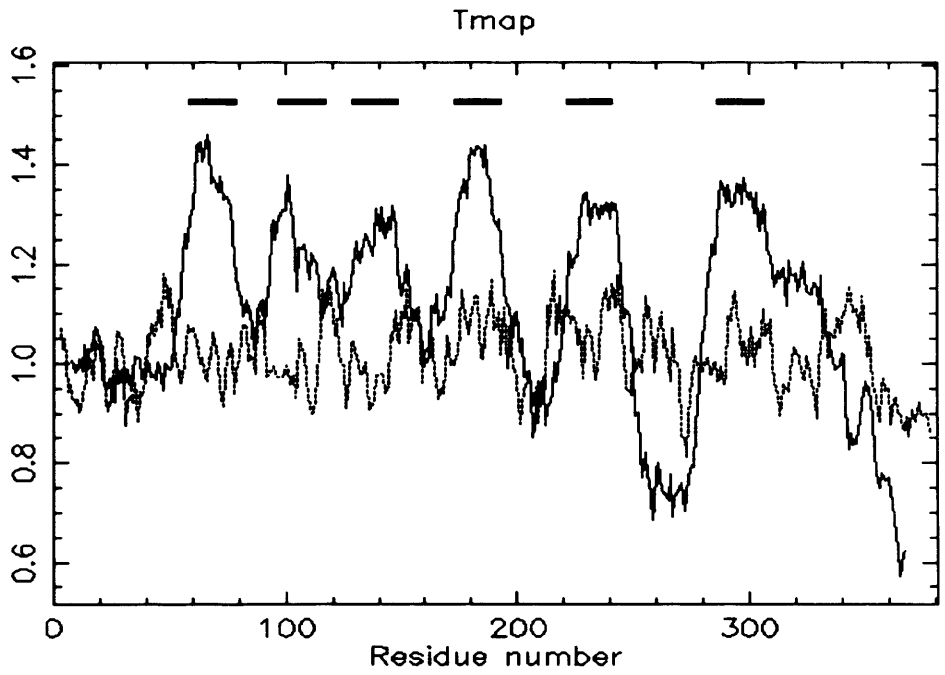


Figure 7.4: The Taverna Output Result

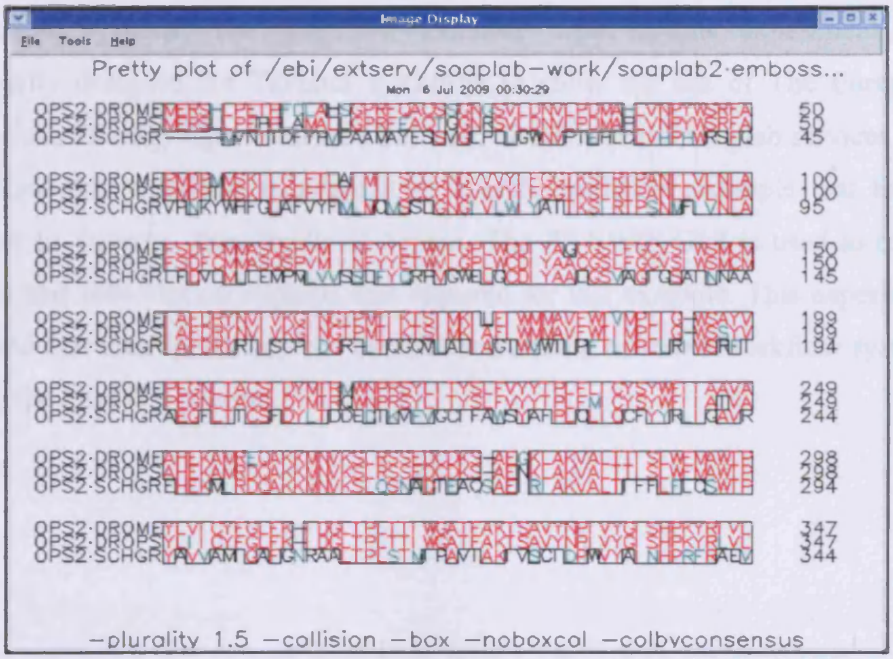


Figure 7.5: The Kepler Output Result

Figure 7.6: The Triana Output Result

7.8 Summary

In this Chapter, the author showed a real example that can be conducted using the PS-SWIF system to achieved workflow interoperability among different

workflow systems. The workflow example used in this experiment was originally designed for Taverna workflow to show the use of The European Molecular Biology Open Software Suite (EMBOSS) based Soaplab services. We modified this workflow to adapt it to a multi-workflow example that linked Kepler to Taverna, then finally to Triana. The PS-SWIF GUI is used to create topics and subscription requests that required for this example. This experiment showed that interoperability can be achieved among different workflow systems using the PS-SWIF system.

CHAPTER 8

Evaluation

This Chapter evaluates the PS-SWIF approach and its system to achieve workflow interoperability among workflow systems.

Evaluation can normally be conducted in two ways; a qualitative approach or a quantitative approach. The qualitative approach requires technical and professional expertise to judge products, while the quantitative approach requires a statistical and numerical analysis to judge the product.

To evaluate the PS-SWIF system, which presents a novel approach in this thesis, it is more appropriate to use a quantitative approach. This is because, the PS-SWIF approach enables a new paradigm for cross-workflow scenarios and therefore contributes in a qualitative way to the community. Moreover, for this to be conclusively proved this requires that the PS-SWIF approach should be recognized by a specialized organization in this field, such as OASIS [156] or OGF for standardization , or used in a number of real life projects. Both options are not possible due to a time limitation on this project.

Therefore, in this thesis, the author has taken a two pronged strategies for evaluation. First, the author has provide in the previous Chapter that different workflow engines can use the PS-SWIF approach to qualitatively improve their capabilities by accessing different workflows from third party systems without internal modification. This result shows the PS-SWIF proof of concept facilitates

a qualitative difference, which could form the basis for futures standardization of the approach in OGF or similar.

Second, the author presents quantitative measurements that show that PS-SWIF can scale in a workflow heterogeneous distributed environment.

The PS-SWIF system therefore is evaluated here using a quantitative approach as follows:

- Assessing the scalability of the system in terms of notification message loads using large number of machines and different sizes of datasets the system can manage;
- Assessing its ability to satisfy all interoperability models provided by WfMC, which shows the flexibility of the PS-SWIF system in terms of variety of workflow systems, supported using different environments.

8.1 Workflow Interoperability Evaluation

This section evaluates the PS-SWIF system to achieve workflow interoperability using Web Services with asynchronous notification messages represented by WS-Eventing standard. This experiment covers different types of communication models provided by WfMC presented in Chapter 5. These models are: Chained processes, Nested synchronous sub-processes, Event synchronous sub-processes, and Nested sub-processes (Polling/Deferred Synchronous). Also, this experiment shows the flexibility and simplicity of the PS-SWIF approach when applied to a variety of workflow systems (Triana, Taverna, Kepler) in local and remote environments.

8.1.1 Experimental Hypotheses

This section presents and explains the experiment hypotheses and how these hypotheses meet the overall hypotheses of this thesis.

1- The experiment involves three different workflow systems, namely Triana, Kepler, and Taverna that run in three different machines to show that the PS-SWIF approach can be applied to different workflow systems that run in remote environments. Moreover, the experiment also involves two different workflow systems, namely Triana and Kepler, to show that the PS-SWIF approach can be applied to different workflow systems that run in a local environment. Choosing the order of running these workflow systems is arbitrary and the experiment can be run in any order.

2- The Experiment uses the PS-SWIF application to mange the exchanging of data between different workflow systems. Four topics are created, namely *Test3M* for Triana workflow run on machine M1, *Test3M_Tavern* for Taverna workflow run on machine M2, *Test3M_Kepler* for Kepler workflow run on machine M1, and *Test3M_Triana* for Triana workflow run on machine M3. Six subscription requests are made: Tavern workflow on M2, Kepler workflow on M1 and Triana workflow on M3 are subscribed to *Test3M* topic which represents the Triana workflow on M1. The Kepler workflow on M1 subscribed to *Test3M_Taverna* topic which represents the Taverna workflow on M2. The Triana workflow on M3 subscribed to the *Test3M_Kepler* topic which represents the Kepler workflow on M1. The Taverna workflow on M2 subscribed to the *Test3M_Triana* topic which represents the Triana workflow on M3. The experiment shows the ability of the system to mange the data through using the PS-SWIF application. The exchange of data depends on these subscriptions and without these subscriptions their data cannot be exchanged between these workflow systems. Moreover, the PS-SWIF allows users to unsubscribe or renew the subscription. These options are considered to be part of managing the data.

3- To prove the ability of the system to control communication between different workflow systems the experiment involves invoking the PS-SWIF Web Services 8 times:

1. The *sendNotification* operation of the Publish Information Web Service is invoked on Triana workflow on M1 to send notification messages to Tavern workflow on M2, Kepler workflow on M1 and Triana workflow on M3.
2. The *receiveNotification* operation the Sink Web Service receives is invoked on Taverna workflow on M2 to receive the notification message from Triana workflow on M1.
3. The *sendNotification* operation of the Publish Information Web Service is invoked on Taverna workflow on M2 to send a notification message to Kepler workflow on M1.
4. The *receiveNotification* operation the Sink Web Service receives is invoked on Taverna workflow on M2 to receive the notification message from Triana workflow on M3.
5. The *sendNotification* operation of the Publish Information Web Service is invoked on Triana workflow on M3 to send a notification message to Taverna workflow on M2. The Triana workflow system support deploys a workflow as a web service, so the Triana workflow will receive the notification message once a subscription is made without the need to invoke the Receive Notification operation of the Sink Web Service.
6. The *receiveNotification* operation the Sink Web Service receives is invoked on Kepler workflow on M1 to receive the notification message from Triana workflow on M1.
7. The *receiveNotification* operation the Sink Web Service receives is invoked on Kepler workflow on M1 to receive the notification message from Taverna workflow on M2.

8. The *sendNotification* operation of the Publish Information Web Service is invoked on Kepler workflow on M1 to send a notification message to Triana workflow on M3.

The control communication between these workflow systems is achieved through invoking the PS-SWIF Web Services at the appropriate stage. Moreover, the invoking of these web services is not arbitrary. They are invoked in order to satisfy and fitful the requirements to achieve the workflow interoperability models provided by WfMC, as explained in section 8.1.8.

8.1.2 Experiment design

Figure 8.1 shows the experiment scenario. Four workflow systems are used: two workflows (Triana and Kepler) are installed on M1, Taverna installed on M2, and Triana installed on M3.

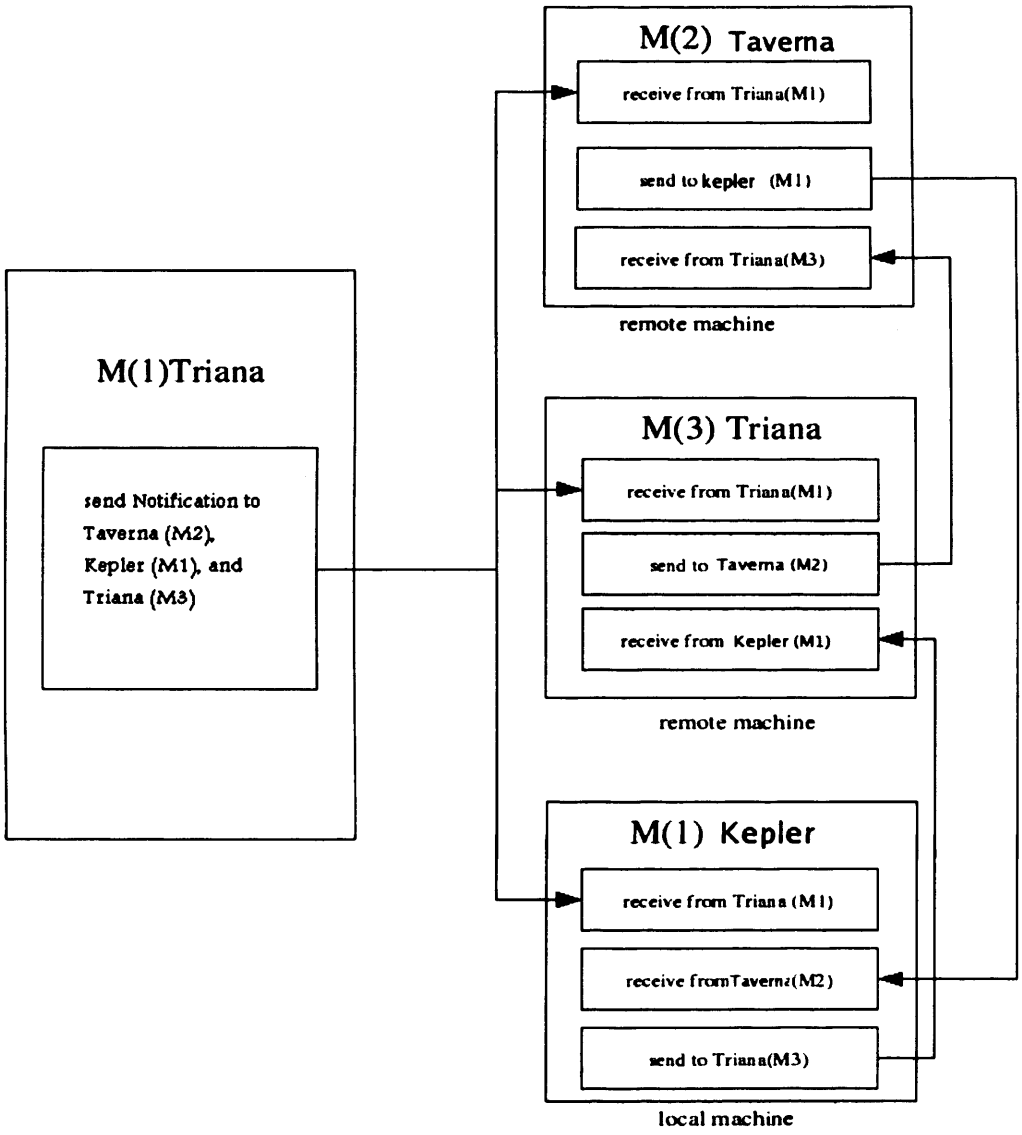


Figure 8.1: Experiment Scenario

The scenario explained:

1. Triana workflow in M1 sends a message to all subscribed workflows, namely; Kepler workflow on M1, Taverna workflow on M2 and Triana workflow on M3.

2. The Taverna workflow on M2 receives a notification message from the Triana workflow on M1. The Taverna workflow does some processing with the message received and sends it to the Kepler workflow on M1. At a later

stage, the Taverna workflow (M2) receives a message from the M3 Triana workflow.

3. The Triana workflow on M3 receives a notification message from Triana workflow on M1. The Triana workflow does some processing with the message received and sends it to Taverna workflow on M2. At a later stage, the Triana workflow on M3 receives a message from Kepler workflow on M1.
4. The Kepler workflow on M1 receives a notification message from Triana workflow on M1. The Kepler workflow does some processing with the received message. At a later stage the Kepler workflow on M1 receives a message from the Taverna workflow on M2 and also does more processing with the received message and then sends it to the Triana workflow on M3.

8.1.3 Test-bed

The test-bed for the experiments includes three machines: the first machine M1 is the same machine M(S) used for the performance experiment section. The other two machines M2 and M3 have similar specifications with a 3.2 Ghz Intel(R) Pentium(R) processor and 1 GB of memory, Fedora 7 as operating system, and Java version 1.6.0.14. All machines were connected through a private Ethernet network which was not shared by other users. M1 installed Triana and Kepler workflow systems, M2 installed a Taverna workflow system and M3 installed the Triana workflow system.

8.1.4 Triana Workflow (M1)

Figure 8.2 shows the Triana workflow that runs on M1, as used to send a message to other workflows. Five tools are used to construct this workflow and Table 8.1 provides a description for each tool. The main tool in this workflow is the *sendNotification* tool which represents the operation of the Publish Information Web Service. The Publish Information Web Service is invoked in Triana using service tools to send message to any subscribed workflows.

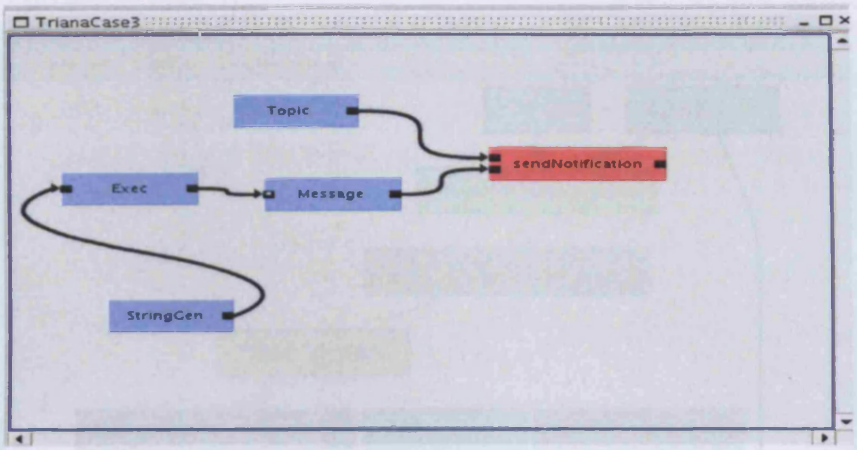


Figure 8.2: Triana Workflow on M1

Triana Tool	Description
sendNotification	An operation of Publish Information Web Service used to send messages to subscribed workflows.
Topic	A tool used to specify the topic name for sendNotification tool
Message	A string value that should be sent by sendNotification tool
Exec	A tool used to execute storetime.sh scrip to store the time.
StringGen	A string unit required to execute the Exec unit.

Table 8.1: Triana Units Description on M1

8.1.5 Taverna Workflow (M2)

Figure 8.3 shows the Taverna workflow on remote machine M2. There are 14 components used to construct this workflow and Table 8.2 gives a brief description for each component. The main components of this workflow are *receiveNotification*, *sendNotification*, and *receiveTrianaNotification* components. The *receiveNotification* is an operation of the Sink Web Service and used to receive a notification message from Triana Workflow on M1. The *sendNotification* is an operation of the Publish Information Web Service and used to send a message to the Kepler workflow on M1. The *receiveTrianaNotification* is an operation of the Sink Web Service to receive a notification message from the Triana workflow on M3. (The original name for this operation is receiveNotification but changed here to distinguish it from the previous operation used earlier in this workflow).

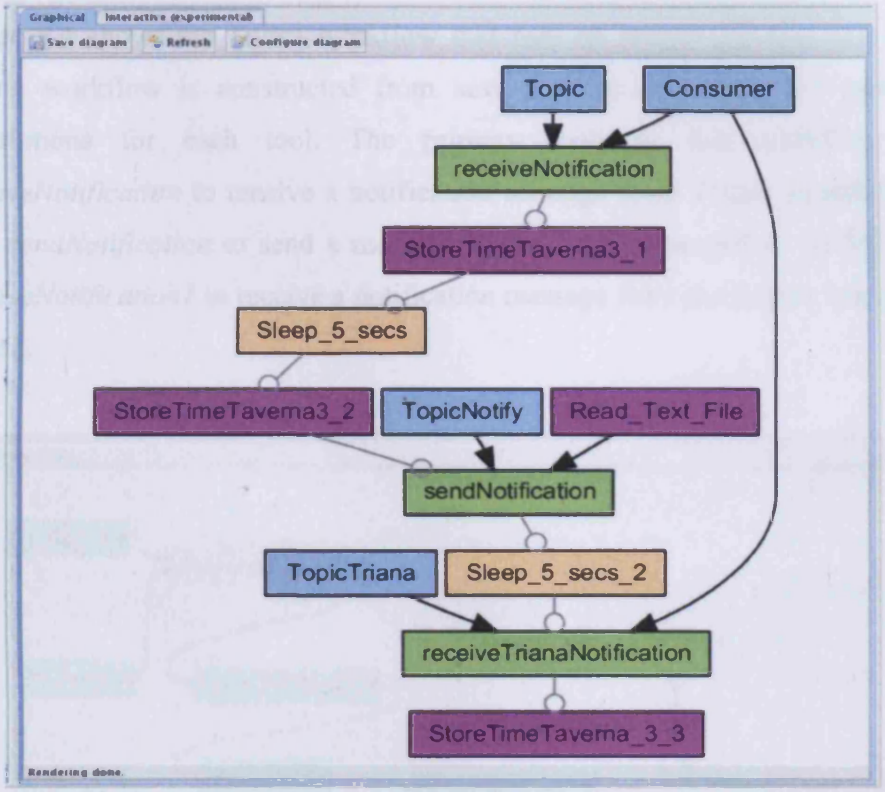


Figure 8.3: Taverna Workflow on M2

Taverna Component	Description
Topic	A component used to specify the topic name for receiveNotification component
Consumer	A component to specify a consumer workflow
receiveNotification	An operation of Sink Web Service used to receive a notification message
StoreTimeTaverna3_1	Store time when the notification message is received
Sleep_5_secs	Used for a process, sleep 5 second thereafter
StoreTimeTaverna3_2	Store time when message is sent by sendNotification operation
TopicNotify	Component used to specify the topic name for sendNotification component
Read_Text_File	Component is used to read a text file
sendNotification	An operation of Publish Information Web Service used to send a notification message
TopicTriana	Component used to specify the topic name for receiveTrianaNotification component
Sleep_5_secs_2	Used for a process, sleep 5 seconds thereafter
receiveTrianaNotification	An operation of Sink Web Service that used to receive a notification message
StoreTimeTaverna_3_3	Store time when the message is received by receiveTrianaNotification components

Table 8.2: Taverna Components

8.1.6 Triana Workflow (M3)

Figure 8.4 shows the Triana workflow that runs on remote machine M3. The Triana workflow is constructed from several tools and Table 8.3 presents descriptions for each tool. The primary tools in this workflow are *receiveNotification* to receive a notification message from Triana workflow on M1, *sendNotification* to send a message to the Taverna workflow on M2 and *receiveNotification1* to receive a notification message from the Kepler workflow on M1.

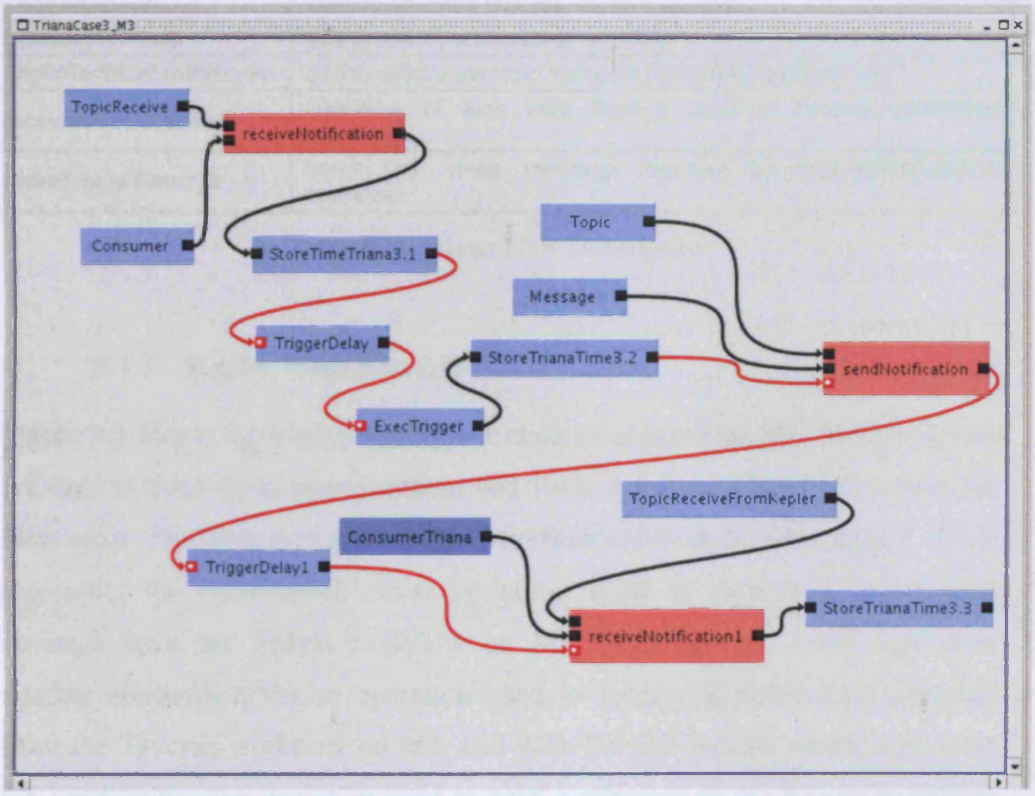


Figure 8.4: Triana Workflow on M3

Triana Unit	Description
TopicReceive	A unit used to specify the topic name for receiveNotification tool
receiveNotification	Operation of Sink Web Service used to receive notification message
Consumer	A unit to specify a consumer workflow
StoretimeTriana3_1	Store time when the message is received by receiveNotification unit
TriggerDelay	Used to do some process, sleep 5 second
ExecTrigger	String unit is used to execute the storetime.sh in the StoreTrianaTime3_2
StoreTrianaTime3_2	Store time when the message is sent by sendNotification operation
Message	String value that should be sent by sendNotification tool
Topic	Unit used to specify the topic name for sendNotification tool
sendNotification	Operation of Publish Information Web Service used to send a notification message
TriggerDelay2	Used to do some process, sleep 5 second
ConsumerTriana	Unit to specify a consumer workflow
TopicReceiveFromKepler	Unit to specify the topic name for receiveNotification1 tool
receviceNotification1	Operation of Sink Web Service used to receive notification message
StoreTrianaTime3_2	Store time when message received by receiveNotification1 operation

Table 8.3: Triana Unit Description

8.1.7 Kepler Workflow M1

Figure 8.5 shows the Kepler workflow run on local machine M1. Various actors are used to build the Kepler workflow and Table 8.4 gives a brief description for each actor. The primary actors in this workflow are Web Service Actor2 which represents the *receiveNotification* operation used to receive a notification message from the Triana workflow on M1, Web Service Actor represents another *receiveNotification* operation used to receive a notification message from the Taverna workflow on M2, and Web Service Actor3 which represents the *sendNotification* operation used to send a message to the Kepler workflow on M1.

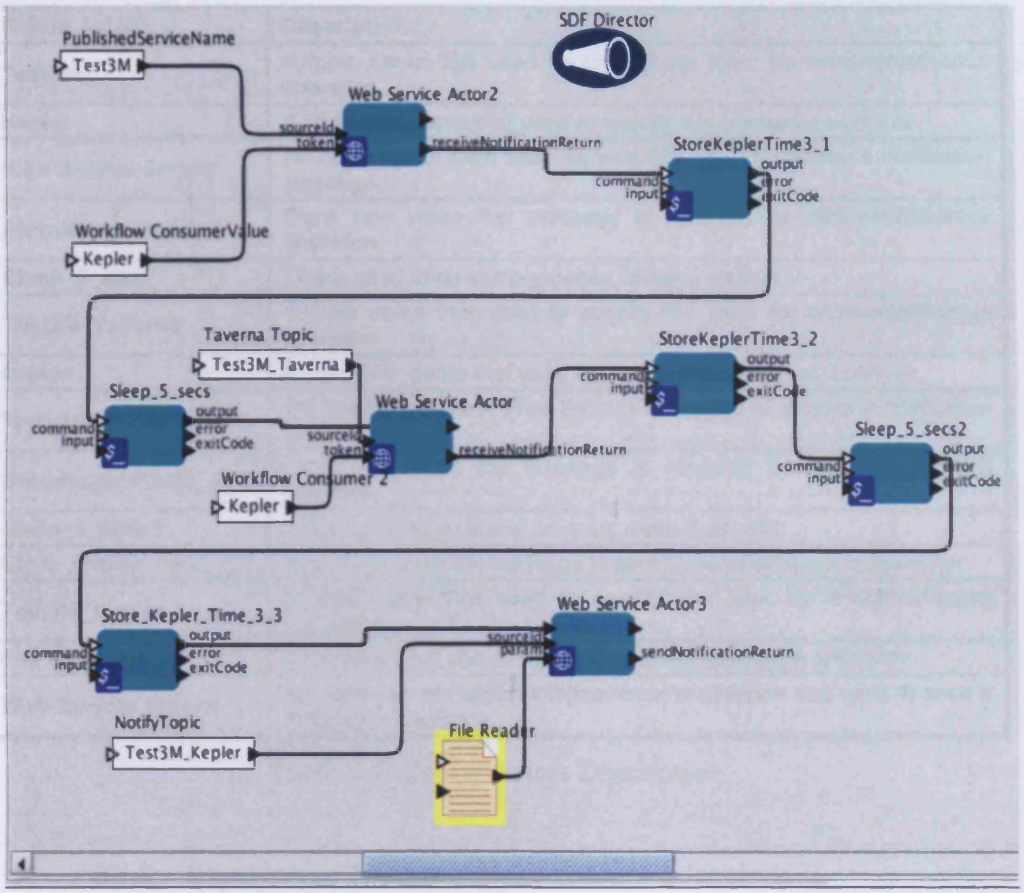


Figure 8.5: Kepler Workflow

Kepler Actor	Description
Test3M	A topic name that used to specify the topic for receiveNotification operation
Kepler	A consumer name that used to specify the consumer workflow
Web Service Actor2	An operation of Sink Web Service that used to receive a notification message
StoreKeplerTime3_1	Store time when the message is received by receiveNotification operation
Sleep_5_secs	This is used to do some process, sleep 5 second
Test3M_Taverna	A topic name that used to specify the topic for receiveNotification operation
kepler	A consumer name that used to specify the consumer workflow
Web Service Actor	An operation of Sink Web Service that used to receive a notification message
StoreKeplerTime3_2	Store time when the message is received by receiveNotification operation
Sleep_5_secs2	This is used to do some process, sleep 5 second
Store_Kepler_time_3_3	Store time when the message is sent by sendNotification operation
Test3M_Kepler	A topic name that used to specify the topic for sendNotification operation
File Reader	A message that should be send by sendNotification operation
Web Service Actor3	An operation of Publish Information Web Service that used to send a notification message

Table 8.4: Kepler Actors Description

8.1.8 Experiment Process

The experiment can be run in any order, no matter which workflow runs first. If the message is sent by the workflow publisher and there is no one to receive it, the message will be held in a queue until pulled by a workflow subscriber. If the workflow subscriber executes first and there is no notification message at this time, the workflow subscriber keeps listening until the notification message arrives.

The only aspect affected by the execution order is the time calculated between the message sent tool of workflow publisher and message tool of workflow subscriber. Taverna workflow on M2 was executed first and then Triana workflow on M3 second and Kepler workflow on M1 third, and the Triana workflow on M1 last, because the Triana workflows on M1 was the initiator of the interactions between these workflow.

The following description explains how the workflow interoperability models provided by WfMC are achieved:

1. The chained processes model is achieved when the Triana workflow on M1 use Publish Information Service to send the notification message to the other workflows.
2. The Nested synchronous sub-process and Event synchronous sub-process models are achieved when the Triana workflow uses the Publish Information Web Service on M1 to send the message to the Taverna workflow on M2. The Taverna workflow receives it through the Sink Web Service and then simulates some processing of the receive message, using the sleep 5 second component, and sends to the Kepler workflow on M1.

   The Nested synchronous sub-process and Event synchronous sub-process models assume that the notification message should be sent back to the first workflow that initiates the communication; which is the Triana workflow on M1 in this case. The PS-SWIF approach can handle this assumption easily but to avoid implementing each model in separate experiments, one experiment that covers all primary aspects of each model is used.
3. The Nested sub-process (Polling/Deferred Synchronous) model is achieved when the Triana workflow on M3 complete their processes except the *receiveNotification1* tool which waits until all other workflows 'Taverna' and 'Kepler' finish the entire workflow processes and send the notification message to the Triana workflow on M3 which explains why the Triana workflow on M3 is the last to finish execution.

8.1.9 Experiment Observation

To observe the experiment, the PS-SWIF application is used to monitor the published topic and the subscription. Figure 8.6 shows that all topics are successfully published, and all the subscriptions are successfully made. The details of these subscriptions are also shown in the same figure. Triana workflow on M1, Taverna workflow on M2, Triana workflow on M3, and Kepler workflow on

M1 successfully invoked the PS-SWIF Web Services and this is shown in Figures 8.2, 8.3, 8.4, and 8.5. The experiment was successfully executed and the data was moved among these workflow systems.

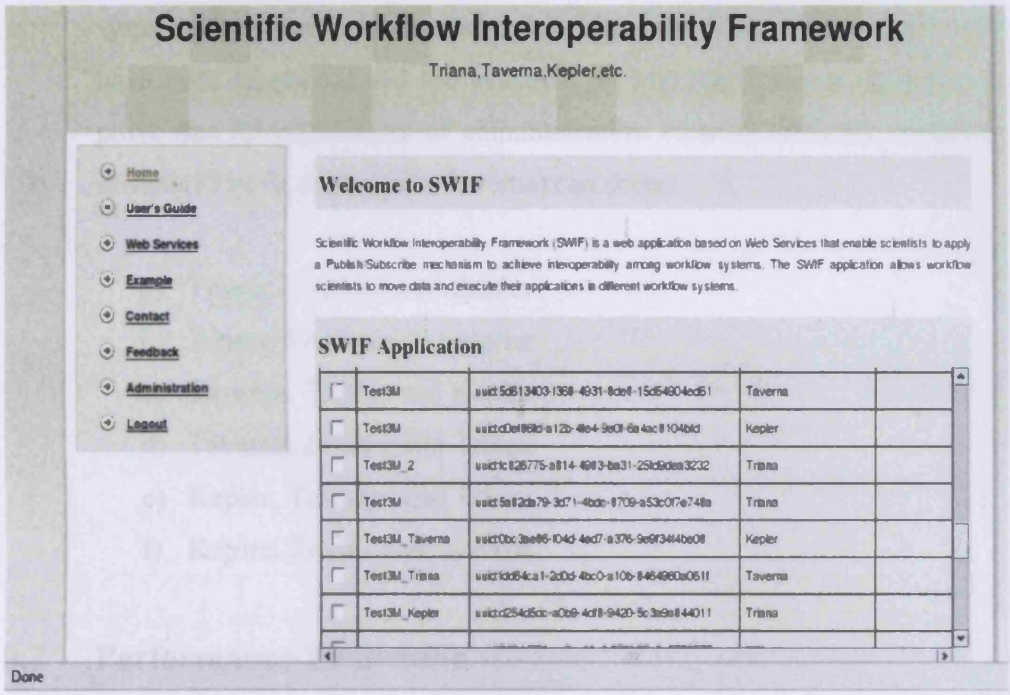


Figure 8.6: Experiment Observation

8.1.10 Experiment Achievements

1. The experiment showed how the Web Services with asynchronous notification messages can be invoked and deployed by different workflow systems, namely Triana, Taverna, and Kepler, to move and mange data between these workflow systems without modification to those systems.
2. The experiment proved that different types of communications between workflow systems can be achieved by satisfying the requirements of workflow interoperability models provided by WfMC.

3. The experiment proved the flexibility and simplicity of the PS-SWIF approach when applied to a variety of workflow systems (Triana, Taverna, Kepler) in local and remote environments.
4. This experiment provides a sophisticated example of how the system can handle different models of interoperability using different types of workflow systems. Moreover, other experiments that cover the following scenario have been conducted and are available on http://swif.cs.cf.ac.uk:8080 to prove that all possibilities of communication between different workflow systems (Tirana, Kepler, and Taverna) can occur.

    a) Tirana, Taverna, and Kepler
    b) Triana, Kepler, and Taverna
    c) Taverna, Triana, and Kepler
    d) Taverna, Kepler, and Triana
    e) Kepler, Taverna, and Triana
    f) Kepler, Triana, and Taverna

8.2 Performance Evaluation

An experiment was conducted to evaluate the performance of the PS-SWIF system with various numbers of machines and using different ranges of data size. A total of 30 tests were undertaken: for each set of machines and sets of data sizes, three tests were undertaken. Measurements of the average value for each machine with a specific data size were taken.

During such experiments, results need to be archived in a log file for each machine. The format of these file must include:

- Test ID;
- A unique identifier for each test performed;
- Total number of machines used in the test;
- Message size sent by a workflow producer to workflow subscriber;
- Duration (sec), of delivery time to transfer the notification message to the workflow subscriber.

Log files and data were then moved to the main machine for analysis. Results are gathered and presented in a meaningful, human readable form. Time is synchronized on each machine by time server "ntp0.cf.ac.uk" using network time protocol (ntp) to guarantee the notification times, between notification sender and notification receiver, are measurable in a standard way.

8.2.1 Test-bed

The test-bed for the experiments includes 30 Linux-based machines. The first machine M(S) with a 1.80 GHz Intel(R) Pentium(R) M processor, 1.5 GB of memory, Fedora 3 as Operating System, and Java version 1.6.0_07. The M(S) machine represents the host machine for the PS-SWIF Server, using a WSPeer framework, and provides deployment for all the PS-SWIF Web Services. The M(S) machine also hosts the PS-SWIF application using Apache Tomcat Version 6.0.10.

The remaining 29 machines (M1 to M29) have a 2.8 GHz Intel(R) Pentium(R) 4 CPU processor and 1.0 GB of memory, Fedora 7 as operating system, and Java version 1.6.0.14. All machines were connected through an Ethernet local-area-network with 100 Mbps.

8.2.2 Experiment Setup

A NTS (Network File System) mounted *Home* directory was created that can be accessed from each machine in the University laboratory. The content of the *Home* directory is described in Table 8.5. The test is based on the remote execution of processes using SSH, and, as this is an automated process, SSH setup is configured to skip the password prompt for each laboratory machine using the *ssh-keygen* command to generate private/public key pair. The public key is copied onto remote machines. After the SSH configuration, the remote machines can be accessed without password prompt, and the remote machines can access the centralized *Home* directory where the data and workflow are installed.

File or Directory	Description
Data folder	Store data files with different size
lib	Present a library for java classes
Workflows	Store Taverna and Kepler workflows
createsubscriptions.sh	A script file to create a subscription
generatefile.sh	A script file to create a file with different size
machines.txt	Store the Linux lab machines name
processlogs.sh	A script file to generate logs files
runmachinetest.sh	A script file used to run the Kepler workflow on lab machines
executeworkflow.sh	A script file used to run the Taverna workflow
runtest.sh	A script file used to run the experiment

Table 8.5: Home Directory Description

8.2.3 Create Topic and Subscriptions

The experiment uses Taverna Workflow and Kepler Workflow as the workflow publisher and workflow subscriber, respectively. A *TestSuite* topic is published manually using the PS-SWIF application. A new Source Web Service is automatically generated and available on M(S) machine and is also available through http://alqaoud:4804/wspeer/TestSuite?wsdl.

Subscriptions are made through the *createsubscriptions.sh* script file. This file subscribes all machine names specified in the *machines.txt* file with the *TestSuite* topic. The machine's name represents the Kepler workflow and the *TestSuite* topic represents the Taverna workflow. Once subscriptions are successfully made, subscription detail is shown on the PS-SWIF application.

8.2.4 Taverna Workflow

The Taverna workflow is constructed using a key component which involves the invocation of the Publish Information Web Service available at http://alqaoud:4804/wspeer/NotifyService?wsdl. The *SendNotification* operation of the Publish Information Web Service is used to send notification messages to any subscribed workflow. The *sendNotification* operation takes two parameters and *TestSuite* is specified for the first parameter representing the target topic, with the second parameter representing the message to be sent to all subscribers to this topic. The message is read from a file specified when the experiment is later run using *runme.sh* from the command line.

8.2.5 Kepler Workflow

The Kepler workflow is constructed using only one Web Service instance, which invokes the Sink Web Service at http://alqaoud:4804/wspeer/SinkService?wsdl. The *receiveNotification* operation of the Sink Web Service receives a message from the Taverna Workflow. Two parameters are specified for this operation. The first parameter represents the Source Web Service the message comes from, and *TestSuite* is specified. The second parameter, the workflow subscriber, specifies when the subscription is made, and, in this case, machine name will be reserved for the parameter. The received massage will be saving in an external file.

8.2.6 Experiment Execution

Every time a test is executed, a unique ID is assigned for this test in the *teseid.txt* file. The test is executed using the *runtest.sh* script file that takes two arguments:

runtes.sh datafile numberofmachines

The *datafile* represents the data size that will be passed to the workflow consumer as a notification message. The *numberofmachines* defines how many machines must be used on the test; if specified the first *n* machines defined in the *machines.txt* file will be used in the test, if not specified all machines will be used. The *runtest.sh* executes all Kepler workflows specified by machines and executes the Taverna workflow that sends notification messages to Kepler workflows.

After any number of tests with any number of machines a *CSV* (Comma-Separated Value) file [157] is generated with the integrated information from all the machines using the *processlogs.sh* script file. The output date is stored in a log file that can be opened with Microsoft Excel or an OpenOffice Spreadsheet for further analysis.

8.2.7 Experiment Analysis

Three sets of runs are conducted; each set representing a specific number of machines representing workflow subscribers, Kepler in this case. Limited to 29

machines available at the University laboratory at the time of the experiment, the three sets are divided into 10 machines, 20 machines and 29 machines. Each set consisted of three groups of three runs each for observation and analysis purposes. The three groups contained 100 Kb, 1 Mb and 10 Mb which represent a message size to be sent to the workflow subscribers. Other experiments are conducted with different message sizes, such as 0 Kb and 25 Mb, to find the overhead time and the limitation of the system respectively. Figures 8.7, 8.8 and 8.9 show performance results for three sets of experiments.

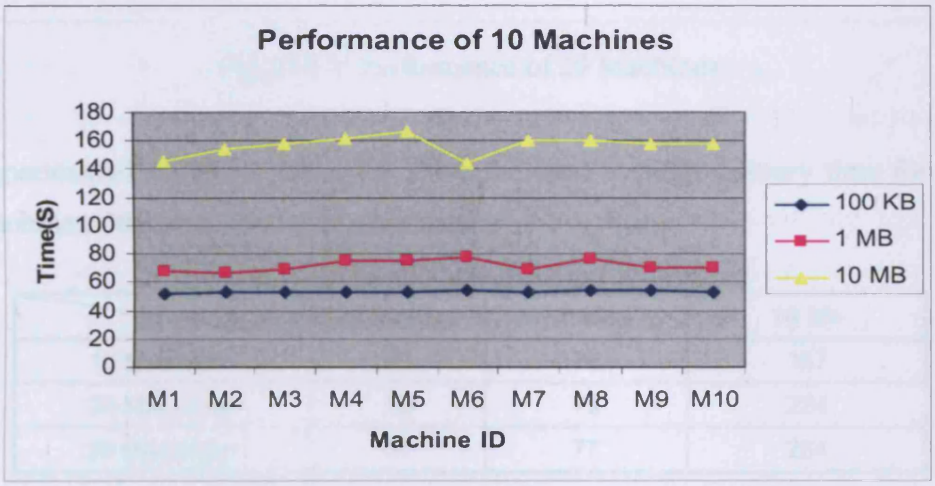


Figure 8.7: Performance of 10 Machines

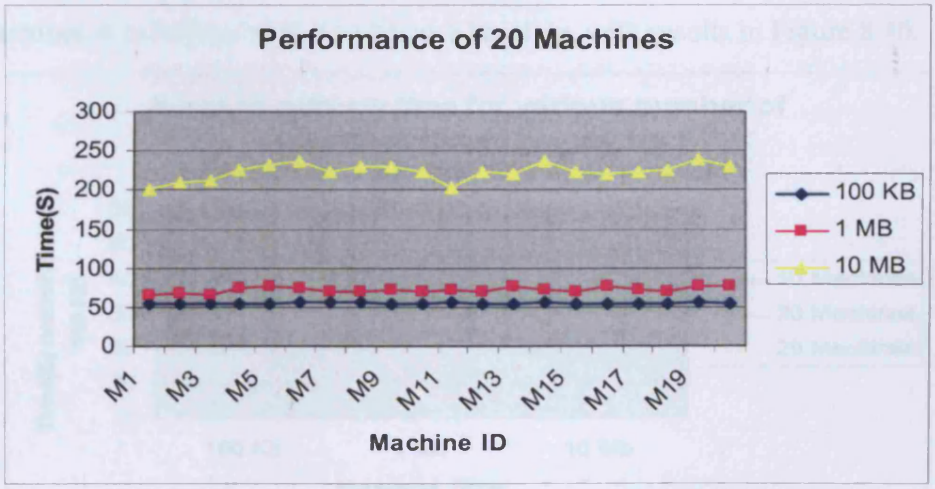


Figure 8.8: Performance of 20 Machines

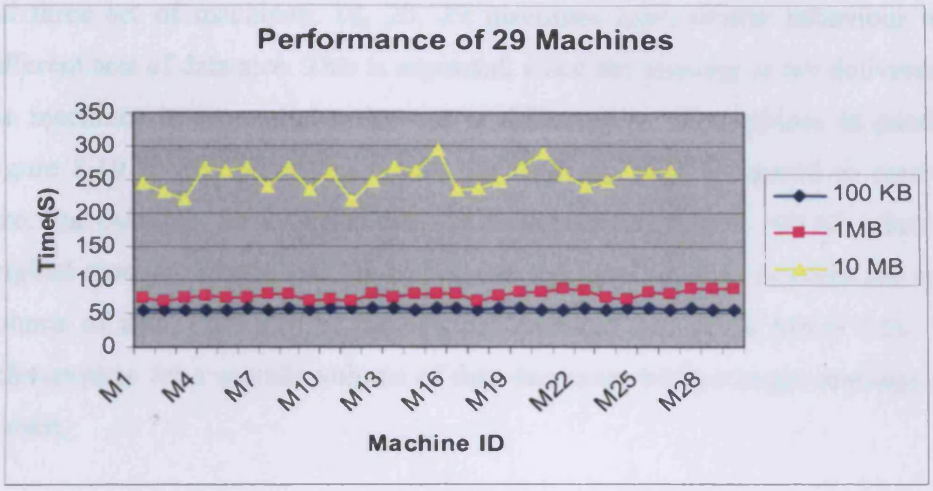


Figure 8.9: Performance of 29 Machines

Experimental results in Table 8.6 show the total average delivery time for 10 machines, 20 machines and 29 machines.

	100 Kb	1 Mb	10 Mb
10 Machines	53	72	157
20 Machines	55	72	224
29 Machines	57	77	254

Table 8.6: Average Delivery Times

The total average delivery time (obtained from Table 8.6) for various numbers of machines is calculated with 100 Kb as a baseline, with results in Figure 8.10.

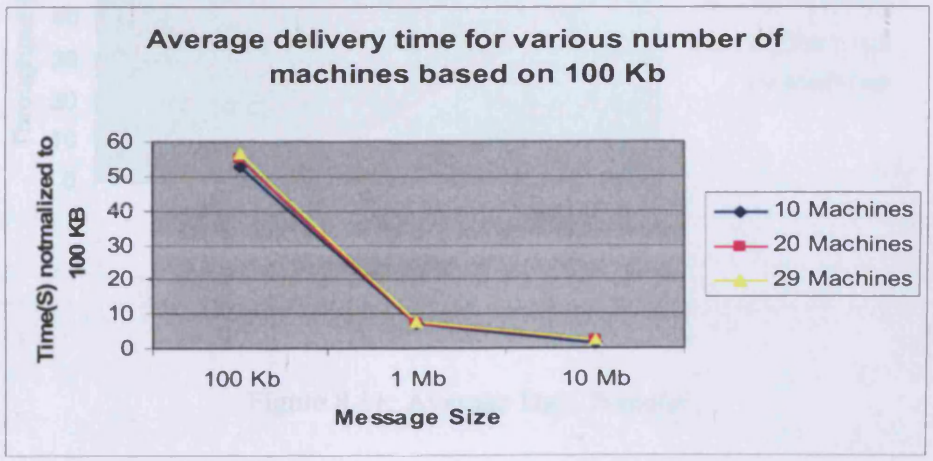


Figure 8.10: Average Delivery Time

All three set of machines; 10, 20, 29 machines have similar behaviour with different sets of data size. This is expected, since the message is not delivered to the machines in sequential order but is delivered to all machines in parallel. Figure 8.10 shows interesting results for delivery time compared to message size. For example, for 29 machines, the time taken to deliver 100 Kb when the original message size is 100 Kb is 57s and the time taken to delivery the same volume of data, (100 Kb), if the original message size of 10 Mb is 2.5s. The delivery time for a specific volume of data decreases when a larger message size is used.

Figure 8.11 shows the average data transferred for the different numbers of machines. To get the throughput in Kb/s, these values of 100, 1,000 and 10,000 are divided by the values in the columns of 100 Kb, 1 Mb and 10 Mb in Table 8.6. Figure 8.11 shows the volumes transferred increases when a large message size is used. For example, the transfer rate for 29 machines is 1.75 Kb/s when 100 Kb is used as message size, whereas the transfer rate for the same number of machines is 39 Kb/s.

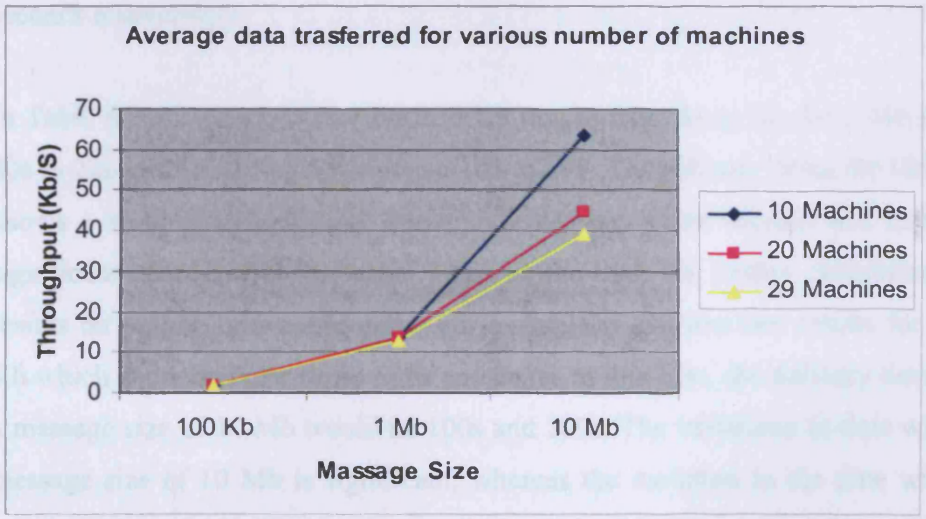


Figure 8.11: Average Data Transfer

Another experiment was conducted to find the fixed overhead time taken to deliver a message to workflow subscribers. The best way to do this is to send a message with a 0 Kb size. To measure this overhead, an empty file is created and a notification message is read from this file. This message is sent to the three different sets of machines 10, 20 and 29. The overhead average results are 52, 52 and 55 for 10, 20 and 29 machines respectively.

A further analysis subtracted the overhead times from the values presented in Table 8.6, with new values displayed in Table 8.7.

	100 Kb	1 Mb	10 Mb
10 Machines	1	20	105
20 Machines	3	20	172
29 Machines	2	22	199

Table 8.7: Average Delivery Time without Overhead Time

The expected result should be if a message with 100 Kb takes 1 second to be delivered, then a message with 1 Mb and 10 Mb should take 10 seconds and 100 seconds respectively.

In Table 8.7 the delivery time for 100 Kb ranges from 1s to 3s, for 1 Mb from 20s to 22s and for 10 Mb ranges from 105 to 199. The delivery times for 100 Kb shows a short period of time which indicates any extra second will make a significant change with this value taken as the base for further calculation of results on higher measurable units. For example if one has two results for 100 Kb which show delivery times of 1s and 2s, as in this case, the delivery time for a message size of 10 Mb would be 100s and 200s. The variations in time with a message size of 10 Mb is significant, whereas the variation in the time with a message size of 100 Kb is quite small. A reasonable explanation for variation values in Table 8.7 for 100 Kb is that the delivery time is recorded in second as the measurable unit. At the analysis stage it was recognized that if the delivery time is recorded in milliseconds, one would get more accurate results. The

results in Table 8.7 for messages 1 Mb and 10 Mb show more reliable consistent values.

To further evaluate the PS-SWIF system, another experiment is conducted to find the system limitation in terms of the maximum load of message size with maximum number of machines the system can handle before failure; essentially to find the system scalability. System limitation is defined when a message is not delivered successfully or takes a very long time to deliver. As shown in Figure 8.9 the system cannot deliver a message size of 10 Mb to more than 26 machines. Moreover, the system manages to deliver a message size of 20 Mb to only 5 machines in 204s. With a message size of 25 Mb, the system was not able to deliver the message to any machines. The reason for these failures was the memory size allocated to the PS-SWIF server that hosts the service. The maximum memory allocated to this server was 1,500 Mb as the total memory available to the machine running this experiment is 1.5 GB. Further investigating of this failure by debugging the PS-SWIF code showed that the problem is caused by WSPeer and/or Axis and not the PS-SWIF code. The current implementation of WSPeer based on the old version of Apache's Axis (1.2) which does not support the delivery of a large message. This can be solved by looking for ways to improve the handling of large messages in Axis and/or WSPeer such as SOAP MTOM (Message Transmission Optimization Mechanism) [158].

8.3 Summary

This Chapter evaluates the PS-SWIF system and approach in a number of ways. Several experiments are conducted to determine scalability of the system when applied with different numbers of machines and a large volume of data that transfers between different scientific workflow systems. The PS-SWIF system is shown to be scalable and reliable with different numbers of machines and large sizes of messages. The PS-SWIF system was not able to delivery a message to more than 26 machines with 10 Mb or any message with a size of 25 Mb. This

failure is not caused by the PS-SWIF system itself but by the current implementation of WSPeer with Axis that does not support the delivery of a large message.

Another experiment proves that the workflow interoperability models provided by WfMC can be achieved using the SWIF approach, which proves the flexibility of the PS-SWIF approach when applied to different workflow systems in different environments.

CHAPTER 9

Conclusion and Future Work

9.1 Research Summary

Scientific workflow is a special type of workflow that emerged for scientists to formalize and structure complex e-Science applications. Many workflow systems have been released to resolve problems in particular domains. In large collaborative projects, it is often necessary to recognize the heterogeneous workflow systems already in use by various partners and any potential collaboration between these systems requires workflow interoperability.

Workflow interoperability was officially addressed for the first time in 1996 by the Workflow Management Coalition (WfMC). Recently workflow interoperability has received much interest from the distributed computing community and many workshops and meetings have been organized to discuss, from different perspectives, how interoperability can be achieved among scientific Workflow Systems. Workflow interoperability is classified on different levels according to the WfMC and other community. In this research, all these classifications and specifications were considered in designing the PS-SWIF model.

Only limited or *ad-hoc* solutions have been attempted to achieve scientific workflow interoperability between e-Science workflow systems. A general approach not limited to specific types of workflow systems is needed.

Many Scientific Workflow systems use Web Service standards to invoke a remote resource or to send a job to be executed on remote resources. Publishing, discovering and availability of services are considered to be part of the composition process. A solution for workflow interoperability at the Web Service layer will facilitate the interaction between workflow systems and also cover a wide range of workflow systems.

In this thesis, the author has presented a novel approach "PS-SWIF" to achieve workflow interoperability. The PS-SWIF approach depends on a Web Service based notification messaging system to provide run-time interoperability. The PS-SWIF approach supports workflow systems written in any languages and running on different operating system.

Most workflow systems do not support the publish/subscribe model directly. The Triana workflow system, being the exception, supports publish/subscribe models by implementing a WS-Notification. The research in this thesis shows WS-Notification is not an appropriate standard to be used as a model within workflow systems to achieve workflow interoperability. WS-Notification is based on the WS-RF specification to deliver the notification messages and most current workflow systems do not support the invocation of Web Services with stateful resources via WS-RF.

Some Workflow systems have the ability to deploy a workflow as a Web Service, for example, the Triana workflow. In contrast, some other workflow systems do support deploying a workflow as a Web Service, for example, the Taverna and Kepler workflow system. In the former case, an asynchronous notification mechanism is applied with the PS-SWIF system. The advantage of this approach is that it maintains a level of decoupling that can allow the coexistence of multiple workflows without the necessity of tight integration or dependency. In the second case where the workflow system does not have the ability to deploy a workflow as a Web Service, a synchronous notification mechanism that blocks and waits for the notification message is applied with PS-SWIF.

WS-Eventing supports both asynchronous and synchronous modes to deliver messages. WS-Eventing provides required functions for the Publish/Subscribe paradigm, such as subscribe, renew, unsubscribe and getStatus. WS-Eventing does not depend on as many specifications as WS-Notification and the only specification required is WS-Addressing, which is also required by WS-Notification. In addition, the WS-Eventing service is represented as a stateless service, so it can be invoked by any workflow system as a normal Web Service. Due to these reasons and its simplicity and features, the WS-Eventing specification is used with some modification in the PS-SWIF approach. WS-Eventing uses Web Service standards to implement their entities and so does the PS-SWIF components.

The advantages of the current design and implementation of PS-SWIF is that it provides decoupling between workflow publisher and workflow subscriber in three domains:

- **Space decoupling domain**: The notification messages are delivered between the workflow publisher and workflow subscriber without the need to know each other. The workflow publisher sends notification messages through the Internal Subscription Manager and the workflow subscriber receive these notifications indirectly through the Internal Subscription Manager.
- **Time decoupling domain**: The workflow publisher and workflow subscriber can communicate with each other even if they are not active at the same time. Specifically the workflow publisher can send notification messages while the workflow subscribers are disconnected, and, in the opposite direction, the workflow subscribers can receive notification messages while the workflow publisher that generates these messages is disconnected.
- **Synchronization decoupling domain**: The workflow publisher is not blocked while generating notification messages and the workflow subscriber can asynchronously receive notification messages while performing some concurrent activity.

To validate the PS-SWIF approach, a prototype is designed and implemented based on the several requirements, gathered through the research period. The PS-SWIF

framework is designed with consideration of scientists that do not have a very strong background in computer science.

The PS-SWIF framework, through suitable, Web interface facilitates user interaction with the other system components in the architecture. It utilizes publishing and subscription tools, such as publish, subscribe and renew, to create topics and manage subscriptions. Moreover, it allows users to view details of subscriptions such as topic, subscription ID and sink.

Different experiments to construct various workflows using Triana, Taverna and Kepler workflow systems have been conducted for proof of the concept of the PS-SWIF system. The author focuses on Triana, Taverna and Kepler workflow systems because these systems are good representatives for a scientific workflow at service level where this approach is applied.

The PS-SWIF system and approach is evaluated in a number of ways. Several experiments are conducted to determine scalability of the system when applied with different numbers of machines and a large volume of data that transfers between different scientific workflow systems. The PS-SWIF system is shown scalable and reliable with different numbers of machines and large sizes of messages. The PS-SWIF system was not able to deliver a large message; not caused by the PS-SWIF code itself but by the current implementation of WSPeer with Axis that does not support the delivery of a large message.

Another experiment conducted proved that the workflow interoperability models provided by WfMC namely, Chained processes, Nested synchronous sub-process, Event synchronized sub-process, and Nested sub-process (Polling/Deferred Synchronous) can be achieved using the PS-SWIF approach, which proves the flexibility of the PS-SWIF approach when applied to different workflow systems in different environments.

9.2 Advantages of PS-SWIF System

In this section, the advantages of using the PS-SWIF approach against other similar related system in the research domain are concluded. Table 9.1 presents a brief comparison of different approaches to scientific workflow interoperability systems. These vary in terms of their strategies and levels of interoperability according to WfMC and workflow lifecycles, their generality, (either limited to specific workflow systems or could be used in any workflow systems), their simplicity of usage – does it need an expert or developer to use it or can it be used by normal scientists – their deployment platform – is the workflow system hosted remotely or locally – and whether or not the approaches support the reusability of the experiments.

Interoperability Approach	Strategy	Levels	Workflow Lifecycle Level	Generality	Simplicity Usage	Deployment Platform	Reusability
PS-SWIF	Message Passing	Complete API	Execution	General	Simple	Remotely	Yes
GEMLCA/P-GRADE	Direct Interaction	Limited API	Execution	Limited	Complex	Locally	No
VLE-WFBus	Direct Interaction	Limited API	Execution	Limited	Complex	Locally	No
IWR	Direct Interaction	Shared Formats	workflow design	Limited	Complex	Locally	No
SIMDAT(Server, Service)	Direct Interaction	Limited API	Execution	Limited	Complex	Remotely	No
SIMDAT (Language)	Direct Interaction	Shared Formats	workflow design	Limited	Complex	Remotely	No
Kepler/Pegasus	Direct Interaction	Limited API	workflow design	Limited	Complex	Locally	No

Table 9.1: Comparison of Workflow Interoperability Approaches

Table 9.1 notes the PS-SWIF approach uses a message passing strategy among different workflow systems to achieve interoperability, whereas the GEMLCA/P-GRADE, VLE-WFBus, IWR, SIMDAT (server, service and language), and Kepler/Pegasus approaches use direct interaction strategies to achieve Workflow Interoperability.

Interoperability is achieved at a Limited Common API Subset Level for GEMLCA/P-GRADE, VLE-WFBus, SIMDAT (server, service), and Kepler/Pegasus

approaches. In the IWR and SIMDAT (language translation) approaches the interoperability is achieved at Shared Definition Format Level. In the PS-SWIF approach, interoperability is achieved at a complete workflow API Level which shares a single standard API among workflow systems. Although there is no API used directly in workflow systems using the PS-SWIF approach, the Web Service technology used in the PS-SWIF approach represents the single standard API shared by workflow systems.

For interoperability, according to workflow lifecycle classification, the PS-SWIF, GEMLCA/P-GRADE, VLE-WFBus, and SIMDAT (server, service) approaches provide a solution at workflow execution level. IWR, SIMDAT (Language translation), and Kepler/Pegasus approaches present a solution at workflow design level.

PS-SWIF provides a general approach and its application can be applied to any workflow system, whereas the other approaches are limited to specific types of workflow systems. However, some approaches can support more general workflow systems, but they need some modifications to their API.

In terms of simplicity, the PS-SWIF application based on a set of Web Services makes the application easy to use, and scientists do not need any programming or deep technical computer background to use the approach, whereas the other approaches require an expert to install the software and to first set up the environments to enable use by scientists. As they implement their approaches using APIs, there are going to be a number of requirements, such as operating system and programming language, which must be satisfied before using the software. These requirements will make it more difficult to use the relevant APIs in many workflow systems.

Since the PS-SWIF, SIMDAT (server, service, and language) approaches are based on Web Services, interoperability can be achieved among workflow systems that run

remotely. In contrast, the other approaches are limited to the workflow systems which run on the users' machines.
Within the PS-SWIF approach the reusability of the same experiment or a similar experiment with different data input and parameters can be achieved without major modifications to the system, and only needs changes to the input parameters. In the other approaches more modifications are needed in the constructed workflow and in the system to allow reusability.

In general, the PS-SWIF approach is not limited to any workflow system. The PS-SWIF approach and tools can be applied to any workflow system that has the capability of invoking a Web Service. Currently, most scientific workflow systems are designed to support Web Service invocation.

As the PS-SWIF application is based on a set of Web Services available for access on the World Wide Web, interoperability can be achieved among workflow systems remotely hosted. Different or similar workflow systems, hosted anywhere on a network and using any operating system, can easily use the full range of PS-SWIF tools to interoperate with others. The PS-SWIF approach is easier for scientists and provides interoperability among a wide range of scientific workflow systems.

9.3 Future Work

This thesis has focused on interoperability among Scientific Workflow Systems at execution level. There are several more extensions possible to the PS-SWIF system. Some of these are worth exploring further. In this section, the author outlines future research directions.

- Future research can look at integrating the PS-SWIF system with the myExperiment project. myExperiment is a social Web site where scientists can safely upload their workflows and experiment plans, share them with groups and find those of others. The myExperiment project is making good progress on sharing Workflows. Most of the workflows uploaded to myExperiment are constructed by a Taverna workflow system. The

myExperiment Web site is well known in this area and has over 2,700 users, 160 groups and 850 workflows [111]. If a tool such as PS-SWIF is integrated with myExperiment, this is going to provide a complete solution for interoperability and sharing workflows. More scientists and users of different workflow systems will be encouraged to participate and publish their workflows and share with others. The users of PS-SWIF should be allowed to add a description for a topic when wanting to publish a new topic. This description should include workflow system type that should participate on this topic and also datatype to be published to this topic. Moreover, users of PS-SWIF should be allowed to upload their workflows into myExperiment, so other users can use and run these workflows in their environment. This feature also provides enhancement for reusability and sharing workflow systems.

- The approach presented in this thesis aims to achieve workflow interoperability at execution level. Future research can consider achieving interoperability at workflow and data provenance level. The PS-SWIF system can be leveraged by adapting the technique provided by the Open Provenance Vision. *The Open Provenance Vision is an approach that consists of controlled vocabulary, serialization formats and APIs (Application Programming Interfaces) that allow provenance from individual systems to be expressed, connected in a coherent fashion, and queried seamlessly* [55]. The Open Provenance Vision introduces an interoperability layer that allows provenance data stored by individual systems to be exposed and uniform queries across these stores transmitted. The Open Provenance Vision is based on the Open Provenance Model [159], which provides a provenance mode to allow different systems to exchange provenance data. More about OPM can be found in these papers [160, 55, 161, 162, 159].

- In this research, the PS-SWIF Web Services do not operate behind a firewall. One possibility to extend is to leverage the current design of PS-SWIF with the Styx protocol. The Styx [163] is a protocol that allows resources to be

exposed as a namespace, such as the UNIX file system. WSPeer framework supports several bindings such as JXTA, P2PS, Styx, and WSKPeer. Combination of P2PS and Styx allows a client behind a NAT to join the P2PS network, and then contact the rendezvous service and queries for resolver services. Then register its logical address with the resolver service .The resolver then creates a virtual file mapped to the logical address of the client and returns the location of this file, which has a Styx address, to the client. The client then initiates a read on the newly created file. Client then subscribes to a topic provided by another service. The combination of P2PS and Styx, allows clients behind NATs and firewalls to receive notification messages.

- PS-SWIF does not presently provide secure message between services. For integrity and confidentiality, public key technologies might be sufficient. For high-frequency notification other mechanisms such as WS-Trust and WS-SecureConversion might be more appropriate. Different security mechanisms should be considered to prevent different message attacks, such as Message Alteration Message disclosure.

APPENDIX A

First Version of PS-SWIF

A.1 Integration WS-Eventing within Triana Workflow

The author implements the WS-Eventing specification within Triana Workflow in the first version of PS-SWIF, to achieve workflow interoperability among different workflow system. WS-Eventing binds to Triana through the GAP interface. Triana has been used to construct a workflow that act as a Source and/or Sink Web Service that generates, or consumes, the notification message.

A Triana workflow is used as a Web Service source generating notification messages and managing subscription requests. The WS-Eventing services in Triana can be either one or more tasks (a group) as a standard Web Service within Triana. A user may select the Web Service binding as the service host to run the Group task, and, selecting the WS-Eventing option, the GAP Interface automatically launches the workflow as a WS-Eventing service. Once WS-Eventing services have been deployed, they replace the equivalent group tasks in the users' workflow to a new WE-Eventing Service task with a different colour.

A.2 Workflow Taverna Launcher

The current state of the Taverna workflow does not support deploying a workflow as a Web Service. The author creates a Taverna Workflow launcher tool that allows user to deploy a workflow in Taverna as a Web Service in order or to send a notification message from Taverna workflow. In Taverna Workflow Launcher a

Web Server is created using WSPeer framework as server on specific port , and reads workflows from configuration file and creates a new Web Service endpoint for each declared workflow in the configuration file. The Web Services are generated dynamically by using the GAP library, an invocation pipe is created with an operation that receives string and returns also string. The workflow is executed using the *WorkflowLuncherWrapper* API that provided by Taverna to run workflows without popping up the Taverna GUI.

A.3 Workflow Kepler Launcher

Moreover the current state of the Kepler workflow does not support the deployment of a workflow as a Web Service. The Workflow Kepler Launcher is similar to Workflow Taverna Launcher WSPeer and GAP library is used to create a Web Service within Kepler Workflow in order to send a notification message from Kepler Workflow. The Kepler workflow is executed through a command line that allow user to load the workflow manually and specify the values of workflow parameters.

APPENDIX B

PS-SWIF API

This approach exposes an API called Publish/Subscribe Scientific Workflow Interoperability Framework (PS-SWIF) that can be implemented in multiple workflow systems to provide run time interoperability. The PS-SWIF application is based on Web Services that enable scientists to apply a Publish/Subscribe mechanism to publish a topic using a workflow system, and enables different workflow systems to subscribe to this topic and receive notification messages when an event is executed. The PS-SWIF API was the basis for the PS-SWIF framework through Web interfaces approach presented in this thesis.

Figure B1 shows the PS-SWIF GUI which provides publish and subscribe tools to achieve workflow interoperability. The main interface is the Subscription Management Console which allows users to access the Manage Subscriptions and Manage Published Services windows. There are two buttons in the Manage Published Services window, namely (1) Publish New Service and (2) Remove Service. The Publish New Service button allows the user to publish a new topic that will be used as a Source Web Service. The Remove Service button allows the user to remove the selected service from the available published services. Once the services are published, users can use the Manage Subscription window to create a new subscription or manage some previous subscription. There are four buttons in the Manage Subscription Window: (1) Create Subscription; (2) Get Status; (3) Renew and (4) Unsubscribe. When the Create Subscription button is selected, a new window is displayed that allows the user to specify Source Event, Sink Event and Expiry Date

and then press the Subscribe button to make a subscription. Once the subscription successfully made, the subscription details, such as Event Source, Event Sink and Expiry Date are shown in the Manage Subscriptions window. Later the user can renew or delete the subscription by selecting the subscription and then choose the appropriate button and inserting the required data.

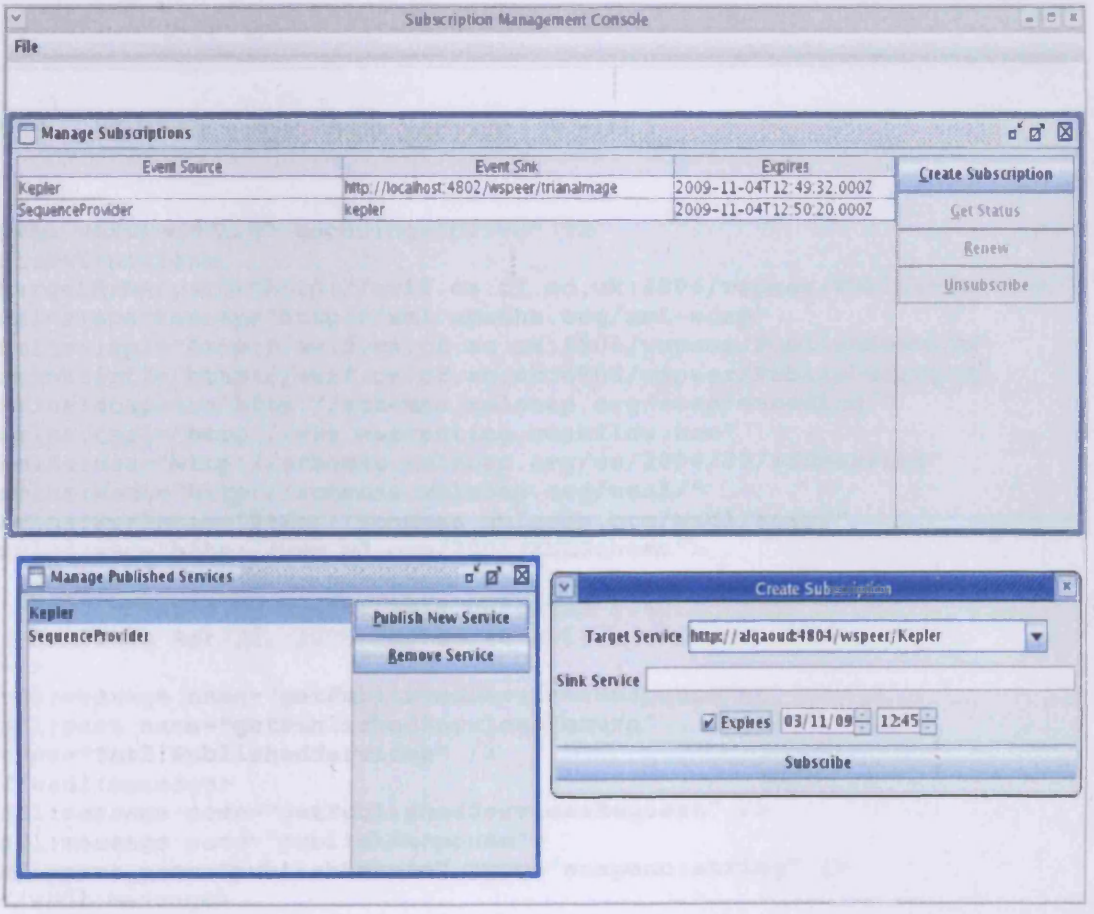


Figure B1: PS-SWIF GUI

APPENDIX C

PS-SWIF WEB SERVICES (WSDL)

C.1 Publish Topic Web Service (WSDL)

```
  <?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
   targetNamespace="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService"
   xmlns:intf="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:tns2="http://www.wseventing.workflow.com"
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <!--
     WSDL created by Apache Axis version: 1.4
     Built on Apr 22, 2006 (06:55:48 PDT)
   -->
 <wsdl:message name="getPublishedServicesResponse">
 <wsdl:part name="getPublishedServicesReturn"
   type="tns2:PublishedServices" />
   </wsdl:message>
 <wsdl:message name="getPublishedServicesRequest" />
 <wsdl:message name="publishResponse">
 <wsdl:part name="publishReturn" type="soapenc:string" />
   </wsdl:message>
 <wsdl:message name="removeResponse">
 <wsdl:part name="removeReturn" type="soapenc:string" />
   </wsdl:message>
 <wsdl:message name="isPublishedRequest">
 <wsdl:part name="serviceName" type="soapenc:string" />
   </wsdl:message>
 <wsdl:message name="publishRequest">
 <wsdl:part name="serviceName" type="soapenc:string" />
 <wsdl:part name="user" type="soapenc:string" />
 <wsdl:part name="password" type="soapenc:string" />
   </wsdl:message>
 <wsdl:message name="isPublishedResponse">
 <wsdl:part name="isPublishedReturn" type="xsd:boolean" />
   </wsdl:message>
 <wsdl:message name="removeRequest"
```

```
<wsdl:part name="serviceName" type="soapenc:string" />
<wsdl:part name="user" type="soapenc:string" />
<wsdl:part name="password" type="soapenc:string" />
  </wsdl:message>
<wsdl:portType name="PublishService">
<wsdl:operation name="remove" parameterOrder="serviceName user
  password">
<wsdl:input message="impl:removeRequest" name="removeRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService/remov
  e" />
<wsdl:output message="impl:removeResponse" name="removeResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService/remov
  eResponse" />
  </wsdl:operation>
<wsdl:operation name="getPublishedServices">
<wsdl:input message="impl:getPublishedServicesRequest"
  name="getPublishedServicesRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService/getPu
  blishedServices" />
<wsdl:output message="impl:getPublishedServicesResponse"
  name="getPublishedServicesResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService/getPu
  blishedServicesResponse" />
  </wsdl:operation>
<wsdl:operation name="isPublished" parameterOrder="serviceName">
<wsdl:input message="impl:isPublishedRequest"
  name="isPublishedRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService/isPub
  lished" />
<wsdl:output message="impl:isPublishedResponse"
  name="isPublishedResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService/isPub
  lishedResponse" />
  </wsdl:operation>
<wsdl:operation name="publish" parameterOrder="serviceName user
  password">
<wsdl:input message="impl:publishRequest" name="publishRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService/publi
  sh" />
<wsdl:output message="impl:publishResponse" name="publishResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService/publi
  shResponse" />
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="PublishServiceSoapBinding"
  type="impl:PublishService">
<wsdlsoap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="remove">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService/remov
  e" />
<wsdl:input name="removeRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService"
  use="encoded" />
  </wsdl:input>
<wsdl:output name="removeResponse">
```

```
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService"
  use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="getPublishedServices">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService/getPu
  blishedServices" />
<wsdl:input name="getPublishedServicesRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService"
  use="encoded" />
  </wsdl:input>
<wsdl:output name="getPublishedServicesResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService"
  use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="isPublished">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService/isPub
  lished" />
<wsdl:input name="isPublishedRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService"
  use="encoded" />
  </wsdl:input>
<wsdl:output name="isPublishedResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService"
  use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="publish">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService/publi
  sh" />
<wsdl:input name="publishRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService"
  use="encoded" />
  </wsdl:input>
<wsdl:output name="publishResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService"
  use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
<wsdl:service name="PublishServiceService">
```

```
<wsdl:port binding="impl:PublishServiceSoapBinding"
  name="PublishService">
<wsdlsoap:address
  location="http://swif.cs.cf.ac.uk:4804/wspeer/PublishService" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

C.2 Source Web Service(WSDL)

The following WSDL document represent an example of Source Web Service called Triana

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
  targetNamespace="http://swif.cs.cf.ac.uk:4804/wspeer/Triana"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://swif.cs.cf.ac.uk:4804/wspeer/Triana"
  xmlns:intf="http://swif.cs.cf.ac.uk:4804/wspeer/Triana"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns2="http://schemas.xmlsoap.org/ws/2004/08/eventing"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <!--
    WSDL created by Apache Axis version: 1.4
    Built on Apr 22, 2006 (06:55:48 PDT)
  -->
<wsdl:message name="subscribe_TrianaResponse">
<wsdl:part name="subscribe_TrianaReturn" type="tns2:SubscribeResponse"
  />
  </wsdl:message>
<wsdl:message name="subscribe_TrianaRequest">
<wsdl:part name="in0" type="tns2:Subscribe" />
  </wsdl:message>
<wsdl:portType name="Triana">
<wsdl:operation name="subscribe_Triana" parameterOrder="in0">
<wsdl:input message="impl:subscribe_TrianaRequest"
  name="subscribe_TrianaRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/Triana/subscribe_Tri
  ana" />
<wsdl:output message="impl:subscribe_TrianaResponse"
  name="subscribe_TrianaResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/Triana/subscribe_Tri
  anaResponse" />
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="TrianaSoapBinding" type="impl:Triana">
<wsdlsoap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="subscribe_Triana">
```

```
 <wsdlsoap:operation
   soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/Triana/subscribe_Tri
   ana" />
<wsdl:input name="subscribe_TrianaRequest">
 <wsdlsoap:body
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   namespace="http://swif.cs.cf.ac.uk:4804/wspeer/Triana" use="encoded"
   />
   </wsdl:input>
 <wsdl:output name="subscribe_TrianaResponse">
 <wsdlsoap:body
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   namespace="http://swif.cs.cf.ac.uk:4804/wspeer/Triana" use="encoded"
   />
   </wsdl:output>
   </wsdl:operation>
   </wsdl:binding>
 <wsdl:service name="TrianaService">
 <wsdl:port binding="impl:TrianaSoapBinding" name="Triana">
 <wsdlsoap:address
   location="http://swif.cs.cf.ac.uk:4804/wspeer/Triana" />
   </wsdl:port>
   </wsdl:service>
   </wsdl:definitions>
```

C.3 Publish Information Web Service (WSDL)

```
 <?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions
   targetNamespace="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService"
   xmlns:intf="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <!--
     WSDL created by Apache Axis version: 1.4
     Built on Apr 22, 2006 (06:55:48 PDT)
   -->
 <wsdl:types>
 <schema
   targetNamespace="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService"
   xmlns="http://www.w3.org/2001/XMLSchema">
 <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
 <complexType name="ArrayOf_soapenc_string">
 <complexContent>
 <restriction base="soapenc:Array">
 <attribute ref="soapenc:arrayType" wsdl:arrayType="soapenc:string[]"
   />
   </restriction>
   </complexContent>
   </complexType>
   </schema>
```

```
  </wsdl:types>
<wsdl:message name="sendNotificationRequest">
<wsdl:part name="sourceId" type="soapenc:string" />
<wsdl:part name="param" type="soapenc:string" />
  </wsdl:message>
<wsdl:message name="sendNotificationValuesResponse">
<wsdl:part name="sendNotificationValuesReturn" type="soapenc:string"
  />
  </wsdl:message>
<wsdl:message name="sendNotificationValuesRequest">
<wsdl:part name="sourceId" type="soapenc:string" />
<wsdl:part name="params" type="impl:ArrayOf_soapenc_string" />
  </wsdl:message>
<wsdl:message name="sendNotificationResponse">
<wsdl:part name="sendNotificationReturn" type="soapenc:string" />
  </wsdl:message>
<wsdl:portType name="NotifyService">
<wsdl:operation name="sendNotification" parameterOrder="sourceId
  param">
<wsdl:input message="impl:sendNotificationRequest"
  name="sendNotificationRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService/sendNo
  tification" />
<wsdl:output message="impl:sendNotificationResponse"
  name="sendNotificationResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService/sendNo
  tificationResponse" />
  </wsdl:operation>
<wsdl:operation name="sendNotificationValues" parameterOrder="sourceId
  params">
<wsdl:input message="impl:sendNotificationValuesRequest"
  name="sendNotificationValuesRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService/sendNo
  tificationValues" />
<wsdl:output message="impl:sendNotificationValuesResponse"
  name="sendNotificationValuesResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService/sendNo
  tificationValuesResponse" />
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="NotifyServiceSoapBinding"
  type="impl:NotifyService">
<wsdlsoap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="sendNotification">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService/sendNo
  tification" />
<wsdl:input name="sendNotificationRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService"
  use="encoded" />
  </wsdl:input>
<wsdl:output name="sendNotificationResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService"
  use="encoded" />
```

```
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="sendNotificationValues">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService/sendNo
  tificationValues" />
<wsdl:input name="sendNotificationValuesRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService"
  use="encoded" />
  </wsdl:input>
<wsdl:output name="sendNotificationValuesResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService"
  use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
<wsdl:service name="NotifyServiceService">
<wsdl:port binding="impl:NotifyServiceSoapBinding"
  name="NotifyService">
<wsdlsoap:address
  location="http://swif.cs.cf.ac.uk:4804/wspeer/NotifyService" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

C.4 Subscriber Web Service (WSDL)

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
  targetNamespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscribe
  rService" xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice"
  xmlns:intf="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns2="http://www.wseventing.workflow.com"
  xmlns:tns3="http://schemas.xmlsoap.org/ws/2004/08/eventing"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <!--
    WSDL created by Apache Axis version: 1.4
    Built on Apr 22, 2006 (06:55:48 PDT)
  -->
<wsdl:message name="unsubscribeResponse" />
<wsdl:message name="subscribeResponse">
<wsdl:part name="subscribeReturn" type="tns3:SubscribeResponse" />
  </wsdl:message>
<wsdl:message name="subscribeRequest">
<wsdl:part name="subscribeRequest" type="tns2:SubscribeRequest" />
```

```
  </wsdl:message>
<wsdl:message name="unsubscribeRequest">
<wsdl:part name="unsubscribeRequest" type="tns2:UnsubscribeRequest" />
  </wsdl:message>
<wsdl:message name="renewResponse">
<wsdl:part name="renewReturn" type="tns3:RenewResponse" />
  </wsdl:message>
<wsdl:message name="getStatusResponse">
<wsdl:part name="getStatusReturn" type="tns3:GetStatusResponse" />
  </wsdl:message>
<wsdl:message name="getStatusRequest">
<wsdl:part name="getStatusRequest" type="tns3:GetStatusRequest" />
  </wsdl:message>
<wsdl:message name="renewRequest">
<wsdl:part name="renewRequest" type="tns2:RenewRequest" />
  </wsdl:message>
<wsdl:portType name="WseRpcSubscriberService">
<wsdl:operation name="subscribe" parameterOrder="subscribeRequest">
<wsdl:input message="impl:subscribeRequest" name="subscribeRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice/subscribe" />
<wsdl:output message="impl:subscribeResponse" name="subscribeResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice/subscribeResponse" />
  </wsdl:operation>
<wsdl:operation name="renew" parameterOrder="renewRequest">
<wsdl:input message="impl:renewRequest" name="renewRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice/renew" />
<wsdl:output message="impl:renewResponse" name="renewResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice/renewResponse" />
  </wsdl:operation>
<wsdl:operation name="unsubscribe"
  parameterOrder="unsubscribeRequest">
<wsdl:input message="impl:unsubscribeRequest"
  name="unsubscribeRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice/unsubscribe" />
<wsdl:output message="impl:unsubscribeResponse"
  name="unsubscribeResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice/unsubscribeResponse" />
  </wsdl:operation>
<wsdl:operation name="getStatus" parameterOrder="getStatusRequest">
<wsdl:input message="impl:getStatusRequest" name="getStatusRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice/getStatus" />
<wsdl:output message="impl:getStatusResponse" name="getStatusResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice/getStatusResponse" />
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="WseRpcSubscriberServiceSoapBinding"
  type="impl:WseRpcSubscriberService">
<wsdlsoap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="subscribe">
```

```
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice/subscribe" />
<wsdl:input name="subscribeRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServi
  ce" use="encoded" />
  </wsdl:input>
<wsdl:output name="subscribeResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServi
  ce" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="renew">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice/renew" />
<wsdl:input name="renewRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServi
  ce" use="encoded" />
  </wsdl:input>
<wsdl:output name="renewResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServi
  ce" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="unsubscribe">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice/unsubscribe" />
<wsdl:input name="unsubscribeRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServi
  ce" use="encoded" />
  </wsdl:input>
<wsdl:output name="unsubscribeResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServi
  ce" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="getStatus">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServ
  ice/getStatus" />
<wsdl:input name="getStatusRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServi
  ce" use="encoded" />
```

```
  </wsdl:input>
<wsdl:output name="getStatusResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServi
  ce" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
<wsdl:service name="WseRpcSubscriberServiceService">
<wsdl:port binding="impl:WseRpcSubscriberServiceSoapBinding"
  name="WseRpcSubscriberService">
<wsdlsoap:address
  location="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriberServic
  e" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

C.5 Subscription Manager Web Service (WSDL)

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
  targetNamespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscript
  ionManagerService" xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService"
  xmlns:intf="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns2="http://schemas.xmlsoap.org/ws/2004/08/eventing"
  xmlns:tns3="http://www.wseventing.workflow.com"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <!--
    WSDL created by Apache Axis version: 1.4
    Built on Apr 22, 2006 (06:55:48 PDT)
  -->
<wsdl:message name="renewRequest">
<wsdl:part name="renew" type="tns2:Renew" />
  </wsdl:message>
<wsdl:message name="isSubscribedRequest">
<wsdl:part name="sinkService" type="soapenc:string" />
<wsdl:part name="targetService" type="soapenc:string" />
  </wsdl:message>
<wsdl:message name="getAllSubscriptionsResponse">
<wsdl:part name="getAllSubscriptionsReturn" type="tns3:Subscriptions"
  />
  </wsdl:message>
<wsdl:message name="getStatusResponse">
<wsdl:part name="getStatusReturn" type="tns2:GetStatusResponse" />
  </wsdl:message>
<wsdl:message name="getStatusRequest">
```

```
<wsdl:part name="statusRequest" type="tns2:GetStatus" />
  </wsdl:message>
<wsdl:message name="clearAllSubscriptionsRequest" />
<wsdl:message name="isSubscribedResponse">
<wsdl:part name="isSubscribedReturn" type="xsd:boolean" />
  </wsdl:message>
<wsdl:message name="clearAllSubscriptionsResponse">
<wsdl:part name="clearAllSubscriptionsReturn" type="soapenc:string" />
  </wsdl:message>
<wsdl:message name="renewResponse">
<wsdl:part name="renewReturn" type="tns2:RenewResponse" />
  </wsdl:message>
<wsdl:message name="unsubscribeRequest">
<wsdl:part name="unsubscribe" type="tns2:Unsubscribe" />
  </wsdl:message>
<wsdl:message name="unsubscribeResponse" />
<wsdl:message name="getAllSubscriptionsRequest">
<wsdl:part name="user" type="soapenc:string" />
<wsdl:part name="password" type="soapenc:string" />
  </wsdl:message>
<wsdl:portType name="WseRpcSubscriptionManagerService">
<wsdl:operation name="renew" parameterOrder="renew">
<wsdl:input message="impl:renewRequest" name="renewRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/renew" />
<wsdl:output message="impl:renewResponse" name="renewResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/renewResponse" />
  </wsdl:operation>
<wsdl:operation name="getAllSubscriptions" parameterOrder="user
  password">
<wsdl:input message="impl:getAllSubscriptionsRequest"
  name="getAllSubscriptionsRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/getAllSubscriptions" />
<wsdl:output message="impl:getAllSubscriptionsResponse"
  name="getAllSubscriptionsResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/getAllSubscriptionsResponse" />
  </wsdl:operation>
<wsdl:operation name="isSubscribed" parameterOrder="sinkService
  targetService">
<wsdl:input message="impl:isSubscribedRequest"
  name="isSubscribedRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/isSubscribed" />
<wsdl:output message="impl:isSubscribedResponse"
  name="isSubscribedResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/isSubscribedResponse" />
  </wsdl:operation>
<wsdl:operation name="unsubscribe" parameterOrder="unsubscribe">
<wsdl:input message="impl:unsubscribeRequest"
  name="unsubscribeRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/unsubscribe" />
<wsdl:output message="impl:unsubscribeResponse"
  name="unsubscribeResponse"
```

```
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/unsubscribeResponse" />
  </wsdl:operation>
<wsdl:operation name="getStatus" parameterOrder="statusRequest">
<wsdl:input message="impl:getStatusRequest" name="getStatusRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/getStatus" />
<wsdl:output message="impl:getStatusResponse" name="getStatusResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/getStatusResponse" />
  </wsdl:operation>
<wsdl:operation name="clearAllSubscriptions">
<wsdl:input message="impl:clearAllSubscriptionsRequest"
  name="clearAllSubscriptionsRequest"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/clearAllSubscriptions" />
<wsdl:output message="impl:clearAllSubscriptionsResponse"
  name="clearAllSubscriptionsResponse"
  wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/clearAllSubscriptionsResponse" />
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="WseRpcSubscriptionManagerServiceSoapBinding"
  type="impl:WseRpcSubscriptionManagerService">
<wsdlsoap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="renew">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/renew" />
<wsdl:input name="renewRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMan
  agerService" use="encoded" />
  </wsdl:input>
<wsdl:output name="renewResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMan
  agerService" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="getAllSubscriptions">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/getAllSubscriptions" />
<wsdl:input name="getAllSubscriptionsRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMan
  agerService" use="encoded" />
  </wsdl:input>
<wsdl:output name="getAllSubscriptionsResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMan
  agerService" use="encoded" />
  </wsdl:output>
```

```
  </wsdl:operation>
<wsdl:operation name="isSubscribed">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/isSubscribed" />
<wsdl:input name="isSubscribedRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMan
  agerService" use="encoded" />
  </wsdl:input>
<wsdl:output name="isSubscribedResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMan
  agerService" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="unsubscribe">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/unsubscribe" />
<wsdl:input name="unsubscribeRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMan
  agerService" use="encoded" />
  </wsdl:input>
<wsdl:output name="unsubscribeResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMan
  agerService" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="getStatus">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/getStatus" />
<wsdl:input name="getStatusRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMan
  agerService" use="encoded" />
  </wsdl:input>
<wsdl:output name="getStatusResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMan
  agerService" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="clearAllSubscriptions">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMa
  nagerService/clearAllSubscriptions" />
<wsdl:input name="clearAllSubscriptionsRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
```

```
    namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMan
    agerService" use="encoded" />
    </wsdl:input>
  <wsdl:output name="clearAllSubscriptionsResponse">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMan
    agerService" use="encoded" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
  <wsdl:service name="WseRpcSubscriptionManagerServiceService">
  <wsdl:port binding="impl:WseRpcSubscriptionManagerServiceSoapBinding"
    name="WseRpcSubscriptionManagerService">
  <wsdlsoap:address
    location="http://swif.cs.cf.ac.uk:4804/wspeer/WseRpcSubscriptionMana
    gerService" />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

C.6 Sink Web Service (WSDL)

```
  <?xml version="1.0" encoding="UTF-8" ?>
  <wsdl:definitions
    targetNamespace="http://swif.cs.cf.ac.uk:4804/wspeer/SinkService"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://swif.cs.cf.ac.uk:4804/wspeer/SinkService"
    xmlns:intf="http://swif.cs.cf.ac.uk:4804/wspeer/SinkService"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <!--
      WSDL created by Apache Axis version: 1.4
      Built on Apr 22, 2006 (06:55:48 PDT)
    -->
  <wsdl:message name="receiveNotificationResponse">
  <wsdl:part name="receiveNotificationReturn" type="soapenc:string" />
    </wsdl:message>
  <wsdl:message name="receiveNotificationRequest">
  <wsdl:part name="sourceId" type="soapenc:string" />
  <wsdl:part name="token" type="soapenc:string" />
    </wsdl:message>
<wsdl:portType name="SinkService">
<wsdl:operation name="receiveNotification" parameterOrder="sourceId
    token">
  <wsdl:input message="impl:receiveNotificationRequest"
    name="receiveNotificationRequest"
    wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/SinkService/receiveN
    otification" />
  <wsdl:output message="impl:receiveNotificationResponse"
    name="receiveNotificationResponse"
    wsa:Action="http://swif.cs.cf.ac.uk:4804/wspeer/SinkService/receiveN
    otificationResponse" />
```

```
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="SinkServiceSoapBinding" type="impl:SinkService">
<wsdlsoap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="receiveNotification">
<wsdlsoap:operation
  soapAction="http://swif.cs.cf.ac.uk:4804/wspeer/SinkService/receiveN
  otification" />
<wsdl:input name="receiveNotificationRequest">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/SinkService"
  use="encoded" />
  </wsdl:input>
<wsdl:output name="receiveNotificationResponse">
<wsdlsoap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://swif.cs.cf.ac.uk:4804/wspeer/SinkService"
  use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
<wsdl:service name="SinkServiceService">
<wsdl:port binding="impl:SinkServiceSoapBinding" name="SinkService">
<wsdlsoap:address
  location="http://swif.cs.cf.ac.uk:4804/wspeer/SinkService" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

APPENDIX D

DATABASE (SQL)

D.1 Subscription Database

```
CREATE TABLE SUBSCRIPTIONS (
SUBSCRIPTION_ID VARCHAR(50) NOT NULL,
SUBSCRIPTION_TS TIMESTAMP NOT NULL,
SOURCE_ID VARCHAR(100) NOT NULL,
EXPIRATION TIMESTAMP,
SUBSCRIBE VARCHAR(4000) NOT NULL,
OWNER VARCHAR(50) NOT NULL,
PRIMARY KEY(SUBSCRIPTION_ID));
```

D.2 Topic Database

```
CREATE TABLE TOPICS(
TOPIC_NAME VARCHAR(100) NOT NULL,
OWNER VARCHAR(50) NOT NULL,
PRIMARY KEY(TOPIC_NAME);
```

D.3 User Database

```
CREATE TABLE USERS (
USER_LOGIN VARCHAR(50) NOT NULL,
FULL_NAME VARCHAR(300) NOT NULL,
IS_ADMIN INT NOT NULL,
USER_PASSWORD VARCHAR(50),
PRIMARY KEY(USER_LOGIN))
```

D.4 SQL Manipulating Statements

The following SQL statements are used to manipulate the tables above.

```
INSERT INTO USERS(USER_LOGIN, FULL_NAME, IS_ADMIN,
USER_PASSWORD) VALUES('alqaoud', 'Administrator',
1,"******");

INSERT INTO SUBSCRIPTIONS(SUBSCRIPTION_ID,
SUBSCRIPTION_TS, SOURCE_ID, EXPIRATION, SUBSCRIBE, OWNER)
VALUES (?, ?, ?, ?, ?, ?)";

DELETE FROM SUBSCRIPTIONS WHERE SUBSCRIPTION_ID=?;

DELETE FROM SUBSCRIPTIONS

"UPDATE SUBSCRIPTIONS SET EXPIRATION=? WHERE
SUBSCRIPTION_ID=?"

SELECT EXPIRATION FROM SUBSCRIPTIONS WHERE
SUBSCRIPTION_ID=?";

"SELECT * FROM SUBSCRIPTIONS WHERE SOURCE_ID=?";

"SELECT * FROM SUBSCRIPTIONS";

"SELECT * FROM SUBSCRIPTIONS WHERE EXPIRATION IS NOT NULL
AND   EXPIRATION <= ?";

"SELECT * FROM SUBSCRIPTIONS WHERE SUBSCRIPTION_ID=?";
```

Index

I

J

K

M

N

O

P

Q

R

S

T

U

V

W

X

Bibliography

[1] A.Alqaoud, I.Taylor, A.Jones,2010, *Scientific Workflow Interoperability Framework.* International Journal of Business Process Integration and Management. (Scientific Workflows).

[2] WfMC, 1995, Workflow Management Coalition The Workflow Reference Model

[3] Fox, G. C. and Gannon, D.,2006, *Workflow in grid systems.* CONCURRENCY AND COMPUTATION:PRACTICE AND EXPERIENCE. **18**(10): p. 1009-1019.

[4] WfMC, 1996, Workflow Management Coalition Workflow Standard - Interoperability Abstract Specification,

[5] WfMC, 2000, Workflow Management Coalition Workflow Standard - Interoperability Internet e-m MIME Binding,

[6] WfMC,2001, *Workflow Management Coalition Workflow Standard - Interoperability Wf-XML Binding.*

[7] Harrison, Andrew, October, 2007, Workflow sharing and Interoperability,

[8] Klingenstein, K and Gannon, D, October, 2007, Improving Interoperability, Sustainability and Platform Convergence in Scientific And Scholarly Workflow, University of Colorado and Indiana University.

[9] Taylor, Ian, February 2008, Workflow Management Research Group - WFM-RG,

[10] Toth, Adrian, May 2007, Levels of the Grid Workflow Interoperability,

[11] Livny, Ewa Deelman and Miron, The Pegasus Approach to Building a Workflow Management System,

[12] Son, H and Li, X,September 2007, *PARMI: A Publish/Subscribe Based Asynchronous RMI Framework for Cluster Computing*, in *High Performance Computing and Communications*, Springer Berlin / Heidelberg. p. 19-29.

[13] The SIMDAT Project. 2004,[cited; Available from: http://www.simdat.org/].

[14] Turner, M, 2007, Aerospace prototype for Interoperability with updated validation results, SIMDAT, Information Society Technologies

[15] W, Wirch, 2006, Documentation of software design, initial implementation and underlying technologies for SIMDAT Automotive Demonstrators for Interoperability Phase, Information Society Technologies

[16] G, Aubert, 2007, SIMDAT Meteorology Application: Evaluation/Validation report and statement of requirements for project phase III, Information Society Technologies

[17] F, Zimmermann, 2007, Documentation of advanced implementation of SIMDAT Pharma Prototypes for Interoperability Phase including evaluation of underlying technologies, Information Society Technologies

[18] Azam, N, 2007, Consolidated Report on Implementation of Workflow Management Infrastructure, on Workflow Interoperability and Business Processes and on Meta-Scheduling Design, Information Society Technologies

[19] Integration, Phoenix, 1995, *ModelCenter Graphical Problem Solving Environmen.*[cited; Available from: http://www.phoenix-int.com/software/phx_modelcenter.php].

[20] SIMULIA, 2004, *FIPER.*[cited; Available from: http://www.simulia.com/about/about.html].

[21] MathWorks, 1994, *MATLAB - The Language Of Technical Computing.*[cited; Available from: http://www.mathworks.com/products/matlab/].

[22] Noesis, 2003, *OPTIMUS* [cited; Available from: http://www.noesissolutions.com/index.php?col=/products&doc=optimus].

[23] Kacsuk, P, 2010, SHaring Interoperable Workflows for large-scale scientific simulations on Available DCIs (SHIWA), EUROPEAN COMMISSION, 7TH RESEARCH FRAMEWORK PROGRAMME (FP7)

[24] Fahringer, T., Prodan, R., Duan, R., et al.,2005, *ASKALON: A grid application development and computing environment.* in *6th International Workshop on Grid Computing*. New York,: IEEE Computer Society Press.

[25] Moteur Workflow Enactor.[cited 2009; Available from: http://www.aci-agir.org/].

[26] Nemeth, C, Dozsa, G, Lovas, R, et al.,2004, *The p-grade grid portal*, in *Computational Science and Its Applications – ICCSA* Springer: Berlin / Heidelberg. p. 10-19.

[27] Cardiff, University, 1998, *The Triana Project.*[cited; Available from: http://www.trianacode.org/].

[28] Foundation, The National Science, *Linked Environments for Atmospheric Discovery (LEAD)*. 2003.

[29] Koranda, Scott,2007, *LIGO Inspiral Analysis Workflow.*

[30] Maechling, Philip,2007, *SCEC Earthquake Wave Propagation and Source Validation Workflow.*

[31] Sommerville, Ian,2001, *Software Engineering.* Sixth ed., Essex: Addison-Wesley Publishers Limited.

[32] Box, D., Cabrera, L. F., Critchley, C., et al., *Web Services Eventing (WS-Eventing).* 2004.

[33] Oinn, T, Addis, M, and Ferris, J, *Taverna: a tool for the composition and enactment of bioinformatics workflows*, in *Bioinformatics.* 2004, Oxford Univ Press.

[34] Foundation, National Science, 2002, The Kepler Project,

[35] Deelman, E., Gannon, D., Shields, M., et al., *Workflows and e-Science: An overview of workflow system features and capabilities.* 2009, Elsevier. p. 528-540.

[36] Yu, J. and Buyya, R., *A taxonomy of workflow management systems for grid computing.* 2005, Springer. p. 171-200.

[37] Sakellariou, R. and Zhao, H., *A low-cost rescheduling policy for efficient mapping of workflows on grid systems.* 2004, IOS Press. p. 253-262.

[38] Berman, F., Fox, G., and Hey, A. J. G.,2003, *Grid computing: making the global infrastructure a reality*: John Wiley & Sons Inc.

[39] DAGMan Application, December 2004],

[40] Deelman, E., Blythe, J., Gil, Y., et al., *Pegasus: Mapping scientific workflows onto the grid.* 2004, Springer. p. 11-20.

[41] McGough, S., Young, L., Afzal, A., et al., *Workflow enactment in ICENI.* 2004. p. 894–900.

[42] Almond, J. and Snelling, D., *UNICORE: Secure and uniform access to distributed resources via the world wide web.* 1999.

[43] Von Laszewski, G., Amin, K., and Hategan, M.,2004, *Gridant: A client-controllable grid workflow system.* in *37th Annual Hawaii International Conference on System Science (HICSS'04),*. Big Island, Hawaii: IEEE CS Press,.

[44] Andrews, T., Curbera, F., Dholakia, H., et al., *Business process execution language for web services, version 1.1.* 2003.

[45] Yu, J. and Buyya, R., *A novel architecture for realizing grid workflow using tuple spaces*. 2004, IEEE Computer Society. p. 128.

[46] Tannenbaum, T., Wright, D., Miller, K., et al., *Condor: a distributed job scheduler*. 2001, MIT Press Cambridge, MA, USA.

[47] Callahan, S., Freire, J., Santos, E., et al.,2006, *Managing the evolution of dataflows with vistrails*. in *IEEE Workshop on Workflow and Data Flow for Scientific Applications*. SciFlow.

[48] Petri, C. A.,1962, *Kommunikation mit automaten*: Rhein.-Westfl. Inst. f. Instrumentelle Mathematik an der Univ. Bonn.

[49] Group., Object Management, *Unified Modeling Language (UML)*.[cited 2009; Available from: http://www.uml.org/].

[50] Hoheisel, A,March 9, 2004, *User tools and languages for graph-based Grid workflows*. in *Grid Workflow Workshop, GGF10*. Berlin, Germany.

[51] Dinda, P.2002, *Online prediction of the running time of tasks*. in *Cluster Computing*. Netherlands: Kluwer Academic Publishers.

[52] Aversano, L, Cimitile, A, Gallucci, P, et al.,2002, *FlowManager: a workflow management system based on Petri nets*. in *26th Annual International Computer Software and Applications Conference*. Oxford, England: IEEE CS Press.

[53] Yahyapour, R., Wieder, P., Pugliese, A., et al., 2004, Grid scheduling use cases, Paper, Global Grid Forum

[54] Cardoso, J., Sheth, A., Miller, J., et al.,2004, *Quality of service for workflows and web service processes*. Web Semantics Journal: Science, Services and Agents on the World Wide Web,. 1(3): p. 281-308.

[55] Moreau, L, 2009, The Foundations for Provenance on the Web, University of Southamption

[56] Simmhan, Y. L., Plale, B., Gannon, D., et al.,2006, *Performance evaluation of the karma provenance framework for scientific workflows*. in *International Provenance and Annotation Workshop, IPAW*. Berlin: Springer.

[57] Swenson, K. D., *ASAP/Wf-XML 2.0 Cookbook*. 2005.

[58] W3C, 2000, Simple Object Access Protocol (SOAP), DevelopMentor, International Business Machines Corporation, Lotus Development Corporation, Microsoft, UserLand Software

[59] Swenson, K. D., Gilger, M. D., and Predhan, S., *Wf-XML 2.0-XML Based Protocol for Run-Time Integration of Process Engines*. 2004.

[60] Marin, M., Norin, R., and Shapiro, R., 2001, Workflow Process Definition Interface—XML Process Definition Language, Workflow Management Coalition Workflow Standard

[61] W3C, 2001, Web Services Description Language (WSDL), Ariba, International Business Machines Corporation, Microsoft

[62] IBM, 2004, Web Services Notification (WS-Notification),

[63] Harrison, A, Taylor, I, Wang, I, et al.,2008, *WS-RF Workflow in Triana.* International Journal of High Performance Computing Applications. **22**(3): p. 268.

[64] Hapner, M, Burridge, R, Sharma, R, et al., *Java message service specification.* 2000.

[65] Dec, H. P. and HyperDesk, N. C. R., *The common object request broker: architecture and specification.* 1991, Object Management Group

[66] OMG, 2004, Event Service Specification, Object Management Group

[67] BEA, Systems,2004, *Notification Service Specification.* in.: Object Management Group.

[68] Gisolfi, D., 2001, *Web services architect, Part 3: Is Web Services the reincarnation of CORBA.*[cited; Available from: URL http://www.ibm.com/developerworks/library/ws-arc3/index.html].

[69] Tuecke, S., Czajkowski, K., Foster, I., et al., *Open grid services infrastructure (ogsi) version 1.0.* 2003. p. 27.

[70] Czajkowski, K., Ferguson, F, Foster, I, et al., *The WS-resource framework version (WS-RF).* 2004, Globus Alliance , IBM

[71] Graham, S. and Niblett, P., Web Services Base Notification (WS-BaseNotification). Technical Specification. 2004,

[72] Graham, S., Niblett, P., Chappell, D., et al., *Web services brokered notification (ws-brokerednotification).* 2004. p. 2006.

[73] Graham, S., Niblett, P., Chappell, D., et al., *Web Services Topics (WS-Topics).* 2004.

[74] The University of Edinburgh, 2002, The OGSA-DAI Project,

[75] Bosworth, A., Box, D., Christensen, E., et al., *Web Services Addressing (WS-Addressing)*. 2003.

[76] Taylor, I.2006, *Triana generations.* in *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*: IEEE.

[77] Gravitational Wave Project GEO600 [cited; Available from: http://www.geo600.org/].

[78] Riposan, A., Taylor, I. J., Rana, O., et al., *The TRIACS Analytical Workflows Platform For Distributed Clinical Decision Support.*

[79] Enabling Desktop Grids for e-Science. 2007,[cited 2009; Available from: http://www.edges-grid.eu/].

[80] OMII, 2007, *Workflows Hosted In Portals.*[cited 2008; Available from: http://www.trianacode.org/whipplugin/].

[81] Taylor, I., Al-Shakarchi, E., and Beck, S. D., *Distributed Audio Retrieval using Triana (DART)*. 2006. p. 6-11.

[82] Pahwa, J. S., Brewer, P., Sutton, T., et al.,2006, *Biodiversity World: A problem-solving environment for analysing biodiversity patterns.* in *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*. Singapore IEEE Computer society.

[83] Rana, O, *Environment for Industrial Design Optimisation (DIPSO).*[cited 2008; Available from: http://www.wesc.ac.uk/projects/dipso/index.html].

[84] Stankovski, V and Swain, M, 2008, *The Data Mining Tools and Services for Grid Computing Environments.*[cited 2008; Available from: http://www.datamininggrid.org/].

[85] Grid Enabled web eNvironment for site Independent User job Submission [cited; Available from: https://genius.ct.infn.it/].

[86] IST, *Grid-Enabled Medical Simulation Services.*[cited; Available from: http://www.it.neclab.eu/gemss/index.html].

[87] Shaikh, Ali, 2005, *Federated Analysis Environment for Heterogeneous Intelligent Mining.*[cited 2007; Available from: http://users.cs.cf.ac.uk/Ali.Shaikhali/faehim/index.htm].

[88] Taylor, I, Wang, I, Shields, M, et al.,2004, *Distributed computing with Triana on the Grid.* in *concurrency and computation: practice and experience.* Wiley InterScience.

[89] Taylor, I, Shields, M, Wang, I, et al.,2005, *Visual grid workflow in Triana.* Journal of Grid Computing. 3(3): p. 153-169.

[90] Taylor, I, Shields, M, Wang, I, et al.,2003, *Triana applications within Grid computing and peer to peer environments.* Journal of Grid Computing. **1**(2): p. 199-217.

[91] Taylor, I, Wang, I, Shields, M, et al., Triana User Guide, Cardiff University

[92] Seidel, E, Allen, G, Merzky, A, et al.,2002, *GridLab—a grid application toolkit and testbed.* Future Generation Computer Systems. **18**(8): p. 1143-1153.

[93] Ananthakrishnan, R, Bester, J, Czajkowski, K, et al., *Grid Resource Allocation Management* [cited 2008; Available from: http://dev.globus.org/wiki/GRAM].

[94] Nabrzyski, J, Schopf, J, and Weqlarz, J, eds.,*2004, Grid Resource Management.* Kluwer Acadmic Publishers: Norwell, Massachusetts, USA.

[95] Oram, A., *Peer-to-peer: Harnessing the power of disruptive technologies.* 2001, O'Reilly & Associates, Inc. Sebastopol, CA, USA.

[96] Andrew, Harrison and Ian, Taylor,2006, *Service-oriented middleware for hybrid environments.* in *Proceedings of the 1st international workshop on Advanced data processing in ubiquitous computing (ADPUC 2006).* Melbourne, Australia: ACM.

[97] Gong, L, 2001, JXTA: A network programming environment,

[98] Arnold, K., Scheifler, R., Waldo, J., et al.,1999, *Jini Specification*: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

[99] Systinet and IBM, 2000, Universal Description, Discovery and Integration (UDDI), OASIS

[100] Wang, I, 2006, WSRF and Triana, Cardiff University

[101] Zhao, Z, Booms, S, Belloum, A, et al.,2006, *Vle-wfbus: a scientific workflow bus for multi e-science domains.* in *Second IEEE International Conference on e-Science and Grid Computing*: IEEE Computer Society.

[102] Kukla, T, Kiss, T, Terstyanszky, G, et al.,2008, *A general and scalable solution for heterogeneous workflow invocation and nesting.* in *The 3rd Workshop on Workflows in Support of Large-scale Science (WORKS08).* Austin, TX: IEEE.

[103] OMII, *myGrid.*[cited 2008; Available from: http://www.mygrid.org.uk/].

[104] The Taverna Project. 2004,[cited 2009; Available from: http://taverna.sourceforge.net/].

[105] European Bioinformatics Institute.[cited 2008; Available from: http://www.ebi.ac.uk/].

[106] European Molecular Biology Laboratory.[cited 2008; Available from: http://www.embl.de/].

[107] National Center for Biotechnology Information [cited 2008; Available from: http://www.ncbi.nlm.nih.gov/].

[108] DNA Data Bank of Japan DDBJ.[cited 2009; Available from: http://www.ddbj.nig.ac.jp/].

[109] Owen, S, 2004, Taverna Workbench Presentaion, University of Manchester

[110] The European Bioinformatics Institute, *Ebi soaplab Web Services*.[cited 2008; Available from: http://www.ebi.ac.uk/Tools/webservices/soaplab/overview].

[111] myExperiment. 2007,[cited 2008; Available from: http://www.myexperiment.org/].

[112] The National Science Foundation.[cited 2009; Available from: http://www.nsf.gov/].

[113] RESearch sURGe ENabled by CyberinfrastructurE. 2005,[cited 2009; Available from: http://ocikbws.uzh.ch/resurgence/].

[114] Science Environment for Ecological Knowledge 2004,[cited 2009; Available from: http://seek.ecoinformatics.org/].

[115] Keller, R, Seber, D, Sinha, A, et al., 2004, The Geosciences Network (GEON): one step towards building cyberinfrastructure for the geosciences European Geophysical Union

[116] Ludaescher, B, Altintas, I, and Moore, R, 2005, Scientific Process Automation (SPA), San Diego Supercomputer Center: University of California and UC DAVIS Genome Center

[117] ROADNet - Real-Time Observatories, Applications, and Data Management Network.[cited 2009; Available from: http://roadnet.ucsd.edu/rtd.html].

[118] Cyberinfrastructure for Phylogenetic Research (CIPRES) project 2003,[cited 2009; Available from: http://www.phylo.org/].

[119] Liu, X, Xiong, Y, and Lee, A,2001, *The Ptolemy II framework for visual languages*. Human-Centric Computing Languages and Environments, IEEE CS International Symposium. p. 50.

[120] Kepler User Manual, 2008,

[121] The R Project for Statistical Computing.[cited 2008; Available from: http://www.r-project.org/].

[122] GNU Operating Systme, 1984, Free Software Foundation, Inc

[123] Cox, S, Chen, L, and Campobasso, S, 2001, *Grid enabled optimisation and design search for Engineering (geodise)*.[cited 2008; Available from: http://www.geodise.org/].

[124] OMII-BPEL (Business Process Execution Language).[cited 2009; Available from: http://www.omii.ac.uk/wiki/BPEL].

[125] Service-Oriented Development In a Unified framework *(SODIUM)*. 2004,[cited 2008; Available from: http://www.atc.gr/sodium/].

[126] Pautasso, C., Heinis, T., and Alonso, G.,2005, *Unified Service Composition Language (USCL)* Service Oriented Development In a Unified fraMework, SODIUM. p. 93.

[127] Tsalgatidou, A, Athanasopoulos, G, and Pantazoglou, M,2006, *Developing scientific workflows from heterogeneous services.* Special Interest Group on Management of Data. **35**(2): p. 22--28.

[128] VisTrails Provenance for a DIGTAL WORLD. 2007,[cited 2009; Available from: http://www.vistrails.com/].

[129] VisTrails User's Guide, 2009, University of Utah

[130] Rowe, A, Kalaitzopoulos, D, Osmond, M, et al.,2003, *The discovery net system for high throughput bioinformatics.* Bioinformatics. **19**: p. 225--231.

[131] Alberto, F, Maurizio, M, Ivan, P, et al., A Grid infrastructure for managing workflows in bioinformatics applications,

[132] Yu, J. and Buyya, R.,2004, *A novel architecture for realizing grid workflow using tuple spaces.* in *5th International Workshop on Grid Computing* IEEE Computer Society.

[133] Delaitre, T, Kiss, T, and Goyeneche, A,2005, *GEMLCA: Running legacy code applications as grid services.* Journal of Grid Computing. **3**(1): p. 75-90.

[134] Kertesz, A., Sipos, G., and Kacsuk, P.,2007, *Brokering multi-grid workflows in the P-GRADE portal*, in *Euro-Par 2006: Parallel Processing* Springer: Berlin / Heidelberg. p. 138-149.

[135] Kacsuk, P and Sipos, G,2005, *Multi-grid, multi-user workflows in the P-GRADE grid portal.* Journal of Grid Computing. **3**(3): p. 221-238.

[136] The Virtual laboratory for e-science. 2004,[cited 2008; Available from: http://www.vl-e.nl/].

[137] Afsarmanesha, H, Bellemana, R, and Bellouma, A, *VLAM-G: A Grid-based virtual laboratory*. 2002, IOS Press. p. 173-181.

[138] Fernando, S. D. I., Creager, D. A., and Simpson, A. C.,2007, *Towards build-time interoperability of workflow definition languages*. in *Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. Timisoara, Romania: IEEE Computer Society.

[139] Ghanem, M and Azam, N, 2006, Consolidated Report on Workflow Interopreability and Business Processes, Information Society Technologies

[140] Ghanem, M and Azam, N, 2007, Report on Grid infrastructure interoperability challenges, Information Society Technologies

[141] Ghanem, M and Azam, N, 2007, SIMDAT Getting Started (SIMDAT Roadmap), Information Society Technologies

[142] InforSense's KDE *(Knowledge Discovery Environment)*. 1999,[cited; Available from: **Error! Hyperlink reference not valid.**.

[143] Mandal, N, Deelman, E, and Mehta, G.2007, *Integrating existing scientific workflow systems: the Kepler/Pegasus example*. in *The 2nd Workshop on Workflows in Support of Large-Scale Science*. Monterey, California, USA: ACM.

[144] Globus, 2002, *The Replica Location Service (RLS)*.[cited; Available from: http://www.isi.edu/~annc/RLS.html].

[145] Clarke, I., Sandberg, O., Wiley, B., et al.,2001, *Freenet: A distributed anonymous information storage and retrieval system*. p. 46-66.

[146] Kubiatowicz, J., Bindel, D., Chen, Y., et al., *Oceanstore: An architecture for global-scale persistent storage*. 2000, ACM. p. 190-201.

[147] W3C, *Extensible Markup Language (XML)*.[cited; Available from: http://www.w3.org/XML/].

[148] Huang, Y and Gannon, D.2006, *A flexible and efficient approach to reconcile different web services-based event notification specifications*. in *Proceedings of the IEEE International Conference on Web Services*. Washington,DC,USA: IEEE Computer Society.

[149] Web Services Axis. 2006,[cited; Available from: http://ws.apache.org/axis/].

[150] Harrison, A, 2006, *The WSPeer Project*.[cited 2006; Available from: http://www.wspeer.org/].

[151] SunMicrosystems, *JavaServer Pages Technology*.[cited; Available from: http://java.sun.com/products/jsp/].

[152] Alqaoud, A, 2009, *Scientific Workflow Interoperability Framework*.[cited; Available from: http://swif.cs.cf.ac.uk:8080/].

[153] Williams, A, 2008, *A workflow version of the EMBOSS tutorial*.[cited 2008; Available from: http://www.myexperiment.org/workflows/159].

[154] Rice, I, Longden, P, and Bleasby, A, 2003, *The European Molecular Biology Open Software Suite (EMBOSS)*.[cited 2008; Available from: http://emboss.sourceforge.net/what/ .].

[155] Alqaoud, A, 2009, *PS-SWIF Web Services*.[cited 2009; Available from: http://swif.cs.cf.ac.uk:4804/wspeer/].

[156] OASIS, 1993, *Advancing open standards for the information society*.[cited 2009; Available from: http://www.oasis-open.org/].

[157] Repici, D, *The Comma Separated Value (CSV) File Format* [cited 2009; Available from: http://www.creativyst.com/Doc/Articles/CSV/CSV01.htm].

[158] W3C, 2005, *SOAP Message Transmission Optimization Mechanism*.[cited; Available from: http://www.w3.org/TR/soap12-mtom/].

[159] Moreau, L, Plale, B, Miles, S, et al., 2008, The open provenance model, University of Southampton

[160] Provenance Challenge.[cited; Available from: http://twiki.ipaw.info/bin/view/Challenge/WebHome].

[161] Moreau, L., Freire, J., Futrelle, J., et al.,2008, *The open provenance model: An overview*, in *Provenance and Annotation of Data and Processes*, Springer: Berlin,Heidelberg. p. 326.

[162] Moreau, L, Ludaescher, B, Altintas, I, et al., *Special Issue: The First Provenance Challenge*. 2008. p. 409-418.

[163] Pike, R and Ritchie, M,1999, *The Styx® architecture for distributed systems*. Bell Labs Technical Journal. 4(2): p. 146-152.