PROVENANCE SUPPORT FOR SERVICE-BASED INFRASTRUCTURE

Shrija Rajbhandari

School of Computer Science
Cardiff University

This thesis is submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

· April 2007 ·

Declaration

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

Signed .... (candidate) Date 17/12/2007

STATEMENT 1

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

Signed .... (candidate) Date 17/12/2007

STATEMENT 2

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

Signed .... (candidate) Date 17/12/2007

STATEMENT 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for interlibrary loan, and for the title and summary to be made available to outside organisations.

Signed .... (candidate) Date 17/12/2007

Abstract

Service-based architectures represent the next evolutionary step in the development of e-science, namely, the transformation of the Internet from a commercial marketplace to a mechanism for sharing multidisciplinary scientific resources. Although scientists in many disciplines have become increasingly reliant on distributed computing technologies for data processing and dissemination, the record of the processing history and origin of a data product, that is its *data provenance*, is often nonexistent, incomplete or impossible to recover by potential users. This thesis aims to address data provenance issues in service-based environments, particularly to answer how a scientist who performs a workflow execution in such an environment can (1) document the data provenance for a data item created by the execution, and (2) use the provenance documentation as a recipe to re-execute the workflow. This thesis proposes a provenance model for delivering data provenance support in a service-based environment. Through the use of an example scenario of a scientific workflow in the Astrophysics domain, we explore and identify components of the provenance model. The provenance model proposes a technique to collect and record data provenance for service-based workflow executions. The technique facilitates the collection of data provenance of workflow execution at runtime. In order to record the collected data provenance, the thesis also proposes a specification to represent provenance to describe the processing history whereby a piece of data was derived. The thesis also proposes query interfaces that allow recorded provenance to be queried, has formulated a technique to construct provenance graphs, and supports the re-execution of past workflows. The provenance representation specification, the collection technique, and the query interfaces have been used to implement a prototype system to demonstrate the proposed model. The thesis also experimentally evaluates the scalability of the components implemented.

Contents

List of Figures

List of Tables

Acknowledgments

I thank my first supervisor Professor David W. Walker for his help and advice throughout the PhD process. I also like to thank him for his insightful comments and feedbacks and for always being wonderfully positive of my work. He has always been there whenever I needed any help. I am very grateful to him for reading my thesis many times - despite his many professional commitments. I could not have completed this thesis without his help.

I would like convey my most special gratitude to my second supervisor Professor W.A Gray. He created an opportunity for me so I could do PhD in the School of Computer Science, Cardiff University. I want to thank him for the opportunity, because I could not have achieved it without his support and leadership.

I thank Dr. Omer Rana for being incredibly understanding and supportive during my PhD write-up process.

I thank Dr. Ian Taylor for hiring me as a student researcher for the Triana project that allowed me to undertake this research.

I owe many thanks to the administrative and technical staff of the School of Computer Science, Cardiff University, particularly Mrs. Margaret Evans and Mrs. Helen Williams for their support.

I thank my partner Vikas Deora for helping me and having confidence in me even when I didn't. I also like to thank him for taking good care of me during difficult times.

Finally, I thank my father Shiva Rajbhandari and my mother Rupa Rajbhandari and dedicate this achievement to them for their foremost support. Their blessings and encouragement have enabled me to be positive.

Chapter 1

Introduction

1.1 Motivation

Multidisciplinary scientific research is often both data intensive and distributed. Scientific investigation and processing relies as much on the effective broadcasting of data between dispersed study sites and collaborating groups as on the conclusions of written publications. In recent years, many scientists have become increasingly reliant on distributed computing technologies as an essential part of their everyday research for data processing and dissemination. Thus, the increasing trend towards the sharing and communication of scientific data is evolving with the growing data processing capabilities of computing research environments. This is contributing to an increase in the propagation of data in various scientific disciplines. Increased transparency in access to data has made researchers realize that the essential documentation of derived scientific data shared online is often incomplete or inadequate. This makes electronically published scientific journal articles the only source of annotations or descriptions of the studies or experiments carried out to produce such scientific data.

Although the concept of sharing distributed scientific data and resources amongst geographically distributed groups is not new, the increasing adoption of Service Oriented Architectures (SOA) based on Grid and Web Services [35, 48, 56] makes the vision of automatic discovery, composition, and consumption of distributed resources more realistic, and support the benefits of Internet standards and common infrastructure to produce optimal efficiencies for intra- and inter-organization computing [54], compared with traditional approaches. SOA technologies have made feasible the use of distributed and heterogeneous resources for scientific disciplines, such as Bioinformatics [32], Astrophysics [75], and Earth Science [98]. Many research scientists make use of such resources in their experimental workflows by using innovative workflow management systems. The concept of workflow, applied to scientific computing, is concerned with the movement of data and the execution of tasks (e.g., on distributed resources) through a work process [106]. This describes how tasks are structured, what their relative execution order is, and how data flows between tasks. An example is a computational experiment performed by a biodiversity scientist that allow the prediction of how species will be distributed under changing climate [60]. In Figure 1.1, given a set of locality data for a species, a climate preference profile is produced by referring to present day climate data to produce a 'climate envelope'. This is then used with a specific selected Open Modeller (OM) algorithm by interpolating the climatic data at the points of locality of specimens producing a bioclimatic model. Such distributions projected upon a world map allow the determination of where a conservation priority area should be in the future for that species. When

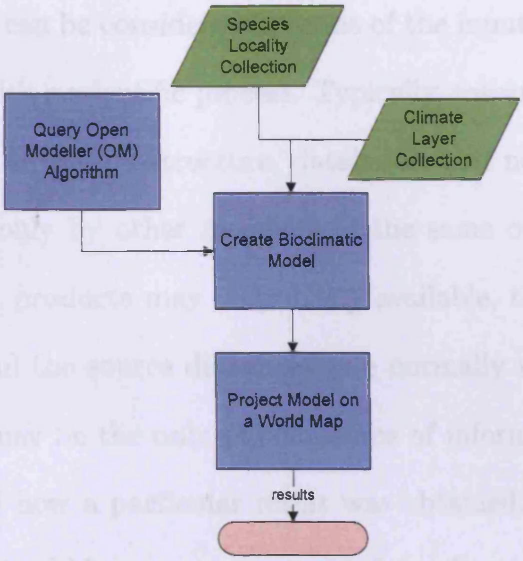

Species Locality Collection
Query Open Modeller (OM) Algorithm
Climate Layer Collection
Create Bioclimatic Model
Project Model on a World Map
results

Figure 1.1: An example of Bioclimatic Modelling workflow from [60].

performing such workflows, scientists may wish to (1) reuse a workflow and data; and, (2) at some point share the produced data with their fellow researchers. Usually such data products are published with metadata that includes the format of the data and a description of what the data represents. This helps researchers, as well as others, to discover and access the archived data products. The appropriate use of such derived scientific data, and the reuse of the workflows that generated the data, relies on understanding the origin and the processing history of the data, that is its *provenance.* Just as a genealogical chart provides documentation that reveals the ancestry of an individual, the provenance of an item describes how it was derived from its source. *Data provenance* refers to the documentation of the processing history of the executed workflow that led to a particular data product.

A scientific article may be a part of the data provenance since it describes the output data's provenance, for example the methods by which such data has been produced. Provenance can be considered in terms of the input dataset and the resultant data associated with a scientific process. Typically, researchers maintain private records of provenance in the file structure, databases, and notebooks which are, in many cases, sharable only by other members of the same organization or project. Although derived data products may be publicly available, the scientific tools, programs or processes, and the source data used, are normally not available for public use. Journal articles may be the only public source of information about the origin of a data product and how a particular result was obtained. We believe that such published documents should be provenance-enabled, to allow scientific journal articles to become partly dynamic. For example, somebody reading an article online would be able to rerun a simulation for which results are presented, possibly with different input parameters, by clicking a button in the provenance-enabled article (see Figure 1.2). The reader of the article can actually use such a capability to re-execute the work, and view and evaluate the results produced on-the-fly, without being aware of how the results have been re-created. We refer to such an interactive document as a "Living Document" [103]. In order to make such a concept a reality, researchers must furnish data products with a special type of metadata that provides an understanding of the processing history that can be used to re-execute the process online. Such metadata, once composed appropriately, provides a view of the processing chain through which the data was derived, i.e., the data provenance.

This vision of a "Living Document" has motivated the research presented in this dissertation. The aim is to capture, represent, and manage the provenance information for a data product so that the provenance can be used to provide a "recipe"

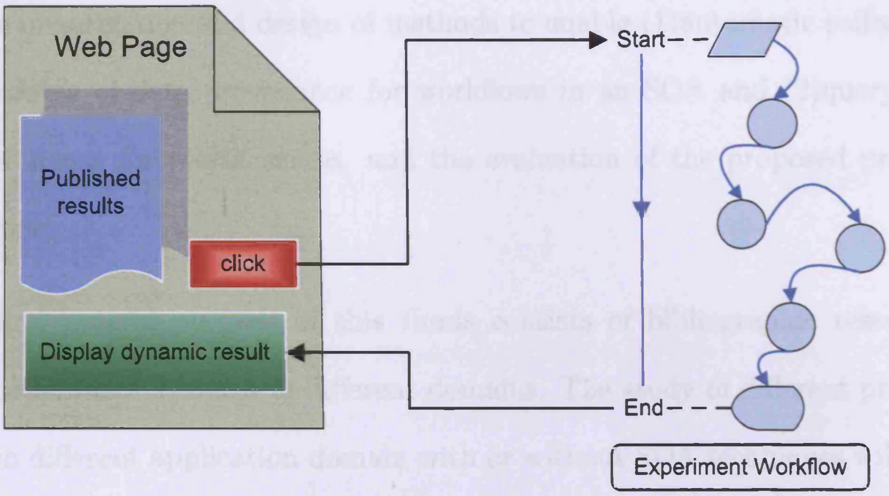


Figure 1.2: The Living Document Concept

for future experiments relating to that data, and also enables the re-execution of the scientific processes that originally created the data in a service-oriented framework.

1.2 Research Objectives and Approach

The preceding discussion illustrates that data provenance is a set of important information that needs to be retained in any scientific experiments in such a way that it can be used to reproduce the original results. To make the "Living Document" concept concrete, a provenance system that caters to a service-based environment is needed. Thus, the objectives of this thesis include the achievement of the following goals:

- The study of existing provenance systems in various scientific application domains.
- The application of provenance systems to workflow enactments in SOAs.

- The investigation and design of methods to enable (1)automatic collection and recording of data provenance for workflows in an SOA and (2)querying such provenance for re-executions, and the evaluation of the proposed provenance system

The first general purpose of this thesis consists of bibliographic research into existing provenance systems in different domains. The study of different provenance systems in different application domain with or without SOA techniques will provide us with a general view of data provenance and provenance systems. The benefits and use of data provenance need to be identified and analyzed in order to evaluate provenance system requirements. This will encompass how a provenance system would best fit within the conceptual vision of an SOA by presenting a provenance system as a Web Service. This directs us to the main research objectives as follows:

1. To design a *Provenance Model* to represent the functionality for: (1) a *provenance collection service*, and (2) a *provenance query service* of a provenance system in an SOA.

2. Modelling provenance (i.e., producing a *provenance format* or *p-format*) for the structured representation of provenance for workflow executions in an SOA.

3. Use of the provenance collection service component to experiment with automated techniques to collect and record the provenance of derived data from a workflow execution.

4. Investigate methods to *browse* and *query* the provenance to be used in the

provenance query service for process re-execution and recreation of the processing chain.

5. Perform evaluations of the functionality of the provenance system to better understand its performance and scalability.

In order to meet these objectives, we make use of the Web Services since it gives us a suitable framework to achieve our purpose of enabling support for data provenance in a service-based distributed environment. The work is presented with an example workflow scenario that represents data analysis and processing in the Astrophysics domain.

1.3 Research Contributions

The main focus of this thesis is to support data provenance by handling automatic recording of composed Web Services executions, and to query such provenance to recreate and re-execute the past process.

The primary contribution of this thesis is the provenance model which incorporates provenance support within Web Services. The model consists of two service components; data provenance (1) collection/recording and (2) querying. We identify metadata that incorporates the provenance documentation, also showing that it is possible to represent and record such provenance for a process execution in a way that it can be queried to re-execute past workflows and construct a provenance graph displaying the processing chain of the derived data. We also allow the intermediate data of an executed workflow to be captured to support enhanced querying capabilities. By evaluating our approach, we have provided a scalable and structured way of automatic collection and recording of data provenance that represents the processing history of a process.

1.4 Outline of the Thesis

Chapter 2 details the study of existing provenance aware systems. In particular, eight provenance systems have been analyzed. The analysis has resulted in the identification of provenance requirements in various types of application processes and domains. The analysis also outlines common benefits arising from the use of data provenance in many application areas. Thus, this chapter first provides us with the descriptions of different provenance systems before formulating our proposed approach. Five of the provenance systems are also evaluated based on a set of criteria (i.e., based on the system's operational model and characteristics). Based on the study carried out, this chapter also presents evaluations to propose a provenance system for an SOA environment. It presents a higher vision of a provenance support framework within an SOA with the notion of exposing a provenance system as a Web Service that can be consumed.

Chapter 3 describes our proposed provenance model based on an SOA. An example scenario of a scientific workflow in the Astrophysics domain is presented to identify and explain the components of the model in the preceding chapters. We identify and describe different classifications for the representation of provenance in a way that provides the processing chain that produced a piece of data. This chapter also presents the high level interactions between the model components for capturing, recording and querying data provenance.

Chapter 4 describes how the different classifications of the representation of provenance in the previous chapter can be modelled, i.e., we define the data structure used to represent the classified types. Based on the modelled structures a common format is produced that represents the documentation of a process that is recorded in the provenance archive. Using the example scenario presented in chapter 3, a simple process is presented in this chapter to describe the provenance capturing and recording mechanism with use of the presented provenance format.

Chapter 5 describes the querying component of our provenance model. The construction of the provenance graph and the re-execution of past processes are discussed. This chapter also presents some examples of how different provenance questions can be queried for with the proposed provenance format.

Chapter 6 presents the software implementation that provides the functionality discussed in the previous chapters. This is followed by chapter 7 that provides an evaluation of the scalability of the implementation of the provenance collection and recording components. Both are illustrated using the enactments of the workflow implementation of the example scenario described in chapter 3.

Finally, chapter 8 concludes this document by restating the main contributions of this thesis and presents proposals for further work. A list of related publications are also included.

Chapter 2

Literature Review

2.1 Introduction

The issue of data provenance is not a new problem, as is evident from the large body of related research in the past few years and which has led to prototype systems that archive and retrieve the provenance of processed data. Provenance-related research in various scientific domains where usefulness of provenance is linked to the granularity at which it is collected has notably increased in recent years [3, 84, 91, 112].

This chapter provides a review and analysis of the importance and use of provenance in various domains and application areas. The criteria for this evaluation are intended to facilitate a coherent understanding of the operations and interactions in the different classes of provenance-enabled models and systems. Table 2.1 summarizes eight major research projects on provenance-enabled systems with their example domains, application type, and provenance applicability. In the literature, some provenance systems are domain-specific; some systems differ in terms of the granularity at which provenance is collected; and others are specific to a particular application architecture, such as database systems and semantic search engines. In reviewing the literature we categorize provenance in terms of:

1. Query-based Systems.

2. Domain-specific Systems.

3. Provenance Middleware.

This chapter provides a detailed discussion of provenance in the context of related work. Furthermore, we discuss provenance granularity and the potential benefits of provenance gathered through the evaluation of different provenance-enabled applications and systems. The chapter is organized as follows. Sections 2.2, 2.3 and 2.4 discuss the relevant research that falls into the above three categories, respectively. Section 2.6 outlines the importance of provenance and its use in light of the literature discussion. Section 2.7 presents a brief introduction to SOA and a comparative evaluation of some of the relevant provenance systems in terms of their operational model and characteristics. This survey highlights the finding that most current provenance systems that cater to the SOA framework are either representative of a particular domain application or inadequate in essential system characteristics. Section 2.7.1 discusses the key areas of development necessary for the establishment of a provenance system in the context of emerging technologies and standards for an SOA environment. In particular, the entities involved in the development of a provenance system, and the high level interactions between them, are identified.

	Domain	**Application Process Type**	**Provenance Use and Benefits**
Chimera [1]	Astrophysics	Service-based workflow enactment	Audit trail and data replication
CMCS [2]	Chemical Science	Informatics-based chemistry research	Information about data products and updates
ES^3 [44]	Earth Science	Script-based workflow enactment	Data lineage information
myGrid [3]	Bioinformatics	Service-based workflow enactment	Re-enactment of workflows and updates
PASOA [4]	General	Service-based workflow enactment	Data lineage information based on asserted causality relationships
Inference Web [104]	General	Query-based in information retrieval	A form of justification and placing some degree of trust in the results
Tioga [11]	General	Database query	Weak inversion to investigate faulty and anomalous data
Trio [105]	Earth Science	Database query	Update propagation and efficient warehouse recovery

Table 2.1: Major Provenance Systems

2.2 Provenance in Query-based Data Processing Systems

Data Processing refers to the means by which processes consume and manipulate data sources, in order to bring about the transformation of the data product. Provenance information for a data product revolves around the two main concepts; the original data sources and the transformations that they underwent to generate the data product. In any systems architecture, data is always being consumed, processed, transformed and copied. In query-based systems, provenance issues are focused on tracing the provenance of data items where the data item is produced or retrieved by querying data archives. The use of provenance in two different query-based systems will now be summarized.

Provenance in Database Systems

The purpose of lineage information about a data item is to find the source data that produced it [108]. Provenance in database systems focuses on the problem of data lineage. The problem of data lineage for a given data item can be summarized as determining the original source data items from which the data item was derived, and the processes by which it was produced [40]. The data lineage problem is relevant in data warehousing systems where the source data goes through a series of transformation steps during analysis and mining to perform data integration, and is modelled as queries over multiple data sources. Cui and Widom [39] focus on tracing the lineage information of materialized data views and general transformations of data items in data warehouses stored in relational databases. A materialized view assembles or represents data contained in other database tables and views, but unlike a database view it contains actual data. In a data warehousing system, since the remote data is cached at the warehouse, the remote data appears as local to the users of the warehouse. Queries are written in terms of materialized views to perform analysis on the warehouse data. For example, consider an analyst wanting to build a warehouse table listing computer products that had a significant sales jump in the last quarter. For this, a complex query needs to be executed on specific warehouse data derived from, for example, Product and Order source tables. That is, the warehouse defines a materialized view Sales Jump, where the view definition is expressed as an SQL query. Using the source tables as inputs, the query performs a series of transformations to produce the Sales Jump table, i.e., the result table. Cui and Widom's work revolves around developing lineage tracing algorithms that can automatically determine the source data from tuples in the result table (a tuple is a row in the table, e.g., a row for a product 'Sony VAIO laptop' in the Sales Jump table). They propose to do this by storing additional relevant information from the query and using it for lineage tracing, i.e., the lineage tracing algorithm uses stored information to execute lineage tracing queries. Thus, the work focuses on a data item lineage derivation where view tuples of interest can be traced down to the sources, e.g., tuples and tables [40]. These lineage tracing algorithms are integrated as a part of the Trio [105] data model, an extended relational DBMS.

Apart from lineage tracing of data items to their source, some important aspects of fine-grained lineage tracing are described by Woodruff and Stonebraker [108]. Through fine-grained lineage tracing, their research aims to provide a capability for scientists to identify and investigate the source of errors and anomalies in data items. They particularly address the problem of tracing the origin of a single element in large arrays of data that went through a series of transformation. For example, to know which pixels of which images were used to construct a given image. Woodruff and Stonebraker proposed methods that are performed within a DBMS. This type of fine-grained lineage processing has been implemented in the Tioga [11] database visualization tool, which is built on top of POSTGRES DBMS with a "drag and drop" approach for programming and managing databases. To enable fine grained lineage tracing, all the user-defined algorithms and their additional functions are registered and stored in Tioga. The additional functions registered are the weak inversion and verification functions that are processed within the DBMS to execute lineage queries. An inversion approach similar to that described in [40] is adopted, and is termed the weak inversion technique for lineage tracing. A weak inversion technique is intended to provide an "approximate lineage" because not all the functions and algorithms are completely invertible. This implies that the weak inversion function provides a flawed but still useful mapping from the output of a given function to the database element input. For functions that cannot be inverted without referring to input data, a verification function with access to the input data for the original function is introduced which is applied to the output data for further refinement and mapping.

Buneman et al. provide an assessment of the issues of data provenance and annotation in shared and distributed scientific databases [29]. Scientific databases are usually "curated" by adding annotations, classifications, and error correction through human intervention. Some databases may contain data items that could be copies of some source database or created by processing the source database. These data sources are likely to be frequently updated. Buneman and his co-authors argue that in this case, the curated source database could not notify other databases when updates to the source occur, nor could the databases that are copied or created from the source be aware of any updates made to the source. Both the source and the copied or created databases lack the ability to track the change histories, provide annotation support, and broadcast such information across all records that are related in some way. Provenance and annotations are necessary in an environment in which data are repeatedly copied and transformed, so the processed data items can be tracked back to the curated source databases, as well as to propagate the annotations on the derived data back to the source database [31]. In [29], the authors argue that such shared scientific databases that change over time require coordination between the interacting databases to maintain data consistency and traceability.

Buneman et al. [30] have defined two terms: "*Why-Provenance*" and "*Where-Provenance*". *Why-Provenance* is the data lineage that provides the reason a particular data item was generated, and specifies what part of the database contributed to its creation, i.e., a set of tuples, and why the source data is in the database. "*Where-Provenance*" is the location of the source data items that created the item of data. Buneman et al. propose a deterministic data model that allows the unique identification of a piece of data through a *path* [27]. This model provides an explicit notion of location that helps to describe the *where-provenance.* The authors have presented research issues and limitations of provenance in current DBMS techniques, and discuss them in terms of scientific databases in application domains such as molecular biology, linguistics, and ecology where the provenance problem is a challenge.

Provenance in the Semantic Web

Provenance, as it relates to the Web, refers to the explanation of the information returned by a web application. The users are unaware of the sources from which the information is derived. The provenance answers questions such as what sources were used, when they were updated, if the information was derived and, if so, how it was derived, and can the user rely on the sources. This type of information is needed to understand and trust the information [70].

McGuinness and Silva [71] point out the lack of support for provenance in the current Semantic Web, and introduce the Inference Web (IW) that addresses the problem of provenance by providing explanations of the answers queried over the Semantic Web. The Inference Web (IW) aims to provide an infrastructure by which the query answers are made transparent, with explanations that describe the path that derived the answers as users may obtain query answers from systems that manipulate data and derive information that was implicit rather than explicit. [71] provides various sets of requirements for the development of IW infrastructure, and [70] defines users of the IW, such as retrieval engines, hybrid programs such as crawlers, merging ontologies and combining knowledge-based systems. The Inference Web provides a Proof Markup Language (PML) based on the OWL specification to represent knowledge provenance or meta-information of different sources used to derive a query answer and derivation history [72]. A web-based registry for storing, manipulating and returning such knowledge provenance and derivation history that is used to enhance explanations is also presented and provides a set of APIs to convert the long and complex PML to a short understandable explanation. A Semantic Web based inference or search engine is used in support of the Inference Web that presents the knowledge provenance and derivation history as proofs and explanations of any responses to queries.

McGuiness and co-workers have extended their work on the Inference Web towards the notion of trust and justification of the answers retrieved from the Web. They argue that the source *meta-information* given with the retrieved answer, which is used to provide explanations, may not be enough to explain the processing steps that generated the answer. In this case, the importance of the reasoning process used to generate the answer, termed the *knowledge process information*, was recognized [73]. Providing the answer with optimal additional information about the sources that were used, the basis on which the answers were selected, and the process used to generate the answers would be essential for the user to trust the answers. A trust infrastructure IWTrust is introduced in [110] where the authors discuss how trust values of the sources with the meta-information can provide a better justification and trustworthiness of the answers generated on the Web.

2.3 Provenance in Domain-Specific Applications

Many research projects focusing on provenance seek to improve scientific collaboration by means of computing environments that capture a generic experiment. Such provenance research usually caters to a specific domain and application system. In the early 1990s, some of the major work on provenance was in the area of Geographic Information Systems (GIS). Lanter [65] contributed to the design of a meta-database for recording GIS procedures and retrieving the lineage of data products within GIS applications. Knowing the quality of a result dataset is critical in GIS. This can be determined via lineage tracing to the source dataset that was used to derive it [66]. This helps GIS users to determine the fitness of use of the data for their application. The most important GIS operation is the overlay of different spatial data sets (e.g., stored in a database system) to produce a new data set. For example, a biologist might want to determine what variables affect the population of dolphins [79]. A GIS enables information to be associated with a feature on a map and the creation of new relationships that can determine the suitability of various sites for development, evaluate environmental impacts, and so on. Lanter's Lineage Information Program (LIP) provides lineage tracing for GIS operations known as Arc/Info (where, ARC handles where the features are on a map, while the INFO component handles the feature descriptions and how each feature is related to others [45]) by examining user input at the command line and from a graphical user interface. Lanter also explored the use of data lineage to optimize the size of spatial databases [66], compare spatial analyses of GIS applications [67], and examine the propagation of error through GIS applications [64]. Another system that incorporates lineage tracing for GIS processes is the Geo-Opera workflow-based system [13]. In Geo-Opera, the data files and transformations (GIS programs or scripts) are distributed and reside outside the system. Such transformations, or *external objects*, are registered in the Geo-Opera system before they are executed and tracked as *task objects.* The relationships between internal task objects are obtained by using control flow connectors to set the order of execution. The system provides lineage recording by using data attributes to point to the latest inputs/outputs of a data transformation.

The Earth System Science Workbench (ESSW) [50] project , now called Earth Science System Server (ES^3), proposes a data storage and management infrastructure which allows researchers to publish their large data sets from environmental models and global satellite-derived image data [51]. The workbench provides a framework for defining and collecting earth science metadata which is based on a conceptual core composed of *science objects.* These are processes, processing steps, science models, inputs and outputs. In ES^3, the meta-model is for tracking the documented processing history of experiments or workflows (referred to as *lineage metadata*). The project has built a client-server application called Lab Notebook that stores the lineage metadata for experiment steps and their associated science objects. A scientist needs to define metadata templates formed of DTD to define XML instance document to publish science objects which are specific to ES^3's fixed set of science objects.

In ES^3 the processing of data products, and the metadata and lineage recording, is based on scripts. ES^3 depends on the script writer to use the templates and libraries to record the metadata for workflow runs. The data products are produced via the scripts that transform the input data (i.e., binary files), where the input data products and scripts are referred to as *software objects.* Each software object has a uniquely identifiable *metadata object*, that contains the details about the software object. The *metadata objects* exist separately from the software objects, so that the same metadata objects can be referred to for different workflow invocations that use the same software objects [23]. Thus, a metadata object corresponds to each software object in a workflow invocation and produces a new data product for which a metadata object is also created. The metadata objects about a workflow invocation are recorded in an XML format in such a way that the lineage of a particular data item can be traced through the parent-child relationships of metadata objects.

In [33], a scientific resource management (SRM) architecture is proposed for managing the distributed scientific resource metadata for an environmental system. Its aim is mainly to publish scientific data and programs thereby making them available on the web. Also the experiments, or the scientific workflow carried out using these programs, and the data need to be registered as well on the "experiments database". The architecture is currently considering the use of Web Services – publishing the programs as Web Services and capturing experiment provenance by keeping track of SOAP messages. The Scientific Publication Model (SPM)[34] is the meta-model (schema) behind the SRM architecture, and is used to provide a semantic representation of scientific resources (data, programs), describing the programs and the associated theory behind it as "the model".

The Collaboratory for Multi-scale Chemical Science (CMCS) project provides a multi-scale informatics toolkit that focuses on "on-demand" metadata creation to support the collaborative management of data, metadata, and data relationships. Generic tools have been developed in the CMCS project to view and browse provenance relationships, and use them for scope notifications and searches. CMCS uses the Scientific Annotation Middleware (SAM) (see section 2.4) project tools for provenance management. There is no facility for the automated collection of provenance from workflow execution in CMCS. The provenance, or lineage, is collected via DAV-aware applications in the workflow, or entered manually by the scientists through a web interface [74], and stored in the SAM repository. Provenance properties can be queried from SAM using generic WebDAV [7] clients.

The myGrid project provides middleware application tools to support *in silico* experiments in the biology domain modelled as workflows in a Web Service environment [3]. *In silico* experiments use databases and computational procedures, rather then laboratory experiments. The middleware developed in myGrid is a set of bioinformatics-specific scientific services that provide for data and computational analysis. myGrid includes resource discovery, semantic descriptions of services, workflow enactment, and metadata and provenance management, thereby enabling the execution of complex bioinformatics computations in a service-oriented environment, and also addressing the semantic complexity of the domain. myGrid workflows are written in an XML-based langauge called XScufl, and executed using the open source FreeFlue/Taverna workflow engine [111].

2.4 Provenance Middleware and Provenance in Other Application Systems

Chimera [49] is the GriPhyN Virtual Data System (VDS) that allows virtual data products and procedures to be described, represented and discovered. Chimera supports the capture and reuse of the lineage of derived data ("virtual data") produced by computations. It captures the lineage in the form of derivation steps for datasets, and uses it for audit tracing, the comparison of datasets, and also to manage the automatic and on-demand re-derivation of derived datasets. Chimera supports data-intensive scientific analysis such as high energy physics simulations (for example, the Compact Muon Solenoid experiment at CERN), the search for galaxy clusters in the Sloan Digital Sky Survey [17], and genome analysis [91]. A set of web interfaces is provided in [112] to interact with the Virtual Data System to query, reuse, share, and trace the lineage of data products. This is being applied in a large collaborative learning project called QuarkNet [19].

The VDS architecture is based on the Chimera Virtual Data Schema where *transformation* elements describe programs, and the arguments describe data input/output. It presents a high-level language, the Virtual Data Language (VDL), that supports data definitions and query statements (for databases) for constructing workflows as directed acyclic graphs (DAGs), and which, when executed on a Data Grid, create a specific data product [49]. The VDL workflows are stored in a Virtual Data Catalog (VDC). The invocations of these workflow procedures are also recorded in the VDC with relevant runtime information about the process, and contain an annotation schema to represent the provenance. The VDL is also used to query the VDC to discover the lineage or computational pipeline that created a particular data object. The main purpose of maintaining such a description is for tracking how the data product was created, and to recreate the data product by recreating the DAG of distributed computations that can then be submitted to the Grid.

The Scientific Annotation Middleware (SAM) is a set of components and services that enable researchers, applications, problem solving environments (PSE) and software agents to create metadata and annotations about data objects, and to document the semantic relationships between them [63]. SAM allows applications to encode metadata within files or to manage metadata at the level of individual relationships, as desired. An Electronic Laboratory Notebook (ELN) is used with the SAM to develop an initial set of SAM-based notebook services to search and browse data and provenance information about data (such as static texts, images, and dynamic images), and also to add provenance about the data. The open source Electronic Laboratory Notebook is a collaborative, distributed, web-based notebook system, designed to provide researchers with a means to record and share their primary research notes and data. This project's aim is to enable the sharing of scientific records among portals and problem-solving environments, software agents, scientific applications, and electronic notebooks that includes annotations and data provenance about the recoded scientific data.

The Collaboratory for Multi-scale Chemical Science (CMCS) research project is one of the projects using the SAM for their pedigree implementation in Grid environments [74]. The CMSC brings together leaders in scientific research and technological development across multiple U.S Department of Energy (DOE) laboratories, other government laboratories and academic institutions to develop an open "knowledge grid" for multi-scale informatics-based chemistry research [97]. Provenance support is provided by adding metadata to files stored in a SAM repository, and SAM also provides configurable, automated metadata extraction and translation of uploaded resources. SAM publishes messages of events whenever a resource is accessed or modified in the SAM server under two topics, one for changes to the data or metadata and one for queries (e.g., a request to view a particular resource) [95]. SAM acts as an open storage system and does not stipulate any specific format for the data and metadata it handles. Thus, it provides an open sharable tool to record resources generated through, e.g., a PSE, and allows researchers to add different types of metadata and annotations about the resources [59]. The provenance about the workflow, or what procedures were invoked within the PSE to generate a particular data product, can either be manually recorded by the PSE user or may be automatically generated within the PSE and then recorded in SAM. Thus, SAM is a middleware storage system and does not participate in the extraction of metadata during the workflow invocation within PSEs or applications.

A middleware system is presented in [89] based on an e-notebook abstraction. The e-notebooks are distributed amongst the users in the research groups, and can record data, and its derivation and transformations, directly via manual user input or from connections to the instruments used. The data and provenance stored in an e-notebook server is represented as a DAG which can be shared with other e-notebook users. The DAG may have nodes representing multiple e-notebooks to show the many individuals participating in a process. When creating a node in a DAG to represent the derivation of a data item, the creator must digitally sign the node to provide support for trust views and credential tracking. This e-notebook approach to provenance recording, along with the users credential information, provides an interesting way of assessing the data's credibility.

2.5 Granularity of Provenance

In certain domains, the usefulness of provenance depends on the level of granularity of the process documentation which is collected by the system. Granularity of documentation refers to the level of detail with which provenance is recorded. A process here means any task performed whose lineage may be documented. In an experimental process, if all the instructions about the scripts used are recorded then this is a fine-grained documentation of the experimental process (compared with recording only the name of the script).

In [108] and [69], fine-grained provenance is recorded about attributes or tuples in a database that represents individual pixels or array elements, respectively. A technique is provided in [69] to trace the lineage in arbitrary array computations. An algorithm called *sub-pushdown* is used that requires the operations or algorithms used to produce a dataset to be described in terms of Array Manipulation Language(AML) operations. The sub-pushdown algorithm has been implemented in a prototype array database system called ArrayDB. In this system, the provenance of an array in ArrayDB can be retrieved. This answers fine-grain questions such as: what points in the intermediate datasets I_1, I_2 and I_3 contribute to a point in the derived dataset D_1?

2.6 Use and Benefits of Provenance

The motivation behind much research in the area of provenance is the benefits it provides to users in the application system domain. Provenance support in scientific computations allows, for example, the verification of derived data products, error propagation, and is a source of information to ensure the integrity and quality of data products. Based on the major benefits that provenance support provides in different application domains, we now outline these benefits under two categories.

2.6.1 Data Quality Benefits

The provenance of a data product can be used to estimate data quality and reliability based on the process that produced the data and the source data used in its derivation. A geographic metadata standard (that includes lineage specification) called the Spatial Data Transfer Standard (SDTS) [90] was produced in 1992 for transferring geospatial data between different application groups and geographic information systems (GIS). Here, the lineage information is attached to the data and transferred as part of a data quality report. Such lineage information is provided so that potential data users can be protected from faulty information and assumptions about the data transformation process, and misinformation on the accuracy of measurements. The propagation of source data errors through GIS data-transformation functions is the focus of [99], which describes lineage-based quality enhancement tools that can be used to improve the quality of derived data products. The issue of data quality becomes more critical as errors introduced by faulty data or misconfigured instruments can grow as they propagate to data derived from them. Errors made at a very low level may never be identified once the data has been integrated and replicated many times unless a detailed lineage is recorded.

In [18] a case is investigated in which the manipulation and misrepresentation of genome data has resulted in research carried out using this faulty data to be worthless. This formal investigation highlights the need for a data verification and integrity system in the research community. Genome data are the protein, DNA and RNA sequences that are annotated by bioinformaticians in academia and public research institutions to give meaning to the sequence data. The annotations are usually performed by referring to the annotations of similar sequences. The source of annotations is usually not recorded so any annotation error is likely to be propagated throughout the database [24, 42, 62]. Lineage metadata about the data, such as the transformations applied to create it or the source of its parent data, can assist the data user in establishing the authenticity of the data and avoid low quality data sources. It provides justification for using the data, enhances the interpretation of data, reduces false data precision, and broadcasts data reliability, accuracy, suitability and currency.

2.6.2 Data Processing Benefits

Using provenance to record the processing history can be beneficial in supporting audit trails, data quality control, and the detection of error sources and faulty data sources. This type of provenance also provides processing "recipes" that can be modified to rerun results from a complete or partial process chain, or to compare the analytical processes of two different experiments. This section describes the benefits of provenance in supporting the management of data processing in the business and scientific communities.

Audit Trails

Provenance serves as a means to perform an audit trial on a piece of data by tracking the origins of the interrelationships between, and the transformations performed on, the data as it moves through distributed processing steps that may cross organizational boundaries. Audit trails are essential for:

- ***Knowledge retention:*** Without provenance, every time a scientific research project or contract ends, critical information is lost about what has been done in terms of computational and laboratory-based research experiments. Such information is important as a reference point for future research and for sharing and using knowledge from the project. Provenance also helps in the version management of data products. For example, provenance helps identify reasons for variations in different versions of data products derived from the same source data at different times.

- ***Impact analysis:*** Any changes in the algorithms used for a computational experiment, or in a laboratory environment, can ripple across the entire procedure. A provenance audit trail permits a big-picture view, and allows an assessment of what must be done to accommodate this change. One recurring use of provenance is to backtrack and locate the source data, or a point in the process, that is the cause of errors found in derived data and apply relevant corrections [53].

- ***Regulatory or industry requirements:*** Pharmaceutical companies must guarantee data has not been corrupted moving from one system to another. Provenance information is important particularly in patenting drug discoveries. Financial organizations must establish auditing to trace the fate of every penny. E-business service providers must protect the privacy of customers. Such certainty is not possible without recording provenance and a comprehensive audit trail.

Process Repetition Recipes

Provenance information containing the processing steps and the source data provides a recipe to recreate a scientific workflow or experiment, and thereby to recreate a data product. If the provenance record contains sufficient context information related to the data's collection and transformation, such as the algorithms and instruments used and their configuration, it may be possible to repeat the data derivation procedure. Repeatability requires the availability of resources similar to those used when the original data was created. The derivation may be repeated to maintain the currency of derived data when some of the source data changes, or if the processing algorithms were modified or updated. It is possible to control such re-execution by only repeating sections affected by the changes in data or operations [21]. Such recipes generated from provenance information are also advantageous and convenient when modified to suit the current processing needs and to rerun the process sequence [108]. Modifications made for the purpose of re-enactment can involve changes to instrument configurations or to input parameters, or may use different source data to perform a comparison or "what if" analysis. Re-execution also works in similar ways in the maintenance of views in data warehousing systems, where following any changes in source tuples, database views derived from underlying source tables and views need to be updated [47]. In some cases it may be cost-effective to maintain provenance for "on-the-fly" data replication, but in some cases the re-execution cost is too high and time-consuming to justify the large amount of data processing.

In summary, provenance promotes the repeatability and reproducibility of experiments. Scientific experiments are often repeated, thus interesting results are generated during more than one run of an experiment. A sufficiently detailed record of the data derivation path, that includes large amounts of metadata and intermediate results, would allow:

1. Other researchers to repeat and validate the experiment.

2. The author of the experiment to repeat the process with different configuration parameters.

In repeating a previously completed process, provenance could be used to apply exactly the same methods, steps and resources, but supplying different configuration parameters to process either the original data, or other input data. Reproducibility of an experiment on the other hand is possible when exactly the same configurations (original raw data from the same version of the database, same tools, same algorithms and versions) are applied to produce the exactly the same results.

Informative documentation

Provenance is the documentation of a process, providing information on derived data that can then be the basis for information discovery and sharing. Data of interest can be located by queries on its provenance, and the effort of repeating a process can be avoided if the same derivation has already been performed. Archiving annotations along with provenance can help to interpret the data in the context it was intended, especially for derived data that may be used some time in the future [61]. Annotations by third party users of the data and its provenance could have added benefits in better understanding the data and processing steps [21]. This gives a clear understanding of data that is specific to the user's application domain. Provenance can also be browsed as a derivation tree, or in other graphical forms, and act as a starting point for exploring other metadata about the data and processes.

Thus, the most obvious importance and use of data provenance is the dissemination of knowledge. The ability to share the techniques and procedures of experiments within a domain is valuable for scientists working in that domain. This gives research scientists a new paradigm for sharing distributed scientific resources. Using the provenance of past experiments to learn from history and apply best practice helps scientists to design and analyze their own experiments. An example is a scenario taken from the Centre for Proteomic Research [77], where multiple experiments with different configurations are conducted to successfully identify proteins from a given sample. Provenance of such experiments would ideally inform later experiments about the sample material, by providing information on the configuration parameters of laboratory machines and the process that lead to the successful protein identification. Furthermore, provenance contributes a great deal to scientists when sharing a result dataset. For example, when a scientist would like to study a derived dataset, the provenance on how, when, and by whom that data was produced is vital in assessing the integrity of the dataset.

2.7 Service Oriented Architecture and Provenance

Service-based infrastructures are at an early stage of evolution and are emerging as the next phase in supporting e-science and e-business. Such services are generally referred to as e-services, where the main idea is to encapsulate an organization's functionality within an appropriate interface and advertise it as independent Web Services [76]. Widely known as an SOA, this is an information systems architecture that enables the creation of applications that are built by combining loosely-coupled and interoperable services. The architecture is not tied to a specific technology and may be implemented using a wide range of technologies including RPC, CORBA, Web Services [35].

While in some cases a Web Service may be used in a stand-alone form, it is normal in the context of an SOA to combine and link several Web Services together to create a new functionality in the form of a Web process or applications. Using such a composite process supplies a desired outcome or encapsulates a business process. In the context of e-science, desired outcomes could be large sets of statistical data. Due to the distributed nature of the process of computation from a collection of information resources and Web Services, it becomes hard to keep track of how and where a certain piece of data has been derived. This creates the need for data provenance support in Web process execution in SOA environments.

As provenance support in service-based infrastructure is a relatively new area, to our knowledge there are no current standards for data provenance support of Web Services. With reference to the literature discussion presented, it would seem that every research group involved in the evaluation and analysis of any provenance-enabled, service-based workflow system makes use of provenance infrastructure that is specific to the domain requirements and scenarios of that research project. It appears highly unlikely that different groups follow a similar approach during the system design and development process. Table 2.2[1] outlines the evaluation of some of the provenance systems based on a set of criteria. The criteria are based on the provenance systems' operational model and characteristics. It can be seen that the predominant approach adopted by them is not truly representative of a dynamic framework, and some research lacks either domain independency in terms of their provenance data model or support for workflow re-execution. The criteria are set to achieve the requirements for the provenance system modelling and development in this thesis. Those requirements will be elaborated in section 3.2. The research

[1] Here, $\surd$ means the criteria is present, X means the critera is not present and — is unknown

	Chimera	CMCS	ES^3	myGrid	PASOA
Service-Based	$\surd$	X	X	$\surd$	$\surd$
Domain Independent	X	X	X	X	$\surd$
Abstract Composing of Workflow	$\surd$	—	$\surd$	$\surd$	X
Run-Time Recording	$\surd$	—	$\surd$	$\surd$	$\surd$
Workflow Re-Execution	$\surd$	X	X	$\surd$	X

Table 2.2: Provenance System Evaluation against a set of Criteria

presented in this thesis intends to meet all the outlined criteria.

2.7.1 Identifying Specific Tasks in a Provenance System

Having specified the use and benefits of provenance and the criteria for designing a Provenance System, the next step is to specify the tasks that the user wishes to perform, aided by the provenance system. These tasks will describe explicitly the nature of the interaction between the user and the system. The design and development of the system will depend on the specific tasks that are defined for the system. We present different representative tasks that the user of a provenance system might wish to perform. These tasks will illustrate the components required in the design of a framework for a provenance system, particularly to provide the basis for supporting the living document concept introduced in chapter 1.

1. A user wants to retain all the experimental information to have a complete historical record. The first and the most obvious task would be to record three aspects of the experiment; a) the resultant data sets, b) the processing steps that led to the result data sets and c) the original input data sets and parameters used. For example, a scientist processing a raw image data set would

not only want to record the result data, but also the metadata on 'what' and 'where' about the tools, algorithms, instrument configuration parameters, and the raw image data used to generate the result. For example, these are needed to perform any re-executions of the procedures.

2. A user wants to browse all recorded details of previously performed workflow processes. For example, revisiting previous experiments would allow a bioinformatician to compare various protein sequence results to draw some conclusions and learn from history.

3. A user would like to annotate workflows with human-readable descriptions of an experiment and the conclusions that were drawn from it. For example, if an experiment is performed with several runs with different input parameters or with updated original data every time, then the scientist might like to highlight such details in a written report that would be linked to the experiment.

4. A user wants to validate a workflow, and wants to know if the experiment still produces appropriate results. For example, a scientist may come across derivation path information of an experiment run by a third party and wants to repeat the experiment to check its validity.

5. A user wants to run a workflow process a number of times with different sets of configuration input parameters. For example, rerunning an experiment more than once would allow the scientist to search for a desired result, or to analyse the results of multiple experiments.

6. A user may want to reuse the methods and steps of a previous experiment to

reproduce exactly the same results.

These tasks illustrate several ways in which users might interact with the provenance system. Among the above tasks the first is the primary one as it makes possible the other subsequent tasks. Having identified the key tasks that the user will perform with the system, the next section presents a high-level model that illustrates how the Provenance System may operate in an SOA environment.

2.7.2 Provenance Web Services

Recent developments in Web Services are leading to the emergence of platforms to support virtual communities of e-services on the Internet. Incorporating a Provenance System as a Web Service would be advantageous so the system can be used as a Provenance Service that provides the necessary "provenance" functionalities for scientists performing experiments in an SOA environment. This way a Provenance Service may be used as any other Web Services that enables invocations of desired Web Services and records the provenance of such invocations.

In order to illustrate the interaction of scientific users with the Provenance Service, we present our high-level architecture of a provenance support framework in Fig. 2.1. This service-oriented framework is characterized by a client being able to request provenance services from several service providers that host provenance systems.

The approach illustrated in Fig. 2.1 provides a high level of flexibility in selecting the desired provenance service that satisfies the client's requirements. This view is consistent with the use of Web Services for discovering services, and for interacting between the client and the service providers. Thus, the model view assumes that

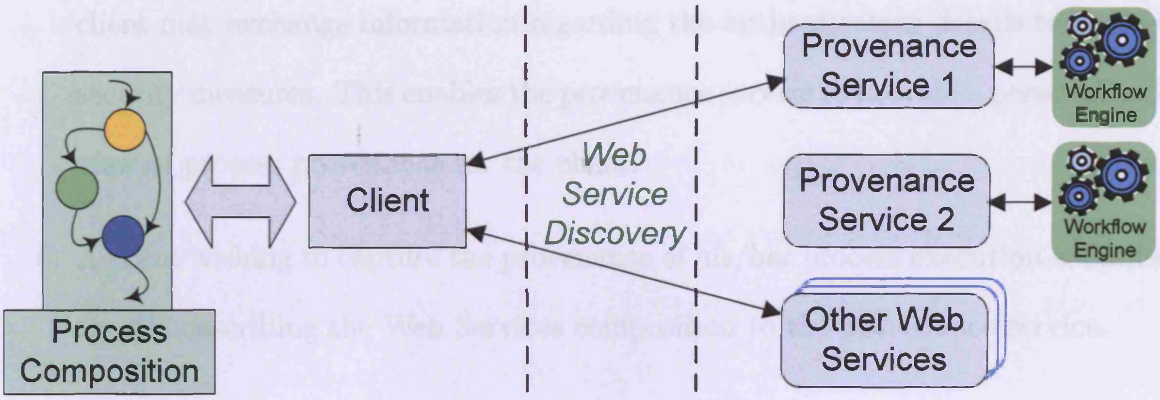


Figure 2.1: Architecture of Provenance Web Services

there are a number of service providers of provenance services for process execution, and each provenance system is exposed as a Web Service. The interaction procedure for this model is as follows.

1. The client discovers one or more Web Services from the service registry.
2. The client selects the Web Services based on its inputs and outputs.
3. All the selected services are then composed by the client using the process composition specification language that best suits him or her.
4. Like any other Web Services, the Provenance Service is also advertised in the registry for discovery. The service provider would advertise its Provenance Service along with its functionality, information about the workflow engine, and the version of the language specification it supports.
5. The client would discover all the available provenance services, and then select a provenance service that matches their requirements, assuming that the selected provenance service is trusted by the client. The provenance service and the

client may exchange information regarding the authentication details to enable security measures. This enables the provenance service to provide a personalized view of process provenance for the client.

6. A client wishing to capture the provenance of his/her process execution submits the file describing the Web Services composition to the provenance service.

7. The Provenance Service is responsible for:

   (a) Interacting with the workflow engine for the execution of the composite workflow.

   (b) Capturing the provenance of the process execution.

   (c) Recording the captured provenance of the process.

8. The Provenance Service returns the final result data of the execution with the process's unique identification number informing the client that the processing task is complete.

9. The client is allowed to browse and query the recorded provenance of the processes that s/he previously executed.

10. The client is able to validate previously-run processes, the incorporated Web Services, and the returned output data through re-execution of the process via its provenance. The client can also change the input parameters of the process during re-execution to perform "what-if" analyses.

The above model is a hypothetical illustration of the functioning of the provenance system as a Web Service. Current research and development in SOAs has enabled this model by providing the functionality of the coordinating entities in the model, namely, Web service discovery and interaction. This includes providing the infrastructure for Web Services registries, e.g., UDDI [78], providing the service access interface Web Service Description Language (WSDL) [100], and an interaction protocol, such as SOAP [101] at the communication layer. For composing independent Web Services, composition languages such as Service Workflow Language (SWFL) [57] and Business Process Execution Language for Web Services (BPEL4WS) [16] are widely used. There is also an emerging acknowledgment of the need for negotiation, trust, and measuring Quality of Service (QoS) which could eventually be used in deciding which service to select in such an environment [41].

Although this thesis assumes an SOA infrastructure, it does not intend to focus on issues relating to service selection and discovery in the model, which are adequately addressed by the current research and developments in industry, academia and standards organizations [8]. Instead, this thesis addresses the question of how data provenance support can be enabled within an SOA by using its infrastructure and technologies, and focuses on the Provenance Service. The research presented in this thesis mainly focuses on the provenance modelling and interaction of the Provenance Service. Provenance modelling refers to identifying the types of provenance required about processes in an SOA forming a provenance representation model. Interaction refers to the methods by which the entities in the model communicate; (1) mainly the clients, the Provenance Service, and the workflow engine with one another to perform provenance capture during workflow execution and, (2) within the Provenance Service for recording and querying provenance about the processes and re-execution of past processes.

2.8 Summary

In this chapter we have discussed the current applications and operations of provenance systems. The importance and use of provenance is seen in various systems and application domains ranging from databases, to the Semantic Web, to workflow applications. Our evaluation of provenance systems, particularly for workflow application systems, suggests that either (1) they do not represent a domain independent and service-based operational model, or (2) they lack essential features, for example, the ability to use the provenance to repeat a workflow. Thus, some service-based provenance systems do not cater for re-execution of workflows. The focus of this thesis is to design a provenance system that provides a domain-independent provenance data model and also caters for workflow re-execution functionality within a service-based environment. We have identified specific functionality for designing the provenance system and presented a model for provenance services that is consistent with the use of Web Services in an SOA environment.

In the following chapters of this thesis we present a provenance model for providing *provenance support* characterized by the capture and recording of the provenance information about processes, and use of the provenance of past processes in different ways within an SOA environment. As identified in this chapter, the two major issues of (a) modelling the representation of provenance, and (b) the interaction requirements for the Provenance Service to provide the provenance support must be addressed to successfully and practically enable provenance support for processes in

SOA. The next chapter presents our provenance model that defines the architecture for a Provenance Service to provide such provenance support.

Chapter 3

Provenance Model for a Web Process

3.1 Introduction

In a service-oriented approach to distributed computing, application resources are regarded as services available on the network that, in collaboration, provide a comprehensive and flexible system solution. Web Services research has provided considerable advances towards the service-oriented vision by allowing Web Services to be automatically discovered and dynamically bound across organizational boundaries. In addition, by assembling these individual Web Services into complex workflows newer and more useful Web processes can be created. Web Services composition is an increasingly important theme in research into SOA for Grid. Although research into various issues relevant to Web Services and Grid computing has deepened, retaining the data provenance of the dynamically assembled Web processes in such an open environment has not been adequately addressed.

In a service-oriented environment, the provenance of a piece of resultant data not only accounts for the transformations that occurred in the original data itself, but also of all the processing steps that lead to the resultant data. Discovering and composing individual Web Services together to form a composite service representing a workflow process in a current SOA can lead to a situation where desired end results are obtained from such dynamically formed Web processes, but the explanations of how we ended up with such results remain unknown. Without the knowledge of how resultant data is obtained, and what it represents, it is impossible to assess its usage and importance. In such a situation it becomes important to capture and record the provenance that leads to the derivation of resultant data that would, for an example, allow a user to study his or her past actions. The data provenance consists of the entire processing history. This includes the identification of the origin of the raw data sets, information on instruments that generated or recorded the original data and the parameters that were set, as well as all the processes that have been applied in the transformation of such data sets. Most research work emphasizes the semi-automatic or manual recording of data provenance and usually the provenance models are specific to domains and research projects, thus making it difficult to apply the models and algorithms to other application areas or to offer more general support for data provenance. The subject of exploiting the recorded data provenance is explored only to the extent that is required by the domain research project. Thus, as discussed in chapter 2, although the significance of data provenance is being realized in many projects, there is currently very limited architectural level support for representing, recording and exploiting data provenance.

From our vision of provenance Web Services discussed in section 2.7.2, we have developed the architecture for a Provenance Service that we present in this chapter. The architecture provides support for provenance in service-based environment, and incorporates provenance capabilities consisting of two main functionalities:

- The ability to collect and archive adequate provenance about the transformation of data occurring during invocation of Web Services, for example, a composite service executed via a workflow engine.
- Allowing the recorded provenance to be accessible and viewable via generic browsers and manipulated though query interfaces. The architecture presented in this chapter focuses on the requirements of the provenance data for complex Web process execution from a user perspective. For example, how is the provenance information about a process collected, represented and recorded, and how is the provenance data queried and reasoned with? It provides the capabilities that help users by preserving adequate process provenance information for 1) tracing the derived data's origin and 2) exploiting and manipulating provenance in numerous ways, such as the recreation and re-execution of a process. Our architecture also provides flexibility to cope with different domains without affecting the underlying provenance recording and representation mechanisms.

3.1.1 Architecture Contributions

1. Added support for provenance in Web Services environments by representing a provenance service implementation which can be discovered and consumed like

any other Web Services.

2. A Provenance Collection Service (PCS) that is capable of capturing and recording the provenance of a Web process. The PCS is able to collect the provenance of all Web Services involved in a Web process execution.

3. Modelled the representation of provenance for a Web process that is captured by the PCS. A predefined structure (the provenance schema) is utilized by the PCS in order to represent the captured process provenance and record it in the provenance database. The recording mechanism depends on the predefined structure.

4. A Provenance Query Service (PQS) has been added to the model that provides ways to query the archived process provenance data, allowing the documented provenance to be viewed and navigated.

5. The PQS can exploit the archived process provenance by allowing the re-execution of the entire process by means of its retrieved provenance. This means that the PQS uses exactly the same services and data as used during a prior process execution, allowing verification of previously run processes and associated services and data. The PQS allows, for example, the performance of "what if" styles of analysis on past processes. In addition to this, it provides tools for recreation and analysis of a process, e.g., to verify if the data sent from a sender is the same as the data received by the receiver.

3.2 An Example Scenario

In this section we present an example scenario that clarifies the vision of provenance Web Services as discussed in section 2.7.2. We have mentioned earlier in section 2.6 various examples to identify system goals, but here we put forward an example scientific process scenario that demonstrates the use of a process provenance service. This example process is a common scenario for many astronomy applications.

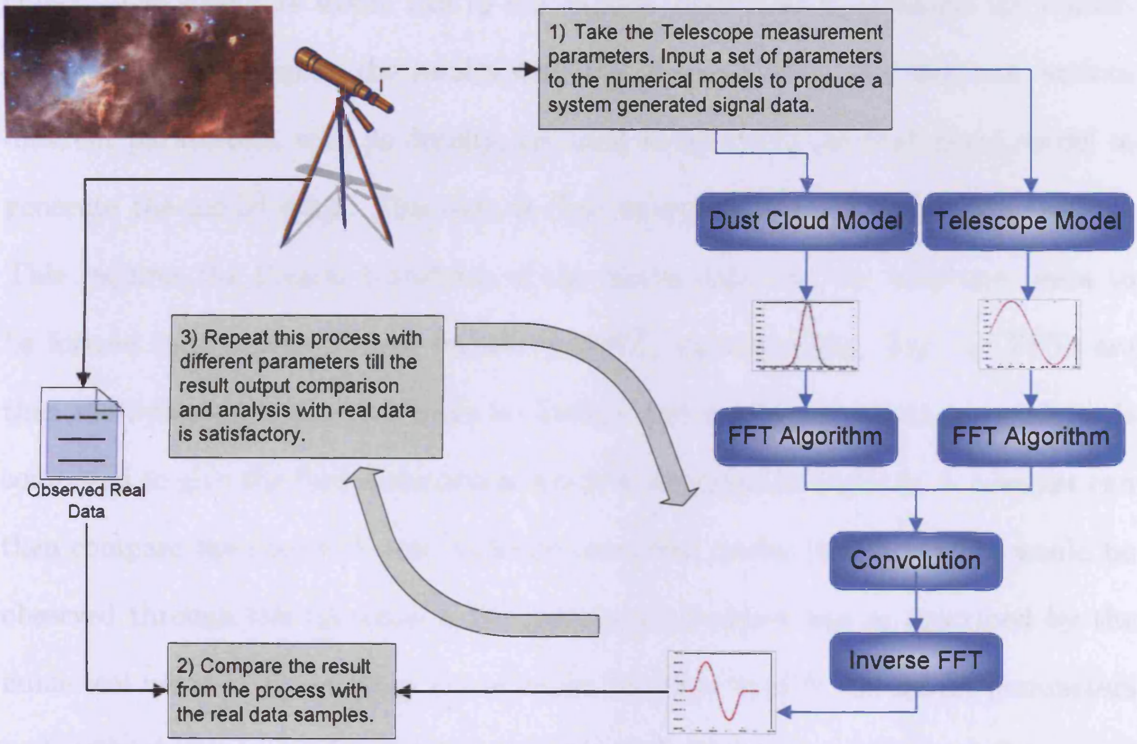

1) Take the Telescope measurement parameters. Input a set of parameters to the numerical models to produce a system generated signal data.
Dust Cloud Model
Telescope Model
FFT Algorithm
FFT Algorithm
Convolution
Inverse FFT
3) Repeat this process with different parameters till the result output comparison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

Figure 3.1: Simple Scenario Example

Astrophysicists seek to gain a better understanding of astronomical objects and events by comparing observations with the output of advanced numerical models. Many research activities in Astronomy and other scientific disciplines are centred on issues such as data representation, storage, retrieval and reuse of data and analysis. Figure 3.1 presents a simple scenario in the astrophysics domain, where an astrophysicist analyses sample data collected from a telescope. The observed data from the telescope is usually stored in some file system. An example of the observed data are the celestial infrared signals from bodies in space such as a dust cloud that surrounds a region of star formation. These observed data are compared by the astrophysicist with mathematical models that are intended to represent the observed astronomical object. The scientists would like to use various algorithms to compute the numerical models and compare the results with the observed data. For instance, various different parameters, such as density, are used as inputs to the dust cloud model to generate the model data. This data is then convolved with the telescope "beam". This requires the Fourier transform of the model data and the telescope beam to be formed using a Fast Fourier transform (FFT) algorithm [80]. The two FFTs are then multiplied together in a pairwise fashion and the inverse Fourier transform is computed to give the final convolution which is displayed in a graph. A scientist can then compare the observed data with the convolved model (which is what would be observed through the telescope if the astronomical object was as described by the numerical model). If necessary the scientist can then modify the model parameters and compute a new model output, convolve with the telescope beam and compare again the numerical results with the observed data. Repetition like this with different parameters in the model allows scientists to perform"what if" analyses and comparing allows the scientist to progressively improve the fit between the data and the model. Figure 3.1 shows all these complex mathematical models and operations combined together, representing a processing step in the data analysis followed by an astrophysicist.

This example produces several "provenance" requirements. In this regards, this section will first discuss a simplified list of the provenance requirements and in the following section the provenance model is presented that provides the functionality to meet these requirements. The requirements are discussed as follows.

Process Modelling: In any process support system, one of the key questions is how to express the experimental process or compose the different steps in some way. Process composition in an SOA is possible in two ways:

1. *Static composition:* use a description language to express the experiment process that can afterwards be executed to generate the intended result, for example using BPEL4WS to describe a workflow for execution.

2. *Dynamic composition:* This involves selecting algorithms or models and executing one step at a time by retaining outputs at each step. In this case, selecting the next step in the process may be based on the scientist's observation of the output from the previous step.

Although dynamism is preferable in terms of composition flexibility, in the case of a large-scale and time-consuming data analysis a structured process construction is desirable. Also, without initial abstract construction of the experiment, support for workflow reusability is lacking. Even though both ways of process composition are effective for modelling with the provenance model, given the intrinsic modularity of the experimental process, static composition is preferable as it provides additional information through an abstract process description. The composition language must be structured, allow nesting and facilitate reusability. In addition, and on account of the complex execution environment, the language must provide support for controlled data flow between different steps and exception handling. Exception handling means a reliable mechanism must be provided to cope with any deviation from the prescribed behaviour, for example, by aborting the execution of the entire process. The language must allow the identification and registration of external objects and programs to the corresponding workflow execution engine. In the case of the astronomical data analysis experiment, both the algorithms and astronomical data will be distributed resources that are external to the provenance system and the execution engine.

It is assumed that the algorithms for the mathematical models are distributed individual Web Services advertised in a registry, i.e., a UDDI registry. With the help of available tools these services can be found by querying the registry, and depending on the requirements specific services are selected and used in the experimental process. As mentioned in section 2.7.2, the discovery procedure relies on several service requirements and criteria. Discussion of this area is not in the scope of this dissertation. After finding the location of the services that are to be used, the next step is to perform some form of composition of the individual Web Services.

Recording, Querying and Analysis Capabilities: The astronomical data analysis process result is a derived data product that cannot be interpreted and recreated without provenance information describing the processing steps used for its creation and the initial data used in the process. This leads to the main problem of lineage tracking. Recording such provenance information answers typical questions such as "which algorithms are used by process W", "which process uses algorithm X", "which dataset is used to get result Y" and "which result may change if the dataset Z is updated". In addition, given the abstract workflow description, a subsequent analysis of the provenance information may be used to evaluate whether the abstract workflow description has been adhered to. Operations to propagate changes are required in the system so that any changes can be reflected by re-executing the process, producing new sets or versions of results. In the case of the resources being distributed, it is rarely possible to rely on an automatic notification of changes from the resource providers. The only way to get information about any changes or updates is either by checking this at set time intervals or only when re-execution of the past experiment is triggered. The derived datasets are often large, so the system should support separate archiving of the derived datasets from the processing steps, but link every program/algorithm with pointers to its associated datasets. This would be truly useful in efficient data management if functionality is provided to track and retrieve data dependencies amongst different processes.

The existence of a common terminology becomes crucial in the astronomy domain in order to understand the traced lineage of data produced by a third person. Definitions of common terminologies are emerging within each domain, e.g., Gene Ontology in the biology domain aims to provide a controlled vocabulary that can be used to describe any organism [38]. Such vocabularies use synonyms and the majority of terms have a textual definition stating references to its source. In the case of the Astronomy community, Unified Content Descriptor (UCD) is one of the vocabularies that is used and defines many core astrophysics units, measurements and concepts [12]. These efforts are likely to be advanced to converge into a standard with the recent service-oriented research in this domain [25]. An agreed common terminology helps in designing analytical tools and it is very useful for these tools to be available with descriptive vocabularies. Re-execution of a previous experiment could result in problems, such as an algorithm/service (tool) being unavailable or moved, preventing the re-execution task. To avoid this, it is desirable for the system to have a mechanism first to identify semantically similar algorithms that could be used so the re-execution could be completed, and secondly select the one with fixed constraints such as high throughput. The latter could be based on the requirement of processing larger datasets.

3.3 Provenance Model

In this section, a provenance model is proposed that integrates the client-server and Web Services models. The provenance model shown in Figure 3.2 is driven by the provenance collection service (PCS) that uses a workflow execution engine to enact the pre-defined workflow by invoking the Web Services in the concrete workflow description. The PCS is exposed as a Web Service with a client GUI interface for workflow invocation, and is able to capture and record the workflow metadata. Thus, by integrating the client-server and Web Services models, the provenance model endeavours to facilitate the capture of the necessary provenance data during the execution of a distributed workflow. The model also provides a structured way of representing the captured process provenance, which is then recorded in the provenance database. The Provenance Query Service (PQS) in the model is used to access the recorded process provenance, allowing users to exploit the provenance information in a number of ways. A Provenance Server is a server machine that hosts both the PCS and the PQS. The model assumes that a mix of client-server and Web Service approaches can lead to more flexible mechanisms for provenance support in a distributed service-oriented environment.

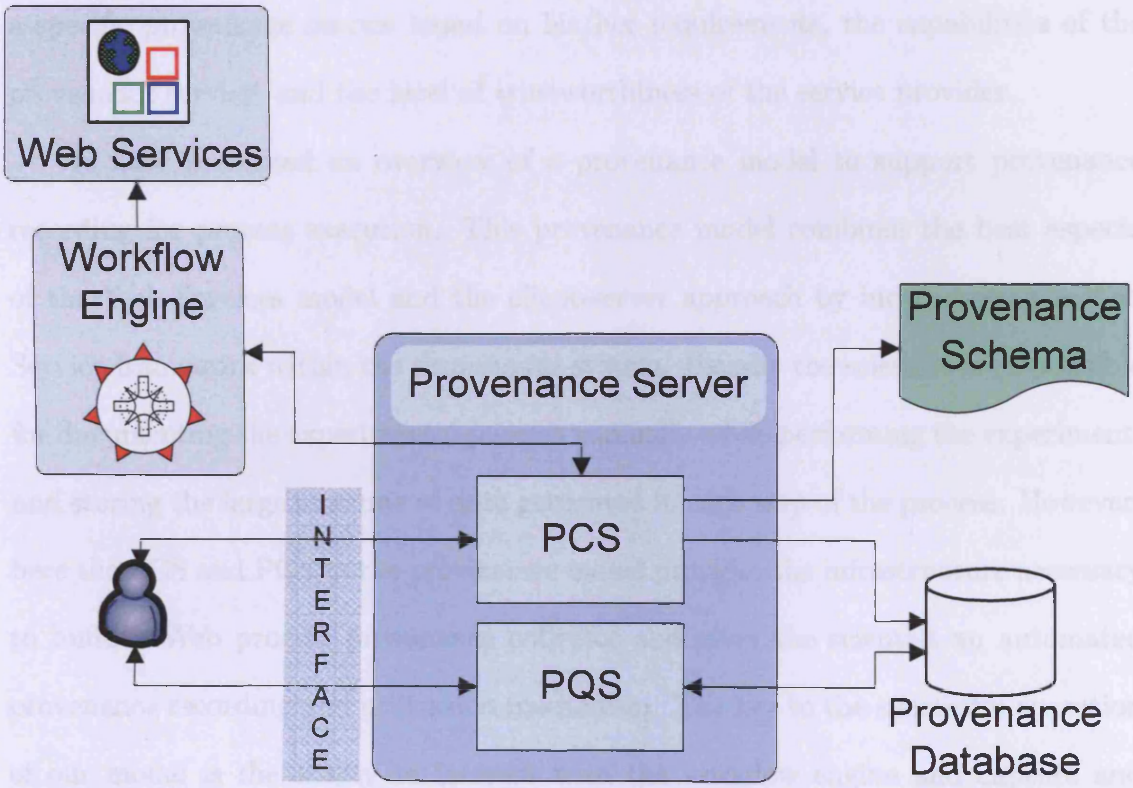


Figure 3.2: Provenance Model

At the conceptual level, the provenance model operates by capturing the provenance of a Web process that integrates various Web Services (composed and submitted by a user) by interacting with the execution engine to gather the provenance information about the Web process execution. The operation of the model is shown with the scientific example scenario illustrated in section 4.2. When the scientist wants to run a composed process and capture its provenance and the data passing between the services, s/he initializes the concrete workflow description using the PCS. The PCS acts as an interface that interacts with the workflow engine and documents the Web process execution. This assumes that the scientist has previously decided to use a specific provenance service based on his/her requirements, the capabilities of the provenance service, and the level of trustworthiness of the service provider.

We have presented an overview of a provenance model to support provenance recording for process execution. This provenance model combines the best aspects of the Web Services model and the client-server approach by incorporating a Web Service framework within the provenance system. Usually the scientist is responsible for documenting the experimental process manually while performing the experiment, and storing the large amounts of data generated in each step of the process. However, here the PCS and PQS in the provenance model provides the infrastructure necessary to build a Web process provenance collector and gives the scientist an automated provenance recording and utilization mechanism. The key to the successful operation of our model is the ability to interact with the workflow engine and capture and record the provenance during process execution, and providing the ability to exploit the stored provenance. In the next subsection, the identified components of a process documentation are outlined and the representation of the provenance of a process is discussed. Following this, the capturing and recording capability in the provenance model is discussed. Finally, the query interface for browsing the documented process, and the various functionalities it provides, are described.

3.3.1 Identifying and Representing Provenance

In chapter 1, it was stated that the provenance of a piece of data is represented in a computer system by appropriately documenting the process that led to its creation. In this section, the key elements that form the representation of provenance in an SOA are introduced; further refinement will ultimately lead to data types for provenance representation in chapter 4. The documentation or provenance information of a process that led to a data product is categorized into three types: involved services, data and passed parameters, and the data flow arrangement.

Involved Services: The provenance of the Web Services that are involved refers to the syntactic metadata that provides information about the location, description and access information for an instance of a service. Dynamic information such as time of execution and the state of the hosting machine or environment could also be part of the service instance's related provenance. This also includes the types of inputs the service accepts and the output it returns. In addition, it could also entail static information on service providers, for example provider name, provider profile, physical location and domain information. Capturing provenance about the services involved in a Web process is largely dependent on what information is made available by the service providers about its services and whether they are accessible to a provenance service. The instances of services involved in a Web process are related to the data that are transformed during their invocation instances.

Data and passed parameters: This refers to the original data that are being supplied to appropriate services for transformation and the parameter values that are passed. Data also refers to the intermediate results that are acquired from all involved services interacting with each other. Considering the example scenario in section 3.2, the telescope data and the model-generated data are the original data that are being transformed, whereas the telescope setting values and other values needed for the model data generation are the parameter data passed to the services. This refers to the messages that are consumed and generated by the involved services during execution. Documenting the intermediate data, e.g., the data generated by the FFT algorithms in the example may be unnecessary for re-execution but it provides additional support, for example, to identify the point of failure in case of an incomplete process.

Data Flow arrangement: This is the actual arrangement of the composed services to form a Web process that could be executed to generate the final result. The information about how the services are linked to create the Web process determines how the data flows through the process to create the result. A workflow description language or scripts can be used to describe aspects of the Web process, i.e., describing the sequence of the service execution at an abstract level. Thus the file containing the workflow description is important and its storage location is metadata that needs to be retained. This provides an "abstract" workflow description outlining which services must be involved in a workflow execution and in which order they should be executed. The abstract workflow description may contain constructs for conditional execution. For example, say service A is expected to output an integer data item (when executed) that is described with a variable dv_1. Say two cases are specified in the abstract process description for dv_1; (a) if ($dv_1 > 100$), than execute a succeeding service B, and (b) if ($dv_1 \leq 100$), than execute a succeeding service C. Due to such conditions, during the actual execution either service B or C would be executed depending on the output from A, i.e., the value of dv_1.

It is important to extract a "concrete" workflow description specifying the particular service instances that were used in a particular Web process enactment. Capturing links or relationships between the service instances is possible by capturing the provenance information in a standard format – thus, automatically providing a concrete workflow description. A simple way to build links is by capturing the interactions between services via the actual dataflow occurring during the workflow enactment.

We further categorize the three components above to provide specific definitions that determine various provenance elements in SOAs.

Definition (service-Provenance) *The provenance information of service instances that are associated with the process; the service-Provenance must include information that allows each service instance to be uniquely identified.*

Definition (process-Provenance) *The documentation of a Web process that consists of one or more service-Provenance items from services involved in the process. It also specifies the links between the service instances in the process. A process-Provenance must include information that allows a process instance to be uniquely identified.*

The three components mentioned previously form the basis for identifying the elements of the provenance representation for process execution. It should be noted that a given process-Provenance provides a representation of multiple pieces of data produced by the involved service instances that ultimately represents the final piece of data or result of a process. When a process-Provenance is created and recorded, it captures the steps in a process in three parts: (1) a process as a whole that may contain an abstract workflow description and manually-entered information about the workflow and required data or parameters, (2) instances of independent services involved in the process and, (3) the dataflow during the interactions between the services instances.

In the context of an SOA, messages are sent from one service to another during interactions. Capturing the messages that are being sent between the services in a standard form allows the entire process for the computation of some data to be represented. With such information one can verify, recreate, re-execute, compare it with similar executions, and evaluate the captured concrete workflow against the abstract workflow, i.e., to determine if the workflow was actually executed according to the abstract description. Describing such messaging between service instances is at the core of documenting the Web process.

In our model, the workflow engine is the mechanism for executing a Web process. The interactions of the services that are present in the composite service described by the abstract workflow occurs through the workflow engine. We assume that the engine is responsible for interacting with each service as specified in the abstract

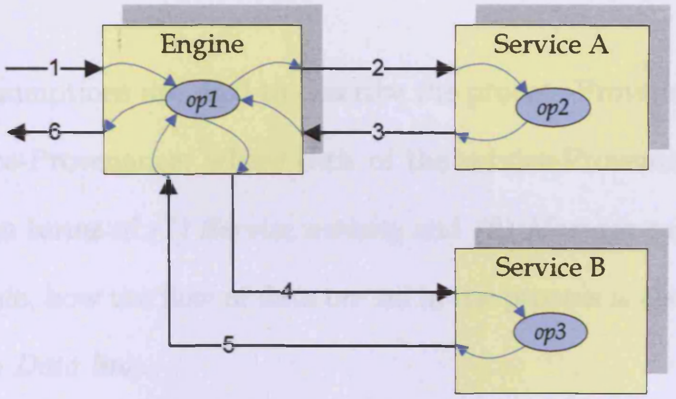


Figure 3.3: Interaction between Engine and Web services

workflow description.

Figure 3.3 shows how the interactions occur through the exchange of messages between the workflow engine (i.e., Engine) and the two services (i.e., remote Web Services). Here, the numbers represents the sequence of the interactions. The following three assumptions are made; (1) the Engine is an interface point of access to a composite service (based on an abstract workflow description) exposed by the underlying engine, (2) the composite service itself is a Web Service with an operation or function that is invoked by a user to execute the process and, (3) the composite service consists of services A and B (any data in the message received during the interaction with A may be sent to B is as specified by the abstract description). This demonstrates that any flow of data between the independent services as described in the abstract workflow description takes place through the engine, and no direct interaction occurs between the services involved in the process. The input data in the message sent to a service during an interaction may be processed and transformed to some output data by the operation or function (e.g., *op*2 of service A in Figure 3.3) of the service.

The above assumptions are used to describe the process-Provenance that consists of a set of service-Provenance; where each of the service-Provenance describes the service instance in terms of *(1) Service activity* and *(2) Message contents.* Using the Figure 3.3 example, how the flow of data occurs in the process is also established and described via the *Data link.*

Definition (Service activity) *A service activity refers to the function or operation that is being performed by a service instance to accomplish a particular task and any other dynamic information linked to the instance of this operation, e.g., execution time [109]. It may also contain static information, e.g., service ownership [86].*

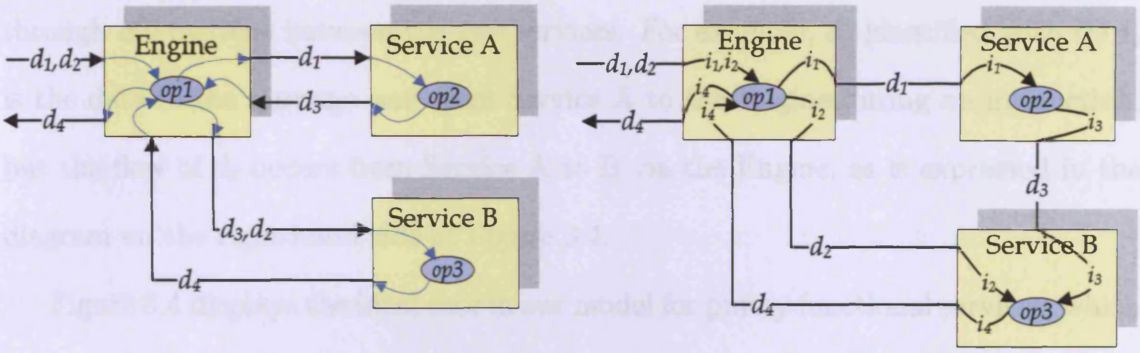
A service may be able to perform one or more functions or operations, but here it is assumed that only one operation is used for a service instance. This is because only one operation can be processed at a particular service invocation. Alternatively, a service referred to in a process may be using other services that are hidden behind that service. Capturing provenance from such hidden services may only be possible if they are provenance-aware, i.e., able to record messages exchanged through some mechanism to the provenance database. Service activity should contain information such as whether the service was invoked successfully.

Definition (Message contents) *This is the contents of the messages exchanged between the instances of services. A message contents consists of messages that are* *inputs received and outputs sent by a given service for its particular instance.*

Which messages are captured depends on the application domain that is performing the message exchanges. Usually the structure and data type of such messages are specified in the service description and are application-specific. We do not intend to define or identify the messages that are exchanged, but simply aim to capture and retain a copy of the messages that are being exchanged between two services. So, the message contents may have the copy of the input and output data for a service instance. A service may send data that is too large or considered confidential. So, instead of the actual data, the service may send a pointer or/and other information about the data.

Definition (Data link) *This forms a part of the process-Provenance that specifies the link between the service instances in the process to identify the flow of data between the services.*

Links (flows of data) are established by the information obtained during the invocations of services in the process. A particular input/output data item for a service instance must be associated with information that uniquely identifies the data. For a service instance; (1) data received (inputs) from a particular service is the "source" of this data, and (2) data sent (outputs) to a particular service is the "target" of this data. The inputs and outputs contained in the message content are identified for a service instance. It is essential to have information that dictates the source and target (e.g., URL addresses) of a particular input and output data item, respectively. Apart from this, a unique identifier for each input and output is crucial when determining the flow of data between the services.



Keys
d = data item
i = ID inserted for the data item

Services	Data Link	Input dataset	Output dataset
Engine	DL1	$d_1 = i_1$ + source(User); $d_2 = i_2$ + source(User); $d_4 = i_4$ + source(Service B)	$d_1 = i_1$ + target(Service A); $d_2 = i_2$ + target(Service B); $d_4 = i_4$ + target(User)
Service A	DL2	$d_1 = i_1$ + source(Engine)	$d_3 = i_3$+ target(Service B)
Service B	DL3	$d_3 = i_3$+ source(Service A); $d_2 = i_2$ + source(Engine)	$d_4 = i_4$ + target(Engine)

Figure 3.4: Expressed data flow of services

As mentioned earlier, the flow of data occurs via the engine through the processing of the abstract workflow description and no direct interaction between the services occurs. Figure 3.4 shows how the flow is data is determined. The left-hand side of Figure 3.4 depicts the data in the message exchange that are sent during the interactions in Figure 3.3. The right-hand side of Figure 3.4 depicts the actual flow of data during the interactions of the services with the engine. Here, let op be the specific operation or function at each service that may transform a particular input data into a particular output data. A unique ID i is inserted to represent a data d. For example, the input data d_2 and d_3 in an interaction between the Engine and Service B on the left-hand side of Figure 3.4 has unique IDs i_2 and i_3 respectively.

This illustrates that in our model, we denote the flow of data through the information that uniquely identifies that data, and this flow does not happen directly through interactions between the two services. For example, d_3 identified with ID i_3 is the data in the message sent from Service A to the Engine during an interaction, but the flow of d_3 occurs from Service A to B via the Engine, as is expressed in the diagram on the right-hand side of Figure 3.4.

Figure 3.4 displays the ideal case in our model for purely functional services, which do not maintain persistent state across invocations. The same approach generalises to persistent services: the data in an output message can be a function of the data received (input message) at that instance of the service invocation. In the lower part of Figure 3.4, is a table with a symbolic representation of the data flow expressed for each service's invocation instance. For each service invocation instance, the data link information consists of the uniquely identified input and output data with their source and target, respectively.

Using such information, one can easily navigate through, or recreate, the flow of data that occurred in a process. For example, to identify the data link of output data d_3 to determine its flow from service A to service B; (1) the target address of the output d_3 in service A instance is identified as the address of the service B instance for that process, (2) search for the input data d_3 with ID i_3 in service B, and (3) match the source address of input data d_3 in service B with the address of the service A instance. A detailed discussion of how the flow of data is represented is given in section 5.2.1.

Hence, data link information in the process-Provenance denotes the flow of data between the services, whereas message contents in the service-Provenance denotes the actual content of all the input/output data for a service instance. Such flows of data are the core elements to reconstruct functional data dependencies in an execution. Thus, the service-Provenance captures the activity and state of services and the content of messages, and the flow of data established in a process forms part of the process-Provenance and is associated with the service instances.

3.3.2 Capturing and Recording Provenance

In this section, a conceptual discussion is presented of the PCS component in the provenance model. Our model, partly based on the client-server model, captures and records documentation about Web process invocations. The mechanism to capture and record provenance is part of the client application and the workflow engine. This means that the independent and distributed services that may be invoked within a process are not concerned with the documentation of its interactions with the workflow engine during its lifetime.

In chapter 2, the notion of a Provenance Service was introduced that eventually led to the provenance model. The documentation of a single execution of a Web process, stored in a provenance database, is handled by the PCS of a particular provenance service. Utilizing a provenance service helps make a workflow engine "provenance aware".

Now, we discuss the main theme of the model of how the recording is achieved via interactions between components. Figure 3.5 illustrates different components within the model and the interactions that occur between them. The PCS can synchronously process the recording of provenance information in the provenance database. Because of the limited control over the workflow execution engine, the interaction between the PCS and the engine is carried out in a synchronous manner, meaning that once the engine starts to execute the submitted Web process, the recording program is prevented from doing any processing until the current execution completes. In other words, upon sending a message, the sender PCS waits until the receiving engine processes it and returns the result of the process execution. The workflow engine upon receiveing the request processes the inputs and it is assumed that the partner Web Services are invoked synchronously based on the abstract process description.

The provenance model provides a GUI interface for the PCS component that acts as an interface to instantiate the process execution, and in turn activates the PCS to capture and record essential and reusable provenance information about the process. The PCS component makes use of the predefined structure to represent such provenance and records it in the provenance database.

Adopting the synchronous approach shown in Figure 3.5, the provenance of a Web process is recorded in two phases.

- Initially the collection begins with the submission of data, input parameters, the process script file, and the URI address of the composite process through the GUI interface. The PCS is activated and creates a unique identifier for the process that is recorded in the provenance database. The recording program starts collecting information received via the interface, labels it appropriately, and stores it in memory along with the unique process identifier. Then it sends

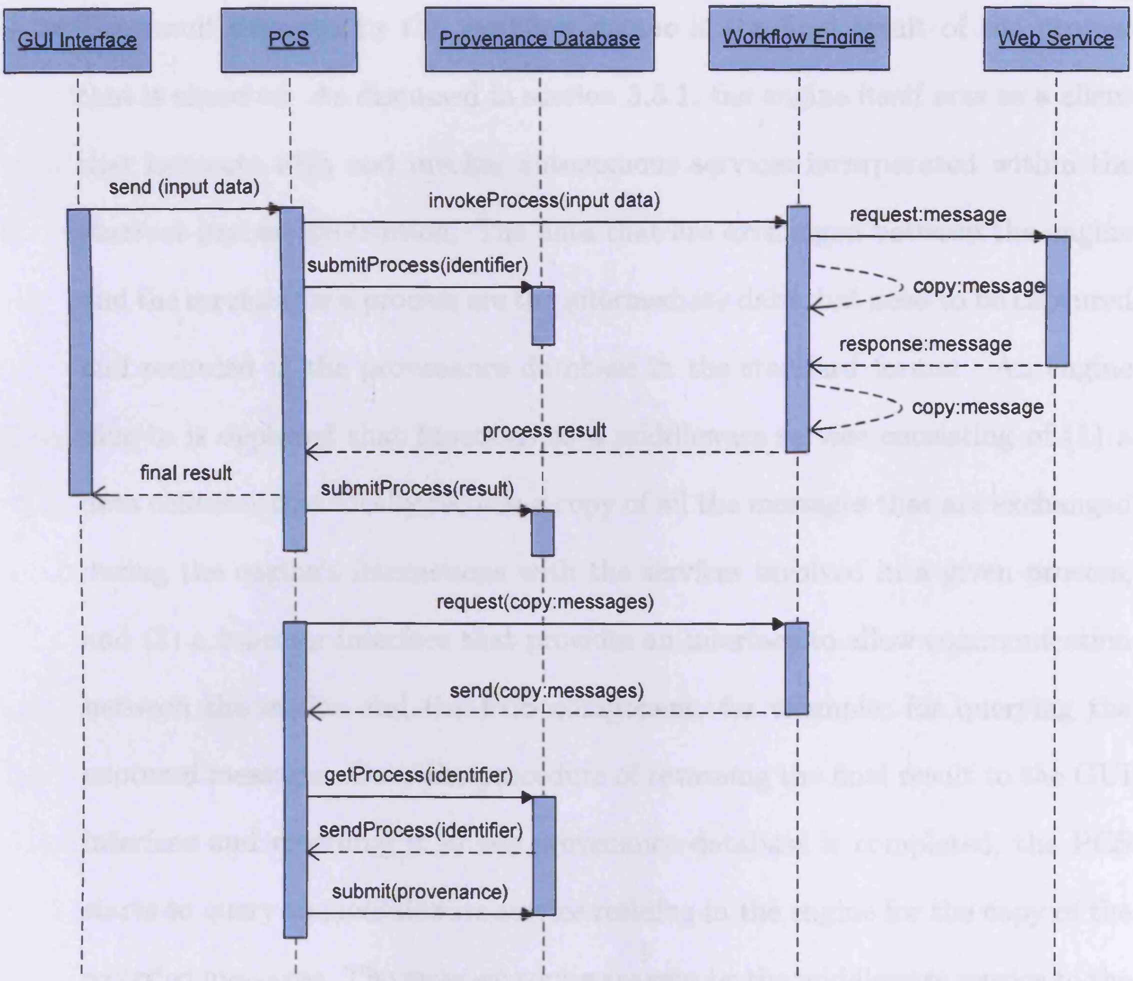


Figure 3.5: Interactions between Components

a message to the engine to start the execution of the process, along with all the required inputs. The PCS then waits for the engine to return the result of the process execution. Once the PCS receives the result from the workflow engine, it stores the result in memory and then returns this result to the GUI interface. With this event completed, an internal thread is triggered within the PCS that starts recording the captured provenance in the provenance database for the given unique process identifier.

- The result returned by the workflow engine is the final result of the process that is executed. As discussed in section 3.3.1, the engine itself acts as a client that interacts with and invokes autonomous services incorporated within the abstract process description. The data that are exchanged between the engine and the services for a process are the intermediate data that need to be captured and recorded in the provenance database in the standard format. An engine plug-in is deployed that functions as a middleware service consisting of (1) a data collector that locally records a copy of all the messages that are exchanged during the engine's interactions with the services involved in a given process, and (2) a collector interface that provides an interface to allow communication between the engine and the PCS component, for example, for querying the captured messages. Once the procedure of returning the final result to the GUI interface and recording it in the provenance database is completed, the PCS starts to query the middleware service residing in the engine for the copy of the recorded messages. The message copies are sent by the middleware service to the PCS. The PCS then (1) gets the most recent (last recorded) process identifier from the provenance database, and (2) records such intermediate messages of the process.

The provenance model uses a standard structure to represent process documentation to determine the approach that will result in the most effective recording. The above discussion illustrates the operation of the PCS component of the provenance model that is involved in recording the process documentation. However, simply recording the provenance may not be sufficient for a successful provenance model.

The effective exploitation of the recorded provenance is as essential as its existence. Without mechanisms for exploiting provenance information, the question of why we need to record provenance would remain unanswered. Consider the case where a process is required to be re-executed to check the validity of a dataset generated in the past. In this case, in order to perform re-execution for validation purposes the provenance recorded during the original execution must be enough for this task to succeed. When re-executing past processes, using exactly the same data, parameters and services (provided by the recorded provenance data) may result in a different set of outputs which may be, for example, due to modified algorithms or services. This may lead to various uncertain conclusions. The ability to provide appropriate and effective explanations of the results generated from the re-execution task would give a basis for any further actions that need to be taken. For example, by comparing the intermediate outputs generated from the re-execution task with the original records of the process, one can identify which intermediate data differs, i.e., it identifies the service/s that is affecting the final output during the re-execution. Thus, various reasoning on the recorded provenance involves querying and comparing information that provides a certain level of explanation. In the following section we formalize the query interfaces within the provenance model. This facilitates the use of the provenance model in using recorded provenance in different possible ways.

3.3.3 Provenance Querying and Reasoning

We discuss the ability to reason about and exploit the recorded provenance of a Web process in the provenance model provided by the Provenance Query Service

(PQS). As discussed above, this in turn forms the basis for subsequent use of the provenance information for a workflow execution. The PQS supports two query interfaces: (1) the *process provenance query interface* through which the contents of an identified *process-Provenance* can be retrieved, and (2) the *provenance reasoning query interface* which allows the querying user to retrieve the provenance of application entities. In this section we introduce these two interfaces.

Process Provenance Query Interface

To retrieve the actual process provenance that makes up the provenance of a Web process instance, the process provenance query interface is used. This interface gives direct access to the process documentation contents, by allowing the querying user to search over, and retrieve parts of, the *process-Provenance.* Provenance query results include the related *service-Provenance* data keys.

The process provenance query interface allows the querying user to perform the following operations:

- Retrieve the unique identifiers for all the process instances.

- Retrieve the contents of a *process-Provenance* for a given unique process identifier.

- Retrieve all the *service-Provenance* recorded, based on the unique identifier of a service.

- Retrieve all the dataflow in the *service-Provenance* recorded with a given unique process identifier or a service identifier.

The actual results of the query depend on the contents of the provenance database to which the query is sent, because the query will only return data contained in that provenance database, and on the access control restrictions placed on the querying user by the database. As the amount of data returned may be large in volume, the process provenance query interface should allow for the iterative retrieval of query results. By this mechanism, a querying user should be able to process the results in manageable chunks.

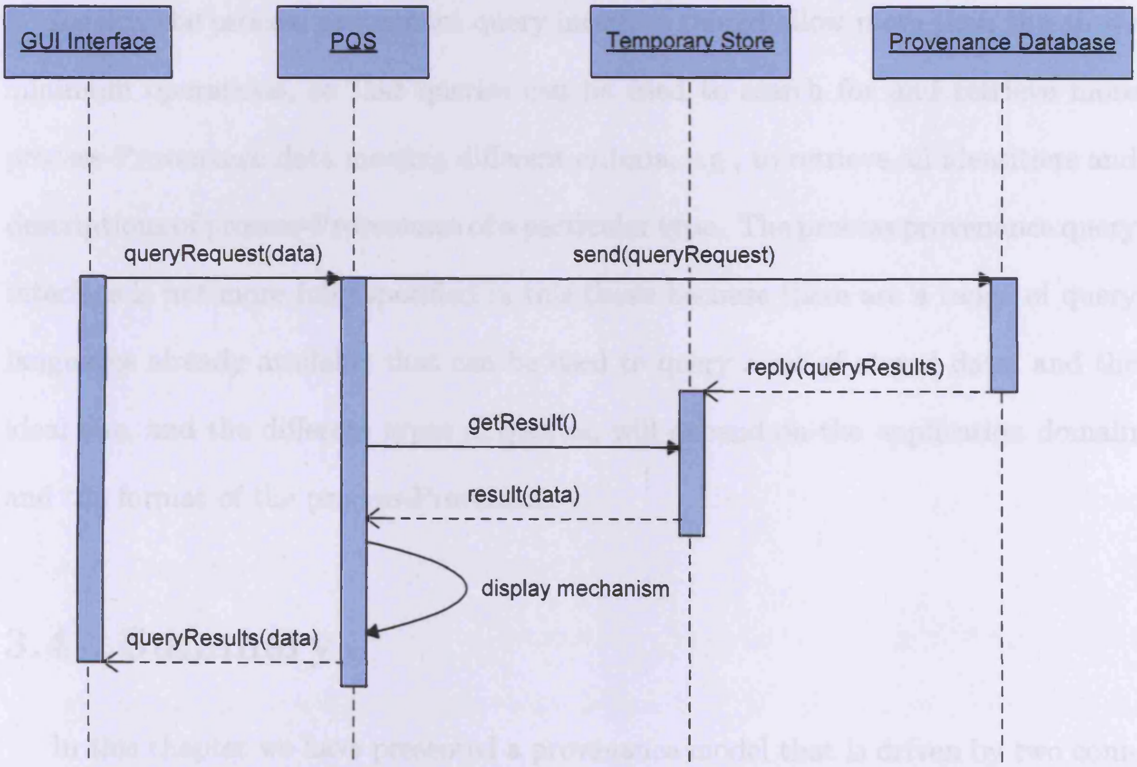


Figure 3.6: Interactions between Query Components

In addition to this, the process provenance query interface consists of a query *display mechanism* to transform the process provenance query results and to display them to the user in a way that they can most easily process. The three main methods of displaying the queried process provenance results are provided based on the previously discussed query operations: (1) simple and structured textual display, (2) tree display, and (3) a graphical display for recreating the high level behaviour of the process.

Figure 3.6 depicts the interactions of the PQS with other components when used by the querying user. Temporary storage is created locally at the PQS during a query to handle the large query results appropriately.

Ideally, the process provenance query interface should allow more than the above minimum operations, so that queries can be used to search for and retrieve more process-Provenance data meeting different criteria, e.g., to retrieve all identitiers and descriptions of process-Provenance of a particular type. The process provenance query interface is not more fully specified in this thesis because there are a range of query languages already available that can be used to query a set of stored data, and the ideal one, and the different types of queries, will depend on the application domain and the format of the process-Provenance.

3.4 Summary

In this chapter we have presented a provenance model that is driven by two components that supports provenance requirements in a Service-Oriented Architecture. The PCS component performs the collection of data provenance about Web process enactments, and stores it in a provenance database using a predefined provenance data structure. The PCS forms the basis for specifying provenance about the results produced from executions of processes in a service-based environment. There is an established need for documenting the processing history of datasets produced in such an environment, and this is a significant contribution of the provenance model. The Provenance Query Service (PQS) provides querying interfaces with respect to the provenance structure. Existing work on provenance recording frameworks in a Service-Oriented Architecture provides a more distributed recording and storage approach [55] where the services interacting with each other are responsible for recording their interactions. The way this work is being modelled based on interactions suffers from the disadvantage of requiring the creation of a very complex query to retrieve a provenance trace (i.e., a process provenance) or other simple information. This is due to the lack of an indexing mechanism in their provenance information storage system. Our model incorporates a simple provenance representation and recording mechanism to support the application of simple and effective queries on the stored provenance.

This chapter has presented the functioning and provenance representation strategy for Web processes of the provenance model. We have discussed the interactions between the components of our provenance model.

The question that we have not addressed thus far is how the identified provenance representation about a process can be modelled, that is the data structure used to define each type of provenance. Thus, in the next chapter we first present the provenance model to define types of provenance about a process to produce a provenance structure, and then discuss the provenance collection process of the PCS that allocates the collected provenance to the appropriate provenance types defined in this thesis.

Chapter 4

Provenance Representation and Capture in SOAs

4.1 Introduction

In chapter 3 we presented our provenance model that consists of a service for enabling the capture of provenance in a standard form during the enactment of a workflow or process in a service-oriented environment. The provenance model supports the need for provenance handling capabilities by;

- Identifying the architectural need for automation in capturing the provenance of workflows enacted in an SOA.
- Using a standard form to represent and record the captured provenance.
- Using a combination of the client-server model and the web service model to capture and record provenance.

The provenance model addresses the need for capturing and recording provenance in a Web services environment by providing mechanisms for capturing, representing and recording the provenance of workflows in a standard way. The key to our model is its ability to automatically and accurately capture the provenance information of enacted workflows in a standard form so that the concrete description of a workflow can be retrieved using the provenance information.

In section 3.3.1, the representation of provenance was discussed, and the notion of *service-Provenance* was introduced and defined as the captured provenance information of a service instance that pertains to a process. In addition, *process-Provenance* was defined as an instance of a process composed from one or more service instances. Two different constituents of service-Provenance were identified: service activity and message contents. As part of process-Provenance, data link was also identified. This section considers how these representations of provenance can be modelled, i.e., the data structure used to represent each type of provenance is defined. Based on the data models, a common structure is produced to structure the documentation of a process in the provenance database.

The provenance collecting and recording functionalities of the PCS are presented using the provenance model. These components were discussed in chapter 3, which also focused on their interactions.

This chapter is organised as follows. Section 4.2 presents the provenance model that is used to represent process documentation in a standard format. Having presented the provenance structure, Section 4.3 discusses how this structure is used when collection and recording is performed for a running or executed process (discussed in

Chapter 3).

4.2 Provenance Modelling

It should be noted that, in general, models used to represent provenance could be defined in different languages, such as XML or RDF [87]. To depict the models, we adopt the graphical representation of RDF schema which is also the method we use to encode the structure to represent the documentation of a process. The models are explained with a simple example data model to illustrate the process documentation structure and use of RDF schema.

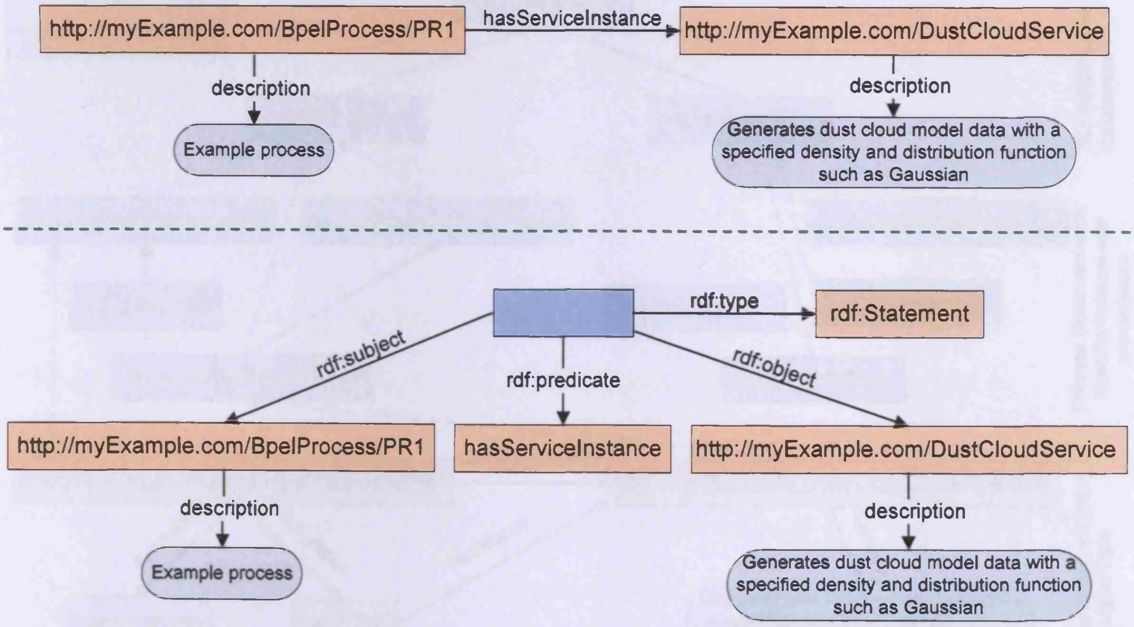

http://myExample.com/BpelProcess/PR1
hasServiceInstance
http://myExample.com/DustCloudService
description
Example process
description
Generates dust cloud model data with a specified density and distribution function such as Gaussian
rdf:type
rdf:Statement
rdf:subject
rdf:predicate
rdf:object
http://myExample.com/BpelProcess/PR1
hasServiceInstance
http://myExample.com/DustCloudService
description
Example process
description
Generates dust cloud model data with a specified density and distribution function such as Gaussian

Figure 4.1: Data model mapped to RDF statements of subject-predicate-object form

Example data model: The top half of Figure 4.1 presents a data model denoting the asserted provenance for an instance of a process with a service and related descriptions. This can be mapped to an RDF model that is based upon the idea of making statements about resources in the form of a subject-predicate-object expression, called a triple in RDF terminology.

The *subject* of an RDF statement is a resource, possibly as named by a Uniform Resource Identifier (URI). Some resources are unnamed and are called blank nodes (Bnodes) or anonymous nodes. They are not directly identifiable. The *predicate* is a resource as well, representing a relationship. The *object* is a resource or a unicode string literal. In the lower half of Figure 4.1, the above example data model is mapped to a simple RDF statement stating that a service, named DustCloudService, has been invoked within a process named PR1.

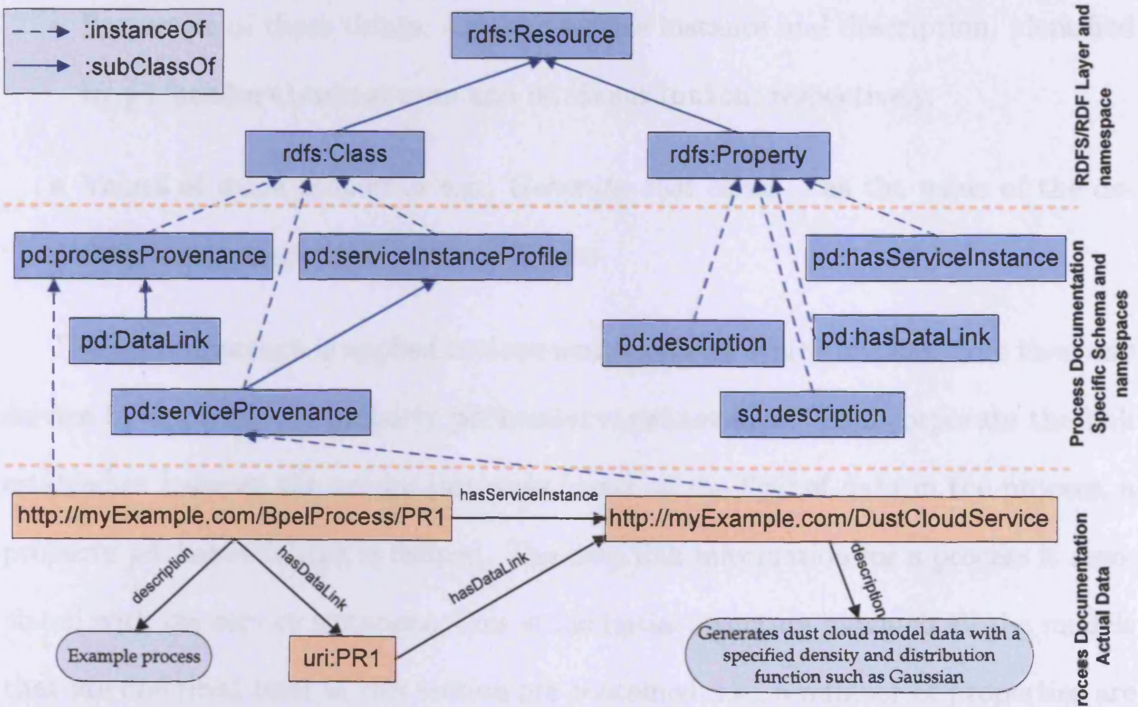

:instanceOf
:subClassOf
rdfs:Resource
rdfs:Class
rdfs:Property
RDFS/RDF Layer and namespace
pd:processProvenance
pd:serviceInstanceProfile
pd:hasServiceInstance
pd:DataLink
pd:description
pd:hasDataLink
pd:serviceProvenance
sd:description
Process Documentation Specific Schema and namespaces
http://myExample.com/BpelProcess/PR1
hasServiceInstance
http://myExample.com/DustCloudService
description
hasDataLink
hasDataLink
description
Example process
uri:PR1
Generates dust cloud model data with a specified density and distribution function such as Gaussian
Process Documentation Actual Data

Figure 4.2: RDFS data model

Figure 4.2 is a graphical illustration of the process documentation schema modelled using an RDFS data model. Here, the instance PR1 is defined to be of rdf:type `pd:processProvenance`, the DustCloudService instance is of rdf:type `sd:serviceProvenance`, and properties are defined such as `pd:hasServiceInstance`, where pd and sd are prefixes used to identify the URI of the process documentation namespace. Thus in RDF, an English statement could be identified using URIs to identify:

- A process instance, e.g., PR1, identified by `http://myExample.com/BpelProcess/PR1`
- Kinds of things, e.g., process provenance, identified by pd:processProvenance
- Properties of those things, e.g., the service instance and description, identified by `pd:hasServiceInstance` and `sd:description`, respectively.
- Values of those properties e.g., *Generate dust cloud...* as the value of the description property of DustCloudService.

The same approach is applied to document a process which invokes more than one service by applying the property `pd:hasServiceInstance`. To incorporate the link established between the service instances based on the flow of data in the process, a property `pd:hasDataLink` is defined. The data link information for a process is associated with the service instances. This is the initial structure in which all the models that are described later in this section are contained, i.e., a number of properties are described to structure the documentation of a process . The structure allows the identification of the context in which provenance capture of a service invocation is made and its association with a process. The structure organizes the service-Provenance in a manner that allows the provenance of a piece of data to be retrieved. Thus, the

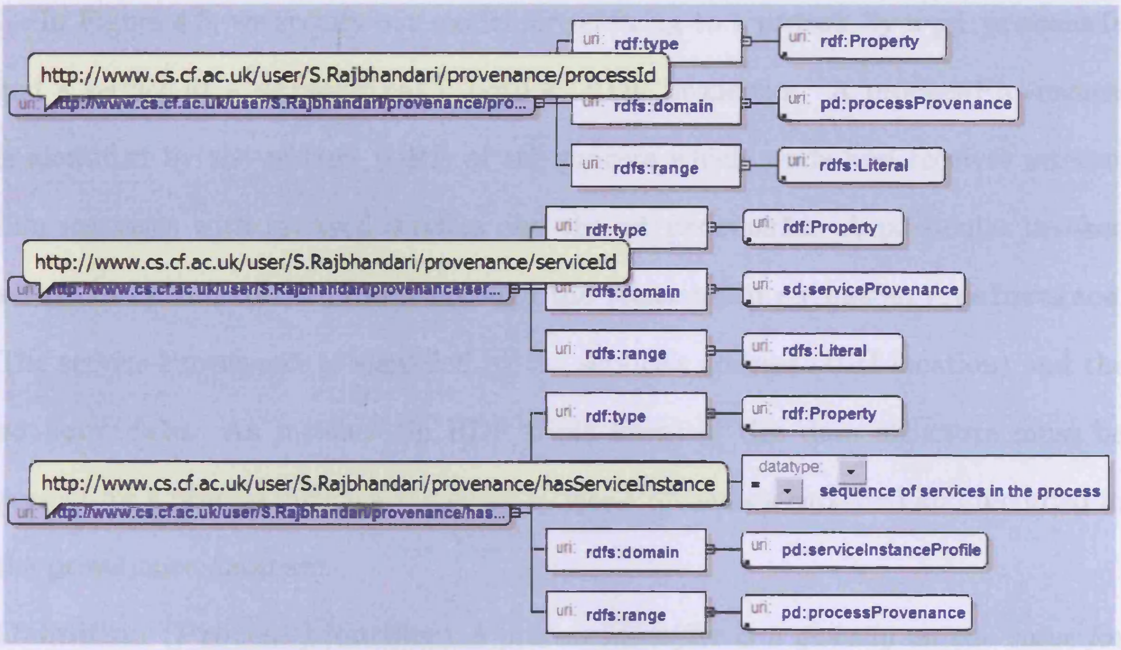

http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/processId
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/pro...
uri: rdf:type
uri: rdf:Property
uri: rdfs:domain
uri: pd:processProvenance
uri: rdfs:range
uri: rdfs:Literal
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceId
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/ser...
uri: rdf:type
uri: rdf:Property
uri: rdfs:domain
uri: sd:serviceProvenance
uri: rdfs:range
uri: rdfs:Literal
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/hasServiceInstance
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/has...
uri: rdf:type
uri: rdf:Property
datatype:
sequence of services in the process
uri: rdfs:domain
uri: pd:serviceInstanceProfile
uri: rdfs:range
uri: pd:processProvenance

Figure 4.3: Model for identifying service instance of a process

captured provenance of a process is recorded in the provenance database using this structure.

4.2.1 Identifying the service-Provenance of a process

Each service-Provenance is defined in the context of a service invocation associated with a process. A service activity consists of the operation and state of a specific service instance, the message contents consists of the details of the data received and sent by a service instance, and the data link contains information that relates a service to other services to which data flows. Therefore, in order to model different parts of service-Provenance and process-Provenance, first a way to relate service-Provenance to a process must be identified.

In Figure 4.3, we specify our model for referring to a process by a `pd:processId` and a service as a `sd:serviceId`, both as RDF properties. A process-Provenance is identified by the address (URI) of the process which sends and receives interaction messages with involved services and the `pd:processId`. A particular invoked service is related to the process through the relationship `pd:hasServiceInstance`. The service-Provenance is identified by the service's address (URI location) and the `sd:serviceId`. An instance (in RDF triple form) of this data structure must be present for a process instance and every service-Provenance captured and recorded in the provenance database.

Definition (Process Identifier) *A process identifier is a globally unique value for identifying a given process instance.*

Definition (Service Identifier) *A service identifier is a globally unique value for identifying each service instance associated with a process.*

The unique IDs are universally unique identifiers or UUID's and are generated using methods that format UUID's according to DCE UUID convention [6].

4.2.2 Identifying process-Provenance

Before discussing the constituent parts of service instances pertaining to a process that is being captured and recorded, the PCS must also retain the information about the abstract process description and its creator. As discussed in Chapter 3, this meta information is useful for matching the concrete process that is documented with the abstract process.

This meta-information is defined as RDF properties that exist in the `pd:process-`

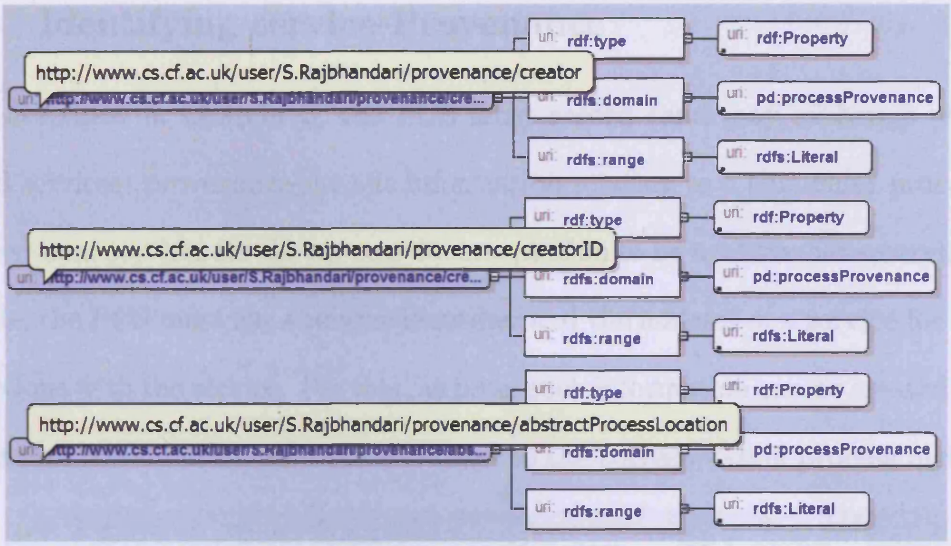

http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/creator
uri http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/cre...
uri rdf:type
uri rdfs:domain
uri rdfs:range
uri rdf:Property
uri pd:processProvenance
uri rdfs:Literal
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/creatorID
uri http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/cre...
uri rdf:type
uri rdfs:domain
uri rdfs:range
uri rdf:Property
uri pd:processProvenance
uri rdfs:Literal
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/abstractProcessLocation
uri http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/abs...
uri rdf:type
uri rdfs:domain
uri rdfs:range
uri rdf:Property
uri pd:processProvenance
uri rdfs:Literal

Figure 4.4: Model for an abstract process

`Provenance` domain and have typed values, e.g., rdfs:range `literal` as depicted in Figure 4.4. The `pd:creator` may consist of the name of the abstract process creator, and `pd:creatorID` is the unique identity given to that creator. The `pd:abstractProcessLocation` gives the location of the file containing the abstract description within the provenance database. The instance of the abstract description must be recorded separately in the provenance database when the process is instantiated by the PCS.

Note that a process instance itself is a composite service, and thus conforms to the service-Provenance structure, particularly the `sd:ServiceActivity` and `sd:MessageContents` that are discussed below. Therefore, in addition to recording the process-specific information, the PCS must also record provenance-specific information about the process in the context of a service.

4.2.3 Identifying service-Provenance

As described in Chapter 3, the PCS must record (and may exchange with the invoked services) provenance-specific information relating to a particular process and its constituent services for the process documentation to be usable when querying. For example, the PCS must use a unique identifier and the address of a service for certain interactions with the service. For this purpose, such information can be created during each service instantiation and a link created to the corresponding process instance.

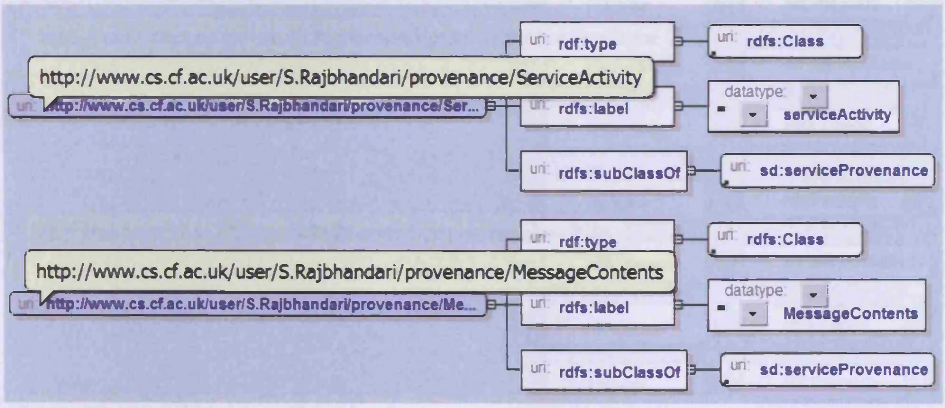

uri: rdf:type
uri: rdfs:Class
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/ServiceActivity
uri: http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/Ser...
uri: rdfs:label
datatype:
serviceActivity
uri: rdfs:subClassOf
uri: sd:serviceProvenance
uri: rdf:type
uri: rdfs:Class
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/MessageContents
uri: http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/Me...
uri: rdfs:label
datatype:
MessageContents
uri: rdfs:subClassOf
uri: sd:serviceProvenance

Figure 4.5: Model for identifying constituents of a service instance

A service instance contains an `sd:serviceID` and a number of properties that are defined within two categories of service-Provenance. These two parts of service-Provenance are defined as being of `RDF` type rdfs:class, and are sub-classes of `sd:serviceProvenance`, as depicted in Figure 4.5.

Identifying Service Activity

The class `sd:ServiceActivity` contains properties to define the state and service-specific static information for a particular service instance, such as the service interface

URL and the start and end times of an interaction. Figure 4.6 outlines the structure for representing such static information. The `sd:wsdlURL` contains the location of the service interface, `sd:serviceName` contains the name defined for that service, and `sd:serviceOperationName` contains the operation executed in the context of the service instance. It should be noted that a service may have more than one operation but only one is invoked for a service instance, and the PCS must be able to capture and record this.

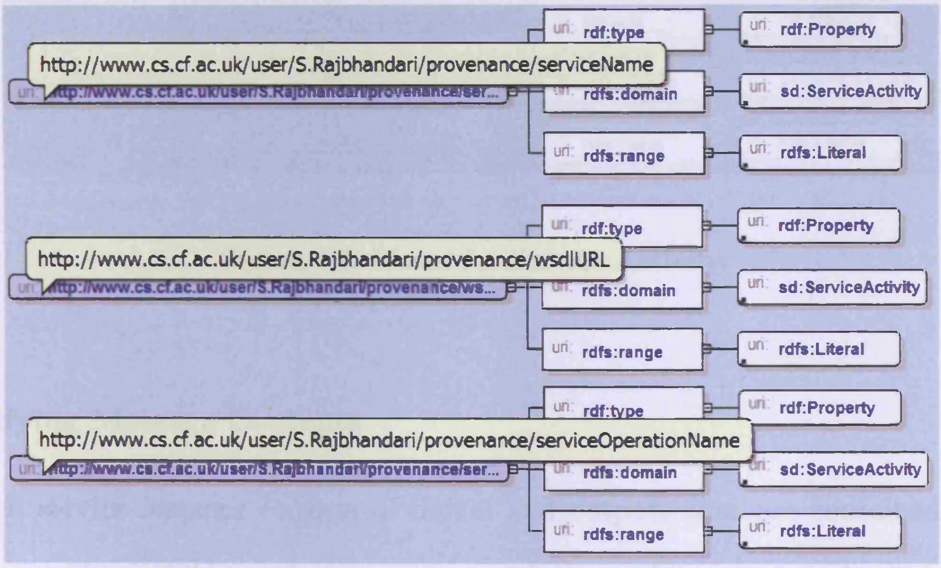

http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceName
uri: http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/ser...
uri: rdf:type
uri: rdf:Property
uri: rdfs:domain
uri: sd:ServiceActivity
uri: rdfs:range
uri: rdfs:Literal
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/wsdlURL
uri: http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/ws...
uri: rdf:type
uri: rdf:Property
uri: rdfs:domain
uri: sd:ServiceActivity
uri: rdfs:range
uri: rdfs:Literal
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceOperationName
uri: http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/ser...
uri: rdf:type
uri: rdf:Property
uri: rdfs:domain
uri: sd:ServiceActivity
uri: rdfs:range
uri: rdfs:Literal

Figure 4.6: Model for service activity

The PCS must also capture the dynamic information, such as the time of the service invocation made by the engine and the time it received the response. In a similar manner, this structure is defined as shown in Figure 4.7. The `sd:serviceStatus` must be assigned a value based on the response by the workflow engine in its interaction with the service. This type of dynamic provenance information must also be recorded for a process instance by mapping the structure in Figure 4.7 to the pd:processProvenance domain.

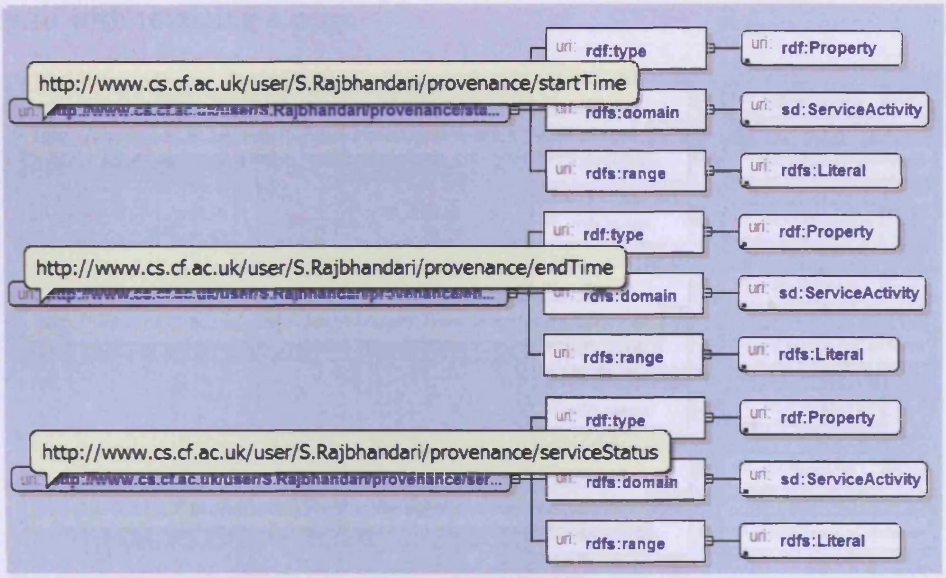


Figure 4.7: Model for Service Activity

Identifying Message Contents

Each service instance consists of inputs and outputs that are contained in the messages exchanged during the interactions with the workflow engine. As discussed in Chapter 3, a reply from an interaction with a service may not contain the actual data but a reference pointing to them. Thus, the message contents are modelled in such a way to support the ability to include: (1) a copy of the actual messages exchanged, for example, a SOAP message; and, (2) specific data within the messages to identify the dataflow. This section discusses the former, which is mandatory (particularly for a process where initial inputs and the final outputs are necessary, for example to perform a what-if analysis using the process). Note that the copy of a SOAP message may also contain attachments. We are not concerned with processing the messages but only to with retaining a copy.

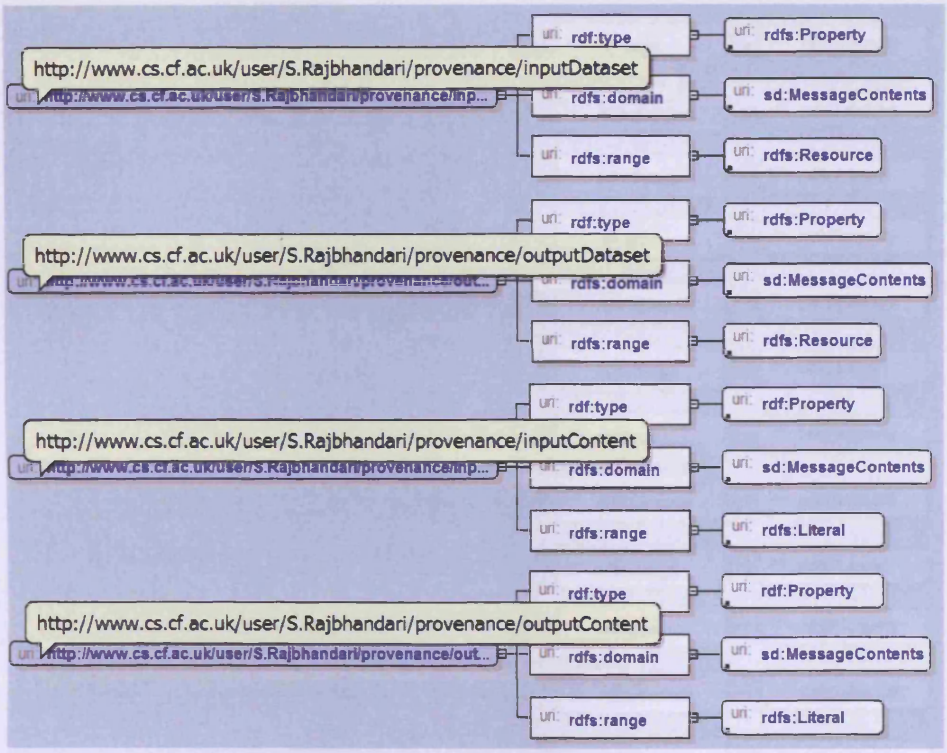
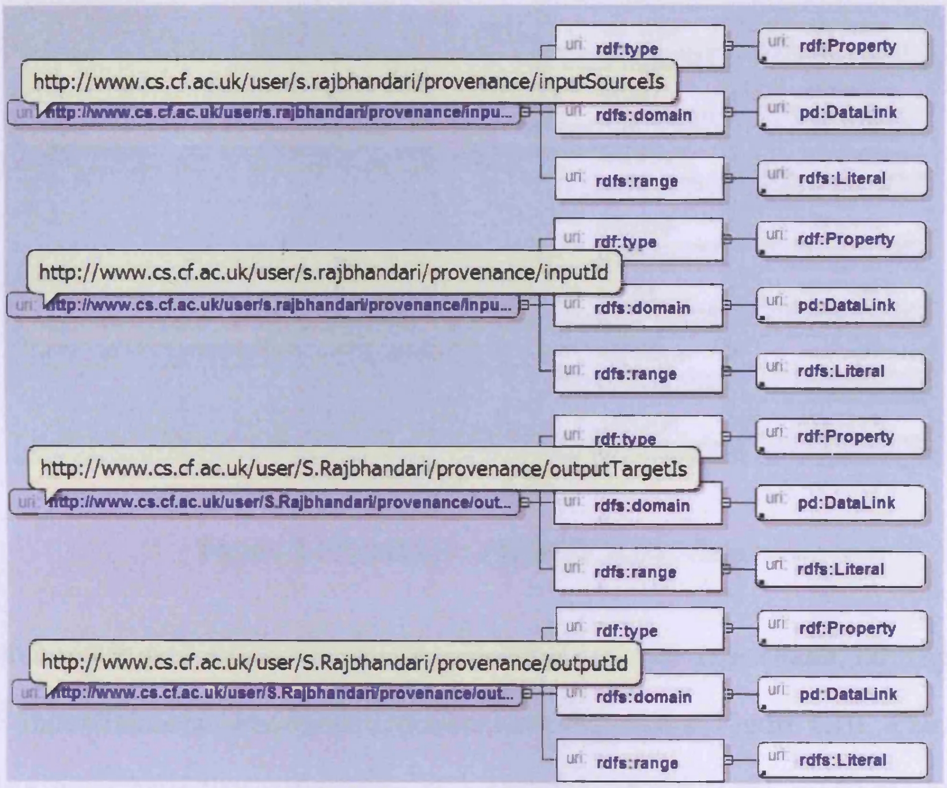

http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/inputDataset
uri http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/inp...
uri rdf:type
uri rdfs:Property
uri rdfs:domain
uri sd:MessageContents
uri rdfs:range
uri rdfs:Resource
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/outputDataset
uri http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/out...
uri rdf:type
uri rdfs:Property
uri rdfs:domain
uri sd:MessageContents
uri rdfs:range
uri rdfs:Resource
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/inputContent
uri http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/inp...
uri rdf:type
uri rdf:Property
uri rdfs:domain
uri sd:MessageContents
uri rdfs:range
uri rdfs:Literal
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/outputContent
uri http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/out...
uri rdf:type
uri rdf:Property
uri rdfs:domain
uri sd:MessageContents
uri rdfs:range
uri rdfs:Literal

Figure 4.8: Model for the I/O messages

First inputs and outputs are defined for a service-Provenance as `RDFS` properties `sd:inputDataset` and `sd:outputDataset`, contained in domain `sd:messageContents` and with rdfs:range `resource` depicted as in Figure 4.8. Using a blank node each is linked to the input and output messages for a service instance, whose structure is depicted in Figure 4.8, where two `RDFS` properties `sd:inputContents` and `sd:outputContents` are defined that have rdfs:ranges `literal` within the `sd:MessageContents` domain. The literals must contain values that are copies of the actual messages exchanged for a service instance during its interactions with the engine. The same must also be defined for a process instance as the process itself is a composite service that consists of input and output messages.



http://www.cs.cf.ac.uk/user/s.rajbhandari/provenance/inputSourceIs
uri http://www.cs.cf.ac.uk/user/s.rajbhandari/provenance/inpu...
uri rdf:type
uri rdf:Property
uri rdfs:domain
uri pd:DataLink
uri rdfs:range
uri rdfs:Literal
http://www.cs.cf.ac.uk/user/s.rajbhandari/provenance/inputId
uri http://www.cs.cf.ac.uk/user/s.rajbhandari/provenance/inpu...
uri rdf:type
uri rdf:Property
uri rdfs:domain
uri pd:DataLink
uri rdfs:range
uri rdfs:Literal
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/outputTargetIs
uri http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/out...
uri rdf:type
uri rdf:Property
uri rdfs:domain
uri pd:DataLink
uri rdfs:range
uri rdfs:Literal
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/outputId
uri http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/out...
uri rdf:type
uri rdf:Property
uri rdfs:domain
uri pd:DataLink
uri rdfs:range
uri rdfs:Literal

Figure 4.9: Model to identify the flow of data

4.2.4 Identifying Data Link

The context information to identify the flow of data conforms to the inputs and outputs of the service instance, and the structure is shown in Figures 4.9, 4.10 and 4.11. As discussed in section 3.3.1, for a service instance, a message received during an interaction with the engine may contain data inputs from different sources. Section 3.3.1 also described how data flow can be determined from such information.

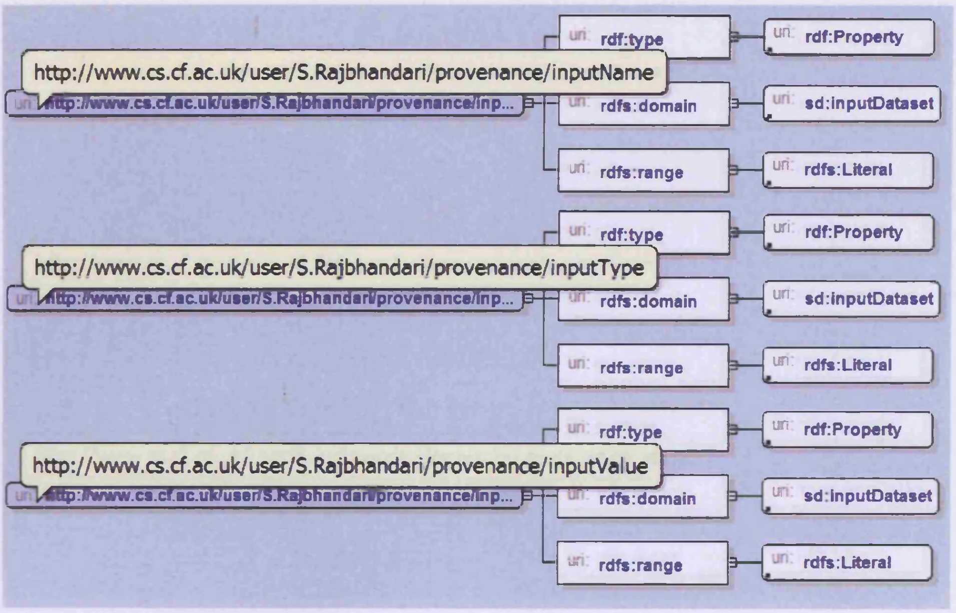

http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/inputName
uri http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/inp...
uri rdf:type
uri rdf:Property
uri rdfs:domain
uri sd:inputDataset
uri rdfs:range
uri rdfs:Literal
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/inputType
uri http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/inp...
uri rdf:type
uri rdf:Property
uri rdfs:domain
uri sd:inputDataset
uri rdfs:range
uri rdfs:Literal
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/inputValue
uri http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/inp...
uri rdf:type
uri rdf:Property
uri rdfs:domain
uri sd:inputDataset
uri rdfs:range
uri rdfs:Literal

Figure 4.10: Model to identify input data

Each input in such a message must be identified with `sd:inputName`, `sd:inputType` and `sd:inputValue` of domain `sd:inputDataset` depicted in Figure 4.10. The `sd:inputValue` must have the actual data or parameters of the input message. In order to provide the data link, each input data must be identified with `sd:inputId` and `sd:inputSourceIs` of domain `pd:DataLink` (Figure 4.9). The `sd:inputSourceIs` must contain the address of the sender of the data and `sd:inputId` must have a unique identifier for that data. Similarly for `sd:outputValue`, the `sd:outputTargetIs` and `sd:outputId` must have the address of the data receiver and the unique key, respectively.

The structure defined differentiates data from different sources and targets contained in the input and output messages for that service instance. By querying and matching this information, particularly `sd:inputId` and `sd:inputSourceIs` with

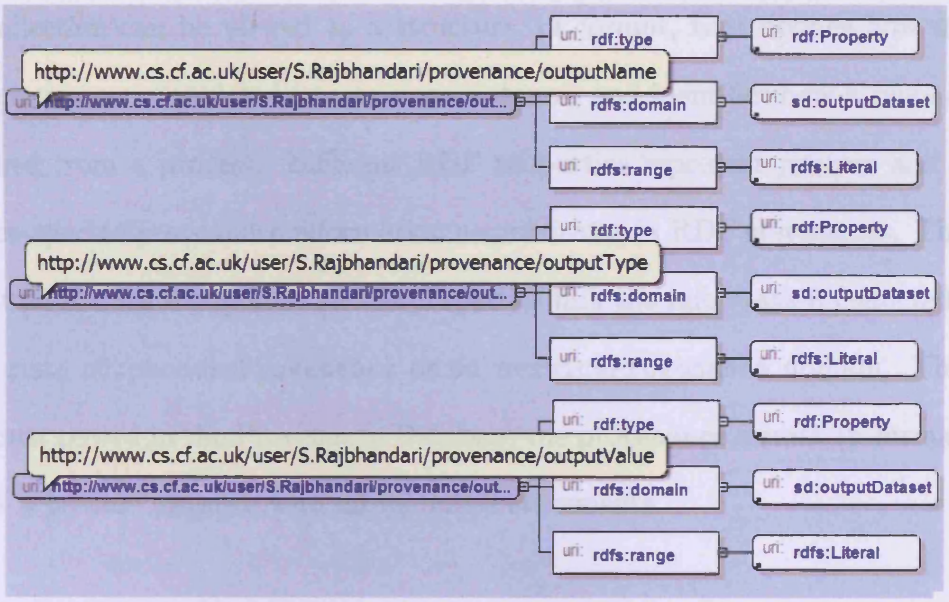

uri: rdf:type
uri: rdf:Property
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/outputName
uri: http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/out...
uri: rdfs:domain
uri: sd:outputDataset
uri: rdfs:range
uri: rdfs:Literal
uri: rdf:type
uri: rdf:Property
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/outputType
uri: http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/out...
uri: rdfs:domain
uri: sd:outputDataset
uri: rdfs:range
uri: rdfs:Literal
uri: rdf:type
uri: rdf:Property
http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/outputValue
uri: http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/out...
uri: rdfs:domain
uri: sd:outputDataset
uri: rdfs:range
uri: rdfs:Literal

Figure 4.11: Model to identify output data

that of the other service instances `sd:outputId` and `sd:outputTargetIs`, one can determine the dataflow.

Thus, during recording, the PCS must use the same unique identifier for a data item that is received from one service (source) and sent to another service (target). From this the flow of the data can be identified. Specially for this purpose, such information can be created during each service instantiation and the data flow link identified by comparing the data in an input message with all data in the messages received from previous invocations.

4.2.5 Provenance Format (p-format)

Up to this point the architecture has assumed that the Provenance Database contains a collection of RDF statements defining the documentation of a process.

This collection can be viewed as a structure, or format, that defines how different service instances are related to a process instance, and identifies how a piece of data is derived from a process. Different RDF properties represent process and service instance-specific provenance information as *predicates* in RDF statements. These are structured in such a way that all the properties link the information collected to the appropriate `pd:processProvenance` or `sd:serviceProvenance` domain. Thus, the PCS must record in the Provenance Database the provenance format (p-format) that depicts a process instance with all its linked statements.

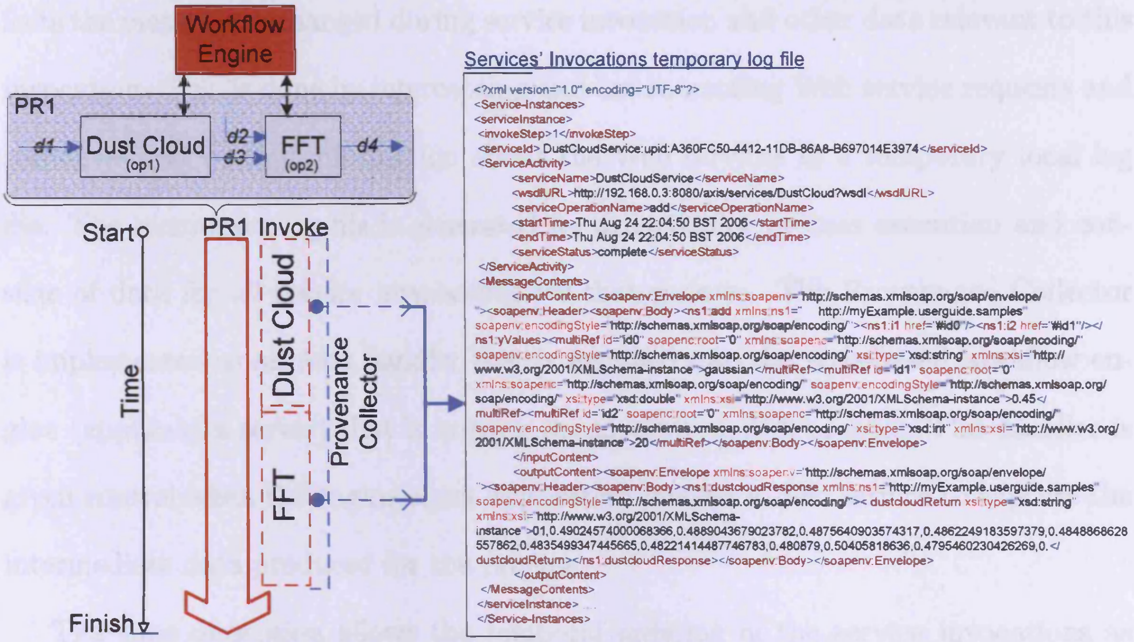
4.3 Provenance Collection Service

In section 3.3.2 a middleware service was introduced, and the interactions of the PCS components in collecting and recording the provenance of a web process were discussed. This section discusses the recording functionalities of the PCS supported by the provenance server and the collection process of the middleware service.

4.3.1 Provenance Recording Interface

The PCS consists of a provenance recording interface for recording the provenance collected about a process in the standard format in the Provenance Database. As discussed in chapter 3, recording occurs in two phases on the client side where provenance is collected. In the first phase, information about the process invocation itself is collected. In the second phase, information about the invocations of the services involved in the process is collected. The PCS provides an interface that collects manually entered data about a process, and also initiates the process execution. The

PCS also collects and records the input data and the resultant data returned by the process. For a process composed of services a middleware component named the Provenance Collector gathers data about each service invocations.

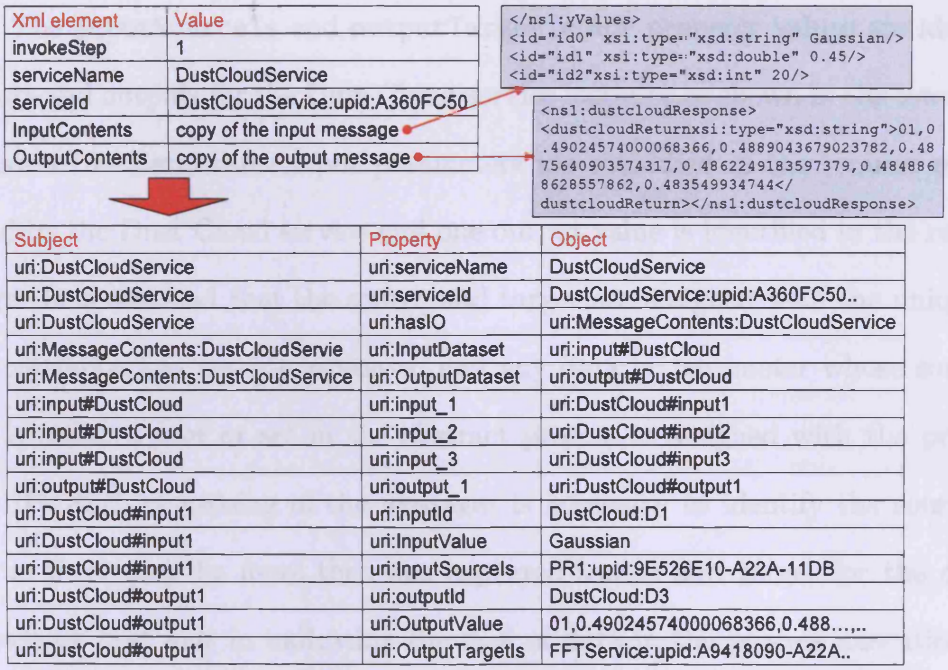


```
<?xml version="1.0" encoding="UTF-8"?>
<Service-Instances>
<serviceInstance>
 <invokeStep>1</invokeStep>
 <serviceId> DustCloudService:upid:A360FC50-4412-11DB-86A8-B697014E3974</serviceId>
 <ServiceActivity>
        <serviceName>DustCloudService</serviceName>
        <wsdlURL>http://192.168.0.3:8080/axis/services/DustCloud?wsdl</wsdlURL>
        <serviceOperationName>add</serviceOperationName>
        <startTime>Thu Aug 24 22:04:50 BST 2006</startTime>
        <endTime>Thu Aug 24 22:04:50 BST 2006</endTime>
        <serviceStatus>complete</serviceStatus>
 </ServiceActivity>
 <MessageContents>
        <inputContent> <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
"><soapenv:Header><soapenv:Body><ns1:add xmlns:ns1="     http://myExample.userguide.samples"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><ns1:i1 href="#id0"/><ns1:i2 href="#id1"/></
ns1:yValues><multiRef id="id0" soapenc:root="0" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xsi:type="xsd:string" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">gaussian</multiRef><multiRef id="id1" soapenc:root="0"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" soapenv:encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/" xsi:type="xsd:double" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">0.45</
multiRef><multiRef id="id2" soapenc:root="0" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xsi:type="xsd:int" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance">20</multiRef></soapenv:Body></soapenv:Envelope>
        </inputContent>
        <outputContent><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
"><soapenv:Header><soapenv:Body><ns1:dustcloudResponse xmlns:ns1="http://myExample.userguide.samples"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">< dustcloudReturn xsi:type="xsd:string"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">01,0.4902457400068366,0.4889043679023782,0.4875640903574317,0.4862249183597379,0.4848868628
557862,0.4835499347445665,0.4822141448774678 3,0.480879,0.50405818636,0.4795460230426269,0.</
dustcloudReturn></ns1:dustcloudResponse></soapenv:Body></soapenv:Envelope>
        </outputContent>
 </MessageContents>
</serviceInstance> ...
</Service-Instances>
```

Figure 4.12: Asynchronous data collection for process execution

The example process PR1 in Figure 4.12 consists of two services, DustCloud and FFT, that expose operations (methods) op1 and op2, respectively. Dust Cloud service takes one data set (consisting of three input parameters) and produces one data set as output. The FFT service takes two data sets (where *d2* is an input parameter set in the abstract process) and produces one data set. The dataflow is represented by an abstract process that is executed by the workflow engine. Each service is invoked by the workflow engine and the flow of data is mediated through the engine. All invocations and dataflow activities during the execution of a process occur at the boundaries that appear between the start and finish of the process depicted in Figure 4.12. Figure 4.12 depicts a Provenance Collector that collects required data about the invocations of a process execution. The Provenance Collector asynchronously collects the messages exchanged during service invocation and other data relevant to this invocation. This is done by intercepting and instrumenting Web service requests and responses and writing information about the Web services to a temporary local log file. The temporary log file is generated for a particular process execution and consists of data for all service invocations for that process. The Provenance Collector is implemented as an Axis handler that is installed into the Axis-based workflow engine (application server) that is hosting the monitored Web process. This handler is given control when the engine client application invokes a Web service to capture the intermediate data produced for the process.

The time dimension allows the temporal ordering of the service invocations as they occur, and makes local log file construction independent of the order in which the invocations are propagated by the Provenance Collector. In the absence of a single globally synchronized clock, we determine the time dimension by associating a logical time stamp which is a counter, called process *invokeStep*, as an element for each invocation. The central workflow engine maintains this *invokeStep* and assigns it for each service invocation event as it occurs. The *invokeStep* is an integer that has no relation to the absolute time. Each process has a "logical clock" that starts at 0 and incremented by 1 on each service invocation event. This sequences the service invocations in the process example in Figure 4.12 as PR1-DustCloud-FFT as seen along the time axis of the invocation chain.



Xml element	Value
invokeStep	1
serviceName	DustCloudService
serviceId	DustCloudService:upid:A360FC50
InputContents	copy of the input message
OutputContents	copy of the output message

```
</ns1:yValues>
<id="id0" xsi:type="xsd:string" Gaussian/>
<id="id1" xsi:type="xsd:double" 0.45/>
<id="id2"xsi:type="xsd:int" 20/>
```

```
<ns1:dustcloudResponse>
<dustcloudReturnxsi:type="xsd:string">01,0
.49024574000068366,0.4889043679023782,0.48
75640903574317,0.4862249183597379,0.484886
8628557862,0.483549934744</
dustcloudReturn></ns1:dustcloudResponse>
```

Subject	Property	Object
uri:DustCloudService	uri:serviceName	DustCloudService
uri:DustCloudService	uri:serviceId	DustCloudService:upid:A360FC50..
uri:DustCloudService	uri:hasIO	uri:MessageContents:DustCloudService
uri:MessageContents:DustCloudServie	uri:InputDataset	uri:input#DustCloud
uri:MessageContents:DustCloudService	uri:OutputDataset	uri:output#DustCloud
uri:input#DustCloud	uri:input_1	uri:DustCloud#input1
uri:input#DustCloud	uri:input_2	uri:DustCloud#input2
uri:input#DustCloud	uri:input_3	uri:DustCloud#input3
uri:output#DustCloud	uri:output_1	uri:DustCloud#output1
uri:DustCloud#input1	uri:inputId	DustCloud:D1
uri:DustCloud#input1	uri:InputValue	Gaussian
uri:DustCloud#input1	uri:InputSourceIs	PR1:upid:9E526E10-A22A-11DB
uri:DustCloud#output1	uri:outputId	DustCloud:D3
uri:DustCloud#output1	uri:OutputValue	01,0.49024574000068366,0.488…,..
uri:DustCloud#output1	uri:OutputTargetIs	FFTService:upid:A9418090-A22A..

Figure 4.13: Service instance given by an RDF triple

The format of the temporary log file conforms to the *service-Provenance* XML Schema. The temporary log file is part of the process execution and is placed in a web-accessible location or URL. The URL location is specified in the configuration file of the engine to place the log file in the given URL. This is queried and interpreted by the PCS in the provenance server to record its contents into the Provenance Database using the provenance-format (see the RDF provenance model discussed in section 4.2). Figure 4.13 shows how a service instance (i.e., DustCloud service) captured in the temporary log file is summarized with RDF triples by the PCS. The data contained in the copy of the messages (i.e., SOAP messages) received and sent are processed by the PCS to uniquely identify the input and output data in the messages (i.e., unique values in properties `inputId` and `outputId` for each input and output value). The `inputSourceIs` and `outputTargetIs` RDF property values are identified for inputs and outputs for the Dust Cloud service instance as shown in the lower table of Figure 4.13. Here, three input parameters are identified in the request message received by the Dust Cloud service and one output value is identified in the response message. It is assumed that the source and target are assigned with the unique IDs of the corresponding service instances, and any data or parameter whose source or target is either a user or set in the abstract process is assigned with the process's unique ID. Such processing of the messages is necessary to identify the source and target as there may be more then one captured source and target for the data in the messages that aids in indicating the flow of data in the process execution. For example, in Figure 4.12, for the FFT service, d2 and d3 are contained in a request message. Here, the source of d2 is assigned with the process ID as this is set in the abstract process and d3's source is assigned as the Dust Cloud service ID as this is the output data from the Dust Cloud service. The algorithm that enables this processing of the messages in the XML log file is carried out as follows:

1. For every service instance in the XML log file, the data in the elements inputContent and outputContent are processed to extract and identify the input and output values in the messages. This assumes the Web services have only simple/primitive data types. In case of an application specific message or complex types, the copies of the entire messages or values in the elements inputContents and outputContents of the XML file are recoded as literal objects in RDF statements.

2. Each input and output is assigned a unique ID associated with that service

instance.

3. For each input value of a given service instance check if any preceding service instances' output value matches to identify the source of this input. Similarly, the target for each output in a service instance is identified. This is performed using simple string matching. The IDs of the input data are overwritten with IDs of the matched preceding output values. If the input values does not match any of the output values of the preceding service instances, then a default value (i.e., process ID) is assigned as the source. The targets for outputs are also determined in a similar manner.

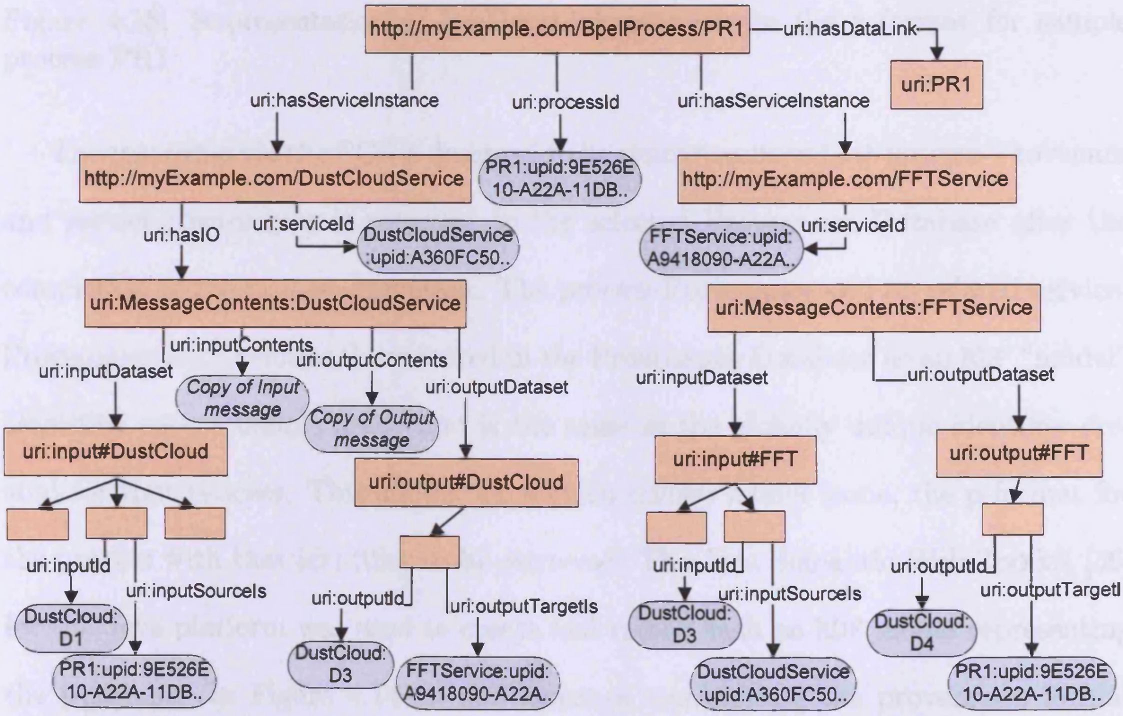


Figure 4.14: RDF instance representing the p-format for sample process PR1

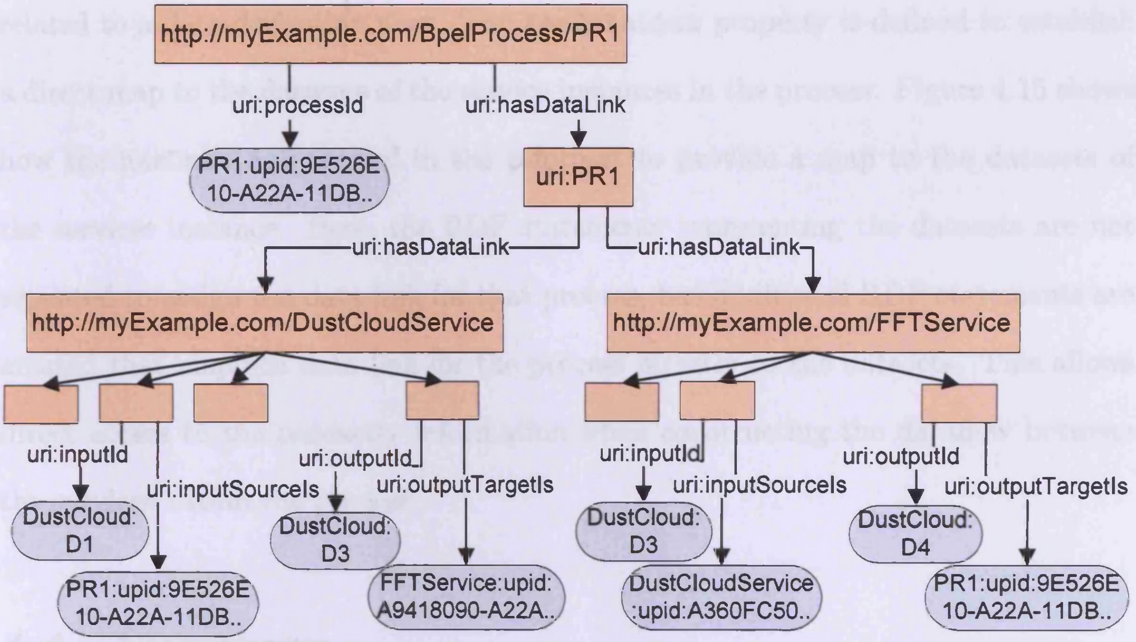

http://myExample.com/BpelProcess/PR1
uri:processId
uri:hasDataLink
PR1:upid:9E526E 10-A22A-11DB..
uri:PR1
uri:hasDataLink
uri:hasDataLink
http://myExample.com/DustCloudService
http://myExample.com/FFTService
uri:inputId
uri:inputSourceIs
uri:outputId
uri:outputTargetIs
DustCloud: D1
PR1:upid:9E526E 10-A22A-11DB..
DustCloud: D3
FFTService:upid: A9418090-A22A..
uri:inputId
uri:inputSourceIs
uri:outputId
uri:outputTargetIs
DustCloud: D3
DustCloudService :upid:A360FC50..
DustCloud: D4
PR1:upid:9E526E 10-A22A-11DB..

Figure 4.15: Representation of hasDataLink property in the p-format for sample process PR1

The recording via the PCS is designed to be synchronous so that process-Provenance and service-Provenance is recorded to the selected Provenance Database after the completion of the process execution. The process-Provenance and its related service-Provenance (or a p-format) are stored in the Provenance Database as an RDF "model" identified with a unique name that is the same as the globally unique identifier created for that process. This allows, for a given unique model name, the p-format for the process with that identifier to be retrieved. The Jena Semantic Web Toolkit [36] for the Java platform was used to create and record such an RDF model representing the p-format. In Figure 4.14 an RDF instance representing the provenance format for a part of the example process PR1 (Figure 4.12) is depicted and shows how the resources are connected with properties defining relationships between the two service instances. These statements represent the process-based view that can also be related to a data-derivation view. The `hasDataLink` property is defined to establish a direct map to the datasets of the service instances in the process. Figure 4.15 shows how the `hasDataLink` is used in the p-format to provide a map to the datasets of the services instance. Here, the RDF statements representing the datasets are not repeated to assign the data link for that process, but additional RDF statements are created that map the data link for the process directly to the datasets. This allows direct access to the necessary information when constructing the dataflow between the services within the process.

4.4 Summary

This chapter first discussed our proposal to model the provenance of a service-based application. We presented a model for how process-Provenance and service-Provenance, and the data contained in them, can be identified, as well as a model of dataflow that is separate from the interactions between the engine and the services. The use of RDF triples was proposed, together with the vocabulary of properties that expresses a process instance and the service instances contained in them, and the relationships in intermediate data of the process instance. The complete structure with the collection of RDF properties that facilitates the specifications of the provenance data for a process is viewed as the provenance format or p-format. The provenance format provides a common knowledge base to represent the provenance about an enacted process so the PCS and PQS can utilize this common p-format for recording and querying the provenance information, respectively. The RDF schemas for both the service-Provenance and process-Provenance (i.e., p-format) are attached as Appendix A, and have been checked using a trial version of Altova SemanticWorks tool [15] for validity and well-formedness.

The PCS is also been discussed in this chapter, particularly the Provenance Collector that intercepts the intermediate data produced by service invocations for a process execution and records them in a temporary XML log file. We demonstrated with an example process how the XML log file is processed by the PCS to map the service instance data to RDF triples of p-format, and discussed formulating RDF triples to represent the data link associated with the process. The service-Provenance XML schema is attached as Appendix B of this thesis, and was created using the XMLSpy tool [14].

In summary, we have presented a provenance representation model for an SOA that is referred to as the p-format. We have also discussed the provenance recording interface of the PCS component, focusing on the functionally of the Provenance Collector that collects service invocation logs for processes and how this is mapped to the p-format standard that includes the data link information, so as to record it in the Provenance Database. Having addressed the provenance representation standard and recording needs in an SOA, we now focus on the provenance query requirements to use the recorded provenance about processes by specifying the functional requirements of the query interfaces of the PQS.

Chapter 5

Provenance Querying and Analysis Tool

5.1 Introduction

Chapter 4 discussed the Provenance Query Service (PQS), and introduced the functionality of its query interfaces: (1) the *process provenance query interface*; and, (2) the *provenance reasoning query interface.* In this section, the functional requirements of the process provenance query interface and the provenance reasoning interface are specified. The query interface provides support to query RDF data using SPARQL [43].

5.2 Process Provenance Query Interface

The process provenance query interface is used to retrieve the documentation of a process that makes up the provenance of a workflow result. The process provenance query interface provides access for the user to perform two types of queries on the selected provenance database:

- retrieve all the identifiers and descriptions for all the process provenance in the provenance database.
- retrieve a process provenance with a specific identifier provided by the querying user.

This allows a user to access globally unique identifiers and descriptions of all the process instances present in the provenance database, so the user can identify and retrieve the p-format for a specific process. The retrieved p-format provenance data that describes a process instance can be used to:(1) construct a process provenance graph to visualize previously executed process in a way that the querying user can interpret appropriately; and, (2) re-execute the process to verify the result of the execution instance for that process.

5.2.1 Process Provenance Graph Construction

Constructing a graph of activities after their occurrence is important because it allows a user to reason about the high level behaviour of the workflow and the interconnection between services and data flow within and across a process. In this section we describe the algorithm by which provenance graphs can be constructed from a p-format describing the series of tasks performed during a process instance. The p-format that is queried with a given unique identifier is based on RDF triple statements. Given access to the process execution instance containing all the statements (i.e., a process-Provenance and one or more service-Provenance instances), it is possible for the PQS to construct the process graph for the execution. A process graph reconstructed from provenance data may be represented as a directed acyclic graph. The nodes of the process provenance graph are services and data sets form the edges representing the dataflow (within the workflow engine and according to the abstract process description). Thus, the edges, according to their direction arrow, represent data consumed or produced by that service. Additional information present in the service instances are represented as attributes of the nodes and edges.

Thus, the algorithm to construct a process provenance graph using the p-format is similar to an algorithm used to construct a graph given a set of node-edge pairs. Given a retrieved p-format for the process, the process provenance graph is constructed by identifying the service instances and the data link between them (as formulated and discussed in chapter 4). Suppose that, for a given process instance's unique identifier, we query a set of service-Provenance instances (SPI) and each service-Provenance instance s^p have a set of inputs I and outputs O as follows:

$$SPI = \{s^p_1, s^p_2, \ldots, s^p_n\}$$
$$I = \{s^p in_1, s^p in_2, \ldots, s^p in_m\} \qquad O = \{s^p ot_1, s^p ot_2, \ldots, s^p ot_o\}$$

where each $s^p_i (1 \leq i \leq n)$ is a service instance that has $s^p in_j (1 \leq j \leq m)$ and $s^p ot_k (1 \leq k \leq o)$ as the inputs and outputs respectively. For the purpose of constructing the process provenance graph, a service instance s^p and its inputs $s^p in$ and outputs $s^p ot$ are represented as follows:

$$s^p = \langle sn, sid, sw, sop \rangle$$

$$s^p in = \langle inm, it, iv, iid, isr \rangle \qquad s^p ot = \langle on, ot, ov, oid, otg \rangle$$

where *sn* is the service name, *sid* is the unique identifier of the service, *sw* is the WSDL location, and *sop* is the service operation. For each input of a service instance, *inm* is the input name, *it* is the input type, *iv* is the actual input value, *iid* is the ID for the input, and *isr* is the source of the input. Similarly, for the output, *on* is the output name, *ot* is the output type, *ov* is the actual output value, *oid* is the ID for the output, and *otg* is the target of the output. We refer to these components using the "." notation, that is, $s^p.sn$ refers to the service name of s^p and $s^p in.inm$ refers to an input name for the service instance s^p.

Our task of constructing a process provenance graph from this information is a two-step procedure. First, nodes (i.e., rectangular boxes) are constructed referring to the service instances, and identified by service name and the other components. Thereafter, edges are constructed corresponding to the nodes by referring to the components of the inputs and outputs of the service instances. These two steps are discussed as follows:

1. *Construct graph nodes for all the* s^p. The process provenance graph algorithm will first query all the service instances for a given process instance and create nodes that are displayed as rectangular boxes with all the components of the service instances. Constructing nodes is a straightforward process. The following procedure explains how this is performed by a part of the process graph algorithm:

```
variables: $SPI = \{s^p_1, s^p_2, \ldots, s^p_n\}$, $Node$,
1 SET $Node$ to an empty set
2 FOR each $s^p_i$ $(1 \leq i \leq n)$
3     $Node = createNode(s^p_i.sn, s^p_i.sid, s^p_i.sw, s^p_i.sop)$
4 RETURN $Node$
```

The algorithm will first traverse the set of service instances and create a node for each by attaching all its components, e.g., service ID that allows each node to be uniquely identified (line 3). This process continues until all the nodes are created.

2. *Determine and construct graph edges connecting the nodes.* In order to construct the edges that connect the nodes (created in Step 1) to depict the high level behaviour of the process, it is necessary to traverse the inputs and outputs of each service instance. With reference to Figure 3.4 of chapter 3, we produce Figure 5.1 as an example to explain this step. In Figure 5.1, the three service instances are represented with their corresponding unique IDs. It is assumed that the unique IDs of the service instances are used as the values to represent the corresponding source of an input and target of an output. Consider the service instance `Service B` uniquely identified with ID `ServiceB_uid:3` and having two $s^p in$ (i.e., input data), d_2 and d_3. The inputs have $s^p in.iid$ (i.e., input ID component) with values i_2 and i_3 respectively, and $s^p in.sr$ (i.e., input source component) values `Engine_uid:1` and `ServiceA_uid:2`, respectively.

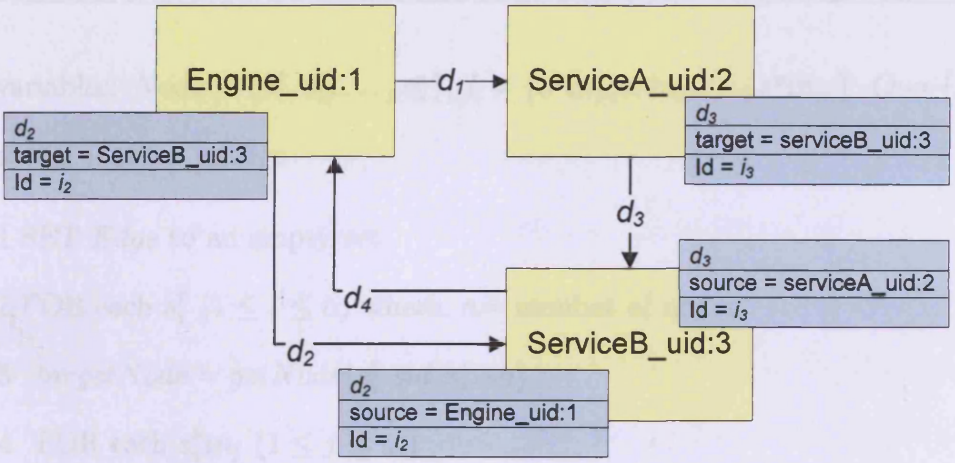


Figure 5.1: Representation of data links of service instances within a process instance by identifying each input and output for a service instance with its source and target, respectively, and the data IDs.

Each input's components, specified in each service instance Node s^p, are used to determine which service instance Nodes created in Step 1 match as the source of the input for that Node s^p. This constructs a link between the two service instance Nodes. That is, the process graph algorithm performs the following:

```
variables: $Node = \{s^p_1, s^p_2, \ldots, s^p_n\}$, $I = \{s^p in_1, s^p in_2, \ldots, s^p in_m\}$, $O = \{s^p ot_1,$
$s^p ot_2, \ldots, s^p ot_o\}$,$Edge$

1 SET $Edge$ to an empty set
2 FOR each $s^p_i$ $(1 \le i \le n)$ where, $n=$ number of nodes
3   $targetNode = getNode(s^p_i.sid, s^p_i.sn)$
4   FOR each $s^p_i in_j$ $(1 \le j \le m)$
5   IF $\{s^p_i in_j.isr == s^p_{i'}.sid, 1 \le i' \le n\}$
6   THEN IF $\{(s^p_i.sid == s^p_{i'} ot_{k'}.otg \quad \& \quad s^p_i in_j.iid == s^p_{i'} ot_{k'}.oid, 1 \le k' \le o)\}$
7             THEN $src = s^p_{i'}.sid \quad \& \quad trg = s^p_i.sid$
8             $data = s^p_i in_j.iv$
9             $sourceNode = getNode(src, s^p_{i'}.sn)$
10            $Edge = createEdge(sourceNode, targetNode, data)$
11 DISPLAY $Node \quad \& \quad Edge$
12 STOP
```

where $s^p_i in_j.isr = s^p_{i'}.sid$ means that each input source value for a Node (*targetNode* in line 3) is compared with the $s^p.sid$ of all the Nodes and must be equal to one of them (line 5). For example, if an input d_3 of a particular service Node ID = *ServiceB_uid* : 3 has source = *serviceA_uid* : 2, then a Node with the ID *serviceA_uid* : 2 is selected at this stage as a source Node. Once a matching source Node is selected, $s^p_i.sid = s^p_{i'} ot_{k'}.otg$ and $s^p_i in_j.iid = s^p_{i'} ot_{k'}.oid$ expresses the fact that the service ID and the input ID for that *targetNode* must be equal to one of the output target and ID values of the matched source Node (line 6). For example, the input data d_3 of the Node ID $=ServiceB\_uid : 3$ has input ID $= i_3$, then the Node ID $= serviceA\_uid : 2$'s output with target $= ServiceB\_uid : 3$ and ID $= i_3$ confirms that the source of this input data is Node with ID $= serviceA\_uid : 2$. The edge is constructed with the identified source and target nodes for that data (line 10). This is a fairly straightforward process, assuming that the algorithm handles any duplicate edges that may occur during the edge creation and display.

The above descriptions explain how a process graph is constructed with the provenance information structured using the p-format. A graphical representation of an executed process created in such a way can be used for a visual comparison with the abstract process description. This enables what was planned to be compared with what actually happened after the enactment of a process. This is useful, particularly when the abstract process description contains conditions such as switch, as discussed in section 3.3.1.

5.2.2 Process Re-Execution

Re-execution is a way of verifying the data product derived from a process execution. Re-execution of a past process mainly serves two purposes:

1. To verify if a result is still up-to-date, meaning whether the information in an input data set used by the process has been updated with new data. In many scientific domains existing data are updated in a database with new data based on recent research findings. In this case, any results from experiments that

were run using the old data may be considered worthless and execution with the updated data becomes desirable in many cases. Such execution may be recorded as a new run of an experiment. Re-execution will either verify the original result, or generate a different result – indicating that an input data set has changed. The provenance database can then be updated with the new result data.

2. To perform a "what-if" style of analysis on the process by changing the input parameters, and setting the algorithm inputs of the services, to investigate interesting results.

In this section the way in which a process can be re-executed given its provenance information is discussed. The re-execution can be performed based on the constructed process provenance graph that defines the "actual" process or by making use of the "abstract" process description. We propose the later as one of our aims is to analyze the process with varying input parameters, which provides the flexibility to make use of any conditions within the abstract process that may produce interesting results from reruns. Such reruns may be recorded, if necessary, in order to produce a process provenance graph for a high-level comparison of different runs (i.e., with different input parameters) of the same process.

5.3 Provenance Reasoning Query Interface

The provenance reasoning query interface accepts a *provenance reasoning query request* and responds with the *provenance reasoning query results.* A provenance reasoning query request defines a search for the provenance of an application element, and the provenance reasoning query result represents provenance for invocations of that element at different instants in time. The query results for an element must be associated with a particular instant in time because the element may have been invoked at a number of different times by different processes. For example, a service element may have different provenance for its invocations within different workflow enactments that happened at different times. For this reason, the provenance for an element must start at a particular instant in time. Thus, the provenance database may contain the records of various process instances with the same elements of different instances. In our recording model, the p-format about a process enactment may be recorded any time after completion of the process enactment. The time instants recorded in the provenance database are the start time and the end time of the process that was captured during its enactment.

Provenance reasoning query request consists of two parts: the *query data command* and the *query data filter*, that are discussed as follows (as depicted in Figure 5.2):

Definition 6.1 (Query Data Command) *The query data command searches over the contents of the provenance database in order to find the records of a specific element for which the querying user wants to retrieve the provenance at a given instant.*

Definition 6.2 (Query Data Filter) *A query data filter is the set of criteria specified by the querying user in order to include only the required information in the reasoning query results. For example, specifying if any given element in a process provenance* *should be included in the query result, constrains the reasoning query results.*

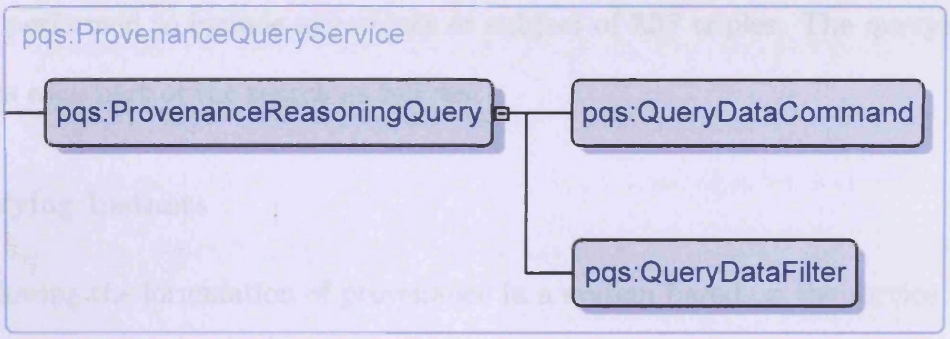


Figure 5.2: Provenance Reasoning Query

5.3.1 Query Data Command

The *query data command* allows provenance questions such as "What is the provenance of element service S at instant I?" to be posed to the provenance database. It provides the user with an identification mechanism so the element is identified in a way that the provenance database can interpret. From the perspective of the PQS, a query data command defines a search for *process-Provenance* and *service-Provenance* data items in `RDF` statements in the process documentation. A query data command is made up of:

- A search over the p-format for the instances where an element may occur.
- A search over the contents of the process-Provenance or service-Provenance to retrieve the data items which are the provenance records of the element.

All the `RDF` statements belonging to a process instance containing the process-Provenance and its related service-Provenance are parts of the p-format represented as an RDF model in the provenance database. Thus, a single search over the p-format can be performed to include any object or subject of RDF triples. The querying user specifies each part of the search as follows:

Identifying Instants

Following the formulation of provenance in a system based on the service invocation instances obtained from processing an abstract process, described in chapter 3, there are ways in which to identify a recorded instant in the past:

- The instant at which a process is invoked.
- The instant at which a service is invoked.

These are apparent in the process-Provenance and service-Provenance of the p-format. Note that there are one or more instances of a service and each is associated to a process instant. The searches in the p-format are as follows:

- On the properties of a process-Provenance.
- On the properties of a service-Provenance associated with a process-Provenance.

The identities of the services involved in the invocation of a process are apparent in the service ID, service name and WSDL URL properties of the service-Provenance. The services' association with a process is apparent through the hasServiceInstance property.

Identifying Data Items

The element for which the querying user wants to find the provenance must be present in the process-Provenance or the service-Provenance in order for the PQS to find it. The p-format defines the copies of the actual messages for a service instance in the properties *inputContents* and *outputContents*, as discussed in section 4.2.3. The element may not always be present as an exact copy of the data itself, but may instead appear as a reference or be inferred by application-specific structures in the p-format. For example, an application message may specify a filename to refer to a data item contained in it, instead of the data itself, and the querying user may wish to get the provenance of the data, rather than the file. It is therefore dependent on the preferred query language the application uses to query application-specific messages.

The query data command includes a search over the contents of p-formats in the database to retrieve the data items that comprise the documentation of the element. This search is expressed in a particular *query language.* The PQS supports query languages for the RDF *data language*, as RDF is our default data structure to document a process. However, application messages may have used a different format, so the PCS may have recorded the data language of the actual messages differently. For example, SOAP messages (in XML format) require a different query language, such as X-Path [88], if it is necessary to retrieve application-specific information in the messages. Therefore, a query data command may specify a *query language mapping* between the data language used for a p-format and the data language required for the search.

Definition 6.3 (Query Language Mapping) *Query Language Mapping defines how to transform one data language to another data language to support the search with a given query language.*

In our case, as the default data language used is RDF, any search with a query language other then RDF is handled with a combination of queries, as discussed in the next section. For example a search with an X-Path query langauge requires first the relevant p-formats in the XML language to be retrieved, and then the search is executed on this.

Combination of Provenance Queries

Primarily a p-format represents a process instance that consists of process-Provenance and its associated service-Provenance. A provenance database consists of one or more p-formats defining different process instances. A query data command is a search for an element within this provenance database. In many cases it may be appropriate to express the query data command as a *combination* of provenance queries, i.e., results from one provenance reasoning query are searched over by another provenance reasoning query. The query data command is identified as the search to be performed for a given element and the range of p-format or data over which it will search. This data is referred to as the *p-format source*, and can have one of two possible forms, as depicted in Figure 5.3.

- The contents of the provenance database that consist of process instances as RDF models. That is, each RDF model is the provenance about a process represented in a p-format.

- The results of another provenance reasoning query that can be in the p-format form of either RDF statements or XML.

Definition 6.4 (P-Format Source) *A p-format source is the expression of the data over which a search for an element is executed.*

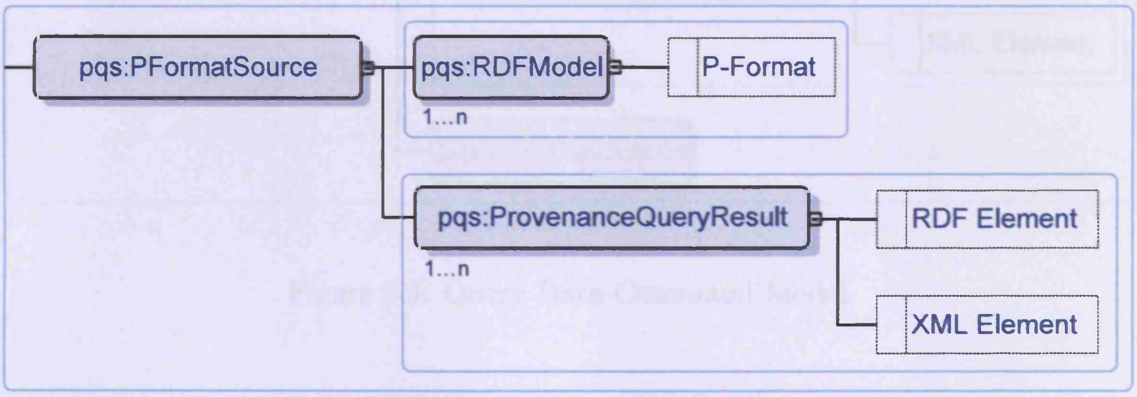


Figure 5.3: P-Format Source Model

Query Data Command Model

The model showing the query data command is presented in Figure 5.4. The *Search* element specifies the search for any query element, in a chosen query language, over the database for the data items within the p-formats that represent process executions. There can be *0* to *n* numbers of query elements specified for the given search. The *P-FormatSource* represents the set of p-formats over which the query will be executed and this may be in either of the forms discussed previously. The

QueryLanguageMapping specifies how the contents of the p-formats are mapped to the data language for the search, particularly, when used for a provenance reasoning query over the results of a previous query.

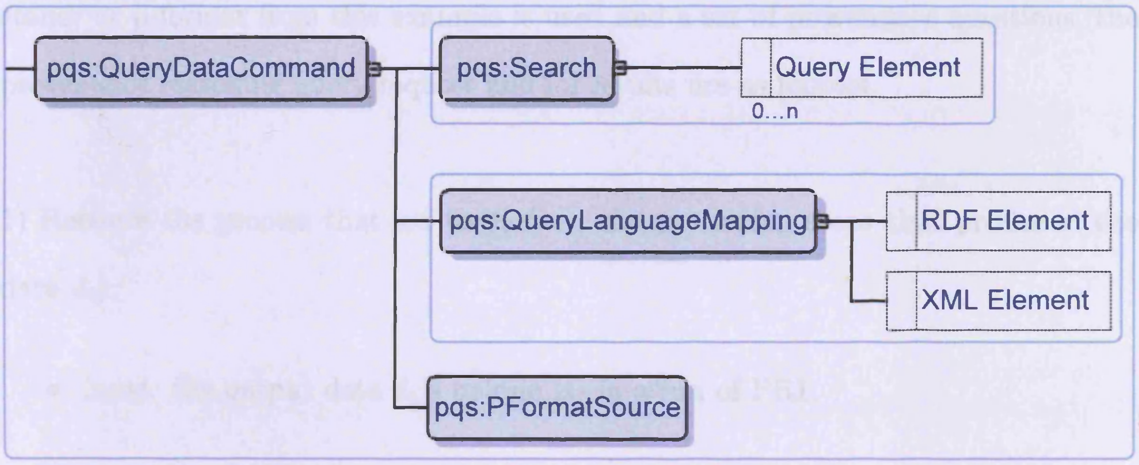


Figure 5.4: Query Data Command Model

5.3.2 Query Data Filter

The element identified by a query data command over a search in the provenance database could be vast, and much of it may be irrelevant to a querying user. Therefore, we need to allow the querying user to specify the scope of the provenance query, i.e., to define what documentation is relevant enough to be part of the results. This is the purpose of the *query data filter.*

Provenance Reasoning Queries Examples

In order to answer provenance-related queries in our proposed RDF based provenance representation, we ask a set of provenance questions and determine the queries that return the provenance reasoning query results. The example process PR1 in Figure 4.12 is mapped to Figure 5.1 depicting the process's data links, where ServiceA and ServiceB are DustCloud service and FFT service, respectively. The process instance or p-format from this example is used and a set of provenance questions, the provenance reasoning query request and its results are as follows.

1) Retrieve the process that led to d_4 (i.e., the processing steps that produced the data d_4).

- *Input:* the output data d_4's unique ID in a run of PR1.
- *Output:* a set of process runs and data that led the d_4.

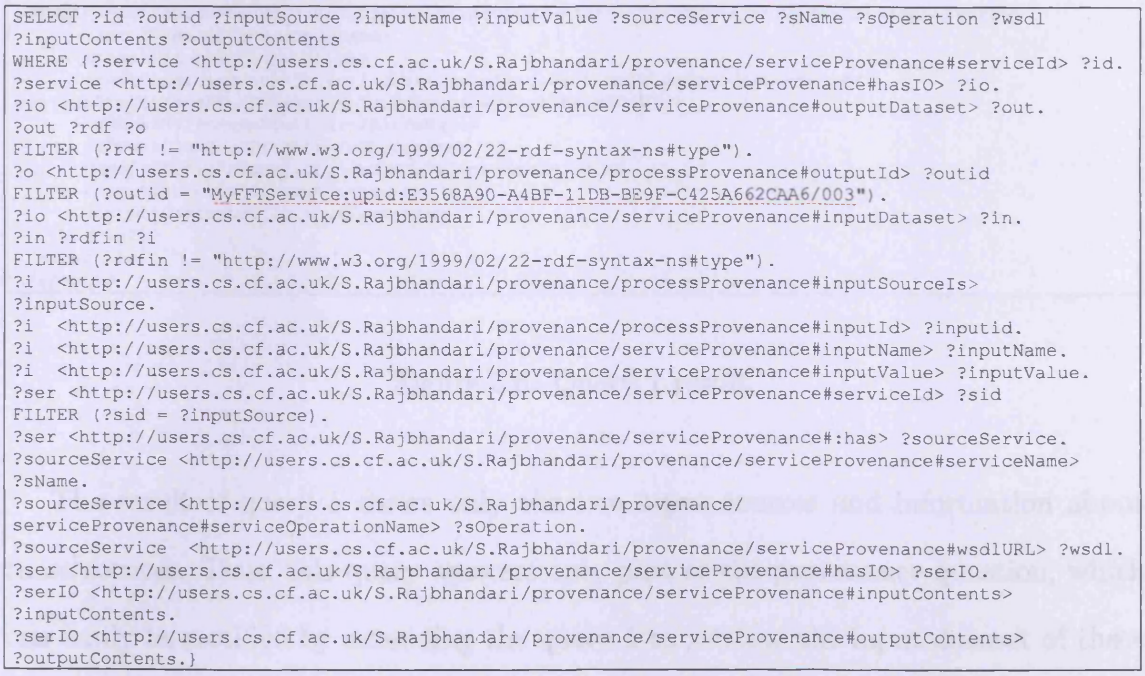
The query requires that all the invocation events and data that contribute to the creation of d_4 during the run of PR1, directly or indirectly, should also be returned, rather than only those contribute directly. Thus, this query is realized in two steps:
Step 1: First get all unique models from the provenance database. For each model containing RDF graphs of a process instance, match the given ID of d_4 with the value of property `sd:outputId`. Return the unique model name that satisfies the match.
Result: This returns the unique model name "PR1:upid:620-A841F-10671F3" created for process instance "uri:BpelProcess/PR1".
Step 2: The query is executed over the returned model by including the unique ID of output d_4. The following describes the query:

1. For a given unique ID of output d_4, the service instance that produced the output is identified. This is done by using the FILTER expression in the query.

2. With the service instance identified, its **inputDataset** is retrieved that may consist of one or more inputs. For each input, **inputName**, **inputSourceIs** and **inputValue** are retrieved.

3. Consider the earlier assumption that the **inputSourceIs** property consists of the unique ID of the service instance that is the source of this input. By comparing the unique IDs of each service instance (within the process) with the value of the **inputSourceIs**, the matching service instance for each input (i.e., the source of the input) is retrieved.

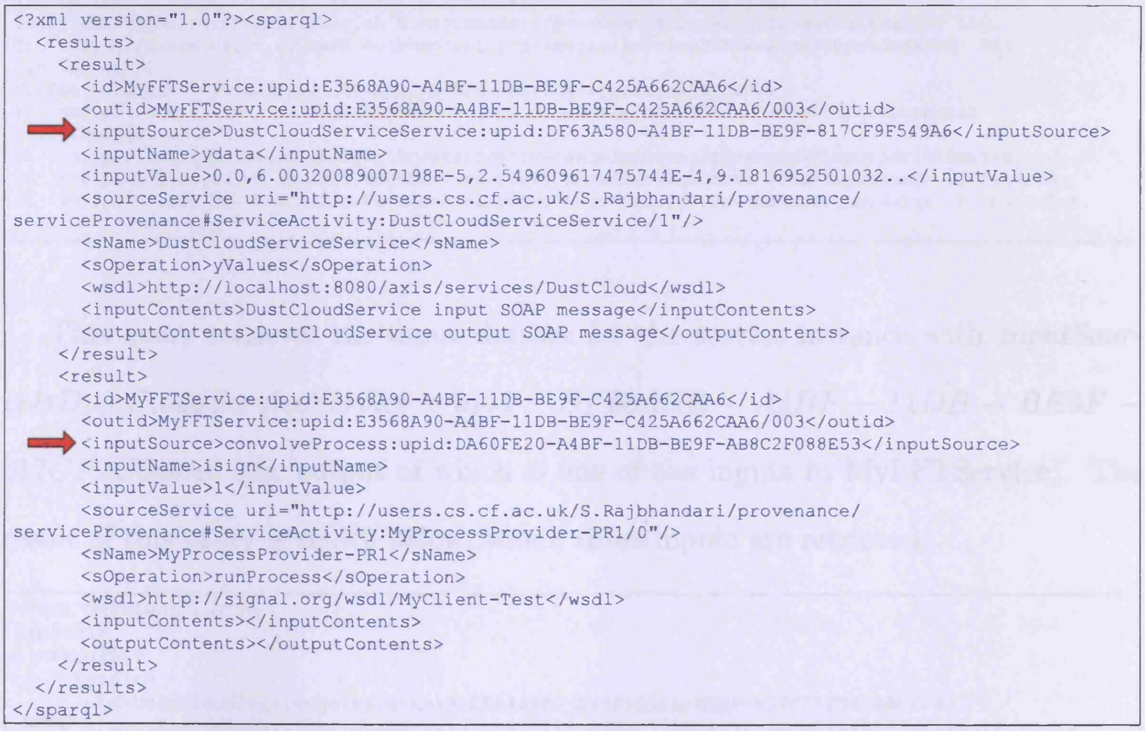
4. For each service instance retrieve the **inputContents**, **outputContents**, **serviceName**, and **serviceOperationName**. This query is shown below in Figure 5.5.



```
SELECT ?id ?outid ?inputSource ?inputName ?inputValue ?sourceService ?sName ?sOperation ?wsdl
?inputContents ?outputContents
WHERE {?service <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#serviceId> ?id.
?service <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#hasIO> ?io.
?io <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#outputDataset> ?out.
?out ?rdf ?o
FILTER (?rdf != "http://www.w3.org/1999/02/22-rdf-syntax-ns#type").
?o <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/processProvenance#outputId> ?outid
FILTER (?outid = "MyFFTService:upid:E3568A90-A4BF-11DB-BE9F-C425A662CAA6/003").
?io <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputDataset> ?in.
?in ?rdfin ?i
FILTER (?rdfin != "http://www.w3.org/1999/02/22-rdf-syntax-ns#type").
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/processProvenance#inputSourceIs>
?inputSource.
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/processProvenance#inputId> ?inputid.
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputName> ?inputName.
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputValue> ?inputValue.
?ser <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#serviceId> ?sid
FILTER (?sid = ?inputSource).
?ser <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#:has> ?sourceService.
?sourceService <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#serviceName>
?sName.
?sourceService <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/
serviceProvenance#serviceOperationName> ?sOperation.
?sourceService  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#wsdlURL> ?wsdl.
?ser <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#hasIO> ?serIO.
?serIO <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputContents>
?inputContents.
?serIO <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#outputContents>
?outputContents.}
```

Figure 5.5: Query 1

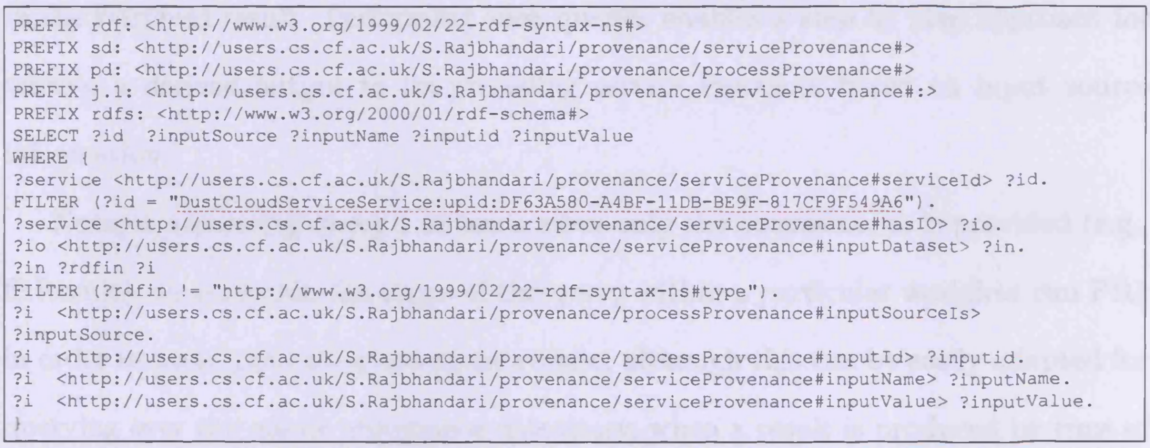
Result: As shown below, this query returns two service instances (sources of inputs for the output with ID d_4, of MyFFTService) and the input data MyFFTService consumed to produce d_4. This also includes all the information specified in the query. Note that, in the query result in Figure 5.6, the inputValue is large, so only part of the data is kept.



```
<?xml version="1.0"?><sparql>
  <results>
    <result>
      <id>MyFFTService:upid:E3568A90-A4BF-11DB-BE9F-C425A662CAA6</id>
      <outid>MyFFTService:upid:E3568A90-A4BF-11DB-BE9F-C425A662CAA6/003</outid>
      <inputSource>DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6</inputSource>
      <inputName>ydata</inputName>
      <inputValue>0.0,6.00320089007198E-5,2.549609617475744E-4,9.1816952501032..</inputValue>
      <sourceService uri="http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/
serviceProvenance#ServiceActivity:DustCloudServiceService/1"/>
      <sName>DustCloudServiceService</sName>
      <sOperation>yValues</sOperation>
      <wsdl>http://localhost:8080/axis/services/DustCloud</wsdl>
      <inputContents>DustCloudService input SOAP message</inputContents>
      <outputContents>DustCloudService output SOAP message</outputContents>
    </result>
    <result>
      <id>MyFFTService:upid:E3568A90-A4BF-11DB-BE9F-C425A662CAA6</id>
      <outid>MyFFTService:upid:E3568A90-A4BF-11DB-BE9F-C425A662CAA6/003</outid>
      <inputSource>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</inputSource>
      <inputName>isign</inputName>
      <inputValue>1</inputValue>
      <sourceService uri="http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/
serviceProvenance#ServiceActivity:MyProcessProvider-PR1/0"/>
      <sName>MyProcessProvider-PR1</sName>
      <sOperation>runProcess</sOperation>
      <wsdl>http://signal.org/wsdl/MyClient-Test</wsdl>
      <inputContents></inputContents>
      <outputContents></outputContents>
    </result>
  </results>
</sparql>
```

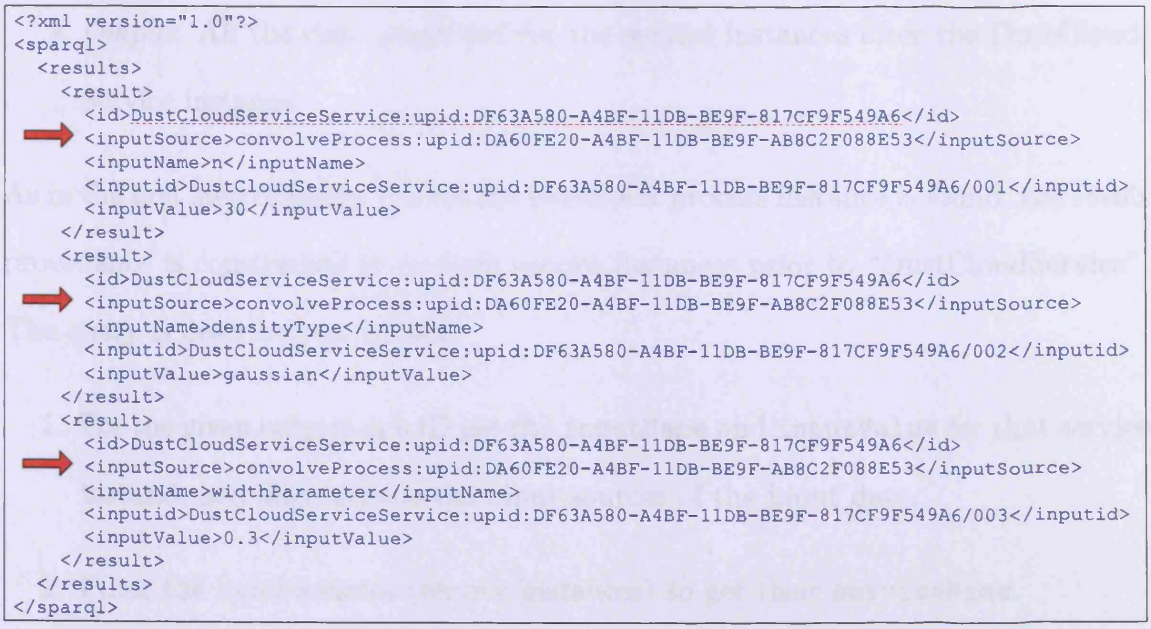
Figure 5.6: Query 1 result

The result of query 1 shows only the two input sources and information about these sources. Thus, this query answers only part of the provenance question, which can easily be rectified by extending the query 1 to retrieve the input dataset of these sources, and the sources of these retrieve inputs, and so forth. Such a query would return a large dataset, and the way results are presented is repetitive and hard to understand. Thus, a similar second query, that uses one of the input sources data from the query 1 result, is executed as follows:



```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sd: <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#>
PREFIX pd: <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/processProvenance#>
PREFIX j.1: <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#:>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?id  ?inputSource ?inputName ?inputid ?inputValue
WHERE {
?service <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#serviceId> ?id.
FILTER (?id = "DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6").
?service <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#hasIO> ?io.
?io <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputDataset> ?in.
?in ?rdfin ?i
FILTER (?rdfin != "http://www.w3.org/1999/02/22-rdf-syntax-ns#type").
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/processProvenance#inputSourceIs>
?inputSource.
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/processProvenance#inputId> ?inputid.
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputName> ?inputName.
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputValue> ?inputValue.
}
```

This query retrieves the input dataset for the service instance with $inputSourceIsDustCloudServiceService : upid : DF63A580 - A4BF - 11DB - BE9F - 817CF9F549A6$ (the output of which is one of the inputs to MyFFTService). The result of this query is shown below, where three inputs are retrieved.



```
<?xml version="1.0"?>
<sparql>
  <results>
    <result>
      <id>DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6</id>
      <inputSource>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</inputSource>
      <inputName>n</inputName>
      <inputid>DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6/001</inputid>
      <inputValue>30</inputValue>
    </result>
    <result>
      <id>DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6</id>
      <inputSource>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</inputSource>
      <inputName>densityType</inputName>
      <inputid>DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6/002</inputid>
      <inputValue>gaussian</inputValue>
    </result>
    <result>
      <id>DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6</id>
      <inputSource>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</inputSource>
      <inputName>widthParameter</inputName>
      <inputid>DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6/003</inputid>
      <inputValue>0.3</inputValue>
    </result>
  </results>
</sparql>
```

The input's source information in this result can be used to perform another similar query. Such a succession of queries ends when there is no input source information in the retrieved result. Performing such queries enables a step by step approach for tracing a derived output to its preceding service instances based on input source information.

Note, in answering query 1 in cases when only the output name is provided (e.g., fftResult), we constrain the scope of the query within a particular workflow run PR1 in order to avoid presenting too many results, although this can be easily adapted for querying over the whole provenance repository, when a result is produced by runs of different processes.

2) Retrieve the process that led to d_4, excluding everything prior to DustCloudService.

- *Input:* The unique ID of output data d_4 and the scope of the query to exclude all the provenance that is prior to DustCloudService.
- *Output:* All the data generated for the service instances after the DustCloudService instance.

As in the first step of query 1, after the particular process instance is found, the result provenance is constrained to exclude service instances prior to "DustCloudService". The query is described as follows:

1. For the given output d_4's ID get the `inputName` and `inputValue` for that service instance and also retrieve the input sources of the input data.
2. Filter the input sources (service instances) to get their `serviceName`.

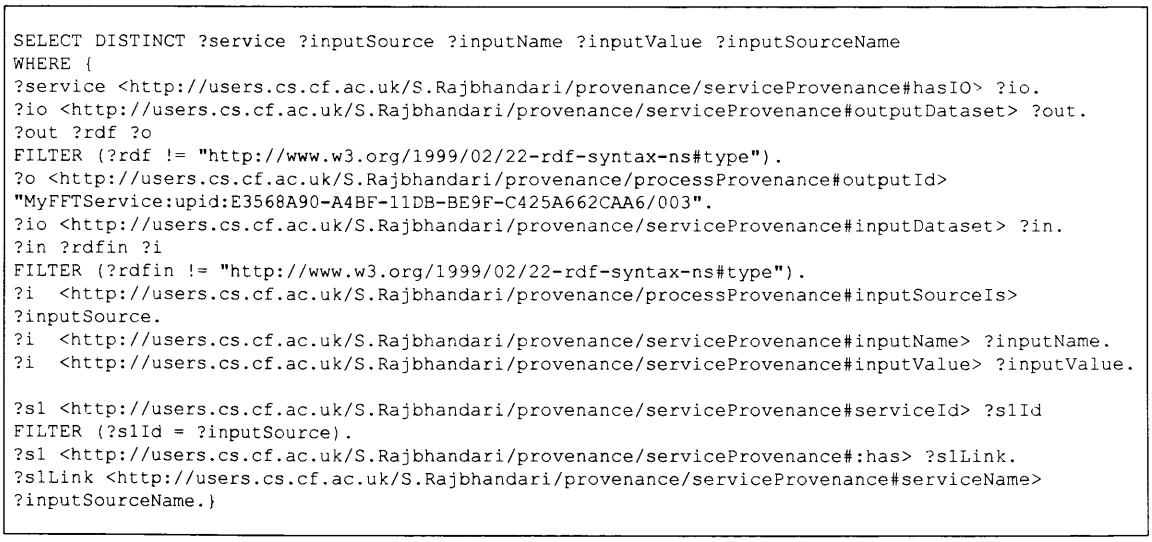
3. For the given service instances retrieve the relevant data and the input sources. This includes the service instance that has the name "DustCloudServiceService".

4. If the retrieved input sources of a service instance are "false" (i.e., the property does not exist), or if the service instance has the name "DustCloudService", then go to 5 else go to 6.

5. Query ends.

6. Perform a similar query as described in step 3 with the retrieve input sources of the service instance.

The query 2 below shows steps 1 and 2 in the above query description.

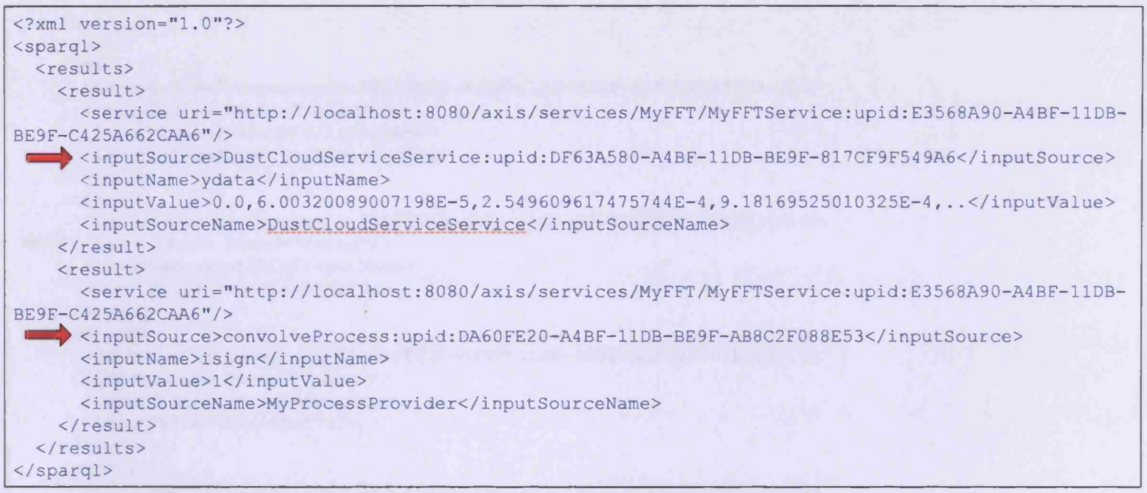


```
SELECT DISTINCT ?service ?inputSource ?inputName ?inputValue ?inputSourceName
WHERE {
?service <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#hasIO> ?io.
?io <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#outputDataset> ?out.
?out ?rdf ?o
FILTER (?rdf != "http://www.w3.org/1999/02/22-rdf-syntax-ns#type").
?o <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/processProvenance#outputId>
"MyFFTService:upid:E3568A90-A4BF-11DB-BE9F-C425A662CAA6/003".
?io <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputDataset> ?in.
?in ?rdfin ?i
FILTER (?rdfin != "http://www.w3.org/1999/02/22-rdf-syntax-ns#type").
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/processProvenance#inputSourceIs>
?inputSource.
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputName> ?inputName.
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputValue> ?inputValue.

?s1 <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#serviceId> ?s1Id
FILTER (?s1Id = ?inputSource).
?s1 <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#:has> ?s1Link.
?s1Link <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#serviceName>
?inputSourceName.}
```

Figure 5.7: Query 2

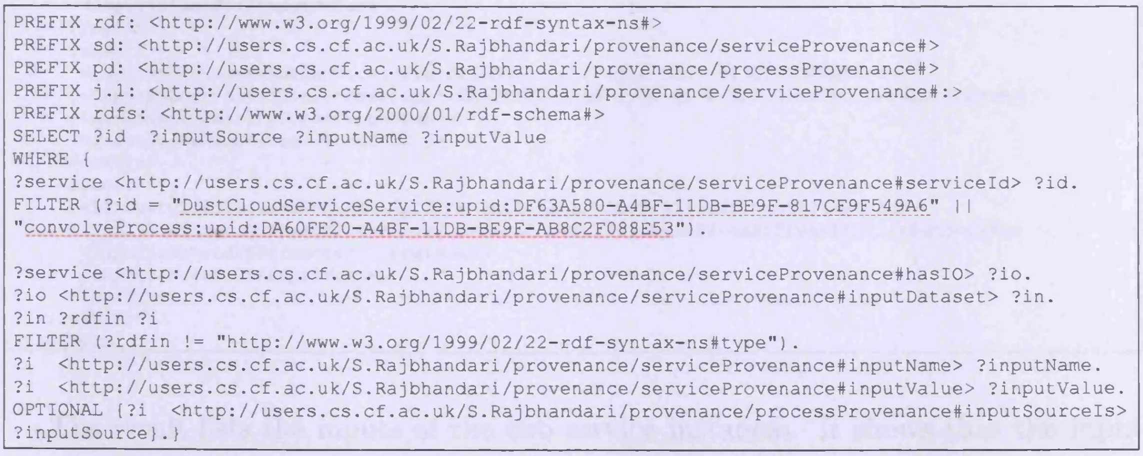
Result: This query results in one service instance "MyFFTService" and the relevant input data with its corresponding sources. The input sources' `serviceName` values



```
<?xml version="1.0"?>
<sparql>
  <results>
    <result>
      <service uri="http://localhost:8080/axis/services/MyFFT/MyFFTService:upid:E3568A90-A4BF-11DB-
BE9F-C425A662CAA6"/>
      <inputSource>DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6</inputSource>
      <inputName>ydata</inputName>
      <inputValue>0.0,6.00320089007198E-5,2.549609617475744E-4,9.18169525010325E-4,..</inputValue>
      <inputSourceName>DustCloudServiceService</inputSourceName>
    </result>
    <result>
      <service uri="http://localhost:8080/axis/services/MyFFT/MyFFTService:upid:E3568A90-A4BF-11DB-
BE9F-C425A662CAA6"/>
      <inputSource>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</inputSource>
      <inputName>isign</inputName>
      <inputValue>1</inputValue>
      <inputSourceName>MyProcessProvider</inputSourceName>
    </result>
  </results>
</sparql>
```

Figure 5.8: Query 2 result

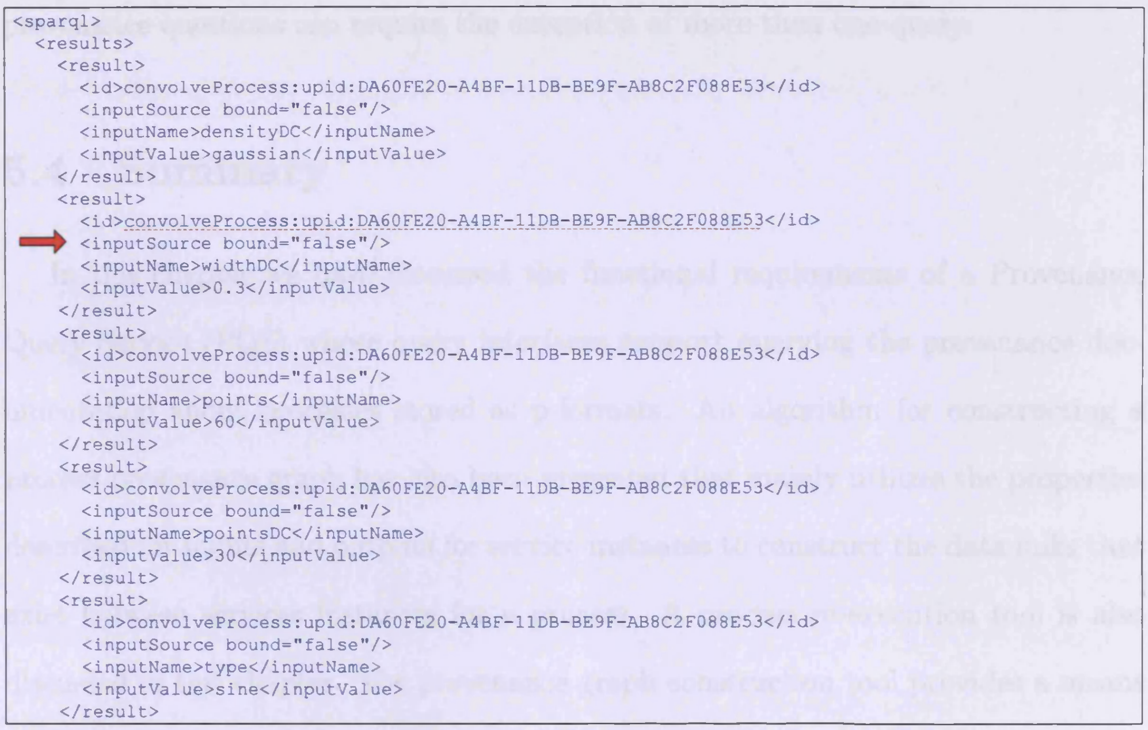
are also retrieved from query 2 (Figure 5.7). The results are shown in Figure 5.8. Using the two retrieved input sources information (see Figure 5.8) from the query 2 result, a second query is performed as follows.



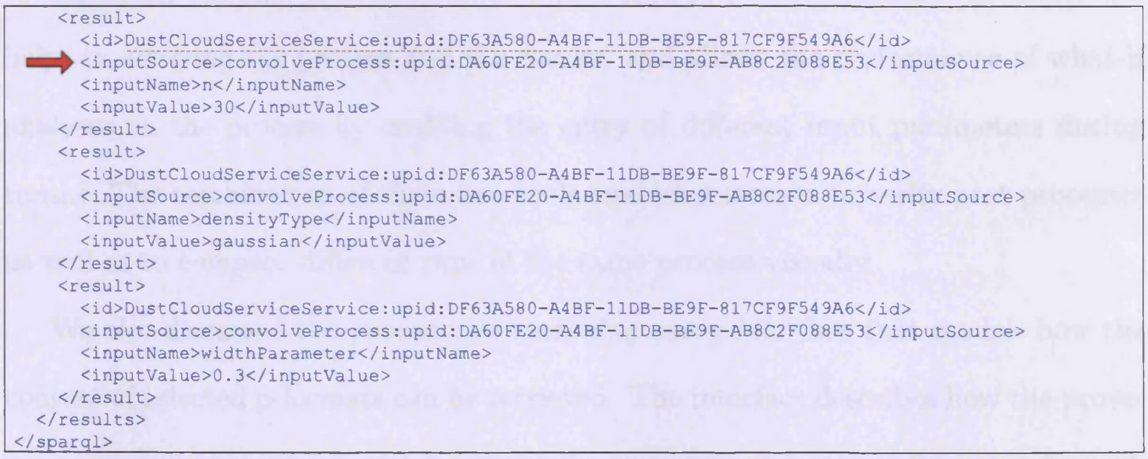
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sd: <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#>
PREFIX pd: <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/processProvenance#>
PREFIX j.1: <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#:>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?id  ?inputSource ?inputName ?inputValue
WHERE {
?service <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#serviceId> ?id.
FILTER (?id = "DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6" ||
"convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53").

?service <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#hasIO> ?io.
?io <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputDataset> ?in.
?in ?rdfin ?i
FILTER (?rdfin != "http://www.w3.org/1999/02/22-rdf-syntax-ns#type").
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputName> ?inputName.
?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/serviceProvenance#inputValue> ?inputValue.
OPTIONAL {?i  <http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/processProvenance#inputSourceIs>
?inputSource}.}
```

This query satisfies step 3 of the query description. This query gives an option to retrieve **inputSourceIs**, so the absence of input source information for any service instance will be known. The result of this query is as follows.



```
<sparql>
  <results>
    <result>
      <id>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</id>
      <inputSource bound="false"/>
      <inputName>densityDC</inputName>
      <inputValue>gaussian</inputValue>
    </result>
    <result>
      <id>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</id>
      <inputSource bound="false"/>
      <inputName>widthDC</inputName>
      <inputValue>0.3</inputValue>
    </result>
    <result>
      <id>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</id>
      <inputSource bound="false"/>
      <inputName>points</inputName>
      <inputValue>60</inputValue>
    </result>
    <result>
      <id>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</id>
      <inputSource bound="false"/>
      <inputName>pointsDC</inputName>
      <inputValue>30</inputValue>
    </result>
    <result>
      <id>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</id>
      <inputSource bound="false"/>
      <inputName>type</inputName>
      <inputValue>sine</inputValue>
    </result>
```



```
    <result>
      <id>DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6</id>
      <inputSource>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</inputSource>
      <inputName>n</inputName>
      <inputValue>30</inputValue>
    </result>
    <result>
      <id>DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6</id>
      <inputSource>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</inputSource>
      <inputName>densityType</inputName>
      <inputValue>gaussian</inputValue>
    </result>
    <result>
      <id>DustCloudServiceService:upid:DF63A580-A4BF-11DB-BE9F-817CF9F549A6</id>
      <inputSource>convolveProcess:upid:DA60FE20-A4BF-11DB-BE9F-AB8C2F088E53</inputSource>
      <inputName>widthParameter</inputName>
      <inputValue>0.3</inputValue>
    </result>
  </results>
</sparql>
```

The result lists the inputs of the two service instances. It shows that the input source information for the service instance "convolveProcess" is not present, and the name of the other one is "DustCloudService". Any service instance before DustCloudService is not needed. Thus, this query satisfies step 4 of the query description to answer this provenance question. The example queries illustrate that, answering provenance questions can require the execution of more then one query.

5.4 Summary

In this chapter we have discussed the functional requirements of a Provenance Query Service (PQS) whose query interfaces support querying the provenance documentation about processes stored as p-formats. An algorithm for constructing a process provenance graph has also been presented that mainly utilizes the properties described for inputs and outputs for service instances to construct the data links that exist between services instances for a process. A process re-execution tool is also discussed in this chapter. The provenance graph construction tool provides a means of displaying the high level behaviour of an executed process. The re-execution tool helps in verifying the process and the results, and allows the performance of what-if analyses on the process by enabling the entry of different input parameters during reruns. The combination of these two tools enables scientists to verify past processes as well as to compare different runs of the same process visually.

We also discussed the *provenance reasoning query interface* that models how the content of selected p-formats can be retrieved. The interface describes how the provenance reasoning query request to *search* for any element in the p-format is formed. The provenance reasoning query interface provides support for RDF and XML query languages namely, SPARQL [43] for RDF and, X-Query[102] and X-Path [88] for XML. A query langauge mapping and the combination of provenance queries are introduced to enable searches for an element that requires (1) execution of more than one query, and (2) use of both RDF and XML query languages when necessary. We also presented some example queries using the SPARQL query language to demonstrate how a particular data item whose provenance is described as a p-format can be successfully traced to its sources, or how it was derived. That is, the service instance and the input data consumed by this service instance to produce this data item, and the preceding service instances whose outputs were used as the inputs to this service instance.

In summary, this chapter provides a comprehensive model for querying provenance and a strategy for (1) tracing the provenance of data items by utilizing the stored provenance information (i.e., p-format) about processes executed in an SOA environment, and (2) exploiting stored provenance information about past processes to re-create process behaviour, and to re-execute them to verify results and analyze the process.

In order to demonstrate the application of our p-format in the context of the PCS and PQS components, we have implemented a prototype provenance system in an SOA environment. We have implemented the automated collection and recording of provenance about processes that uses the p-format and interfaces to support querying of the recorded provenance. The architecture and implementation of this prototype are presented in chapter 6. We have thus far presented the theoretical models to provide support for provenance in an SOA environment. Chapter 6 and chapter 7 present the implementations and experimental evaluation of the developed model.

Chapter 6

Provenance Prototype: Implementation

6.1 Introduction

The preceding chapters in this thesis have presented the theoretical aspects of our research, namely, the Provenance Model, the collection of data provenance and the format for recording provenance, and the querying capabilities needed to enable provenance support in a service-oriented environment. This chapter presents the implementation of the Prototype Provenance System in a service-based environment. The prototype encompasses the concepts developed in this research and validates the feasibility of the implementation of the models proposed in this thesis. It demonstrates the ability of the Provenance Collection Service and the provenance format to support automatic recording of data provenance in a standard format for a composed Web process execution. It also provides interfaces for querying, re-execution and provenance graph construction. The implementation provides the basis for experimental validation and analysis of the provenance recording techniques and the Provenance Model that will be discussed in Chapter 8.

This chapter is organized as follows. In Section 6.2, the architecture and operation of our Prototype Provenance System is presented. This section discusses the implementation of the components that support the provenance collection and the p-format which was described earlier in Chapters 4 and 5. These components primarily include the Provenance Collector, the Provenance Recorder, and the Workflow Engine. Section 6.3 presents the implementation of the query interface, and the tools for provenance graph construction and workflow re-execution (described in Chapters 4 and 5).

6.2 Architecture of the Provenance Collection Service

This section presents the architecture (shown in Figure 6.1) and operations of our prototype implementation of the Provenance Service for the collection and recording of the provenance of a Web process execution in service-oriented environment. It should be noted that a service-oriented environment typically has multiple service providers hosting different provenance services. Our implementation is a prototype system that provides a Provenance Service as a proof-of-concept.

The prototype consists of the following components:

- *MySQL Database.* This component stores the provenance documentation that

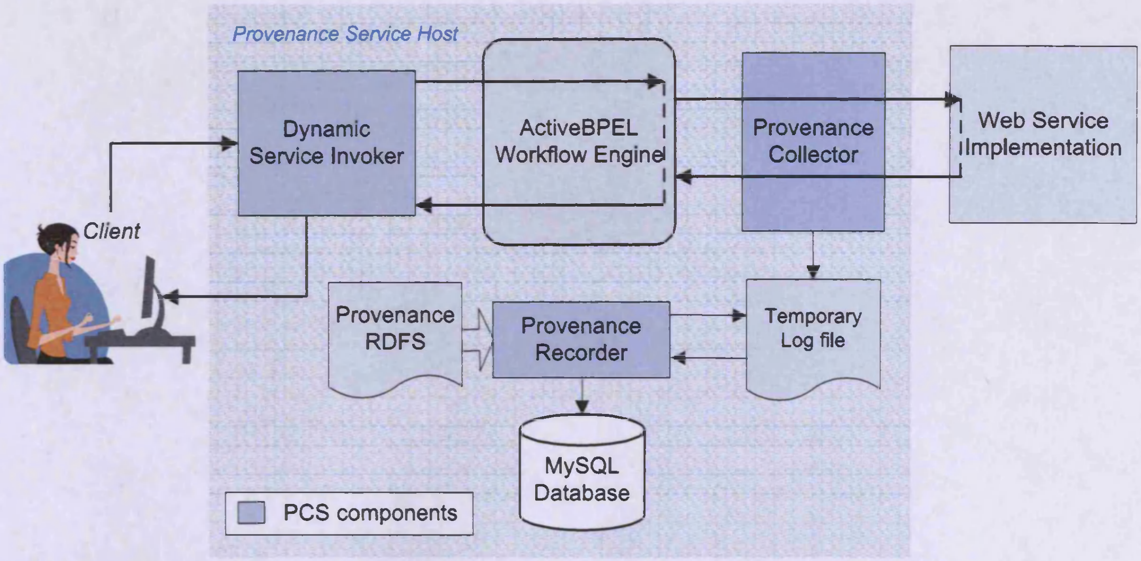
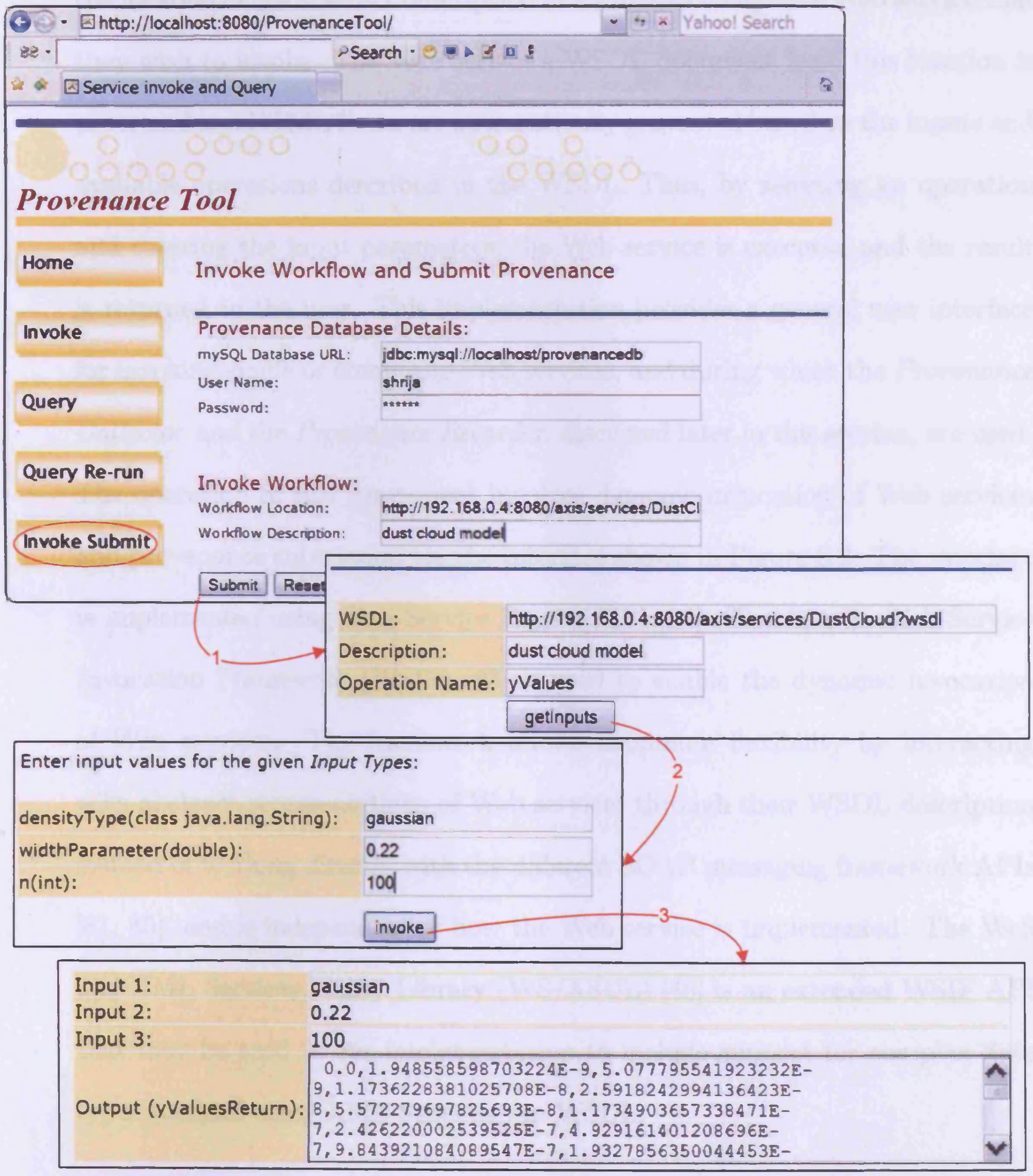


Figure 6.1: Architecture of Prototype Implementation of the PCS

is captured automatically at runtime from process executions. In our implementation a mySQL relational database system server [5] is used for storing the RDF triples represented by the p-format, i.e., the RDF schema discussed in Chapter 4 and specified in Appendix A.

- *ActiveBPEL Workflow Engine.* The ActiveBPEL engine [9] is an open-source implementation of a Business Process Execution Language (BPEL) [16] engine, written in Java. It reads BPEL process definitions (and other inputs such as WSDL [100] files) and creates representations of BPEL processes. BPEL is an XML language for describing business process behaviour based on Web services. The ActiveBPEL engine is used to deploy and invoke the workflows created using the BPEL standard. The ActiveBPEL engine supports the invocation of deployed BPEL processes as Web services.



Invoke Workflow and Submit Provenance

Provenance Database Details:

mySQL Database URL:	jdbc:mysql://localhost/provenancedb
User Name:	shrija
Password:	******

Invoke Workflow:

Workflow Location:	http://192.168.0.4:8080/axis/services/DustCl
Workflow Description:	dust cloud model

WSDL:	http://192.168.0.4:8080/axis/services/DustCloud?wsdl
Description:	dust cloud model
Operation Name:	yValues

Enter input values for the given *Input Types*:

densityType(class java.lang.String):	gaussian
widthParameter(double):	0.22
n(int):	100

Input 1:	gaussian
Input 2:	0.22
Input 3:	100

Output (yValuesReturn):

```
 0.0,1.948558598703224E-9,5.077795541923232E-
9,1.1736228381025708E-8,2.5918242994136423E-
8,5.572279697825693E-8,1.1734903657338471E-
7,2.426220002539525E-7,4.929161401208696E-
7,9.843921084089547E-7,1.9327856350044453E-
```

Figure 6.2: Interface for Process Invocation and Provenance Submission in the PCS

- *Dynamic Service Invoker.* A web interface is provided that allows users to enter the location of the abstract description of a single or composite Web service that they wish to invoke. The Web service's WSDL document from this location is processed and HTML forms are automatically generated based on the inputs and available operations described in the WSDL. Thus, by selecting an operation and entering the input parameters, the Web service is executed and the result is returned to the user. This implementation provides a general user interface for invoking single or composite Web services, and during which the *Provenance Collector* and the *Provenance Recorder*, discussed later in this section, are used. The operation of this component involves dynamic invocation of Web services and provenance submission via the interface shown in Figure 6.2. The interface is implemented using Java Service Pages (JSP) [94]. The Apache Web Service Invocation Framework (WSIF) [83] is used to enable the dynamic invocation of Web services. The framework allows maximum flexibility by interacting with abstract representations of Web services through their WSDL description instead of working directly with the different SOAP messaging framework APIs [81, 85], and is independent of how the Web service is implemented. The Web and XML Services Utility Library (WS/XSUL) [46] is an extended WSIF API that may be used in our implementation to include support for complex data types (defined using XML Schemas) in WSDL.

- *Provenance Collector.* The Provenance Collector applies the interactions discussed in Chapter 4 to record the invocations of the services involved in the process in a temporary log file. The log file is structured in a standard way by using XML tags (see Appendix B for service-Provenance XML schema) defined as properties in the service-Provenance RDF schema. This is implemented using client-side Axis handler [82] APIs to log SOAP messages associated with service invocations.

The Provenance Collector (PC) is deployed with the ActiveBPEL workflow engine in the Apache Tomcat Server (V5.5.12) [96] in order to intercept and log the intermediate messages of the services involved in an enacted process. The Provenance Collector is activated from within the engine so the engine acts as the client sending messages for invoking services. Thus, the PC processes each service's outgoing message first and then the incoming message. A logical clock is also implemented that increases by integer value 1 each time the Provenance Collector is called as a result of a service invocation. This determines, for a given process enactment (occurring in the engine), how many services were invoked and in what order. It saves the logs as an XML file on the server side and the URL location of this file may be sent in the SOAP header to the client that enacted the process. The Provenance Collector for the engine is an added optional functionality that captures intermediate data for a process. The Provenance Collector is also used within the Dynamic Service Invoker, together with the Provenance Recorder, to record the initial inputs and the final output for a process enactment.

- *Provenance Recorder.* The Provenance Recorder uses the provenance RDFS, or p-format, presented in Chapter 4 to generate process documentation. For a process execution instance, the process documentation or data provenance generated is uniquely identified in the database. This component primarily provides two methods:

1. To generate and record RDF triples describing the process-Provenance that includes the inputs and the final output (using service-Provenance properties) for a process returning the unique process ID.
2. To generate and record the service-Provenance for the service instances and the data-links in the process-Provenance. This is done by processing the temporary log (XML file) and the BPEL document of the enacted process.

The first method is mandatory, whereas the second is used only when the Provenance Collector at the workflow engine end is activated, i.e., it returns the XML file location in the SOAP header of the process's response SOAP message. By processing the XML file containing service instances for a process and using Jena APIs [36], RDF triples of service-Provenance instances are created and stored with process-Provenance as a Jena model in the database. A Jena model is identified by a model name in the database to which the RDF triples are allocated. A unique process ID is created and stored as the model name to identify a process instance and, hence, all the RDF triples describing the provenance documentation for that process.

To summarize, the implementation involves the integration of the following technologies and languages: Java [92], Jena – a Semantic Web framework for Java, Resource Description Framework (RDF) [87], Apache Axis [82] to implement Java Web services, the Business Process Execution Language (BPEL4WS) standard, and the ActiveBPEL workflow engine [9]. The primary language used was Java. RDF was used to represent the provenance from an enacted process, based on an RDF schema (p-format). The Provenance Recorder used the Jena packages, which provide an API and tool for automating the mapping between the RDF triples of the captured provenance and mySQL database objects. The API handles all the details of RDF parsing and formatting, and the structure of the RDF storage within the database.

The algorithms required for the Astrophysics example workflow presented in Chapter 4 were implemented according to the details in [80], and deployed as Axis Web services. An abstract BPEL process description was created using the ActiveBPEL designer [10] (BPEL development environment) depicted in Appendix C. The BPEL process description specifies what the process can do and the inputs and outputs of each of the parties (i.e., Web services) involved, but does not describe how anything gets done, i.e., it does not reveal their internal behaviour. This BPEL document is deployed in the ActiveBPEL workflow engine and executed using dynamic service invocation to experimentally validate the provenance model presented in Chapter 4.

This concludes the discussion of the operation of the different components of the Provenance Collection Service that has been developed to perform the actual collection and recording of provenance for process enactments in a service-oriented environment. The implementation and operation of the Provenance Query Service will now be presented in Section 6.3.

6.3 Interface Implementation of the Provenance Query Service

This section discusses the implementation and operation of the Provenance Query Service (PQS). This is a part of the Provenance Service that has been developed to enable the querying and exploitation of data provenance. Furthermore, the PQS supports the re-execution and re-creation of past workflows to facilitate the verification of their results, and allows "what-if" analyses to be carried out on the process instances.

The languages and tools used in the implementation of the PQS include Java, JSP and Jena. The SPARQL Query Language for RDF [43] is a query language for extracting information from RDF graphs or triples. SPARQL is used for running both standalone queries and to programmatically call Jena's SPARQL capabilities directly. SPARQL queries were created and executed with Jena via the jena.query package by passing in the *Query String* to execute and the Jena *Model* to run it against. Because the data for the query is provided programmatically, the query does not need a FROM clause. The PQS has been developed with web interfaces to perform different queries, re-execution and visual re-creation of processes stored as RDF triples in the Provenance Database. The PQS implementation architecture is depicted in Figure 6.3

The components of the PQS are as follows:

- *Query Processor.* The Query Processor contains different set query tasks that enable simple data provenance queries. The Query Processor uses the methods that implement the Query Result Format to provide query results in four dif-

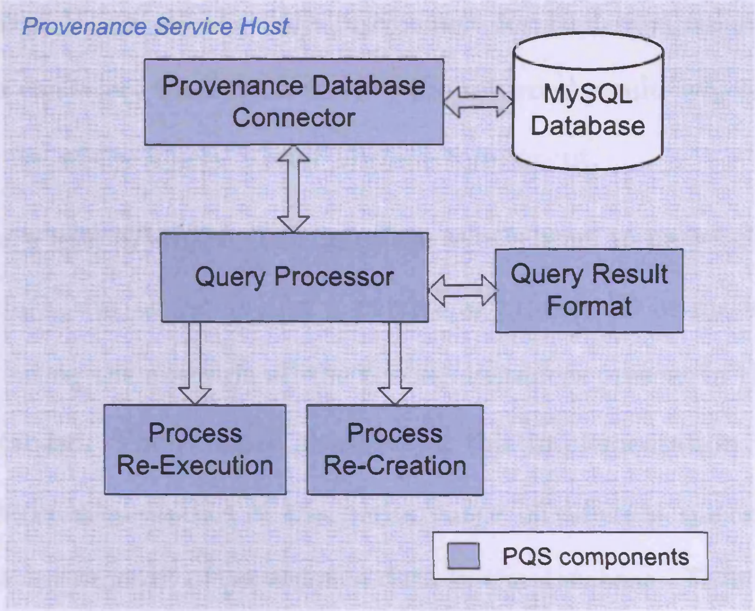


Figure 6.3: Architecture of the Prototype Implementation of the PQS

ferent formats: plain text, RDF triple, RDF/XML, and tree view. The Query Processor can perform the following query tasks, which are presented as links in the web interface:

- *Query all the process IDs.* This task is responsible for initiating the database connection and querying the process IDs that represent process instances stored as Jena models in the database. A mechanism is provided to query only ten IDs at a time and a link is given to return the next ten IDs, and so forth. Each ID has a hyperlink which when clicked returns the XML/RDF view of that particular process instance. The web interface is depicted in Figure 6.4.
- *Query with a given process ID.* This interface allows the user to enter the

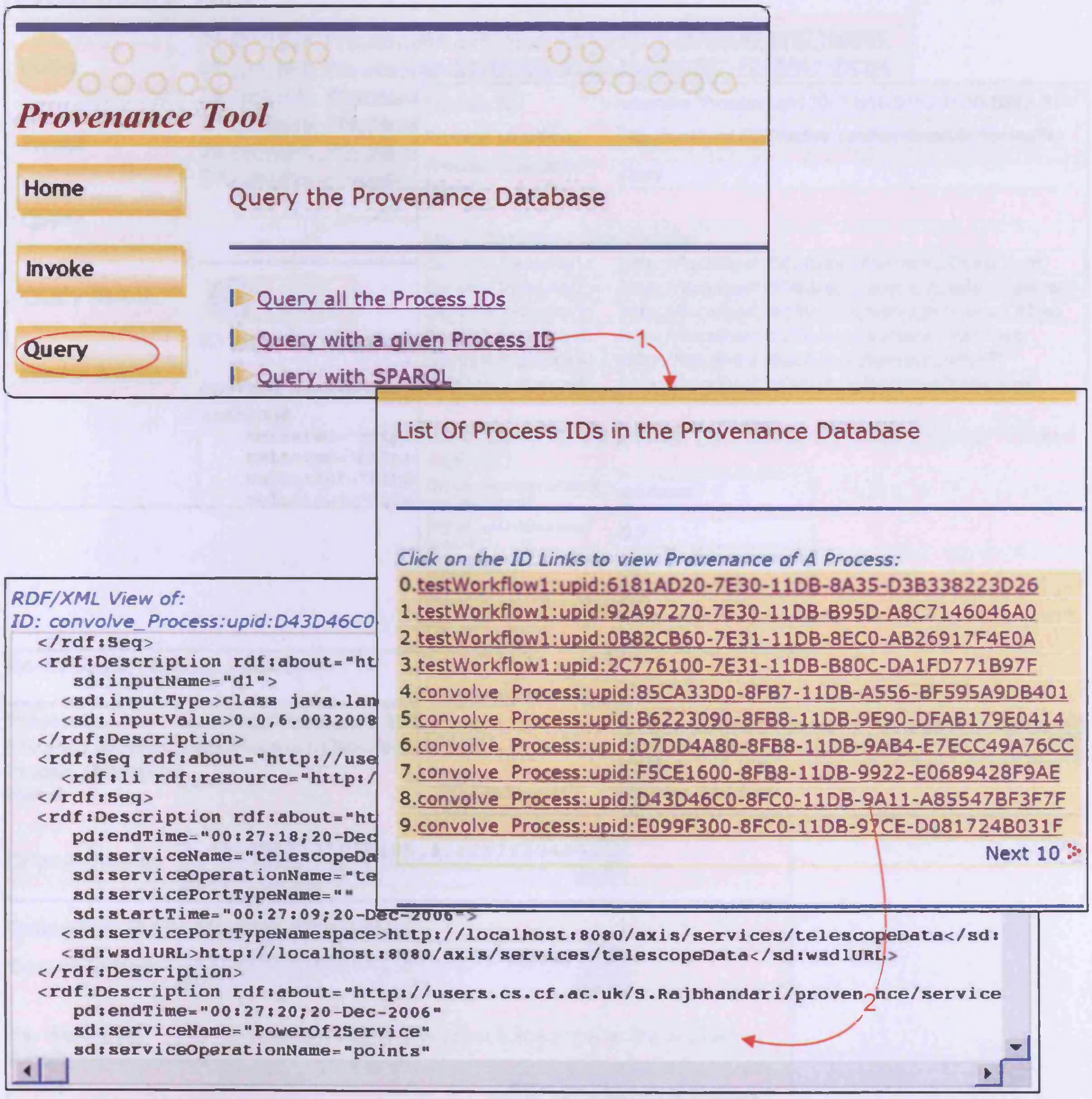
process ID to get the data provenance for that particular process. The user can select the format to view the returned result, which demonstrates the use of the Query Result Format component.

  - *Query with SPARQL.* This interface allows users to pass a SPARQL query string in two ways: (1) for a particular process ID or model, and (2) for executing the query on all the process instances present in the Provenance Database. The primary objective of this implementation is to facilitate experimental studies of the performance of different queries with an increasing amount of provenance data in the database. These experimental results are presented and analyzed in Chapter 8.

- *Provenance Database Connector.* This component uses the request sent through the web interface to build a connection with the mySQL database that contains the data provenance as RDF triples. The connection object is given to the Query Processor.

- *Process Re-execution.* The Process Re-execution component uses the query task (that retrieves process instances) determined by the Query Processor to re-execute a particular process. Web interfaces have been developed to display the process location, the original inputs and generated output for the process. These data are placed in an automatically generated form that is used for the re-execution task. The form consists of two buttons:

  1. The *re-execute* button is for enabling re-execution of the process with the same input parameters and performs a check to determine if the original



Query the Provenance Database

- Query all the Process IDs
- Query with a given Process ID
- Query with SPARQL

List Of Process IDs in the Provenance Database

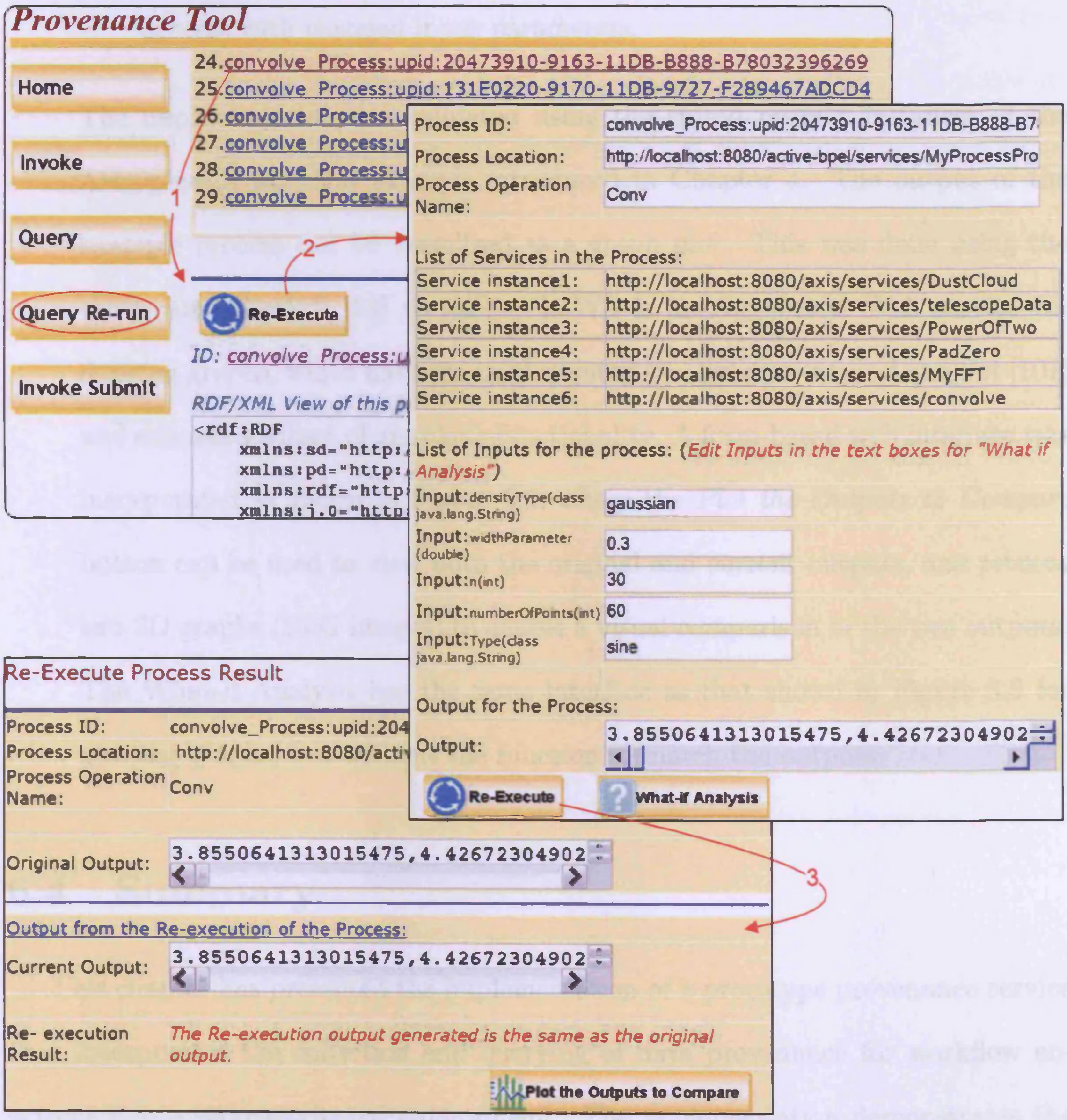
Click on the ID Links to view Provenance of A Process:

0.testWorkflow1:upid:6181AD20-7E30-11DB-8A35-D3B338223D26
1.testWorkflow1:upid:92A97270-7E30-11DB-B95D-A8C7146046A0
2.testWorkflow1:upid:0B82CB60-7E31-11DB-8EC0-AB26917F4E0A
3.testWorkflow1:upid:2C776100-7E31-11DB-B80C-DA1FD771B97F
4.convolve_Process:upid:85CA33D0-8FB7-11DB-A556-BF595A9DB401
5.convolve_Process:upid:B6223090-8FB8-11DB-9E90-DFAB179E0414
6.convolve_Process:upid:D7DD4A80-8FB8-11DB-9AB4-E7ECC49A76CC
7.convolve_Process:upid:F5CE1600-8FB8-11DB-9922-E0689428F9AE
8.convolve_Process:upid:D43D46C0-8FC0-11DB-9A11-A85547BF3F7F
9.convolve_Process:upid:E099F300-8FC0-11DB-97CE-D081724B031F

RDF/XML View of:
ID: convolve_Process:upid:D43D46C0

```
  </rdf:Seq>
  <rdf:Description rdf:about="ht
     sd:inputName="d1">
    <sd:inputType>class java.lan
    <sd:inputValue>0.0,6.0032008
  </rdf:Description>
  <rdf:Seq rdf:about="http://use
    <rdf:li rdf:resource="http:/
  </rdf:Seq>
  <rdf:Description rdf:about="ht
     pd:endTime="00:27:18;20-Dec
     sd:serviceName="telescopeDa
     sd:serviceOperationName="te
     sd:servicePortTypeName=""
     sd:startTime="00:27:09;20-Dec-2006">
    <sd:servicePortTypeNamespace>http://localhost:8080/axis/services/telescopeData</sd:
    <sd:wsdlURL>http://localhost:8080/axis/services/telescopeData</sd:wsdlURL>
  </rdf:Description>
  <rdf:Description rdf:about="http://users.cs.cf.ac.uk/S.Rajbhandari/provenance/service
     pd:endTime="00:27:20;20-Dec-2006"
     sd:serviceName="PowerOf2Service"
     sd:serviceOperationName="points"
```

Figure 6.4: Interface in the PQS for querying process IDs displaying ten results at a time



24.convolve_Process:upid:20473910-9163-11DB-B888-B78032396269
25.convolve_Process:upid:131E0220-9170-11DB-9727-F289467ADCD4
26.convolve_Process:u
27.convolve_Process:u
28.convolve_Process:u
29.convolve_Process:u

ID: convolve_Process:u

RDF/XML View of this p

```
<rdf:RDF
     xmlns:sd="http:/
     xmlns:pd="http:/
     xmlns:rdf="http:
     xmlns:j.0="http:
```

Process ID: convolve_Process:upid:20473910-9163-11DB-B888-B7
Process Location: http://localhost:8080/active-bpel/services/MyProcessPro
Process Operation Name: Conv

List of Services in the Process:

Service instance1:	http://localhost:8080/axis/services/DustCloud
Service instance2:	http://localhost:8080/axis/services/telescopeData
Service instance3:	http://localhost:8080/axis/services/PowerOfTwo
Service instance4:	http://localhost:8080/axis/services/PadZero
Service instance5:	http://localhost:8080/axis/services/MyFFT
Service instance6:	http://localhost:8080/axis/services/convolve

List of Inputs for the process: (*Edit Inputs in the text boxes for "What if Analysis"*)

Input:densityType(class java.lang.String)	gaussian
Input:widthParameter (double)	0.3
Input:n(int)	30
Input:numberOfPoints(int)	60
Input:Type(class java.lang.String)	sine

Output for the Process:
Output: 3.8550641313015475,4.42672304902

Re-Execute Process Result

Process ID: convolve_Process:upid:204
Process Location: http://localhost:8080/activ
Process Operation Name: Conv

Original Output: 3.8550641313015475,4.42672304902

Output from the Re-execution of the Process:

Current Output: 3.8550641313015475,4.42672304902

Re- execution Result: *The Re-execution output generated is the same as the original output.*

Figure 6.5: Interface to re-execute a process

output matches the currently generated output for verification purposes.

2. The *What-if Analysis* button is used to perform the re-execution of the process with changed input parameters.

The implementation was validated using the stored process instances of the Astrophysics workflow example introduced in Chapter 4. The output of the example process can be visualized as a graph plot. This was done using the third-party PlotWS [52] service. PlotWS is an Axis-based Web service for drawing graphs, which has been implemented as a wrapper around gnuplot [107] and exposes a subset of gnuplot's functionality. A form-based web interface was incorporated as shown in Figure 6.5, where the *Plot the Outputs to Compare* button can be used to view both the original and current outputs, and returns two 2D graphs (SVG images) to enable a visual comparison of the two outputs. The What-if Analysis has the same interface as that shown in Figure 6.5 for plotting graphs, but without the function to match the outputs.

6.4 Summary

This chapter has presented the implementation of a prototype provenance service that incorporates the collection and querying of data provenance for workflow enactments in a service-oriented environment. The implementation demonstrates the feasibility of the provenance representation language, and the interactions of the components and SOA technologies to support processes such as collecting, recording and query manipulation of data provenance. This chapter has also discussed the implementation and operation of the Provenance Service that has been developed to demonstrate the Provenance Model and its components and features, including the automated recording of data provenance during process execution. The Provenance Service provides an appropriate basis for experimental validation of the concepts developed in this thesis, including the performance and scalability of provenance recording and querying as the size of the Provenance Database grows, as well as the re-execution of the queried processes.

Chapter 7

Evaluation

7.1 Introduction

The implementation of our Provenance System and its main components were presented in Chapter 6. The Provenance System incorporates the conceptual Provenance Model presented in this thesis to enable provenance support in service-oriented environments. The system is based on an SOA infrastructure and includes a recording mechanism for automatic collection and storage of data provenance about process executions, and query interface tools to perform the re-execution and re-creation of workflows to aid in the verification of past processes and to perform "what-if" style analyses. The Provenance Model proposed in Chapter 4 aims to support the collection of data provenance, and the querying and re-execution of process executions by using a combination of client-server and Web service models. The PCS performs dynamic invocation of processes, and the collection and recording of data provenance about such invocations. Recording data provenance is important from the perspective of the notion of a "Living Document", introduced in Chapter 1, since re-execution of a recorded process enables clients to view and evaluate process results produced on-the-fly. The Provenance Model's PCS and PQS are therefore implicitly targeted towards providing functionalities to conform to the notion of a living document. In order to support the recording of data provenance in a structured form, data provenance representation formats were modelled and collection mechanisms and component interactions were proposed in Chapter 4. A collection and query mechanism for recording data provenance about the execution of a service-based workflow (i.e., in an SOA environment), and the re-execution of such recorded workflows, respectively, were developed. The questions that need to be addressed in the context of the mechanisms in the proposed model are:

- *Scalability of the PCS.* The scalability of the PCS for collection and recording data provenance is important to process execution, since the collection of data provenance is active at run-time and recorded in the database. Good scalability would reflect the PCS's ability to handle the collection of data provenance for complex workflows with minimal execution overhead.

- *Performance of re-execution in the PQS.* The PQS component can be used to query the data provenance of an enacted process in order to re-execute it. In such cases, it is important to establish that the query tasks result in reasonable performance (measured by the response time) as the number of concurrent clients increases.

The provenance system has been implemented to enable the experimental evaluation of the above issues, and thereby to analyze the Provenance Model embodied in the PCS and the PQS. The Provenance System was implemented with supporting user interfaces for the PCS and PQS. Thus, to evaluate these two components in terms of the issues discussed above, in this chapter experiments are conducted that use a set of client applications and the results are analysed.

This chapter is organized as follows. In Section 7.2 the experimental evaluation of the scalability of the collection and recording of data provenance in the PCS is presented. In Section 7.3 the experimental evaluation of the performance of query re-execution is presented. First experimental results are presented that compare the process running time when the provenance collection and recording functions are used with the running time without these functions. Then the experimental results of clients concurrently querying and re-executing processes are presented in order to analyse the performance of the PQS component. Finally, Section 7.4 concludes this chapter.

7.2 Scalability of the Provenance Collection Service

The primary purpose of this experiment is to show empirically that, by using the PCS component, data provenance about a process executing in a service-oriented environment can be scalably collected and recorded. The two workflow scenarios used in the evaluation are described and presented in Figure 7.1.

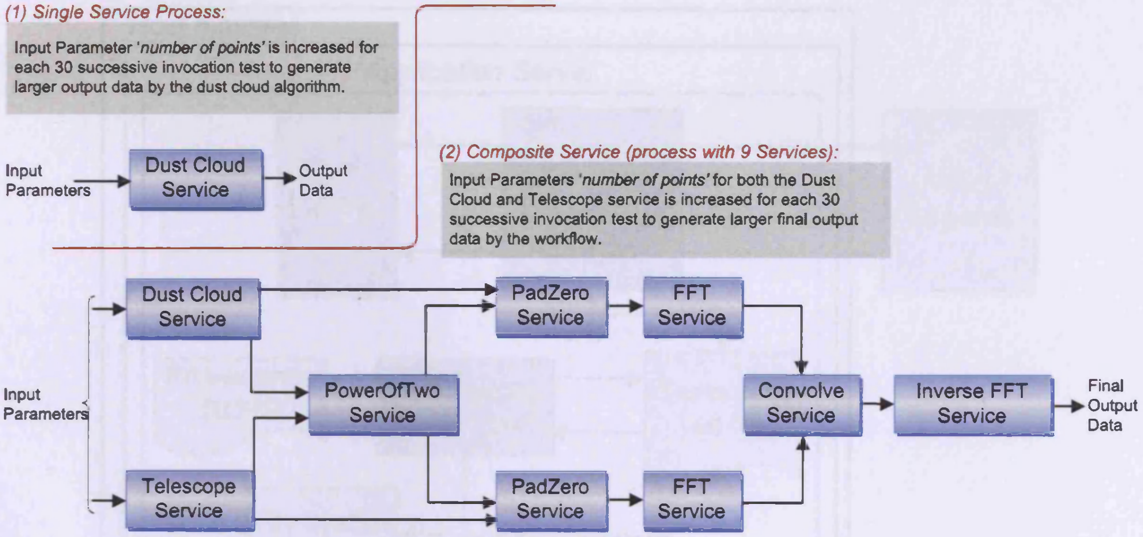

(1) Single Service Process:
Input Parameter 'number of points' is increased for each 30 successive invocation test to generate larger output data by the dust cloud algorithm.
Input Parameters
Dust Cloud Service
Output Data
(2) Composite Service (process with 9 Services):
Input Parameters 'number of points' for both the Dust Cloud and Telescope service is increased for each 30 successive invocation test to generate larger final output data by the workflow.
Dust Cloud Service
PadZero Service
FFT Service
Input Parameters
PowerOfTwo Service
Convolve Service
Inverse FFT Service
Final Output Data
Telescope Service
PadZero Service
FFT Service

Figure 7.1: The Web services and workflow used to generate data provenance for the experiments. The rectangular boxes denote the Web services implemented to demonstrate the Astrophysics Example Workflow from the example scenario presented in Chapter 4. The arrows denote the dataflow occurring in the workflow.

7.2.1 Summary of Setup

This section presents a summary of the setup for the experiments. The components of the Provenance Service are deployed on a laptop running Windows XP with service pack 2 with an Intel Pentium M processor operating at 1.7Ghz, and 1Gbytes of memory. The components include the PCS, the ActiveBPEL engine deployed in an Apache Tomcat 5.1 server, and the mySQL database server. The client applications that activate the PCS to collect and record provenance also run on the same machine. The Web service implementations for the experiments are deployed on a separate Windows XP desktop PC with an Intel Pentium processor operating at 2.80Ghz, and 512Mbytes of memory. The two machines are connected through GigaBit Ethernet.

Experiments were performed to evaluate the performance of the provenance record-

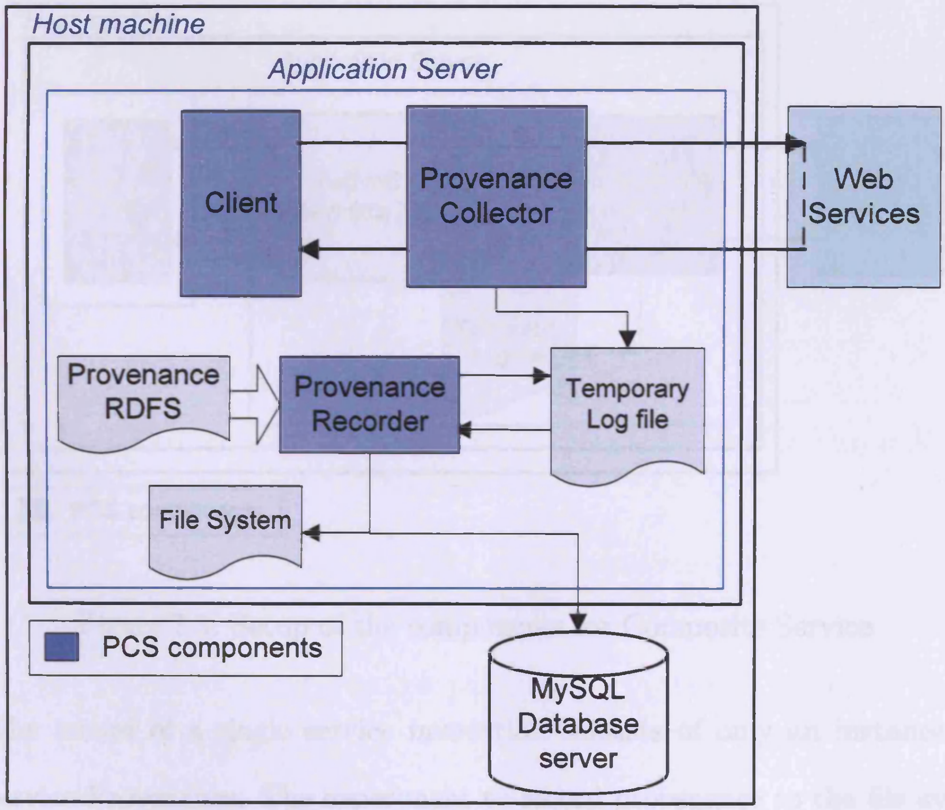


Figure 7.2: Setup of the components for Single Service

ing code of the PCS for documenting provenance of process executions using the process scenarios depicted in Figure 7.1. The scenarios are based on algorithms and mathematical models from the Astrophysics domain, and an example scenario is discussed in Chapter 4. The PCS was used with two setups of the components: (1) without the engine as shown in Figure 7.2 for single service and (2) with the ActiveBPEL engine as shown in Figure 7.3 for composite service.

1. *Single Service.* In this experiment a single service, such as the DustCloud service (Figure 7.1(1)), is invoked and its provenance recorded at run-time in (a) the file system and (b) the Provenance Database (i.e., the mySQL database).

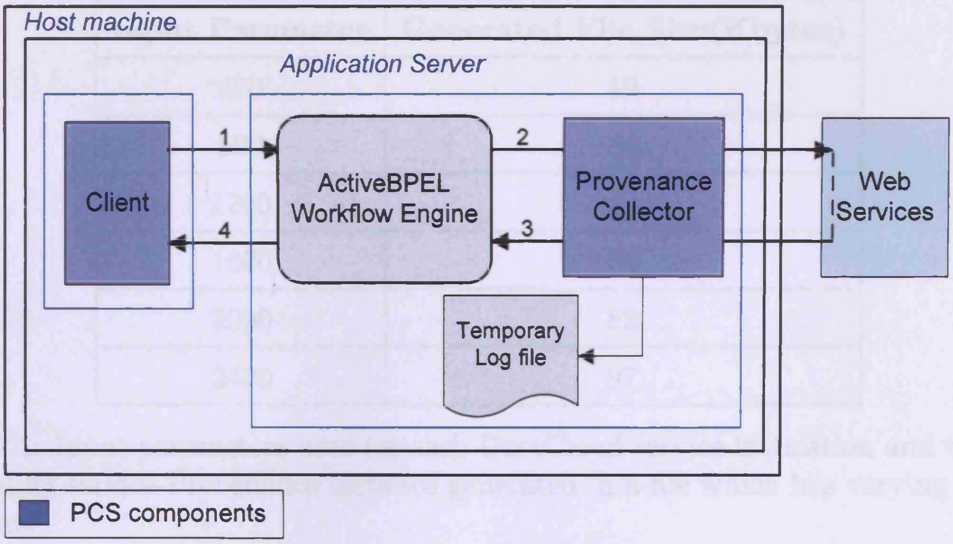


Figure 7.3: Setup of the components for Composite Service

The record of a single service invocation consists of only an instance of the service-Provenance. The experiment to record provenance to the file system is performed by creating an instance of the service-Provenance RDF file for each service invocation. This is done to demonstrate the approximate collection times of service-Provenance instances compared with the service invocation instances when provenance is not collected. The DustCloud algorithm takes input parameters and uses APIs from Java Math and the Java 2D Graph Package [26] to generate output data with X and Y data points that may be plotted as a graph. In order to get variation in the results of the tests performed, the DustCloud service's input parameter (the number of points) is increased for each test producing a service-Provenance record. This increases the total invocation time due to the increasing processing time taken by the service to produce output of increasing size. For example, if the number of points input is 100, the

Input Parameter	Generated File Size(Kbytes)
400	19
800	34
1200	48
1600	66
2000	82
2400	97

Table 7.1: Input parameters used for each DustCloud service invocation and the corresponding service-Provenance instance generated in a file which has varying output data size.

DustCloud service produces an output array of 200 data points (i.e., 100 times 2), and the values depend on the *width* input parameter. Table 7.1 shows the data used to perform the tests.

The tests to evaluate the performance of recording service invocations in the mySQL database is performed by increasing the number of service-Provenance instances in the database. For this experiment, the invocation tests are performed with fixed input parameters for the DustCloud service such that each test returns an instance of service-Provenance with a record size of 7 Kbytes.

2. *Composite Service.* A complex workflow composed of nine Web services (Figure 7.1(2)) representing a scenario in the Astrophysics domain has been developed to test the performance of the Provenance Collection Service. The implementations of the Web services described in the composite service in Figure 7.1(2) are used, and an abstract workflow constructed using BPEL4WS (abstract because the Web services implementations are not embedded in the

Input Parameters	Generated File Size(Mbytes)
400/400	0.44
800/600	0.51
1200/600	1.23
1600/600	1.42
2000/600	1.89

Table 7.2: Input parameters used (for DustCloud and Telescope services) for each complex Astrophysics Workflow invocation and the corresponding service-Provenance data generated by the Provenance Collector as an XML log file.

BPEL document which only provides references through the WSDL interfaces). The BPEL document, and the WSDL documents for the Web services, are deployed in the activeBPEL engine that provides an accessible WSDL interface for the Astrophysics Example Workflow (AEW) as a composite service. This AEW BPEL document is attached as Appendix C of this thesis. A Provenance Collector component (discussed in Section 4.3) is deployed within the engine to collect provenance about the Web service instances invoked by the engine. The invocation record of a composite service, such as the AEW, consists of an instance of process-Provenance and nine service-Provenance instances. The service-Provenance data for a process are collected by the Provenance Collector asynchronously in an XML file as a temporary log. After completion of a given process invocation, the XML file is then processed by the PCS to be recorded as service-Provenance instances (i.e., in RDF) in the MySQL database. Similar to the single service case, the AEW invocations are performed to collect and record the provenance at run-time, both in the file system and in a mySQL database. The tests are performed to demonstrate the approximate collection times of the process-Provenance instances compared with the AEW process invocation times when the Provenance Collector component is not active in the engine. Table 7.2 shows the data used to perform the tests and the size of the XML file generated for each test.

Benchmarks were performed, and the time measured to process the recording code of the PCS with various execution tests for both single service and complex service scenarios.

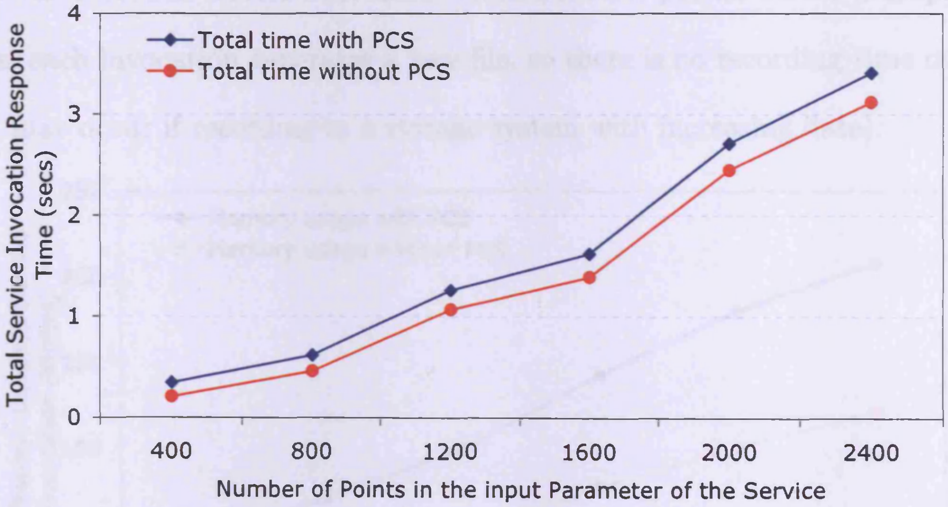


Figure 7.4: Single Service Recording: Invocation Computation Time

Results: Single Service Scenario

The results obtained for the experiments that recorded the provenance for the single service case in the file system and the mySQL database are now presented. For the experiments performed with the file system, the results are shown as graphs (in Figures 7.4 and 7.5) that illustrate the total time taken, and memory usage to invoke the DustCloud service, when (1) service-Provenance is actively being collected and recorded at run-time, and (2) no provenance is collected and recorded. For each graph, the x-axis shows the value of the number of points parameter passed to the DustCloud service. The y-axis in Figure 7.4 denotes the total invocation response time (i.e., the time taken from sending a request to receiving the result back) and the y-axis in Figure 7.5 denotes the total memory usage for the service invocations for different tests. The client applications that perform the invocation tests are set up so that the recorded service-Provenance data does not persist across multiple runs. That is, each invocation generates a new file, so there is no recording time overhead (which may occur if recording in a storage system with increasing data).

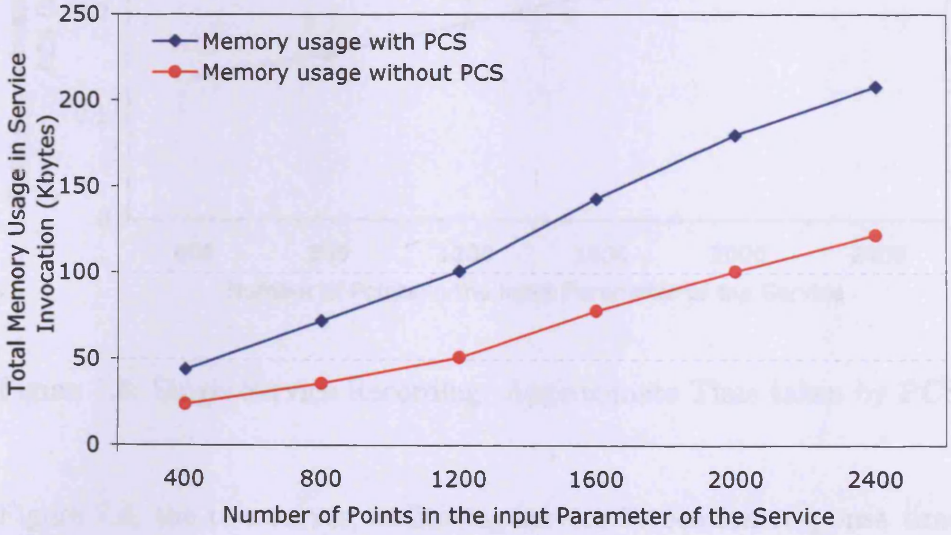


Figure 7.5: Single Service Recording: Memory Usage

Each point plotted in the graphs represents the average time of 30 successive service invocation tests. Each set of 30 successive invocation tests was carried out with the given input number of points both with and without recording provenance. As the size of the data output for each test increases with the increasing number of points input (Table 7.1), the corresponding amount of memory used to collect and record this also increases as shown in Figure 7.5. The memory usage represents the amount of heap memory used by the Java Virtual Machine executing the recording code of the PCS. The values has been collected using Java Management eXtension (JMX) [93]. The graph shows a linear trend as is expected for the problem size used in the tests.

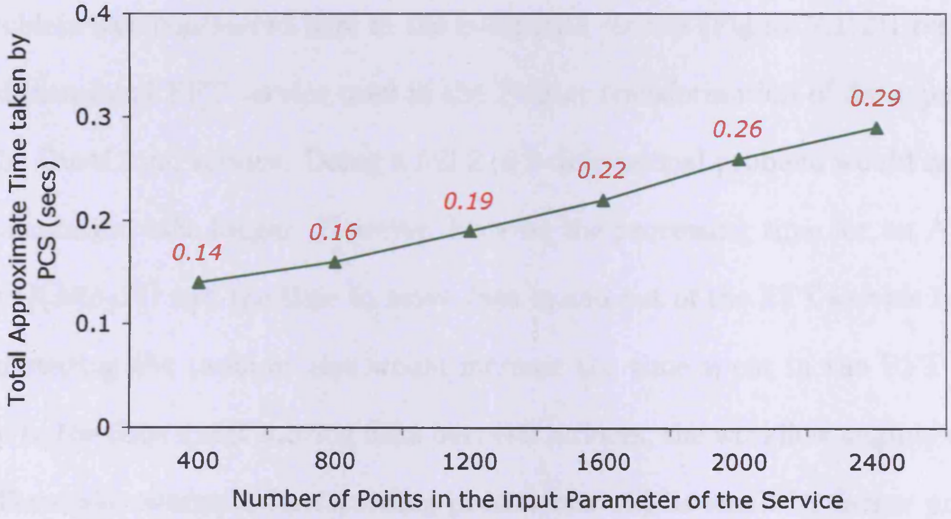


Figure 7.6: Single Service Recording: Approximate Time taken by PCS

In Figure 7.4, the two curves, indicating service invocation response times with and without recording provenance, rise in parallel and the increase is because the service's processing time is increasing. The difference between the response times (in seconds) plotted in these two curves is shown in Figure 7.6, where the x-axis denotes the number of points, as shown in Table 7.1. This provides the approximate times taken by the PCS to record service-Provenance instances as RDF files. As expected, the approximate times or overhead time taken by the PCS increases with the increasing number of points but the increment is negligible since the amount of data to record is not that large. In a real world application, the overhead of using the PCS may depend on the size and amount of the input and output data to be recorded but not the processing time of the services used. For example, a scientific application might involve the evaluation of a multidimensional Fourier transform. This consists of a series of one-dimensional FFTs, each of which is of few hundred or thousand points. Thus, this problem size considered here in the composite service (Figure 7.1(2)) represents a one-dimensional FFT service used in the Fourier transformation of data generated from the DustCloud service. Doing a full 2 or 3 dimensional problem would make the FFT calculation take longer. However, because the processing time for an N-points FFT is $O(NlogN)$ and the time to move data in and out of the FFT service is $O(N)$, then increasing the problem size would increase the time spent in the FFT service relative to the time spent moving data between services, the workflow engine, and the PCS. Thus, the overhead for recording provenance will be small for larger problems as well.

The second experiment for single service provenance recording was performed using the MySQL database (used as the Provenance Database). These results are shown in the graph in Figure 7.7, which shows the total response time to invoke the DustCloud service and record the provenance about the invocations, as the size of the service-Provenance in the database increases. The x-axis shows the number of service-Provenance records (instances) present in the database. The y-axis denotes

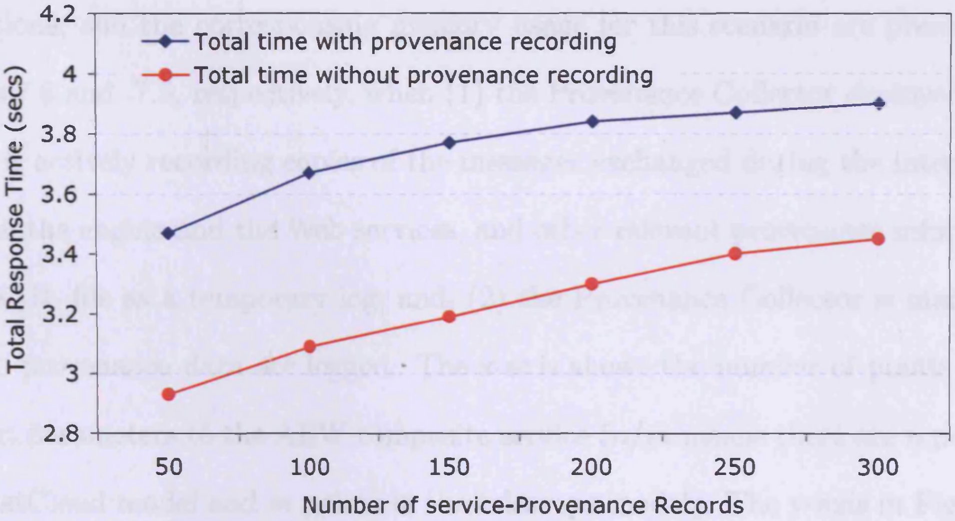


Figure 7.7: Single Service Recording in Database: Invocation Computation Time

the total time to invoke the service and record an instance of service-Provenance. Each value plotted in the graph represents the average value of 30 successive service invocation tests. Each of the 30 successive invocation tests was carried out for the total number of existing service-Provenance records in the database shown on the x-axis. The recording of provenance for a single service invocation in the database shows a steady increase with respect to the increasing number of database records. The additional time incurred with recording in the database is partly due to time taken to connect to the database, open and update the Jena model, used to store provenance data in RDF format.

Results: Composite Service Scenario

The results obtained from the experiments performed using the complex Astrophysics Example Workflow will now be discussed. The response times results for AEW invocations, and the corresponsing memory usage for this scenario are presented in Figures 7.8 and 7.9, respectively, when (1) the Provenance Collector deployed in the engine is actively recording copies of the messages exchanged during the interactions between the engine and the Web services, and other relevant provenance information in an XML file as a temporary log; and, (2) the Provenance Collector is inactive so that no provenance data are logged. The x-axis shows the number of points passed as input parameters to the AEW composite service (n/m means there are n points in the DustCloud model and m points in the telescope model). The y-axis in Figure 7.8 denotes the total response time and the y-axis in Figure 7.9 is the total memory usage for the AEW invocations for different tests. In the invocation tests the data collected in the XML file does not persist across multiple tests. That is, in the initial stage of every test run, the database is empty or a new temporary XML file is created to log provenance for each process invocation.

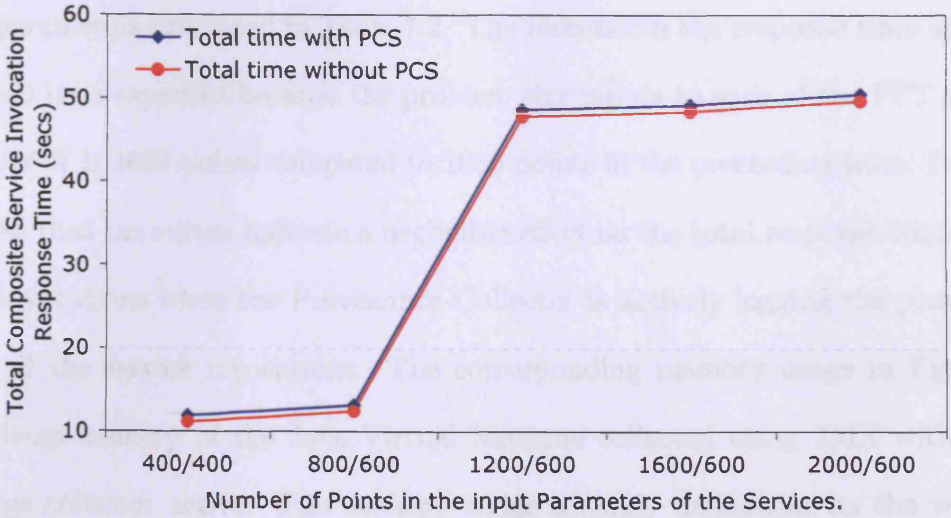


Figure 7.8: Composite Service Recording: Invocation Computation Time

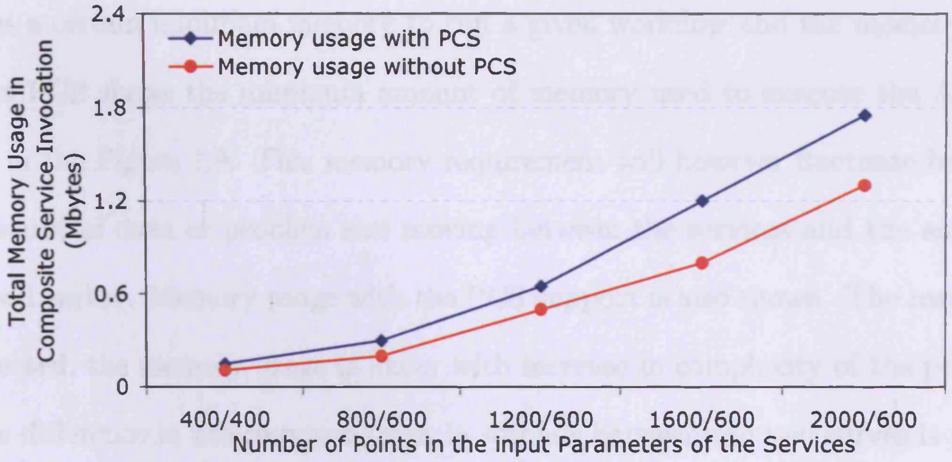


Figure 7.9: Composite Service Recording:Memory Usage

As in the single service evaluation, each value plotted represents the average of 30 successive AEW invocation tests. Each set of 30 successive tests was carried out with and without the collection of intermediate data (i.e., SOAP messages) by the Provenance Collector. The tests are performed with the number of points input parameters presented in Table 7.2. The increase in the response time at points 1200/600 is as expected because the problem size inputs to each of the FFT services in the AEW is 4026 points compared to 1056 points in the preceeding tests. It can be observed that the curves indicate a negligible effect on the total response time for the AEW invocations when the Provenance Collector is actively logging the provenance about all the service invocations. The corresponding memory usage in Figure 7.9 is the heap memory of the Java Virtual Machine collected using JMX with Java's garbbage collector active. The memory usage is largly dependent on the workflow engine requirements. The tomcat server where the engine is deployed also requires certain memory to run which is not included in this graph. The workflow engine requires a certain minimum memory to run a given workflow and the memory usage without PCS shows the miminum amount of memory used to execute the AEW as shown in the Figure 7.9. This memory requirement will however fluctuate based on the amount of data or problem size moving between the services and the engine as discussed earlier. Memory usage with the PCS support is also shown. The results are as expected, the memory usage is linear with increase in complexity of the problem.

The difference in the response times in seconds between the two curves is plotted in Fig. 7.10, which gives the approximate time taken by the Provenance Collector to intercept and collect data about Web service invocations in the engine for the AEW process. Here, the x-axis denotes the number of points of the input parameters (Table 7.2). This provides the approximate time or overhead incurred by adding the PCS support. The results in this graph indicate that the time taken by PCS increases with the increasing number of points or problem size as expected. The negative and positive error bars are also plotted for each points indicating the minimum and maximum approximate times.

It should be noted that the recording of the service-Provenance instances and the data links using the p-format occur after the invocation of the AEW composite service is completed and the final result is returned. Instead of the logging approach that has been adopted, if the Provenance Collector recorded the service-Provenance instances in RDF during the run-time of the AEW invocation, the response times would be larger, judging by the considerably higher time to record service-Provenance for a single service invocation depicted in Figure 7.6.

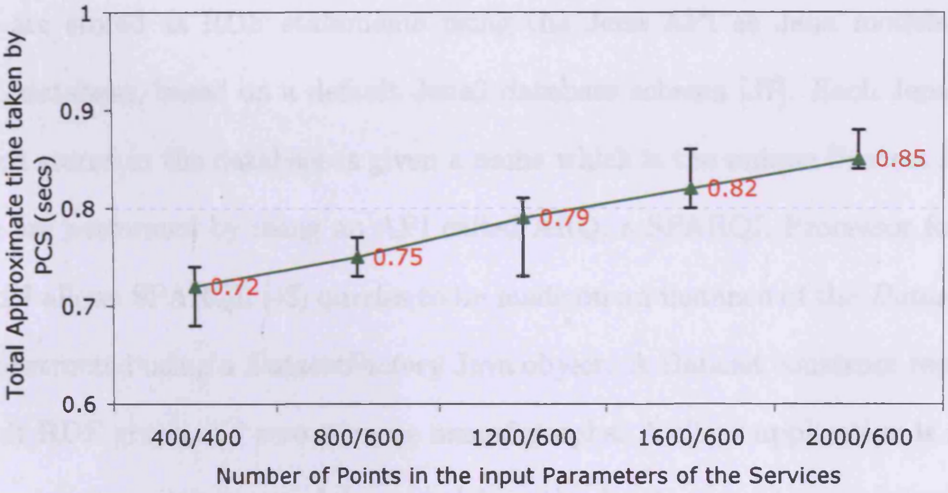


Figure 7.10: Composite Service Recording: Invocation Computation Time

7.3 Evaluation of the Provenance Query Service

This section presents experiments that show the affects of clients simultaneously performing various provenance queries and process re-executions using the Provenance Query Service. The experiments also demonstrate how the performance of the PQS varies as the size of the provenance records retrieved increases, and this is shown with various queries. The experiments investigate the scalability of the PQS.

7.3.1 Setup Summary

The hardware and software setup is the same as in Section 7.2.1. The client applications implemented to demonstrate simultaneous use scenarios, that exercise the components of the PQS, run on the same machine as the PCS. The PQS is evaluated using the p-format records of the composite service of the Astrophysics Example Workflow (AEW) presented in Fig. 7.1(2). The process instances in p-format are stored as RDF statements using the Jena API as Jena models in the mySQL database, based on a default Jena2 database schema [37]. Each Jena model or record stored in the database is given a name which is the unique Process ID. The queries are performed by using an API called ARQ, a SPARQL Processor for Jena. This API allows SPARQL [43] queries to be made on an instance of the *Dataset* Java class constructed using a *DatasetFactory* Java object. A Dataset construct represents a default RDF graph and zero or more named graphs. A client application is created that queries the existing named Jena models in the database, and represents them as named graphs of the Dataset object to execute the SPARQL queries over the models representing process instances. Figure 7.11 shows the setup and flow of data for the queries for the following experiments.

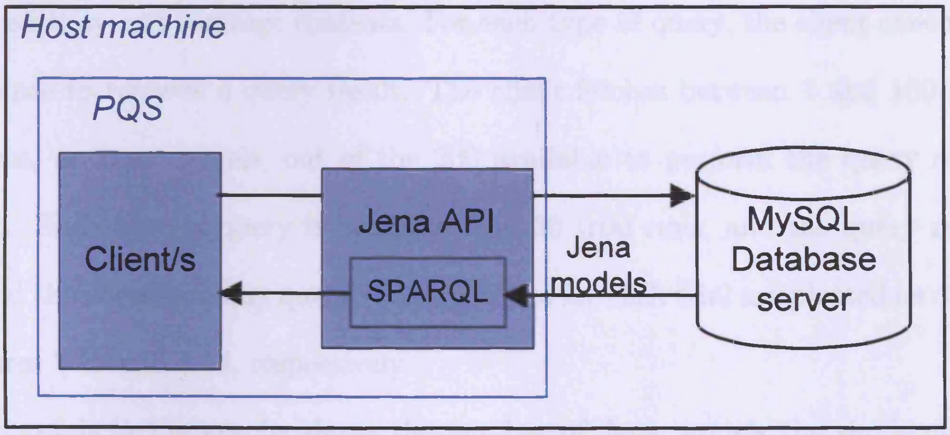


Figure 7.11: Setup and flow of data for the queries

The Size of the Results of a Provenance Query

This experiment measures the query response time as the number of provenance records retrieved increases. The experiments are performed with two queries on the

Provenance Database that is loaded with 200 instances of AEW enactments. Each AEW instance has one process-Provenance and ten service-Provenance instances. Thus, the provenance database simply consists of 200 process-Provenance and 2000 service-Provenance instances. A web client interface (presented in Chapter 7) is provided to query for one or more provenance records using the SPARQL query language. Using this client interface, *two types of query*, that specify RDF properties of (1) *pd:processId* and (2) *sd:serviceId*, were performed to retrieve a provenance query result, each as an XML document. The query performed with the RDF property *pd:processId* retrieves all the RDF statements of a matching process instance record, consisting of a process-Provenance and 10 service-Provenance instances. The service ID query retrieves only the service-Provenance instances as literal values, including service activity and message contents. For each type of query, the client executes the query once to retrieve a query result. The client fetches between 1 and 100 process instances, or Jena models, out of the 200 available to perform the query over the dataset. Each type of query is averaged over 30 trial runs, and the query response time and the corresponding query results file size for each trial are plotted as depicted in Figures 7.12 and 7.13, respectively.

The x-axis in the graphs shows the number of Jena models the single client retrieves from the database over which the queries are performed. The y-axis in Figure 7.12 shows the average response time. The bar graphs in Figure 7.13 represent the query result sizes for each of the corresponding query experiments. It can be seen from the graphs that the query response time plot for the query on service ID shows a linear trend beyond 15 retrieved result sets, the curve elevates giving an average

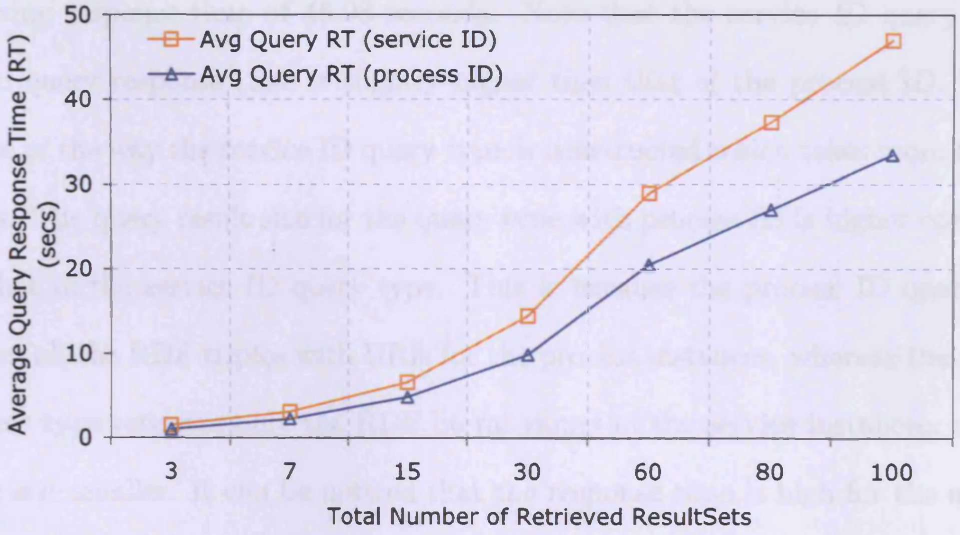


Figure 7.12: Average query response time for two types of query as the retrieved result set increases.

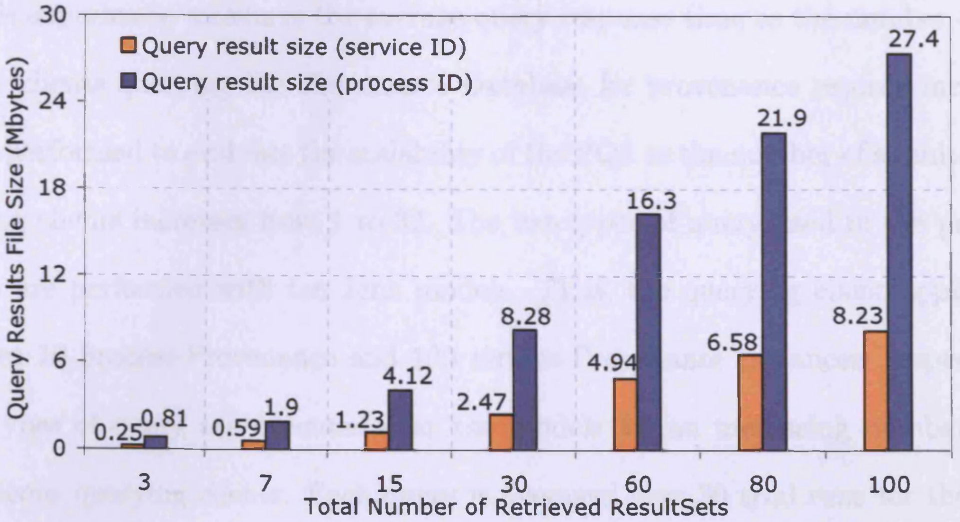


Figure 7.13: Query result file size for two types of query as the retrieved resultset increases.

response time of 14.45 seconds and beyond this point till 100 models shows a gradual rise giving response time of 46.98 seconds. Note that the service ID query type's average query response time is slightly higher then that of the process ID. This is because of the way the service ID query type is constructed which takes more time to process. The query result size for the query type with process ID is higher compared with that of the service ID query type. This is because the process ID query type retrieves all the RDF triples with URIs for the process instances, whereas the service ID query type retrieves only the RDF literal values of the service instances, making the file size smaller. It can be noticed that the response time is high for the amount of data queried in the tests. This is because of the lack of optimization of SPAPQL and also the indexing mechamism of Jena is not optimal.

Simultaneous Querying by Clients

This experiment measures the average query response time as the number of concurrent clients querying the Provenance Database for provenance records increases. This is performed to evaluate the scalability of the PQS as the number of simultaneous querying clients increases from 1 to 32. The two types of query used in the previous section are performed with ten Jena models. Thus, the querying client application retrieves 10 process-Provenance and 100 service-Provenance instances, respectively. Both types of query are executed over ten models for an increasing number of simultaneous querying clients. Each query is averaged over 30 trial runs for the given number of concurrent querying clients. The experimental results are presented in Figures 7.14 and 7.15.

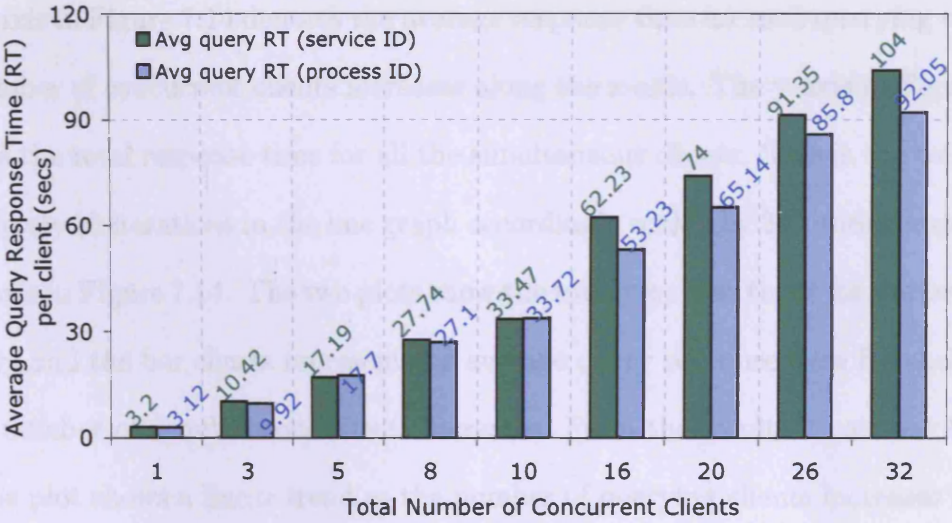


Figure 7.14: Average query response time per client as the number of concurrent clients performing the two kinds of query increases.

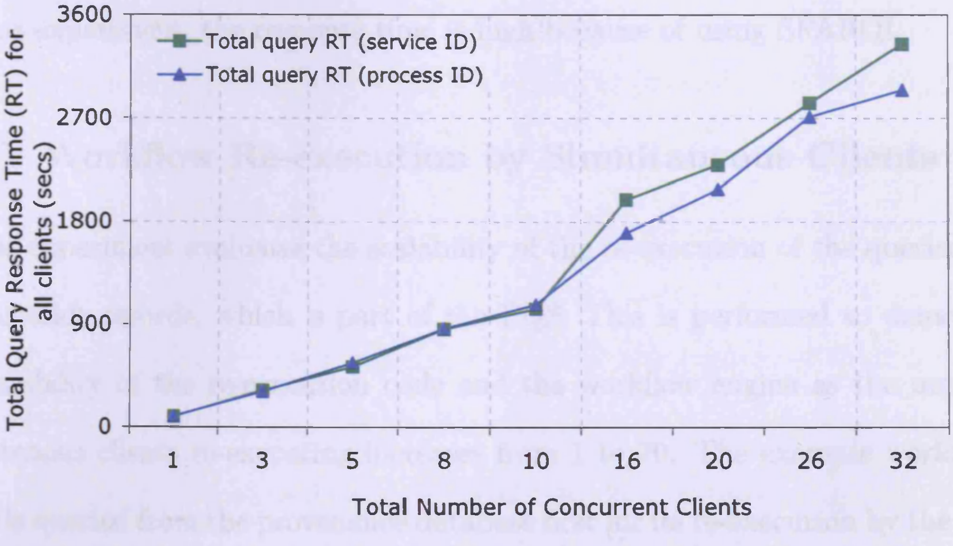


Figure 7.15: Total response time for all the clients as the number of concurrent clients performing the two kinds of query increases.

The x-axis in both the graphs shows the number of simultaneous querying clients. The y-axis in Figure 7.14 denotes the average response time for each querying client as the number of concurrent clients increases along the x-axis. The y-axis in Figure 7.15 denotes the total response time for all the simultaneous clients. This is the total time to complete 30 iterations in the line graph accordingly scaled by 30 times compared to the y-axis in Figure 7.14. The two plots show the total response times for the two types of query and the bar charts represent the average query response time for each client as the number of simultaneous clients increases. From the results it can be observed that the plot shows a linear trend as the number of querying clients increases beyond 10, taking an average of 33.72 seconds and 92.05 seconds to retrieve the provenance trace given the process ID for 10 and 32 clients respectively. The results for the query with service ID similarly indicates that the query component within the PQS has good scalability when the number of simultaneous clients increases. As in the previous experiment, the response time is high because of using SPARQL.

7.3.2 Workflow Re-execution by Simultaneous Clients

This experiment evaluates the scalability of the re-execution of the queried workflow instance records, which is part of the PQS. This is performed to demonstrate the scalability of the re-execution code and the workflow engine as the number of simulatenous clients re-executing increases from 1 to 70. The example workflow instance is queried from the provenance database first for its re-execution by the clients. The re-execution of the AEW is performed with the same input parameters for each client. Here, only the response time of the re-execution task is taken into account.

Each re-execution response time is averaged over 30 trials for the given number of simultaneous clients. The hardware and network configuration and the setup used for the experiment is the same as described in section 7.2.1 and in Figure 7.3, only the PCS is not included here.

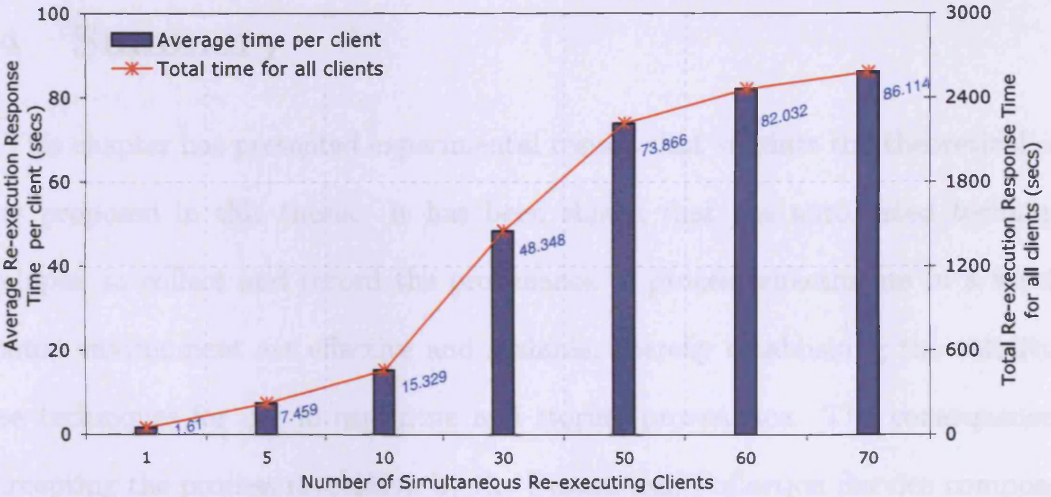


Figure 7.16: Average re-execution response time as the number of simultaneous clients re-executing a past workflow increases.

The results of this experiment are presented in Figure 7.16, which shows the average re-execution response time along the left y-axis as the number of simultaneous clients increases along the x-axis. The times are averaged over 30 iterations and the total time for all the clients to complete the 30 iterations is shown on the right y-axis. The PQS timings show a sub-linear trend as the number of clients increases beyond 10, taking an average of 15.32 seconds and 86.11 seconds, respectively, to re-execute the queried workflow on 10 and 70 clients. As seen from the graph, the slope decreases as the number of clients goes beyond ten, which may be due to the operating system or the machine hosting the workflow engine reaching its threshold. Thus, the response time for re-execution is highly dependent on the hosting environment of the workflow engine and the Web services. During the 30 iterations of re-execution for the increasing number of clients, no errors were encountered, which shows the stability and reliability of the activeBPEL engine that was used in the experiments.

7.4 Summary

This chapter has presented experimental results that validate the theoretical concepts proposed in this thesis. It has been shown that the automated techniques developed to collect and record the provenance of process enactments in a service-oriented environment are effective and scalable, thereby establishing the validity of these techniques for use in capturing and storing provenance. The consequence of intercepting the process invocation by the Provenance Collection Service component has been experimentally established, and it has been shown that the increase in the response time of the process invocation is negligible. In real world cases, this enables the use of larger problem sizes to be handled effectively by the PCS. Also the experiment conducted for the asynchronous recording of the collected provenance information in the database indicates reasonable performance. The results obtained from the Provenance Query Service experiments have established the good scalability and performance of the PQS component particularly for simultaneous clients querying and re-executing the processes. Due to the lack of optimization of SPARQL, the response time is high for the given result datasets. This problem can easily be solved by using appropriate SQL queries on the mySQL database.

The experimental results have therefore established that the primary components of our provenance model satisfy the requirements of a data provenance facility in a service-oriented environment. The proposed provenance model has been evaluated with a real scientific workflow example, and it has been demonstrated that it facilitates the capture and recording of data provenance for process invocations, so that past processes can be re-executed and verified. This brings the thesis to its concluding chapter where the contribution of this research will be summarised, and future directions of our work will be outlined.

Chapter 8

Future Work and Conclusions

8.1 Research Summary

The research presented in this thesis has focussed on addressing the specific requirements of the data provenance of Web processes in service-oriented environments. The emergence of the Service-Oriented Architecture (SOA) paradigm as the dominant infrastructure for e-service delivery, and the realization that support for data provenance is both necessary and viable for SOAs, has resulted recently in significant research interest in data provenance support in SOAs. Discussion in the data provenance research community (see for example [22, 28]) has revealed that numerous researchers in a variety of scientific disciplines are recognizing the importance of providing provenance for their dedicated data products to research partners and/or other potential data users. These discussions have also revealed the relevance of different application-level views of provenance, and demonstrate that operational systems to achieve provenance-aware applications are not yet common. A recent provenance workshop [20] has reported on provenance research being conducted that is inspired by previous research that relates to both provenance and workflow. Very little research has contributed towards providing a provenance-aware framework in a service-based environment – most work has focused on workflow applications with provenance tracking facilities for scientific domains.

In an open, large-scale and distributed environment of the type used in numerous disciplines of modern computational science, a provenance system helps scientific users to trace and evaluate research results of interest. In an SOA, there is a significant distinction between the abstract workflow document (or workflow specification) and the provenance document about the results produced from the invocation of that workflow. The work presented in this thesis has sought to elucidate the interrelations and differences between these two concepts, so that the system designed to support provenance tracing in an SOA is generic and well-suited to the needs of research scientists using a service-based infrastructure in their everyday computational investigations. This thesis has proposed a model to provide support for provenance in service-based environments, and has advanced the state-of-the-art by realizing the necessity of a generic and simple provenance system for use by research scientists. This thesis has focused on addressing the constraints imposed on scientists who need to retain a history of their computations when interacting in service-based environments.

This chapter concludes the thesis by summarizing the research contributions of this work in Section 8.2, and outlining the future directions that we intend to pursue in Section 8.3.

8.2 Research Contributions

The primary theme of this thesis is concerned with how researchers performing scientific experiments in a service-based environment can best compose and express data provenance for their new scientific data products, and how to query and reuse them. This thesis has contributed to the state-of-the-art of provenance support in service-oriented environments in a broad and non domain-specific way. This section presents the precise research contributions of this thesis with reference to the research objectives posed in Chapter 1. These contributions are as follows:

- *Development of the provenance architectural model.* The thesis has proposed a provenance model that supports the requirements of a data provenance facility for distributed, service-based workflows. These requirements are to provide a capability for capturing and storing data provenance, and for querying, exploiting and reusing the captured provenance information. The provenance model combines features of the widely-used Web service and client-server models, which facilitates the exposure of parts of the PCS and PQS components of the provenance model as Web services.

- *Development of a provenance format for representing the data provenance of workflow executions in an SOA.* Although there are standards to represent the provenance of a document, such as the Dublin Core [58], there is no Web Services standard for representing provenance information, especially for services and process enactments. Some existing provenance representation techniques are either domain-specific or technically too complex [84]. For example, the model presented in [84] is based on records about interactions between services. Here, provenance assertions about interactions are required to be recorded by all the involved services based on the unique identifiers of the interactions generated and passed between the services. There are different identifiers about various provenance assertions that require in depth learning to understand and use the recording mechanism and to perform queries. This thesis has proposed a provenance representation model, called p-format, that is simple to learn, expand, and adapt. The extensibility and flexibility of the p-format is due to the RDF data structure that has been used, which enables the creation of the new vocabulary needed to add additional meaningful provenance information with little or no change in the PCS and PQS components. The p-format is designed, in particular, to structure provenance for atomic and composite services, and for the data links that occur between the services of a composite service execution. Thus, the p-format facilitates the structuring and retrieval of the provenance for a piece of data in a way that is meaningful and interpretable by humans.

- *Development of a PCS with automated capturing and recording of data provenance.* Scientific research by its nature often involves complex computational processing of sensitive and large-scale data sets in a distributed environment. This necessitates a mechanism for collecting provenance in an SOA with minimal impact on the total time to process the complex computation. This dissertation has developed a provenance collection mechanism that intercept and logs provenance information at different service invocation points in the execution of a composite process. The logs for a process execution are processed following the completion of the execution, to record the provenance in the Provenance Database according to the provenance format. The time expended on assigning and recording provenance information in the database does not significantly burden the process enactment itself. Existing provenance support in SOAs addresses provenance recording but does not focus on the need to record provenance with little user interaction and execution overhead. This thesis has developed the PCS component to automatically collect and record provenance for process enactments. In order to support implementations, logs are provided as XML documents which can then be processed by the PCS for asynchronous provenance recording.

- *Development of the PQS for querying, re-execution, and re-creation of workflows.* This thesis has also developed the PQS component that provides the ability to query and reuse the recorded data provenance of process executions. The PQS component provides interfaces to retrieve process instances. In addition, the interface supports the re-execution of processes with different input parameters. Existing provenance research work in SOAs also has provided querying tools, but few deal with re-execution of previous workflows, and those that do are either application-specific or do not cater for a service-based environment. This thesis provides a mechanism for the re-execution of past processes that fulfills the requirements of research scientists to verify results and to perform what-if analysis on the processes. This assumed that the required services exists during re-execution.

- *Experimental evaluations of the PCS and PQS.* Finally, the PCS and PQS components of our provenance model have been evaluated, and experimental results have been presented and analyzed.

The above discussion has highlighted the principal contributions of this thesis. Future directions of this research work will now be briefly discussed.

8.3 Research Directions

The previous section has described the primary contributions of our research, and this section concludes the thesis by outlining areas for future work.

The dissertation has focussed on modelling, recording and querying the provenance of workflow enactments in a service-based environment. Motivated by the idea of a "Living Document" as introduced in Chapter 1, we have considered the recording of provenance, such that the captured provenance enables re-execution of past workflows. At the start of the re-execution of a past workflow problems may arise from services being unavailable, moved, or no longer existing at that point in time, thereby causing the re-execution to fail. Enhancing the re-execution model to tackle such problems by searching for, and selecting, a similar service (i.e., one that performs the same operation), and substituting it in the workflow to be re-executed, is being considered. The approach to searching may be closely related to the increasingly active area of semantic-based service description, discovery, and matching. We intend to study the searching and matching of semantically described services, and develop an integrated approach to provide a way to handle re-execution failures caused by unavailable, moved or terminated services. We also intend to consider the notion of "smart" re-execution, where only the services whose inputs are changed are re-executed in the workflow. This is valuable for scientists investigating larger workflows as it reduces the re-execution time, and subsumes the concept of checkpointing. To incorporate this requires the workflow engine to interpret the provenance data, which therefore requires further investigation.

While part of the work in this thesis is specific to re-execution of past workflows, a future extension to provide re-execution without using BPEL may be considered. Although the PCS was utilized in Section 7.2 to record provenance of a service invocation without using BPEL, a further investigation on the provenance data model is required in this case to see how re-execution could be supported.

Future development of the PCS for automatic provenance recording will move in the direction of creating APIs for recording data provenance directly by the researcher performing the experiments. This makes it easier for researchers to directly use the data provenance recording functionality. This would help to include additional information, for example causality relationships, by manual assertions as some information may not be automatically recorded and interpreted. This approach would complement the trend toward increasing online propagation of scientific data by providing data provenance consisting of human-understandable details to explain the workflow process that led to a particular piece of data.

The implementation presented to re-create workflows using a graphical display provides a high-level view of what happened to produce some piece of data. Applying this as a tool to enable researchers to place some degree of trust in the produced data is being considered. In our implementation of the PQS we have experimented with multiple aspects of exploiting data provenance. We intend to investigate how trust in an enacted workflow may be established and modelled with respect to data provenance.

Data provenance has thus far been the method of choice to enhance scientific workflows or applications that deal with the production of data with added scientific value. On one hand, the distributed Web-based approaches adopted by researchers focus on supporting the needs of many researchers in setting up an infrastructure for sharing scientific data and meta-data in order to propagate scientific resources. However, service-based adaptation of scientific workflows has the potential to bring data provenance infrastructure of the type presented here to a level at which scientific data could be verified by re-executing the workflow that produced that data. This benefits researchers trying to study published works by giving greater insight into the research of others, and by bringing new opportunities and challenges to the research community. Future extensions of our research on provenance support for service-based scientific workflows will focus on the exploitation of data provenance. One future focus will be the investigation of better ways to automatically generate the data provenance of workflows in order to capture richer semantics that would help a later search for a method to solve a particular problem. For example, given a set of data provenance of workflows, it would be interesting to solve provenance questions like, how can a system discover or synthesize a workflow if the only input the user provides is a desired outcome (that may only be formed by combining parts of workflow recipes and data from enacted workflows [68]).

8.4 Research Publications

Different aspects of this research have been validated and presented in the proceedings of peer reviewed international conferences:

- Shrija Rajbhandari and David W. Walker. Support for Provenance in a Service-based Computing Grid. In Proceedings of UK e-Science All Hands Meeting, Notthingham, U.K., 2004.

- Shrija Rajbhandari and David W. Walker. Incorporating Provenance in Service Oriented Architecture. In Proceedings of 2nd International Conference on Next Generation Web Services Practices (NWeSP), Seoul, South Korea, 2006.

Bibliography

[1] ATLAS Chimera. http://grid.uchicago.edu/atlaschimera/, 2004.

[2] Collaboratory for Multi-Scale Chemical Science. http://cmcs.org/home.php, 2004.

[3] myGrid Project. http://www.mygrid.org.uk/, 2004.

[4] Provenance Aware Service Oriented Architecture (PASOA) Project. http://www.pasoa.org/, 2004.

[5] MySQL Database Server. http://www.mysql.com/, 2005.

[6] Universally Unique Identifier (UUID). http://en.wikipedia.org/wiki/UUID, 2006.

[7] Web-based Distributed Authoring and Versioning (WebDAV). http://www.webdav.org/, 2006.

[8] The World Wide Web Consortium (W3C). http://www.w3.org/, 2007.

[9] Active Endpoints, Inc. ActiveBPEL open source engine. http://www.active-endpoints.com/active-bpel-engine-overview.htm, 2005.

[10] Active Endpoints, Inc. ActiveBPEL designer. http://www.active-endpoints.com/active-bpel-designer.htm, 2006.

[11] Alexander Aiken, Jolly Chen, Michael Stonebraker, and Allison Woodruff. Tioga-2: A direct manipulation database visualization environment. In *IEEE Proceedings for 12th International Conference on Data Engineering*, 1996.

[12] International Virtual Observatory Alliance. An IVOA Standard for Unified Content Descriptor. http://www.ivoa.net/Documents/latest/UCD.html, 2005.

[13] Gustavo Alonso and Claus Hagen. Geo-Opera: Workflow Concepts for Spatial Processes. In *Proceedings of GIS/LIS Conference*, pages 144–153, 1997.

[14] Altova. Altova XMLSpy 2007, the industry-standard XML editor. http://www.altova.com, 2007.

[15] Altova. SemanticWorks 2007, visual semantic web tool. http://www.altova.com/downloadtrialsemanticworks.html, 2007.

[16] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business process execution language for web services (BPEL4WS). ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, 2005.

[17] James Annis, Yong Zhao, Jens Voeckler, Michael Wilde, Steve Kent, and Ian Foster. Applying chimera virtual data concepts to cluster finding in the sloan

sky survey. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–14, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[18] M Beasley, S Datta, H Kogelnik, H Kroemer, and D Monroe. Report of the investigation committee on the possibility of scientific misconduct in the work of hendrik schon and co-authors. In *Technical report*, 2002.

[19] Marjorie Berdeen, Eric Gilbert, Thomas Jordan, Paul Nepywoda, Elizabeth Quigg, Michael Wilde, and Yong Zhao. The quarknet/grid collaborative learning e-lab. In *Second International Workshop on collaborative and learning Applications of Grid Technology and Grid Education*, 2005.

[20] Dave Berry, Peter Buneman, Ian Foster, and James Frew. Proceedings of the international provenance and annotation workshop (IPAW 2006). http://www.ipaw.info/ipaw06/proceedings.html, Chicago, Illinois, May 2006.

[21] Dave Berry, Peter Buneman, Michael Wilde, , and Yannis Ioannidis. Workshop on data provenance and annotation, December 2003. National e-Science Institute, Edinburgh.

[22] Dave Berry, Peter Buneman, Michael Wilde, and Yannis Ioannidis. Workshop on data provenance and annotation. http://www.nesc.ac.uk/esi/events/304/, December 2003.

[23] Rajendra Bose and James Frew. Composing lineage metadata with xml for custom satellite-derived data products. In *16th International Conference on*

Scientific and Statistical Database Management (SSDBM'04), Santorini Island, Greece, 21-23 June 2004.

[24] S.E Brenner. Errors in genome annotation. In *Trends in Genetics*, volume 15, pages 132–133, 1999.

[25] D.O Briukhov, L.A Kalinichenko, and V.N Zakharov. Diversity of domain descriptions in natural science:virtual observatory as a case study. In *Proceedings of the 7th Russian Conference on Digital Libraries (RCDL 2005)*, Yaroslavl, Russia, 2005.

[26] Leigh Brookshaw. Java 2D graph package. http://www.sci.usq.edu.au/staff/leighb/graph/, 2006.

[27] Peter Buneman, Alin Deutsch, and Wang-Chiew Tan. A deterministic model for semistructured data. In *Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, pages 14–19, 1999.

[28] Peter Buneman and Ian Foster. Workshop on Data Derivation and Provenance. http://www-fp.mcs.anl.gov/ foster/provenance/, Chicago, October 17-18, 2002.

[29] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Data provenance: Some basic issues. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1974 / 2000, page 87, New Delhi, India, December 2000. Lecture Notes in Computer Science Publisher:Springer-Verlag.

[30] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *International Conference on Database*

Theory (ICDT 2001), volume 1973 / 2001, pages 316–330. Lecture Notes in Computer Science Publisher:Springer-Verlag, January 2001.

[31] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. On propagation of deletions and annotations through views. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 150–158, New York, NY, USA, 2002. ACM Press.

[32] Nicola Cannata, Flavio Corradini, and Emanuela Merelli. A resourceomic grid for bioinformatics. *Future Generation Computer System*, 23(3):510–516, 2007.

[33] Maria Claudia Cavalcanti, Marta Mattoso, Maria Luiza Campos, Eric Simon, and Francois Llirbat. An architecture for managing distributed scientific resources. In *14th International Conference on Scientific and Statistical Database Management (SSDBM)*, 2002.

[34] Maria Cludia Cavalcanti, Marta Mattoso, Maria Luiza Campos, Francois Llirbat, and Eric Simon. Sharing scientific models in environmental applications. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 453–457, New York, NY, USA, 2002. ACM Press.

[35] Kishore Channabasavaiah, Kerrie Holley, and Edward Tuggle. Migrating to a service-oriented architecture, ibm developerworks. http://www-128.ibm.com/developerworks/library/ws-migratesoa/, December, 2003.

[36] Hewlett-Packard Development Company. Jena a semantic web framework for java. http://jena.sourceforge.net/, 2004.

[37] Hewlett-Packard Development Company. Jena2 database interface - database layout. http://jena.sourceforge.net/DB/layout.html, 2004.

[38] The Gene Ontology Consortium. Gene ontology database. http://www.geneontology.org/.

[39] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'00)*, San Diego, California, February 2000.

[40] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. In *ACM Transactions on Database Systems*, volume 25, pages 179–227, June 2000.

[41] Vikas Deora, J Shao, W. A Gray, and N. J Niddian. Modelling quality of service in service oriented computing. In *Second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06)*, pages 95–101. IEEE Computer Society, November 2006.

[42] D Devos and A Valencia. Intrinsic errors in genome annotation. In *Trends in Genetics*, volume 17, pages 429–131, 2001.

[43] W3C Working Draft. SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 4 October 2006.

[44] University of California Environmental Information Laboratory. The Earth System Science Server (ES3). http://essw.bren.ucsb.edu/projects/proj-essw.htm, 2004.

[45] Environmental System Research Institute, Inc. Understanding GIS-the Arc/Info method. http://www.ciesin.columbia.edu/docs/005-331/005-331.html, 1992.

[46] Extreme Lab, Indiana University. Web and XML Services Utility Library(WS/XSUL). http://www.extreme.indiana.edu/xgws/xsul/, 2004.

[47] Hao Fan. Tracing data lineage using automed schema transformation pathways. In *BNCOD 19: Proceedings of the 19th British National Conference on Databases*, pages 50–53, London, UK, 2002. Springer-Verlag.

[48] Ian T. Foster. The anatomy of the grid: Enabling scalable virtual organizations. In *First IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, pages 6–7, Brisbane, Australia, May 15-18 2001.

[49] Ian T. Foster, Jens-S. Vöckler, Michael Wilde, and Yong Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *14th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 37–46, 2002.

[50] James Frew and Rajendra Bose. Earth science workbench: A data management infrastructure for earth science products. In *SSDBM '01: Proceedings of the Thirteenth International Conference on Scientific and Statistical Database Management*, pages 180–189, Washington, DC, USA, July 18-20 2001.

[51] James Frew and Jeff Dozier. Data management for earth system science. *SIG-*

MOD Record (ACM Special Interest Group on Management of Data), 26(1):27–31, 1997.

[52] G-Hydroflex. PlotWS, southampton regional e-science centre. http://www.soton.ac.uk/ ghydflex/plotws/, 2006.

[53] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian-Augustin Saita. Improving data cleaning quality using a data lineage facility. In *Design and Management of Data Warehouses*, page 3, 2001.

[54] Dan Gisolfi. Web service architecture: Part 1. http://www.ibm.com/developerworks/webservices/library/ws-arc1/, April 2001.

[55] Paul Groth, Simon Miles, and Luc Moreau. Preserv: Provenance recording for services. In Simon J. Cox and David W. Walker, editors, *Proceedings of the UK e-Science All Hands Meeting 2005, published on CD*, Nottingham, UK, September 2005.

[56] W3C Working Group. Web services architecture. http://www.w3.org/TR/ws-arch/, February 2004.

[57] Yan Huang. Service workflow language (swfl). http://users.cs.cf.ac.uk/Yan.Huang/GridWF/SWFL.htm, 2003.

[58] Dublin Core Metadata Initiative. Dublin Core Metadata Element Set, Version 1.1. http://dublincore.org/documents/dces/, 2006.

[59] James D Myers Jens Schwidder, Tara Talbott. Bootstrapping to a semantic grid. In *Proceedings of the Semantic Infrastructure for Grid Computing Applications Workshop at IEEE/ACM International Symposium on Cluster Computing and the Grid CCGRID 2005*, May 9-12 2005.

[60] A. Jones, R. White, N. Pittas, W. Gray, T. Sutton, X. Xu, O. Bromley, N. Caithness, F. Bisby, N. Fiddian, M. Scoble, A. Culham, and P.Williams. BiodiversityWorld: An architecture for an extensible virtual laboratory for analysing biodiversity patterns. In *UK e-Science All Hands Meeting, EPSRC*, pages 759–765, Nottingham, UK, september 2003.

[61] J.Zhao, C.A. Goble, M. Greenwood, C. Wroe, and R. Stevens. Annotating, linking and browsing provenance logs for e-science. In *1st Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data in conjunction with 2nd International Semantic Web Conference*, 20th October 2003.

[62] Peter Karp. What we do not know about sequence analysis and sequence databases. In *Bioinformatics*, pages 753 – 754, 1998.

[63] Pacific Northwest National Laboratory. Scientific Annotation Middleware (SAM). http://collaboratory.emsl.pnl.gov/software/sam/, 2006.

[64] D.P. Lanter. A lineage meta-database program for propagating error in geographic information systems. In *Proceedings of GIS/LIS Conference*, pages 144–153, 1990.

[65] D.P. Lanter. Design of a lineage-based meta-database for gis. In *Cartography and Geographic Information Systems*, pages 255–261, 1991.

[66] D.P. Lanter. A lineage meta-database approach toward spatial analytic database optimization. In *Cartography and Geographic Information Systems*, pages 112–121, 1993.

[67] D.P. Lanter. Comparision of spatial analytic applications of GIS. In *Environmental information management and analysis;echosystem to globel scales*, pages 413–425, 1994.

[68] Shalil Majithia, David W. Walker, and W.A. Gray. Automated composition of semantic grid services. In *Proceedings of UK e-Science All Hands Meeting*, Notthingham, U.K., 2004.

[69] Arunprasad P. Marathe. Tracing lineage of array data. In *Proceedings of Thirteenth International Conference on Scientific and Statistical Database Management SSDBM*, pages 69 - 78, 18-20 July 2001.

[70] Deborah L. McGuinness and Paulo Pinheiro da Silva. Inference web: Portable and shareable explanations for question answering. In *Proceedings of the American Association for Artificial Intelligence, Spring Symposium Workshop on New Directions for Question Answering*, pages 67–71. AAAI Press, March 2003.

[71] Deborah L. McGuinness and Paulo Pinheiro da Silva. Infrastructure for web explanations. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, pages 113–129. Springer, October 2003.

[72] Deborah L. McGuinness and Paulo Pinheiro da Silva. Explaining answers from the semantic web: The inferenceweb approach. In *Journal of Web Semantics*, volume 1, pages 397–413, October 2004.

[73] Deborah L. McGuinness and Paulo Pinheiro da Silva. Trusting answers from web applications. In Mark T. Maybury, editor, *New Directions in Question Answering*. AAAI/MIT Press, October 2004.

[74] James D. Myers, Carmen Pancerella, C Lansing, Schuchardt K.L, and B Didier. Multi-scale science: supporting emerging practice with semantically derived provenance. In *ISWC 2003 Workshop: Semantic Web Technologies for Searching and Retrieving Scientific Data*, October 20 2003.

[75] Grid Physics Network. GriPhyN Project. http://www.griphyn.org/, 2006.

[76] University of Chicago. Open grid services architecture (OGSA). http://www.globus.org/ogsa/, 2007.

[77] University of Southampton. Centre for proteomic research. http://www.proteome.soton.ac.uk/, 2004.

[78] OASIS Open. Universal Description Discovery and Integration (UDDI) Specification Version 3. http://uddi.org, 2006.

[79] G Scott Owen. Geographical Information Systems (GIS). http://www.siggraph.org/education/materials/HyperVis/applicat/gis/gis.htm, 1999.

[80] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C:The Art of Scientific Computing.* Cambridge University Press, West Sussex, England, 1992.

[81] Apache Web Service Project. Apache SOAP (WS-SOAP). http://ws.apache.org/soap/, 2003.

[82] Apache Web Service Project. Axis. http://ws.apache.org/axis/, 2004.

[83] Apache Web Service Project. Web Services Invocation Framework (WSIF). http://ws.apache.org/wsif/, 2004.

[84] EU Provenance Project. Enabling and supporting provenance in grids for complex problems. http://gridprovenance.org, 2006.

[85] SOAP Lite Project. SOAP::Lite. http://www.soaplite.com/, 2005.

[86] Shrija Rajbhandari, Ali Shaikh Ali, Ian Wootten, and Omer.F Rana. Evaluating provenance-based trust for scientific workflows. In *Sixth IEEE International Symposium on Cluster Computation and the Grid(CCGrid06)*, pages 365–372, Singapore, May 2006.

[87] W3C RDF Core Working Group. Resource Description Framework (RDF). http://www.w3.org/RDF/, 2004.

[88] W3C Recommendation. XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath, 2007.

[89] P. Ruth, D. Xu, B. K. Bhargava, , and F. Regnier. E-notebook middleware for acccountability and reputation based trust in distributed data sharing communities. In *Proceedings on 2nd International Conference on Trust Management*, Oxford, UK, 2004. LNCS Springer.

[90] Spatial Data Transfer Standard (SDTS). U.s. geological survey. http://mcmcweb.er.usgs.gov/sdts/SDTS_standard_nov97/part1b12.html, 2001.

[91] D. Sulakhe, A. Rodriguez, M. D'Souza, M. Wilde, V. Nefedova, I. Foster, and N. Maltsev. GNARE: an environment for grid-based high-throughput genome analysis. In *Cluster Computing and the Grid, CCGrid 2005 IEEE International Symposium on*, pages 455–462. IEEE International Press, 2005.

[92] Sun Microsystems, Inc. Java 2 Software Development Kit (J2SDK). http://java.sun.com/, 2005.

[93] Sun Microsystems, Inc. Java Management eXtension. http://java.sun.com/products/JavaManagement/, 2005.

[94] Sun Microsystems, Inc , Sun Developer Network(SDN). JavaServer Pages (JSP) technology 2.0. http://java.sun.com/products/jsp/, 2003.

[95] Tara Talbott, Michael Peterson, Jens Schwidder, and James D. Myers. Adapting the electronic laboratory notebook for the semantic era. In *Proceedings of the 2005 International Symposium on Collaborative Technologies and Systems (CTS 2005)*, May 15-20 2005.

[96] The Apache Software Foundation. Apache Tomcat. http://tomcat.apache.org/, 2006.

[97] The National Collaboratories Program, U.S. Department of Energy. Collaboratory for Multi-Scale Chemical Science (CSMC). http://cmcs.org/index.php, 2005.

[98] EGEE User and Application Portal. Earth science application. http://www.eu-egee.org/.

[99] H. Veregin and Peter D.Lanter. Data-quality enhancement techniques in layer-based geographic information systems. In *Computers, Environment and Urban Systems*, pages 23–36, 1995.

[100] World Wide Web Consortium (W3C). Web Services Descriptipn Language(WSDL)1.1. http://www.w3.org/TR/wsdl, 15 March, 2001.

[101] World Wide Web Consortium (W3C). SOAP version 1.2 specification, messaging framework. http://www.w3.org/TR/soap12-part1/, 24 June, 2003.

[102] W3C Recommendation. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/, 2007.

[103] D. W. Walker, M. Li, O. F. Rana, M. S. Shields, and Y. Huang. The software architecture of a distributed problem-solving environment. *Concurrency: Practice and Experience*, 12(15):1455–1480, December 2000.

[104] Inference Web. Semantic Web infrastructure for provenance and justification. http://www.inference-web.org/, 2006.

[105] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Second Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 262–276, January 2005.

[106] Wikipedia. Workflow defination. http://en.wikipedia.org/wiki/Workflow.

[107] Thomas Williams, Colin Kelley, Russell Lang, Dave Kotz, John Campbella, Gershon Elber, and Alexander Woo. Gnuplot. http://www.gnuplot.info/, 2006.

[108] Allison Woodruff and Michael Stonebraker. Supporting fine-grained lineage in a database visualization environment. In *IEEE Proceedings for 13th International conference on Data Engineering*, 1997.

[109] Ian Wootten, Shrija Rajbhandari, and Omer Rana. Automatic assertion of actor state in service oriented architectures. In *IEEE International Conference on Web Services)*, Salt Lake City, Utah, USA, July 2007.

[110] Ilya Zaihrayeu, Paulo Pinheiro da Silva, and Deborah L. McGuinness. Iwtrust: Improving user trust in answers from the web. In *Proceedings of 3rd International Conference on Trust Management (iTrust2005)*, Rocquencourt, France, October 2005. Springer.

[111] Jun Zhao, Carole Goble, Robert Stevens, and Sean Bechhofer. Semantically linking and browsing provenance logs for e-science. In *Proc. of the 1st International Conference on Semantics of a Networked World*, pages 155–176, Paris, France, January 2004.

[112] Yong Zhao, Michael Wilde, Ian Foster, Jens Voeckler, James Dobson, Eric

Gilbert, Thomas Jordan, and Elizabeth Quigg. Virtual data grid middleware services for data-intensive science. In *Concurrency and Computation: Practice and Experience*, pages 455–462, 2005.

Appendix A

RDFS of Provenance Format

This Appendix shows the rdf schema to structure a web process.

```
<rdf:RDF xml:base="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/processProvenance#" xmlns:pd="http://
www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/processProvenance#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:sd="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/
serviceProvenance#">
<!--Provenance for a Process instance  -->
        <rdfs:Class rdf:about="http://www.cs.cf.ac.uk/user/s.rajbhandari/provenance/processProvenance">
                <rdfs:label>pd</rdfs:label>
        </rdfs:Class>
<!-- Service Instance Profile -->
        <rdfs:Class rdf:about="http://www.cs.cf.ac.uk/user/s.rajbhandari/provenance/serviceInstanceProfile">
                <rdfs:label>serviceInstanceProfile</rdfs:label>
        </rdfs:Class>
        <rdfs:Class rdf:about="DataLink">
                <rdfs:label>dataLink</rdfs:label>
                <rdfs:subClassOf rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/
processProvenance#processProvenance"/>
        </rdfs:Class>
<!-- Presenting a profile, of web services if provided is a composite service-->
        <rdf:Property rdf:about="presents">
                <rdfs:domain rdf:resource="pd:processProvenance"/>
                <rdfs:range rdf:resource="pd:serviceInstanceProfile"/>
        </rdf:Property>
        <rdf:Property rdf:about="presentedBy">
                <rdfs:domain rdf:resource="pd:serviceInstanceProfile"/>
                <rdfs:range rdf:resource="pd:processProvenance"/>
        </rdf:Property>
        <rdf:Property rdf:about="hasServiceInstance">
                <rdfs:comment>sequence of services in the process</rdfs:comment>
                <rdfs:domain rdf:resource="pd:serviceInstanceProfile"/>
                <rdfs:range rdf:resource="pd:processProvenance"/>
        </rdf:Property>
        <rdf:Property rdf:about="hasDataLink">
                <rdfs:domain rdf:resource="pd:DataLink"/>
                <rdfs:range rdf:resource="pd:processProvenance"/>
        </rdf:Property>
```

```
          <rdf:Property rdf:about="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/processId">
                    <rdfs:domain rdf:resource="pd:processProvenance"/>
                    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
          </rdf:Property>
          <rdf:Property rdf:about="processDescription">
                    <rdfs:domain rdf:resource="pd:processProvenance"/>
                    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
          </rdf:Property>
          <rdf:Property rdf:about="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/abstractProcessLocation">
                    <rdfs:domain rdf:resource="pd:processProvenance"/>
                    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
          </rdf:Property>
          <rdf:Property rdf:about="startTime">
                    <rdfs:domain rdf:resource="pd:processProvenance"/>
                    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
          </rdf:Property>
          <rdf:Property rdf:about="endTime">
                    <rdfs:domain rdf:resource="pd:processProvenance"/>
                    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
          </rdf:Property>
          <rdf:Property rdf:about="status">
                    <rdfs:domain rdf:resource="pd:processProvenance"/>
                    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
          </rdf:Property>
          <rdf:Property rdf:about="creatorID">
                    <rdfs:domain rdf:resource="pd:processProvenance"/>
                    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
          </rdf:Property>
          <rdf:Property rdf:about="creator">
                    <rdfs:domain rdf:resource="pd:processProvenance"/>
                    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
          </rdf:Property> </rdf:RDF>
```

```
<rdf:RDF xml:base="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#" xmlns:pd="http://www.cs.cf.ac.uk/
user/S.Rajbhandari/provenance/processProvenance#" xmlns:profileHierarchy="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provena
nce/profileHierarchy#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-
schema#" xmlns:sd="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#">

<rdfs:Class rdf:about="serviceProvenance">
  <rdfs:label>serviceProvenance</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenanceprocessProvenance#serviceInstanceProfile"/
>
</rdfs:Class>
<rdfs:Class rdf:about="ServiceActivity">
     <rdfs:label>serviceActivity</rdfs:label>
     <rdfs:subClassOf rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#serviceProvenance"/
>
</rdfs:Class>
<rdfs:Class rdf:about="MessageContents">
     <rdfs:label>MessageContents</rdfs:label>
     <rdfs:subClassOf rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#serviceProvenance"/
>
</rdfs:Class>
<rdfs:Class rdf:about="DataFlow">
     <rdfs:label>DataFlow</rdfs:label>
     <rdfs:subClassOf rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#serviceProvenance"/
>
</rdfs:Class>
<rdf:Property rdf:about="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceId">
     <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#serviceProvenance"/>
     <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
```

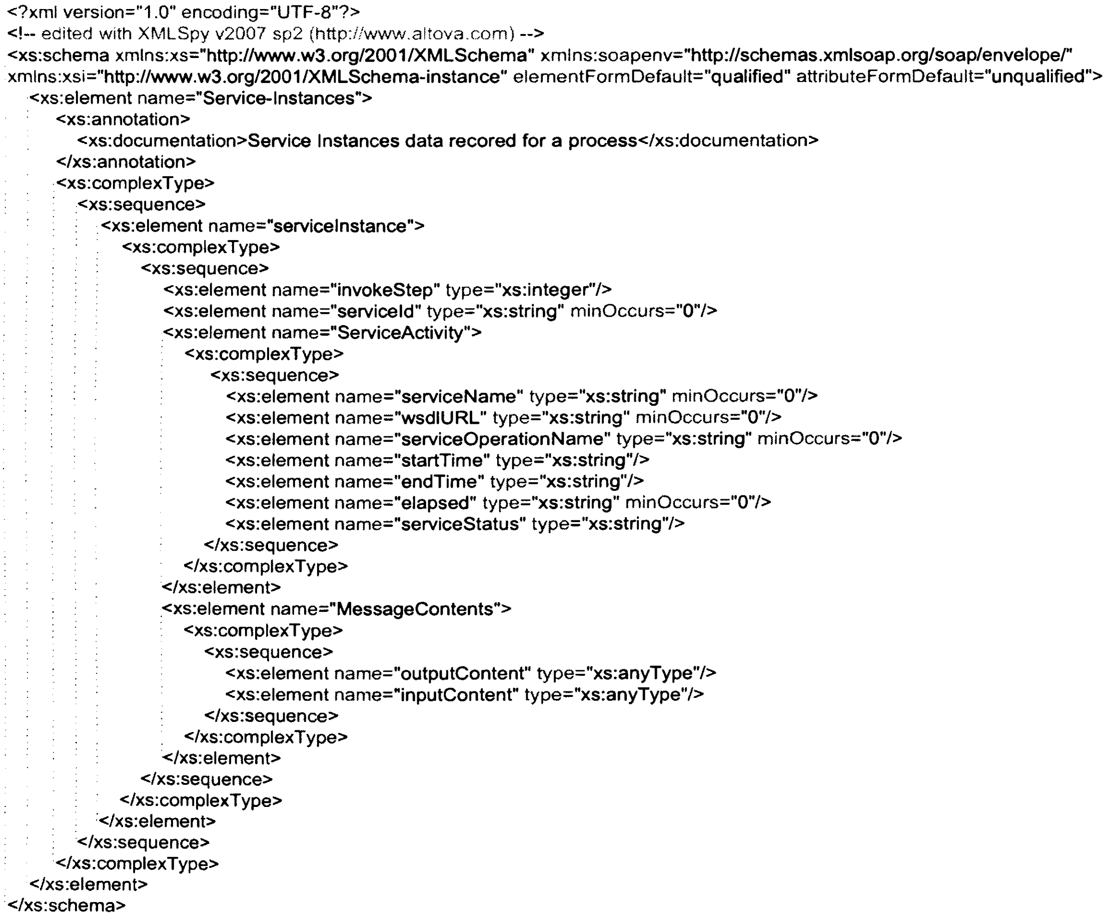
```
<!--
 Service Activity- metadata
 -->
<rdf:Property rdf:about="startTime">
        <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#ServiceActivity"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="endTime">
        <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#ServiceActivity"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="serviceName">
        <rdfs:domain rdf:resource="sd:ServiceActivity"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="serviceNamespace">
        <rdfs:domain rdf:resource="sd:ServiceActivity"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLschema.xsd#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="wsdlURL">
        <rdfs:domain rdf:resource="sd:ServiceActivity"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="serviceOperationName">
        <rdfs:domain rdf:resource="sd:ServiceActivity"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="servicePortTypeName">
        <rdfs:domain rdf:resource="sd:ServiceActivity"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="serviceStatus">
        <rdfs:domain rdf:resource="sd:ServiceActivity"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
```

```
<!--
input Dataset and output Dataset
-->
<rdfs:Property rdf:about="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/inputDataset">
        <rdfs:domain rdf:resource="sd:MessageContents"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
        <!-- <rdfs:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Seq" /> -->
</rdfs:Property>
<rdf:Property rdf:about="inputName">
        <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#inputDataset"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="inputType">
        <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#inputDataset"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="inputValue">
        <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#inputDataset"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="inputContents">
        <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#inputDataset"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdfs:Property rdf:about="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/outputDataset">
        <rdfs:domain rdf:resource="sd:MessageContents"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdfs:Property>
<rdf:Property rdf:about="outputName">
        <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#outputDataset"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="outputType">
        <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#outputDataset"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="outputValue">
        <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#outputDataset"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="outputContents">
        <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/serviceProvenance#inputDataset"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>


<!--
To provide the data flow link between the services instances in the workflow, we link the input and output data of this service
with the source and target of those data received and send to.
-->
        <rdf:Property rdf:about="http://www.cs.cf.ac.uk/user/s.rajbhandari/provenance/inputSourceIs">
                <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/processProvenance#DataLink"/>
                <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
        </rdf:Property>
        <rdf:Property rdf:about="http://www.cs.cf.ac.uk/user/s.rajbhandari/provenance/inputId">
                <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/processProvenance#DataLink"/>
                <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
        </rdf:Property>
        <rdf:Property rdf:about="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/outputTargetIs">
                <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/processProvenance#DataLink"/>
                <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
        </rdf:Property>
        <rdf:Property rdf:about="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/outputId">
                <rdfs:domain rdf:resource="http://www.cs.cf.ac.uk/user/S.Rajbhandari/provenance/processProvenance#DataLink"/>
                <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
        </rdf:Property>
</rdf:RDF>
```

Appendix B

service-Provenance XML Schema



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2007 sp2 (http://www.altova.com) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="Service-Instances">
    <xs:annotation>
      <xs:documentation>Service Instances data recored for a process</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="serviceInstance">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="invokeStep" type="xs:integer"/>
              <xs:element name="serviceId" type="xs:string" minOccurs="0"/>
              <xs:element name="ServiceActivity">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="serviceName" type="xs:string" minOccurs="0"/>
                    <xs:element name="wsdlURL" type="xs:string" minOccurs="0"/>
                    <xs:element name="serviceOperationName" type="xs:string" minOccurs="0"/>
                    <xs:element name="startTime" type="xs:string"/>
                    <xs:element name="endTime" type="xs:string"/>
                    <xs:element name="elapsed" type="xs:string" minOccurs="0"/>
                    <xs:element name="serviceStatus" type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="MessageContents">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="outputContent" type="xs:anyType"/>
                    <xs:element name="inputContent" type="xs:anyType"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Appendix C

BPEL Workflow

This Appendix shows the example workflow constructed using BPEL.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
BPEL Process Definition
Edited using ActiveBPEL(tm) Designer Version 2.1.0 (http://www.active-endpoints.com)
-->
<!-- FFT convolution WorkFlow example - generated with the help of oracle bpel designer -->
<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/" xmlns:DCService="http://131.251.49.136:8080/
axis/services/DustCloud" xmlns:MyFFT="http://131.251.49.136:8080/axis/services/MyFFT" xmlns:PadZero="http://
131.251.49.136:8080/axis/services/PadZero" xmlns:PowerOfTwo="http://131.251.49.136:8080/axis/services/PowerOfTwo"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/" xmlns:convolve="http://131.251.49.136:8080/axis/
services/convolve" xmlns:ora="http://schemas.oracle.com/xpath/extension" xmlns:telescopeData="http://
131.251.49.136:8080/axis/services/telescopeData" xmlns:tns="http://signal.org/wsdl/MyClient-Test" xmlns:xsd="http://
www.w3.org/2001/XMLSchema" name="MySignalProcess" suppressJoinFailure="yes" targetNamespace="http://signal.org/
wsdl/MySignalProcessing">
  <partnerLinks>
    <!-- The 'client' role represents the requester of this service. -->
    <partnerLink myRole="MyProcessProvider" name="Client" partnerLinkType="tns:ClientLink"/>
    <partnerLink name="DustCloudService" partnerLinkType="tns:DustCloudServiceLink"
partnerRole="DustCloudServiceProvider"/>
    <partnerLink name="TelescopeData" partnerLinkType="tns:telescopeDataLink" partnerRole="telescopeDataProvider"/>
    <partnerLink name="PowerOfTwo" partnerLinkType="tns:PowerOf2Link" partnerRole="PowerOf2Provider"/>
    <partnerLink name="PadWithZero" partnerLinkType="tns:padZeroLink" partnerRole="padZeroProvider"/>
    <partnerLink name="FFTService" partnerLinkType="tns:MyFFTLink" partnerRole="MyFFTProvider"/>
    <partnerLink name="ConvolveService" partnerLinkType="tns:convolveLink" partnerRole="convolveProvider"/>
  </partnerLinks>
```

```
 <variables>
    <!-- Reference to the message that will be returned to the requester -->
    <variable messageType="tns:ClientRequest" name="userRequest"/>
    <variable messageType="DCService:yValuesRequest" name="inputDC"/>
    <variable messageType="DCService:yValuesResponse" name="outputDC"/>
    <variable messageType="telescopeData:telDataRequest" name="inputTEL"/>
    <variable messageType="telescopeData:telDataResponse" name="outputTEL"/>
    <variable messageType="PowerOfTwo:pointsRequest" name="inputPOTwo"/>
    <variable messageType="PowerOfTwo:pointsResponse" name="outputPOTwo"/>
    <variable messageType="PadZero:zeroPadedDataRequest" name="inputZeropad1"/>
    <variable messageType="PadZero:zeroPadedDataResponse" name="outputZeropad1"/>
    <variable messageType="PadZero:zeroPadedDataRequest" name="inputZeropad2"/>
    <variable messageType="PadZero:zeroPadedDataResponse" name="outputZeropad2"/>
    <variable messageType="MyFFT:realFTRequest" name="inputFFT1"/>
    <variable messageType="MyFFT:realFTResponse" name="outputFFT1"/>
    <variable messageType="MyFFT:realFTRequest" name="inputFFT2"/>
    <variable messageType="MyFFT:realFTResponse" name="outputFFT2"/>
    <variable messageType="convolve:ConvRequest" name="inputConvolve"/>
    <variable messageType="convolve:ConvResponse" name="outputConvolve"/>
    <variable messageType="MyFFT:realFTRequest" name="inputFFT3"/>
    <variable messageType="MyFFT:realFTResponse" name="outputFFT3"/>
    <variable messageType="tns:ClientResponse" name="userResponse"/>
  </variables>

<!-- ================================================================ -->
  <!-- ORCHESTRATION LOGIC                              -->
  <!-- Set of activities coordinating the flow of messages across the  -->
  <!-- services integrated within this business process               -->
  <!-- ================================================================ -->
  <!-- Start of main sequence                              -->
  <sequence name="PR1">
    <!-- Receive input from requester. -->
    <receive createInstance="yes" name="receiveRequest" operation="runProcess" partnerLink="Client"
portType="tns:ProcessClientPT" variable="userRequest"/>
    <!-- START OF FLOW-1 === DUST CLOUD AND TELESCOPE -->
    <flow>
      <sequence name="S1-DustCloud">
        <assign name="DCInput1-DensityType">
          <copy>
            <from part="densityDC" variable="userRequest"/>
            <to part="densityType" variable="inputDC"/>
          </copy>
        </assign>
        <assign name="DCInput2-Width">
          <copy>
            <from part="widthDC" variable="userRequest"/>
            <to part="widthParameter" variable="inputDC"/>
          </copy>
        </assign>
        <assign name="DCInput3-NumOfPoints">
          <copy>
            <from part="pointsDC" variable="userRequest"/>
            <to part="n" variable="inputDC"/>
          </copy>
        </assign>
        <invoke inputVariable="inputDC" name="invoke-DustCloudService" operation="yValues" outputVariable="outputDC"
partnerLink="DustCloudService" portType="DCService:DustCloudService"/>
      </sequence>
 <sequence name="S2-Telescope">
        <assign name="SIGInput1-WaveType">
          <copy>
            <from part="waveType" variable="userRequest"/>
            <to part="Type" variable="inputTEL"/>
          </copy>
        </assign>
        <assign name="SIGInput2-NumOfPoints">
          <copy>
            <from part="points" variable="userRequest"/>
            <to part="n" variable="inputTEL"/>
          </copy>
        </assign>
        <invoke inputVariable="inputTEL" name="invoke-TelescopeService" operation="telData" outputVariable="outputTEL"
partnerLink="TelescopeData" portType="telescopeData:telescopeData"/>
      </sequence>
    </flow>
<!-- END OF FLOW-1 -->
<!-- COPY OUTPUT FROM FLOW 1== TO== POWER OF TWO CODE -->
    <assign name="assignData1-OutputDustCloud-to-PowerOfTwo">
      <copy>
        <from part="yValuesReturn" variable="outputDC"/>
        <to part="d1" variable="inputPOTwo"/>
      </copy>
    </assign>
    <assign name="assignData2-OutputTelescope-to-PowerOfTwo">
      <copy>
        <from part="telDataReturn" variable="outputTEL"/>
        <to part="d2" variable="inputPOTwo"/>
      </copy>
    </assign>
    <invoke inputVariable="inputPOTwo" name="invokePowerOfTwoService" operation="points"
outputVariable="outputPOTwo" partnerLink="PowerOfTwo" portType="PowerOfTwo:PowerOf2"/>
    <!-- COPY OUTPUT (INT VALUE) TO== ZEROPAD WS        -->
    <!-- START FLOW-2 == PAD WITH ZERO AND FFT THE OUTPUT -->
    <flow name="flow-2">
      <!-- SEQUENCE-1 -->
      <sequence name="S3-FFTService-seq1">
        <assign name="OutputPOT-To-ZEROPAD1">
          <copy>
            <from part="pointsReturn" variable="outputPOTwo"/>
            <to part="points" variable="inputZeropad1"/>
          </copy>
        </assign>
        <assign name="OutputDustCloud-To-ZP1">
          <copy>
            <from part="yValuesReturn" variable="outputDC"/>
            <to part="d" variable="inputZeropad1"/>
          </copy>
        </assign>
        <invoke inputVariable="inputZeropad1" name="invoke-ZeropadService1" operation="zeroPadedData"
outputVariable="outputZeropad1" partnerLink="PadWithZero" portType="PadZero:padZero"/>
        <assign name="assign-OutputZP1-To-FFT1">
          <copy>
            <from part="zeroPadedDataReturn" variable="outputZeropad1"/>
            <to part="ydata" variable="inputFFT1"/>
          </copy>
        </assign>
        <assign name="FFTisign1-from-expression">
          <copy>
            <from expression="1"/>
            <to part="isign" variable="inputFFT1"/>
          </copy>
        </assign>
        <invoke inputVariable="inputFFT1" name="invoke-FFTService1" operation="realFT" outputVariable="outputFFT1"
partnerLink="FFTService" portType="MyFFT:MyFFT"/>
      </sequence>
  <!-- SEQUENCE-2 -->
      <sequence name="S4-FFTService-seq2">
        <assign name="OutputPOT-To-ZEROPAD2">
          <copy>
            <from part="pointsReturn" variable="outputPOTwo"/>
            <to part="points" variable="inputZeropad2"/>
          </copy>
        </assign>
        <assign name="OutputTelescope-To-ZP2">
          <copy>
            <from part="telDataReturn" variable="outputTEL"/>
            <to part="d" variable="inputZeropad2"/>
          </copy>
        </assign>
        <invoke inputVariable="inputZeropad2" name="invoke-ZeropadService2" operation="zeroPadedData"
outputVariable="outputZeropad2" partnerLink="PadWithZero" portType="PadZero:padZero"/>
        <assign name="assign-OutputZP2-To-FFT2">
          <copy>
            <from part="zeroPadedDataReturn" variable="outputZeropad2"/>
            <to part="ydata" variable="inputFFT2"/>
          </copy>
        </assign>
        <assign name="FFTisign2-from-expression">
          <copy>
            <from expression="1"/>
            <to part="isign" variable="inputFFT2"/>
          </copy>
        </assign>
        <invoke inputVariable="inputFFT2" name="invoke-FFTService2" operation="realFT" outputVariable="outputFFT2"
partnerLink="FFTService" portType="MyFFT:MyFFT"/>
      </sequence>
    </flow>
    <!-- END OF FLOW-2 -->

<assign name="FFT1ToConvolve">
      <copy>
        <from part="realFTReturn" variable="outputFFT1"/>
        <to part="fft1" variable="inputConvolve"/>
      </copy>
    </assign>
    <assign name="FFT2ToConvolve">
      <copy>
        <from part="realFTReturn" variable="outputFFT2"/>
        <to part="fft2" variable="inputConvolve"/>
      </copy>
    </assign>
    <invoke inputVariable="inputConvolve" name="invoke-ConvolveService" operation="Conv"
outputVariable="outputConvolve" partnerLink="ConvolveService" portType="convolve:convolve"/>
    <assign name="ConvolveToInvFFT">
      <copy>
        <from part="ConvReturn" variable="outputConvolve"/>
        <to part="ydata" variable="inputFFT3"/>
      </copy>
    </assign>
    <assign name="isignToInvFFT">
      <copy>
        <from expression="-1"/>
        <to part="isign" variable="inputFFT3"/>
      </copy>
    </assign>
    <invoke inputVariable="inputFFT3" name="invoke-InverseFFTService" operation="realFT" outputVariable="outputFFT3"
partnerLink="FFTService" portType="MyFFT:MyFFT"/>
    <!-- map the final output to the output the user expects -->
    <assign name="finalToClient">
      <copy>
        <from part="realFTReturn" variable="outputFFT3"/>
        <to part="finalReturn" variable="userResponse"/>
      </copy>
    </assign>
    <!-- respond to the user -->
    <reply name="replyResponse" operation="runProcess" partnerLink="Client" portType="tns:ProcessClientPT"
variable="userResponse"/>
  </sequence>
</process>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
    <wsdl:definitions targetNamespace="http://131.125.49.1368080/axis/services/DustCloud"
    xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://131.125.49.136:8080/axis/services/DustCloud"
    xmlns:intf="http://131.125.49.136:8080/axis/services/DustCloud"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types />
     <wsdl:message name="yValuesResponse">
       <wsdl:part name="yValuesReturn" type="xsd:string" />
         </wsdl:message>
          <wsdl:message name="yValuesRequest">
       <wsdl:part name="densityType" type="xsd:string" />
       <wsdl:part name="widthParameter" type="xsd:double" />
       <wsdl:part name="n" type="xsd:int" />
     </wsdl:message>
          <wsdl:portType name="DustCloudService">
          <wsdl:operation name="yValues" parameterOrder="densityType widthParameter n">
          <wsdl:input message="impl:yValuesRequest" name="yValuesRequest" />
          <wsdl:output message="impl:yValuesResponse" name="yValuesResponse" />
          </wsdl:operation>
          </wsdl:portType>
          <wsdl:binding name="DustCloudSoapBinding" type="impl:DustCloudService">
          <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
          <wsdl:operation name="yValues">
       <wsdlsoap:operation soapAction="" />
     <wsdl:input name="yValuesRequest">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://131.125.49.136:8080/axis/services/DustCloud" use="encoded" />
         </wsdl:input>
       <wsdl:output name="yValuesResponse">
       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="http://131.125.49.136:8080/axis/services/DustCloud" use="encoded" />
         </wsdl:output>
         </wsdl:operation>
         </wsdl:binding>
       <wsdl:service name="DustCloudServiceService">
       <wsdl:port binding="impl:DustCloudSoapBinding" name="DustCloud">
       <wsdlsoap:address location="http://131.125.49.136:8080/axis/services/DustCloud" />
     </wsdl:port>
     </wsdl:service>
  </wsdl:definitions>
```

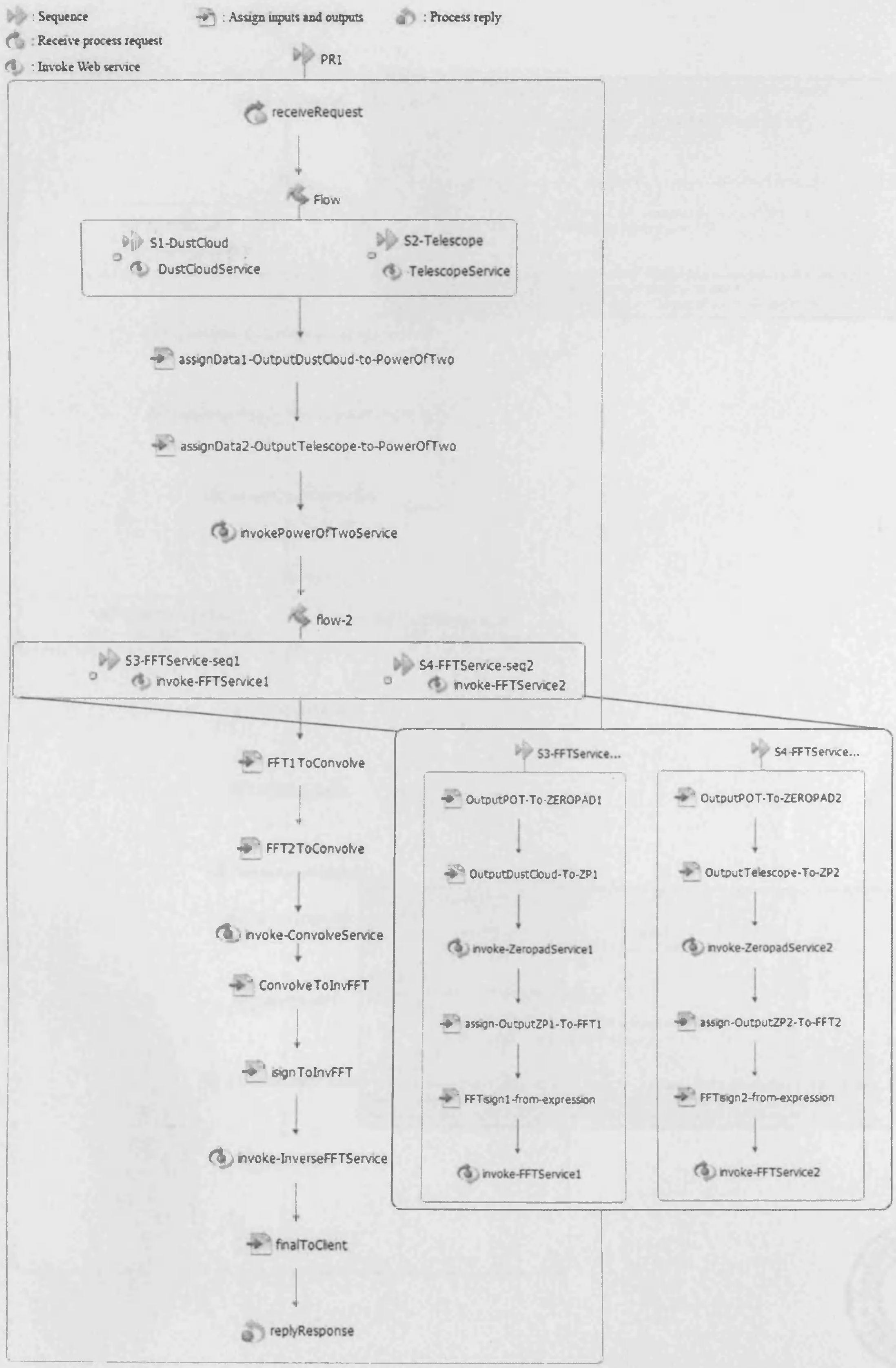
```
<?xml version="1.0" encoding="UTF-8" ?>
   <wsdl:definitions targetNamespace="http://131.125.49.136:8080/axis/services/telescopeData"
   xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://131.125.49.136:8080/axis/services/telescopeData"
   xmlns:intf="http://131.125.49.136:8080/axis/services/telescopeData"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <wsdl:types />
   <wsdl:message name="telDataResponse">
     <wsdl:part name="telDataReturn" type="xsd:string" />
    </wsdl:message>
    <wsdl:message name="telDataRequest">
     <wsdl:part name="n" type="xsd:int" />
     <wsdl:part name="Type" type="xsd:string" />
    </wsdl:message>
    <wsdl:portType name="telescopeData">
      <wsdl:operation name="telData" parameterOrder="n Type">
        <wsdl:input message="impl:telDataRequest" name="telDataRequest" />
        <wsdl:output message="impl:telDataResponse" name="telDataResponse" />
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="telescopeDataSoapBinding" type="impl:telescopeData">
     <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
       <wsdl:operation name="telData">
       <wsdlsoap:operation soapAction="" />
       <wsdl:input name="telDataRequest">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   namespace="http://131.125.49.136:8080/axis/services/telescopeData" use="encoded" />
      </wsdl:input>
       <wsdl:output name="telDataResponse">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   namespace="http://131.125.49.136:8080/axis/services/telescopeData" use="encoded" />
      </wsdl:output>
     </wsdl:operation>
    </wsdl:binding>
   <wsdl:service name="telescopeDataService">
    <wsdl:port binding="impl:telescopeDataSoapBinding" name="telescopeData">
    <wsdlsoap:address location="http://131.125.49.136:8080/axis/services/telescopeData" />
    </wsdl:port>
   </wsdl:service>
   </wsdl:definitions>
```

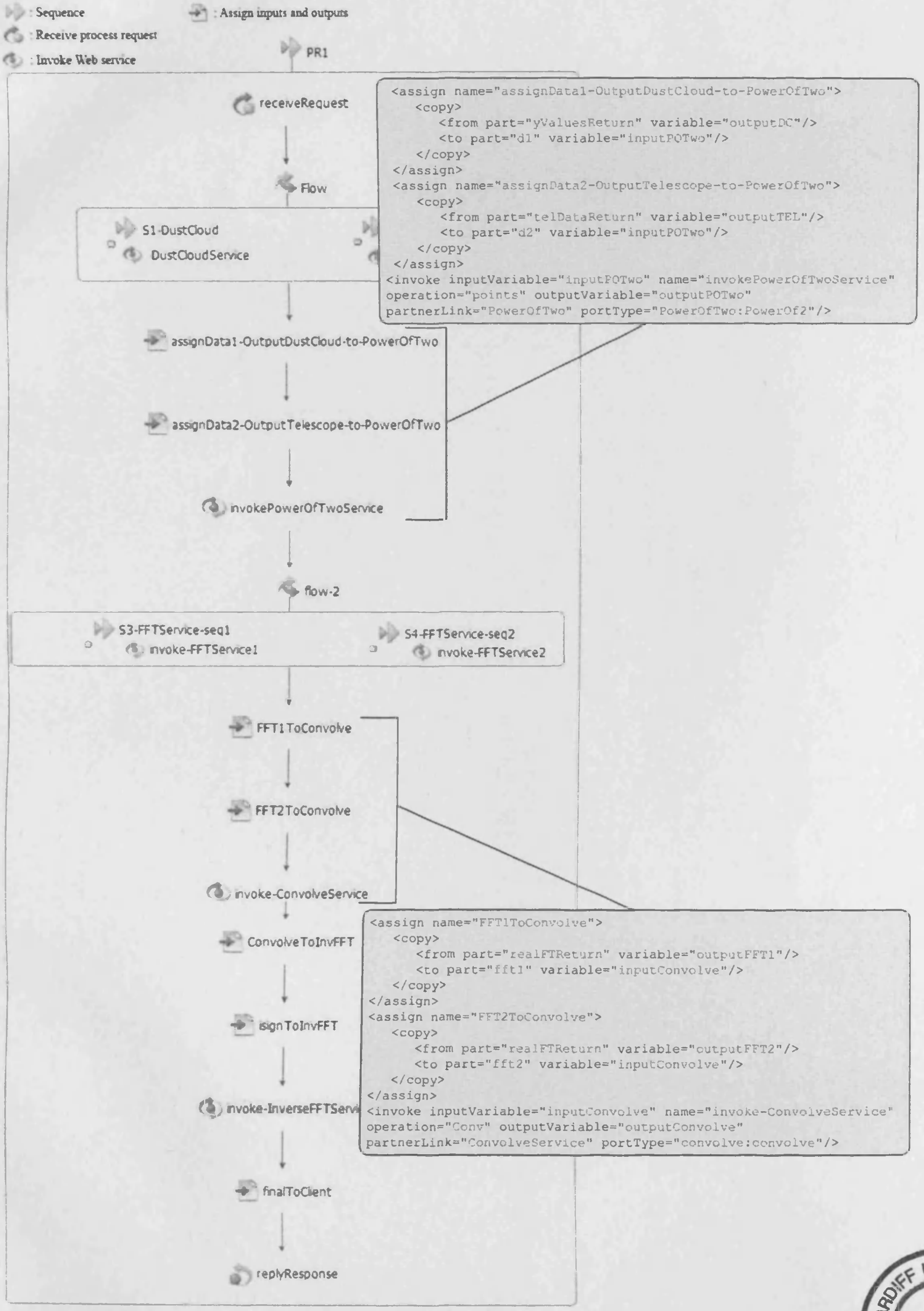
```
<?xml version="1.0" encoding="UTF-8" ?>
  <wsdl:definitions targetNamespace="http://131.125.49.136:8080/axis/services/PowerOfTwo"
  xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://131.125.49.136:8080/axis/services/PowerOfTwo"
  xmlns:intf="http://131.125.49.136:8080/axis/services/PowerOfTwo"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types />
    <wsdl:message name="pointsResponse">
      <wsdl:part name="pointsReturn" type="xsd:int" />
      </wsdl:message>
    <wsdl:message name="pointsRequest">
      <wsdl:part name="d1" type="xsd:string" />
      <wsdl:part name="d2" type="xsd:string" />
    </wsdl:message>
    <wsdl:portType name="PowerOf2">
    <wsdl:operation name="points" parameterOrder="d1 d2">
      <wsdl:input message="impl:pointsRequest" name="pointsRequest" />
      <wsdl:output message="impl:pointsResponse" name="pointsResponse" />
    </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="PowerOfTwoSoapBinding" type="impl:PowerOf2">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
      <wsdl:operation name="points"><wsdlsoap:operation soapAction="" />
        <wsdl:input name="pointsRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://131.125.49.136:8080/axis/services/PowerOfTwo" use="encoded" />
        </wsdl:input>
        <wsdl:output name="pointsResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://131.125.49.136:8080/axis/services/PowerOfTwo" use="encoded" />
        </wsdl:output>
     </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="PowerOf2Service">
     <wsdl:port binding="impl:PowerOfTwoSoapBinding" name="PowerOfTwo">
      <wsdlsoap:address location="http://131.125.49.136:8080/axis/services/PowerOfTwo" />
    </wsdl:port>
    </wsdl:service>
   </wsdl:definitions>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
    <wsdl:definitions targetNamespace="http://131.125.49.136:8080/axis/services/PadZero"
    xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://131.125.49.136:8080/axis/services/PadZero"
    xmlns:intf="http://131.125.49.136:8080/axis/services/PadZero"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <wsdl:types />
        <wsdl:message name="zeroPadedDataRequest">
           <wsdl:part name="d" type="xsd:string" />
           <wsdl:part name="points" type="xsd:int" />
        </wsdl:message>
        <wsdl:message name="zeroPadedDataResponse">
           <wsdl:part name="zeroPadedDataReturn" type="xsd:string" />
        </wsdl:message>
        <wsdl:portType name="padZero">
        <wsdl:operation name="zeroPadedData" parameterOrder="d points">
       <wsdl:input message="impl:zeroPadedDataRequest" name="zeroPadedDataRequest" />
       <wsdl:output message="impl:zeroPadedDataResponse" name="zeroPadedDataResponse" />
          </wsdl:operation>
          </wsdl:portType>
      <wsdl:binding name="PadZeroSoapBinding" type="impl:padZero">
       <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
     <wsdl:operation name="zeroPadedData">
     <wsdlsoap:operation soapAction="" />
       <wsdl:input name="zeroPadedDataRequest">
       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://131.125.49.136:8080/axis/services/PadZero" use="encoded" />
       </wsdl:input>
       <wsdl:output name="zeroPadedDataResponse">
       <wsdlsoap:body encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
          namespace="http://131.125.49.136:8080/axis/services/PadZero" use="encoded"/>
       </wsdl:output>
     </wsdl:operation>
     </wsdl:binding>
     <wsdl:service name="padZeroService">
       <wsdl:port binding="impl:PadZeroSoapBinding" name="PadZero">
          <wsdlsoap:address location="http://131.125.49.136:8080/axis/services/PadZero" />
       </wsdl:port>
     </wsdl:service>
    </wsdl:definitions>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
    <wsdl:definitions targetNamespace="http://localhost:8080/axis/services/MyFFT"
    xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://localhost:8080/axis/services/MyFFT"
    xmlns:intf="http://localhost:8080/axis/services/MyFFT"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <wsdl:types />
        <wsdl:message name="realFTResponse">
          <wsdl:part name="realFTReturn" type="xsd:string" />
        </wsdl:message>
        <wsdl:message name="realFTRequest">
          <wsdl:part name="isign" type="xsd:int" />
          <wsdl:part name="ydata" type="xsd:string" />
        </wsdl:message>
        <wsdl:portType name="MyFFT">
         <wsdl:operation name="realFT" parameterOrder="isign ydata">
           <wsdl:input message="impl:realFTRequest" name="realFTRequest" />
           <wsdl:output message="impl:realFTResponse" name="realFTResponse" />
        </wsdl:operation>
        </wsdl:portType>
        <wsdl:binding name="MyFFTSoapBinding" type="impl:MyFFT">
           <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="realFT"><wsdlsoap:operation soapAction="" />
            <wsdl:input name="realFTRequest">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://localhost:8080/axis/services/MyFFT" use="encoded" />
          </wsdl:input>
        <wsdl:output name="realFTResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://localhost:8080/axis/services/MyFFT" use="encoded" />
         </wsdl:output>
        </wsdl:operation>
        </wsdl:binding>
      <wsdl:service name="MyFFTService">
       <wsdl:port binding="impl:MyFFTSoapBinding" name="MyFFT">
         <wsdlsoap:address location="http://localhost:8080/axis/services/MyFFT" />
       </wsdl:port>
     </wsdl:service>
    </wsdl:definitions>
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
    <wsdl:definitions targetNamespace="http://131.125.49.136:8080/axis/services/convolve"
    xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://131.125.49.136:8080/axis/services/convolve"
    xmlns:intf="http://131.125.49.136:8080/axis/services/convolve"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsdl:types />
    <wsdl:message name="ConvRequest">
       <wsdl:part name="fft1" type="xsd:string" />
       <wsdl:part name="fft2" type="xsd:string" />
     </wsdl:message>
     <wsdl:message name="ConvResponse">
       <wsdl:part name="ConvReturn" type="xsd:string" />
     </wsdl:message>
     <wsdl:portType name="convolve">
       <wsdl:operation name="Conv" parameterOrder="fft1 fft2">
         <wsdl:input message="impl:ConvRequest" name="ConvRequest" />
         <wsdl:output message="impl:ConvResponse" name="ConvResponse" />
       </wsdl:operation>
     </wsdl:portType>
     <wsdl:binding name="convolveSoapBinding" type="impl:convolve">
       <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
           <wsdl:operation name="Conv">
       <wsdlsoap:operation soapAction="" />
     <wsdl:input name="ConvRequest">
       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://131.125.49.136:8080/axis/services/convolve" use="encoded" />
          </wsdl:input>
     <wsdl:output name="ConvResponse">
       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://131.125.49.136:8080/axis/services/convolve" use="encoded" />
          </wsdl:output>
          </wsdl:operation>
          </wsdl:binding>
      <wsdl:service name="convolveService">
       <wsdl:port binding="impl:convolveSoapBinding" name="convolve">
        <wsdlsoap:address location="http://131.125.49.136:8080/axis/services/convolve" />
       </wsdl:port>
       </wsdl:service>
    </wsdl:definitions>
```


: Sequence
: Assign inputs and outputs
: Process reply
: Receive process request
: Invoke Web service
PR1
receiveRequest
Flow
S1-DustCloud
DustCloudService
S2-Telescope
TelescopeService
assignData1-OutputDustCloud-to-PowerOfTwo
assignData2-OutputTelescope-to-PowerOfTwo
invokePowerOfTwoService
flow-2
S3-FFTService-seq1
invoke-FFTService1
S4-FFTService-seq2
invoke-FFTService2
FFT1ToConvolve
FFT2ToConvolve
invoke-ConvolveService
ConvolveToInvFFT
isignToInvFFT
invoke-InverseFFTService
finalToClient
replyResponse
S3-FFTService...
OutputPOT-To-ZEROPAD1
OutputDustCloud-To-ZP1
invoke-ZeropadService1
assign-OutputZP1-To-FFT1
FFTsign1-from-expression
invoke-FFTService1
S4-FFTService...
OutputPOT-To-ZEROPAD2
OutputTelescope-To-ZP2
invoke-ZeropadService2
assign-OutputZP2-To-FFT2
FFTsign2-from-expression
invoke-FFTService2



```
<assign name="assignData1-OutputDustCloud-to-PowerOfTwo">
   <copy>
      <from part="yValuesReturn" variable="outputDC"/>
      <to part="d1" variable="inputPOTwo"/>
   </copy>
</assign>
<assign name="assignData2-OutputTelescope-to-PowerOfTwo">
   <copy>
      <from part="telDataReturn" variable="outputTEL"/>
      <to part="d2" variable="inputPOTwo"/>
   </copy>
</assign>
<invoke inputVariable="inputPOTwo" name="invokePowerOfTwoService"
operation="points" outputVariable="outputPOTwo"
partnerLink="PowerOfTwo" portType="PowerOfTwo:PowerOf2"/>
```

```
<assign name="FFT1ToConvolve">
   <copy>
      <from part="realFTReturn" variable="outputFFT1"/>
      <to part="fft1" variable="inputConvolve"/>
   </copy>
</assign>
<assign name="FFT2ToConvolve">
   <copy>
      <from part="realFTReturn" variable="outputFFT2"/>
      <to part="fft2" variable="inputConvolve"/>
   </copy>
</assign>
<invoke inputVariable="inputConvolve" name="invoke-ConvolveService"
operation="Conv" outputVariable="outputConvolve"
partnerLink="ConvolveService" portType="convolve:convolve"/>
```