The Exploitation of Provenance and Versioning in the Reproduction of e-Experiments

Thesis by
Dayang Hanani Abang Ibrahim

A thesis submitted for the degree of

Doctor of Philosophy

Newcastle University
Newcastle Upon Tyne, UK

April 2016

Dedication:

Specially dedicated to My husband; Zamsury, My children; Ameera Nailya, Adlyna Nailya, Muhammad Arief Naufal and Ayeesha Nailya, My parents and in laws; Abang Ibrahim and Khadijah, Kushaili and Normah. Thank you for all your patience, continuous support and encouragement throughout the years. Thank you very much!

Acknowledgements

In the name of Allah, the Most Gracious and the Most Merciful. First and foremost, Alhamdulillah, all praises to Allah for the strengths and His blessing in completing this thesis.

This thesis would not have been possible without the help and guidance of several individuals who in one way or another contributed in the preparation and completion of this study.

My gratitude goes to my supervisor Professor Paul Watson for his constant support, valuable guidance, insights and encouragement throughout the course of research and writing of this thesis. His sincerity, patience and understanding on numerous occasions when I needed to hurdle all the obstacles in the completion this research work. "Nearly there", a phrase that I will never forget, such motivating words to keep me going.

I would like to thank Professor Michael Harrison for his valuable suggestions and insights, his willingness to share thoughts which were helpful during my years in Newcastle upon Tyne.

I would like to thank my employer, the Universiti Malaysia Sarawak (UNIMAS) and the Ministry of Higher Education Malaysia for being my sponsor and supporter throughout my PhD study.

I cannot forget the amount of support I received from my family during this entire journey. My husband, Zamsury Kushaili and my four children Ameera Nailya Zamsury, Adlyna Nailya Zamsury, Muhammad Arief Naufal Zamsury and Ayeesha Nailya Zamsury who continuously support me in every step in my life. You have all been my source of joy.

I can never express enough gratitude for the support I received from my parents, Abang Ibrahim Abang Junaidi and Khadijah Ibrahim, who have

constantly encouraged me, at times pushed me - when I used to get frustrated. Thanks also to my parents in law, Kushaili Su'ut and Normah Kadir, my siblings and all family members for their encouragement and support.

I am indebted to my close friends, Dyg Mariana, Dr. Johari, Dr. Rouaa Yassin, Dr. Chiew Kang Leng, Dr. Nadianatra, Dr. Halikul and all my dearest friends for their valuable assistance in the completion of this study. Thank you for being supportive and caring friends.

I owe my gratitude to the Management of the Faculty of Computer Science and Information Technology for their continuous support and encouragement.

Last but not least, thanks to all individuals who have directly and indirectly given me the possibility to complete the thesis.

Abstract

Reproducibility has long been a cornerstone of science, and is now becoming a key research area for e-Science. This is because it provides a way to validate, and build on, previous results. Underpinning reproducibility in e-Science is provenance, which has the potential to provide scientists with a complete understanding of data generated in e-experiments, including the services that produced and consumed it. This thesis explores the issues in exploiting provenance for reproducibility. Based on this, a reproducibility framework is designed and implemented to allow past experiments to be reproduced. Seven aspects of reproducibility are considered: 1) experiments, 2) reproducibility, 3) provenance, 4) provenance models, 5) provenance and versioning, 6) automatic transformation of provenance to support reproduction, and 7) a reproducibility taxonomy. A key to reproducibility is the provenance model: a data model that structures information about an e-experiment. A review of existing provenance systems shows that the problem caused by services being updated has been neglected. This can have a severe impact on the ability to reproduce experiments and it is therefore argued that the issue of service versioning must be addressed. Even after information on the provenance of an execution, and versioning of services, is captured there is the need for a method to transform this knowledge into a form that allows past experiments to be reproduced: that is another output of this thesis. The thesis focuses on the use of workflow as a means to represent the composition, and to execute experiments. This work explores how workflows can be automatically generated to re-execute past experiments. In order to do this, a transformation algorithm is described that maps a past experiment's execution log data into a workflow format that can be read and processed by the workflow system. The thesis also introduces a Reproducibility Taxonomy that captures and structures the information required for reproducibility in the presence of versions and provenance.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

e-Science is a term that describes scientists working collaboratively, often in large distributed project teams, to solve scientific problems using computers. Throughout their work, they are typically dealing with large datasets, accessing scientific literature, analysing and reanalysing their findings and discussing the results within the group. According to the European Bio-Informatics Institute (EBI),

> *"e-Science is about exploiting digital technology to support all aspects of scientific activity by the development of a communication and computational infrastructure to underpin the work of scientists."[8]*

In an e-Science environment, scientists such as bio-informaticians, chemists and physicians are working with open and large-scale distributed systems, thus they need to access various databases and use computational methods to interact with their own and others' programs. As an example, one of the first UK e-Science pilot projects, myGrid [9], and its successor projects, delivered a collaborative environment that makes information and tools available to scientists.

The myGrid project provides middleware to help the scientists to develop, manage and share their research artifacts among different institutions and communities. The project has developed Taverna, a workflow system that provides scientists with a way to create, generate, manage and share their experiments. This practice can improve the processes involved in conducting experiments, and therefore help in solving scientific problems.

This thesis focuses on the reproducibility of e-Science experiments, which requires us to consider the seven aspects shown in Figure 1.1: 1) experiments, 2) reproducibility,

3) provenance, 4) provenance models, 5) provenance and versioning, 6) automatic transformation and reproduction, and 7) a taxonomy of reproducibility. If reproducibility is to be achieved then these aspects need to be considered as a unified way. To achieve this, this thesis considers them in the order shown in Figure 1.1. A brief overview of the reasons for including each aspect is now given.

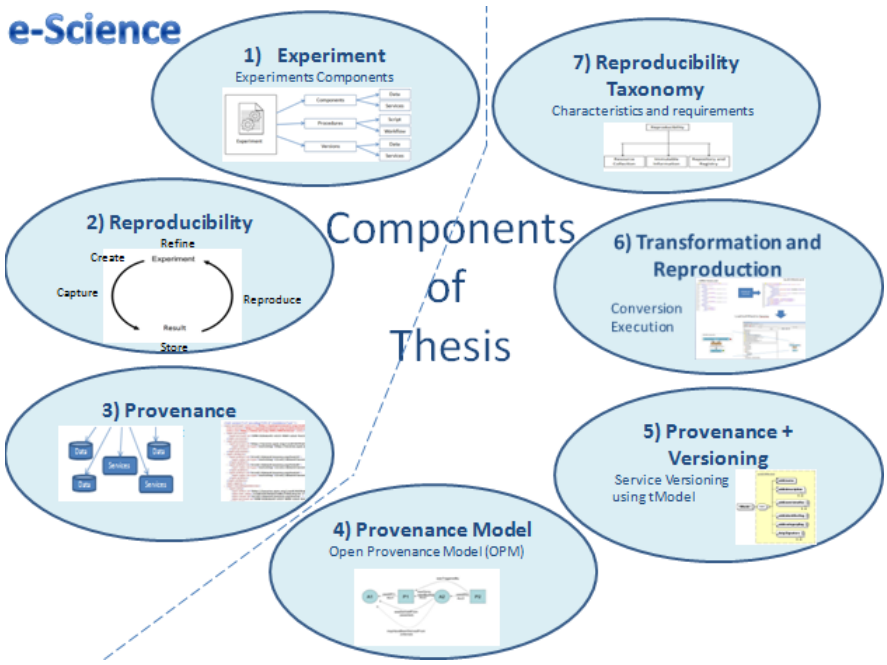


Figure 1.1: The components addressed in this thesis

In e-Science, experiments are conducted to test a hypothesis. For scientists, the results of experiments are their primary interest. This raises two key questions: How to interpret experimental results and how to reproduce experiments in order to explore existing and new concepts from past experiments. Experimental reproducibility is concerned with being able to re-execute past experiments in a different workflow environment and to see if a prior result can be confirmed. This is because it is not guaranteed that past experiments can be re-executed successfully if the experiments were created in a different workflow environment. This maybe due to a different workflow structural differences and missing data, services or processes. To reproduce experiments, the original experimental entities must be accessible. To achieve this, reproducibility requires

provenance information that captures all the important entities in an experiment. For this to be successful, the entities must be described by a provenance model. A major issue is that the experimental entities may be changed from time to time: for example new versions of services used in an experiment may be deployed. Therefore, in this thesis we argue that versioning is an essential mechanism needed to support experimental reproducibility. Even once execution provenance and entity versioning have been addressed, another key component is a technique to actually transform this information in a way that allows the reproduction of past experiments. Workflow is widely used in e-Science as a means to represent the composition of experiments, and as a way to execute them. This work focuses on e-experiments based on workflows and considers how to automatically generate workflows using provenance and versioning information in order to re-execute past experiments. In order to achieve this, a transformation algorithm is needed to map the past experiment's execution log into a workflow format, taking into account versioning information.

As these aspects are described in more detail throughout the thesis, it will become clear that each has specific linkages to other aspects. However, in order to achieve the reproducibility of e-experiments, three key fundamental concepts emerge and are used throughout this thesis:

- Experiments
- Reproducibility
- Provenance

In Figure 1.1, these three concepts are separated by dashed lines from the rest of the components which are the main focus of the thesis. Overall, the key results from the thesis are:

- Conceptualisation of an experiment using provenance data model descriptions that are able to completely describe an experiment and so support reproducibility.
- Incorporation of service versioning into provenance. Mechanisms to support service versioning have been neglected in previous work. Therefore, the thesis will explore how service versioning can be introduced into provenance in order to be able to reproduce experiments involving services that may be updated over time.

- Design of a technique to allow the reproduction of past experiments.
- Creation of a reproducibility taxonomy that identifies the requirements for facilitating the reproduction of experiments.

1.1 Motivation

e-Science experiments deal with computations over data, therefore in reproducing experiments, it is the computations, and not just the data that are important. The three elements identified to represent an experiment: components, procedures and versions are shown in Figure 1.2. The components of an e-Science experiment deal with data and services. Data can be sets of independent and dependent variables. Independent variables are input datasets, and parameters to the experiment. These can be retrieved from database records, spreadsheets or as direct inputs. Dependent variables are the data products (outputs) and the intermediate results. A service is the actions or computations performed on the data so that they are transformed and produced a result. Therefore a service is a task or a process that take inputs and produces outputs. An output from a service can be an input for another service (for example in a workflow). However, data and services need to be accompanied by procedures which detail the steps and actions taken towards assigning values to the dependent variables. There are several ways people can describe experiments, for example using scripts, or workflow systems. However workflow systems have the advantage of providing a visualisation computational process of the experiment. This will be described further in the literature review where workflow systems are described.

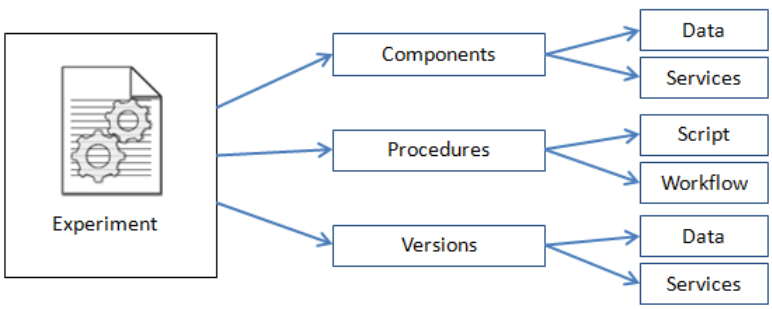


Figure 1.2: Experiment elements

Another important mechanism is versioning, as data and services can change after an experiment had been executed. Thus, versioning is a key focus of this thesis, with respect to the recording and retrieval of data and services in experiments.

Reproducibility is a cornerstone of science and is becoming a key research area in e-Science. This is because it provides ways for continuous improvement by supporting knowledge transfer through the re-use of an existing body of knowledge and methods. For example, a scientist (Scientist A) carries out an experiment on sequence data from microbial proteins and publishes his work. Five years later, Scientist B reads the paper which explains the theory, experimental implementation and results. Scientist B is very interested in the data and would like to exactly reproduce the experiment. If Scientist B is able to do so, he can learn from the knowledge generated by the past experiment. He can then observe and reflect on this experience, and may recognise problems or discover new opportunities to build on the work. This scenario enables Scientist B and other research communities to continue to learn from past experience. According to one of the most widely studied and cited learning process models, the Kolb experiential learning theory, experience from the past can be taken as the source of learning for the future [10].

However, how can Scientist B reproduce the experiment? Is there a database where he can download all the required microbial protein sequence data? Bowker [11] points out that in the standard scientific model, *"one collects data, publishes a paper or papers then gradually loses the original dataset"*. In addition to Bowker's concern, not only do datasets need to be preserved if experiments are reproducible, but also the computations that generated them. e-Science experiments deal with computations, therefore reproducing experiments involving computations is what is important.

Today, if a scientist wants to build on another's previous work, it is often a painful process involving a tremendous amount of reimplementation. The scientist has to write his own scripts and code in order to process the data, if the data is available. The scientist also needs to verify and test whether the reimplementation produces the same results as the previous one. Only then can the scientist proceed with building on the results of this earlier experiment.

Therefore, reproducibility creates opportunities for scientists to share, analyse and explore new problems and refine the past experiments. The ideal *"virtuous cycle"* of

reproducibility aimed to be realised through this work is presented in Figure 1.3. However, achieving this is not straightforward, and is therefore the key focus of the work described in this thesis. The key question is how to reproduce experiments that involve computations and data? This requires a way to preserve computations, data and methods so that reproduction is achievable. This leads to the reason why provenance has become another key research area.

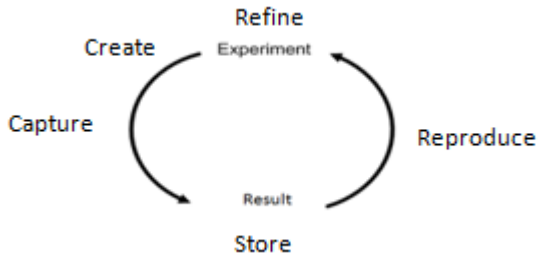


Figure 1.3: The ideal cycle of reproducibility

Provenance enables reproducibility. According to the Merriam-Webster Online Dictionary [12], provenance is defined as i) the origin, source and ii) the history of ownership of a valued object or work of art or literature. There are many papers that give definitions that are similar to this [13, 14, 15]. They all regard provenance as the derivation from a particular source (e.g. origin) to the specific state of an item. Scientists are interested in provenance because it allows them to view the steps involved in collecting and processing data. Provenance allows scientists to verify how results were achieved. Storing and preserving data alone does not provide sufficient information to allow experiments to be reproduced. Preserving services that represent the computations is also important in order to keep track of services that have been invoked. Exposing the relationships between data and services for an experimental run can be achieved using a provenance trace [6]. The need to have a provenance trace of the experiment (execution) that documents data and services explicitly is a precondition for reproducibility. This trace will give the scientist who is interested in the experiment a complete understanding of the experiment data, including the services that have been consumed and produced the data. However, as we will see, a typical provenance trace does not contain all the information needed to ensure that it is possible to reproduce the experiment.

There are number of models that describe provenance such as Provenir [2] and the Open Provenance Model (OPM) [4]. These two provenance models are discussed in Chapter 2. This work shows how OPM can be used to represent an experiment. The question *"Is OPM expressive enough to describe the provenance of data and services used in the experiment so that it can be reproduced?"* is explored and answered in this thesis.

The work shows that versioning is particularly important because data and services may be modified as time goes by. For example, services can up upgraded to improve functionality or fix bugs. Thus, it is argued that the versioning of data and services is needed to prevent overwrites and deletions from preventing reproducibility. However, while the current provenance literature does address data versioning, it is lacking in addressing service versioning. There are problems if the external services are removed by the service provider or owner that makes the services no longer available or inaccessible. There is no mechanism to record the version number of external services into provenance. The common practice of researchers dealing with non-versioned services is that when a service is upgraded, the earlier version is overwritten. Therefore, the old versions of services are not available after new versions of a service are deployed. If service version is not applied, it is difficult for the user to know whether the service in the past provenance trace is the same as the latest service available. There are existing best practices that support service versioning for Web Services, such as using XML namespaces [16] and adding a version number to the existing tModels in the UDDI registry [17]. These practices will be discussed further in Chapter 3. In this work, the SOAP WSDL-specified web services are used, not covering RESTful web services. Therefore a specific programming or scripts is out of the scope of this work. User can use the available web services through workflow execution environments. Today, service versioning is not directly supported in OPM or other provenance models. Therefore, in this work, these best practices are explored, their relationship to reproducibility examined, and the results incorporated into a provenance model.

As discussed earlier, OPM provides a way to capture provenance information from an e-experiment. It does not directly provide a way to reproduce the past experiment in a workflow environment if it is different from the workflow environment when the experiment was created and run. Therefore, this work designs and describes a method to transform OPM to create a workflow that can be executed to reproduce an e-experiment.

In this work, the Taverna Workflow System is used as the vehicle to re-execute past experiments.

With such reproducibility needs and provenance in mind, another concern to address is the reproducibility taxonomy. This taxonomy identifies the reproducibility requirements for facilitating the reproduction of experimental results, such as the descriptive provenance trace, organised collection of information, accessible repository and registry, and implementable reproducible technique. These requirements are discussed at Chapters 3, 4 and 5.

1.2 Contribution

This thesis investigates the conceptual and architectural aspects of provenance and reproducibility, with particular reference to service versioning. Its overall contribution can be summarised as:

> **Provenance has the potential to provide scientists with a complete understanding of data, including the services that produced and consumed it. Exploiting provenance, a reproducibility framework is designed to allow past experiments to be reproduced. A review of existing provenance systems shows that the issues of service versioning have been neglected, preventing reproduction from being achieved. This thesis describes these issues, and proposes, implements and evaluates a solution.**

The specific contributions of this thesis are:

- **Contribution 1:** It shows how the Open Provenance Model (OPM) can be used to describe a class of experiments, so forming the basis for reproducibility.
- **Contribution 2:** It shows how OPM is then extended to incorporate service versioning in provenance.
- **Contribution 3:** It describes the design, implementation and evaluation of a reproducibility system - ReProduX - that transforms the provenance of past experiments to allow experiments to be reproduced.
- **Contribution 4:** It describes a reproducibility taxonomy which describes the requirements for a reproducibility system.

1.3 Structure of the Thesis

In Chapter 2, a review and discussion of existing work on aspects related to the thesis is presented. The review takes into account the reproducibility and provenance concepts in general, and critically analyses existing work on how provenance can support reproducibility. There are discussions about the requirements a reproducibility system should have. A reproducibility taxonomy is proposed.

Chapter 3 describes in greater detail the conceptualisation of an experiment using the OPM, and motivates the need to incorporate service versioning in provenance via OPM Generator which is a contribution of this thesis. Though this thesis uses existing web service versioning best practices, applying the practices to the work to accommodate the needs of versioning in provenance is a contribution of this thesis.

Chapter 4 introduces a technique (ReProduX) to transform the OPM to Simple Conceptual Unified Flow Language (SCUFL) (used in Taverna Workflow System), which forms another contribution of this thesis.

Chapter 5 presents an evaluation of a design and implementation of this approach. The chapter discusses the implementation and the experimental setup, the tests, and the difficulties and limitations in achieving reproducibility. In this chapter, the proposed reproducibility taxonomy outlined in Chapter 2 is tested.

Chapter 6 concludes this thesis, draws conclusions, and proposes some areas which can be explored further in future research.

Chapter 2

Background and Related Work

This chapter describes the background behind the thesis, focusing on the work relevant to the concept of a reproducibility framework. Firstly, the chapter reviews previous work on existing reproducibility systems and current approaches to facilitate reproducibility, followed by provenance systems. The review includes provenance trace, provenance models, the capabilities and limitations of provenance systems, and current efforts where provenance is a critical enabler to achieve reproducibility, which then leads to a discussion of issues and gaps in current practice such as service versioning. Secondly, this chapter reviews the technologies used to implement service versioning. The review includes Service-Oriented Architectures (SOA) and scientific workflow-based systems. After this review, a taxonomy that describes the requirements needed to facilitate the reproduction of experiments is presented. Then the overall discussion of existing work is summarised at the end of this chapter.

2.1 Reproducibility

Reproducibility requires the author of any publication, such as the description of an experiment, to document the creation of the experimental results from the inputs and processes so that others can reproduce the results.

As researchers have realised that reproducibility can promote sharing, and give other advantages to the scientific community, there has been a growth in work on reproducibility [18, 19, 20, 21]. These works discuss the motivation for reproducibility, as well as describing infrastructure to support it.

Reproducibility research was pioneered by Claerbout [22] and his research groups at the Stanford Exploration Projects in the geophysics domain [23]. Vegue [24] reproduced experiments using a scripting language, Python in the Clawpack package. Buckheit and Donoho [25] used Matlab-based tools in creating and executing the experiments. Peng and Eckel [26] used the cacher package in the R language. These are reproducibility systems that mostly use programming/ scripting methods to reproduce the experiment without the use of provenance. For example, Buckheit and Donoho develop WaveLab to reproduce the computations that underly the figures and tables in the Wavelet articles published by the Wavelet community group. When creating the figures in the article, all the data and code will be a subdirectory of WaveLab containing scripts used to generate the figures. When the readers of the articles are interested to know how the figures and tables are generated, using Matlab, they can see what algorithms were used and parameters were set in producing the figures.

Reviewing previous reproducibility efforts has shown that all materials needed for experiments are recorded and documented and collected in a package to distribute to other interested researchers. Each work on their own internal environment such as UNIX and Matlab. Likewise, Matlab's ability to re-run scripts is beneficial if the research is entirely contained within Matlab, as scripts can be re-run in any computational environment that supports Matlab.

Such programming or scripting systems do not provide a visual model to illustrate the experiments graphically to make them more understandable to other researchers who may be interested in re-using them. The visual model has several advantages over programming or scripting systems where several of activities can be designed and connected the outputs and inputs of one another. Therefore, for users who are not programming savvy, a visual model is preferred. Many have argued that reproducibility is more likely to occur in cases where users can visualise as a graph the data and services of an experiment, along with their connections. There are many works on visualising computations, mostly using workflow systems [27].

There are several existing works on reproducibility in science, such as Trident Scientific Workflow System [18] [28], Research Objects [29] [30], Vistrails [31] and e-Science Central [21] [32]. Trident Scientific Workflow System is a commercial workflow management system that provides users to compose a workflow of an experiment, visualise, execute and manage the execution of experiment. In order to capture the information

about the execution, Trident has its own provenance model to automatically collect, store, query and view provenance of a workflow in a workflow and provenance schema. By having the provenance mechanism, Trident is giving better understanding to other researcher to understand and be able to reproduce the experiment result. The Trident provenance model is also compatible to the OPM specification. In research, the experiment that was carried out may change and get updated from time to time. This means that the workflow will evolve as new versions of experiments has changed, that leads to versioning issues. In supporting reproducibility Trident has incorporates Trident EVF, a Workflow Evolution Framework that has a versioning strategy to handle the issues of managing workflow evolution. In EVF, the versioning is handled using direct evolution and contributions. When a researcher needs to edit the workflow, the direct evolution will take place to record the changes, thus the version of the workflow evolves. The contributions feature will enable the tracking of versioned workflows and correlates them with the workflow results. Therefore, Trident EVF preserves all artifacts that relate to the execution of experiments in Trident Registry. In Trident versioning data model, Trident EVF also provides ability to work with other versioning system by providing an extension points to add new versioning system. This is believed may be extended to also incorporate web service versioning from third-party services.

Research Objects (RO) supports reproducibility by providing a container to keep the collection of all resources that associate with a particular research or an experiment so that the resources can be sharable within community or in public. RO provides rich annotation to keep details of all its resources such as authors, versions and citations to support preservation and sharing. Research Objects use the Object Exchange and Reuse (ORE) model to represent aggregation of web resources includes workflows, datasets and document. RO has its life cycle that starts with Empty Life RO where RO can be filled by workflows and datasets, then followed by Live RO that represents work in progress, and after it has passed the state where it can be preserved, Live RO will be kept into Snapshot RO where the record of past activity is kept. The final stage is Archived RO where at this stage RO is stable and appropriate for long term preservation. For new discovery, the existing Archived RO can be reproduced as a starting point to new research and it will initiate the first cycle again.

Vistrails has an advantage of visualisation where it shows the versions of workflows into one canvas, uniquely support for data analysis and visualisation. This is benefi-

cial where we can see straightaway the differences to compare the results side by side, whereas in many other workflows each of workflow version needs to be called and run one by one. Vistrails can reproduce and validate computational experiments directly from the publications. The VisTrails system provides researchers with the ability to capture and explore the data involved in the researchers' work. An interesting feature of this work is where data, computations and results of the experiments are associated with embedded links. These links allow readers to access the results in the paper and explore them with new inputs or original parameters to produce variations of the results. Vistrails has an automatic workflow upgrade mechanism to support the workflow evolution and displays all of the workflow versions in a version tree. The comprehensive provenance support in Vistrails maintains detailed history information including the versions of a workflow.

e-Science Central is a workflow based programming environment that aim to achieve reproducibility of past experiments by verifying the original experiments results. Whenever there is a change in the experiments' components such as data changes, service updates and workflow evolves, e-Science Central provides a framework to support reproducibility in computation experiments. Woodman et. al (2011) have described how e-Science Central overcomes the issues of handling the service and workflow versioning in achieving reproducibility by automatically stores and retains all the versions, stores a full provenance trace for all executions and automatically transform a provenance trace into a workflow. Missier et.al (2014), have introduced provenance trace divergence detection in e-Science Central to compare the two execution traces to determine whether the reproducibility is achieved. However their focus is not on handling external dependencies (ED) such as third-party services that not within the control of e-Science Central.

Comparing to the above reproducibility systems, this thesis makes two arguments that firstly, the service versioning support is also needed in the design and execution, and the versioning support should be considered at the early stage of service development. At this stage, the service provider or service owner should be allowed to provide multiple versions of services, and can be placed and registered by service registry, in dedicated directories. The works in Trident EVF and Vistrails do not discuss the service versioning issues of the computations (services) at the service creation. Secondly, this thesis looks at the issues of handling external dependencies such as third-party web services that is

not within the control of the workflow environment. The other reproducibility works including e-Science Central mentioned in this thesis are on the other hand, where they are the owners of the data, services and workflow, therefore they are in control to do all the amendments and updates of the services. Normally, they are able to get access to their own local data, services and workflow.

The past experiments results need to be accompanied by the necessary data and code needed to reproduce them. In this work the code is encapsulated in the services. The interested users may have difficulties reproducing the experiments when the services may be missing or not available at all time. This may be due to new version of a service has been created and updated, and the old version is no longer used. Is there a mechanism to handle this? The available access on versions of services can guarantee that the past experiment can be re-executed. However, it is not guaranteed to produce exact results due to service update has overwrites the previous version of the service. In this case, the service version identified in the past experiment trace is different from the one is currently available.

Reproducible Research (RR) [20, 33] gives guidelines on the requirements for making research reproducible. These include the proper submission of research papers, data, experiments, the results of the experiment, and auxiliary materials. RR creates a reproducible research repository and links to five working examples of reproducible research publications, specifically in image and signal processing. The links provide the full paper, archived data and code to reproduce all images and tables from the paper. However, there the guidelines are lacking as the issue of service versioning is not been discussed.

Peng [34] has described the barriers existing in reproducible research such as making the data and code available. People, in the main, do not support reproducibility because there is too much effort needed in preparing experiments to support reproducibility later. Exploiting reproducibility requires the sharing of resources and tools across disciplines and between a set of individuals and/or institutions. Groth [35] relates this to multi-institutional scientific systems which he defines as, *"Systems that require sharing and coordination of resources across multiple institutions for a particular scientific domain or question"*. He has identified three key properties: the sharing of resources to carry out tasks, support for heterogeneous resources, and dynamicity in participation amongst users of the system. Groth et al. [36] have shown that the readiness of reproducibility

is in place where the provenance architecture of capturing, recording and sharing the resources have been implemented in the community. However they do not discuss the issues of versioning of services in the identified properties, and this is what differentiates this work.

2.2 Provenance Enabled Reproducibility

In this thesis, provenance is shown to be an essential element in reproducibility. However how exactly can provenance enable reproducibility? To answer this question, this section reviews the meaning of provenance and existing provenance models and systems.

2.2.1 Provenance

Provenance can be defined as the act of tracking the origin, source or history of an object [37]. In a computation, the object can be drawn from the inputs, the processes, or the final output (the experimental result). Therefore, the provenance of experiment results is concerned with the tracking of how results were derived, what data sets are used by processes or services that make up the whole experiment.

Scientists are often interested in provenance because it provides them with a more complete understanding of the data they can access, and also the process of conducting an experiment. Zhao et al. [13] describes three purposes of provenance from the viewpoint of the scientist:

1. **Debugging:** The scientist can view a log of events, recording what services were accessed and with which data, in order to detect bugs that cause undesirable results.

2. **Validity Checking:** Validation ensures results from the experiments are of good quality and meaningful. A framework for validating workflow executions enables scientists to verify the correctness of their own experiments or review the correctness of their peers' work.

3. **Updating:** The scientist needs to know any changes that affect the previous results. Therefore, any updates on the datasets and services should be visible.

Another purpose of provenance, which is the focus of this thesis, is

4. **Reproducing computations:** Reproducing archived e-experiment results can involve reproducing computations using provenance to provide all the necessary information. A scientist can also re-run the experiment with a different algorithm and parameters, examine each result, and then compare them to see the difference. These issues are covered by Freire et al. [38] and involve capturing, storing, and querying computational tasks.

Provenance involves capturing the relationships between the inputs, processes and outputs. Taking an analogy of researching the genealogy of a family, we often hear about lost or hard to find connections in families. This problem is due to a lack of information and linkage between families. The origin of Person C can be found by looking at the processes and other objects involved, as illustrated in Figure 2.1. For this case, the ancestry relationship involves the process M (produce), that involves Person A and Person B. The process M produces a product, Person C. The relationships that may be involved are Person A "is spouse to" Person B, if they are married. Person C "is generated by" Process M, "produce". Having these connections recorded helps to discover the family tree, and reunites families.

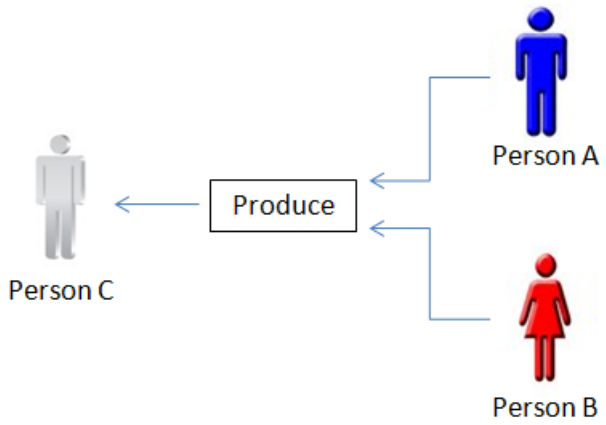


Figure 2.1: The analogy to genealogy

When C later on meets another person, say D and they may produce E and F. If one day F needs to know who his ancestors are, this path on the family tree can assist in tracing their history. Not only the path, but the information of the genealogical

information such as their family names, dates of birth and name of places are also beneficial. There can be many reasons and motivations behind having this family tree, one of the greatest reasons is *"...to preserve the past for future generations"* [39]. This family tree helps to identify the origin of a family.

The concept of provenance is analogous to the above. In the context of a scientific experiment; the provenance of a data product is the process that led to that product. The provenance records the data sets, parameters and computational processes that were involved in deriving the data product. A review of how provenance is documented and is modelled follows.

2.2.2 Documentation for Provenance

e-Science experiments are composed of experimental components that are data and services. This work refers to a service as a component that takes data inputs and produces data outputs. When an experiment is conducted, the execution process must be captured, not only capturing the experimental components, but also the interactions that take place and how those interactions are related. Relating back to the definition of a provenance, Moreau [1] summarizes provenance as *"The provenance of a piece of data is the process that lead to that piece of data"*. This means there exists documentation that describes the process, taking into account the experimental components and the interaction between them.

To address the needs of the above scenario, PASOA [40] has developed an architecture for determining the provenance of data. The architecture proposes three phases in a provenance life cycle; 1) create a description of the execution; 2) record the description into a provenance store; and 3) query the execution result in order to obtain the provenance record. Creating a description of execution that shows the detail of every interaction between data and services will produce documentation. This documentation is called an execution log [41], process documentation [42] or provenance trace [21, 43]. This work uses provenance trace as a term for the recording of an execution. The provenance trace may also need to be managed and maintained in the provenance store.

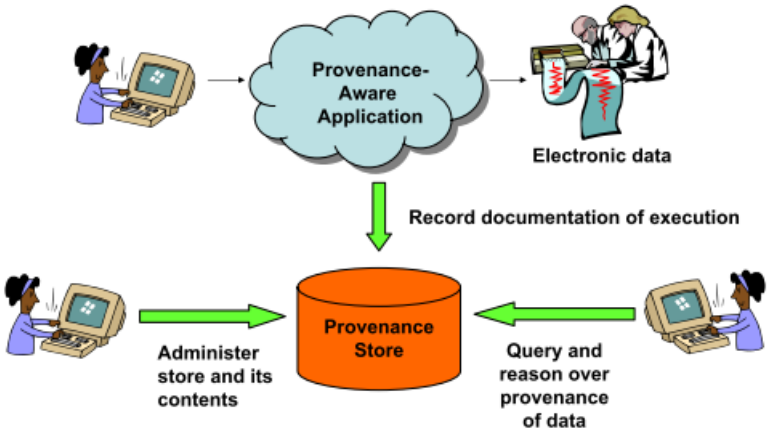


Figure 2.2: Provenance Lifecycle [1]

Realising the phases in the provenance architecture, it is important to ensure that the provenance trace is properly structured. In [44] it is claimed that experimental documentation should be structured in a way that can help to construct a record of how execution takes place. By having a good experimental structure is also helpful at the query stage. To ensure this is achieved, a provenance data model is needed.

2.2.3 Provenance Modelling

If a provenance trace records the execution that shows the derivation of how experimental results were obtained, then there must be a model that represents all entities involved in all computational paths. The data model is a representation of the data, and the computations involved in consuming and producing that data. Therefore the provenance model is a model of the documentation for the provenance trace. In this work the following four provenance models are now described in more detail: Provenance Authoring and Versioning Ontology (PAV) [45], Provenir Ontology (PO) [2] , Provenance (PROV data model) [3] and Open Provenance Model (OPM) [4].

Provenance Authoring and Versioning Ontology (PAV) [45] is an ontology to present the general needs of description for tracking provenance, authoring and versioning. PAV distinguishes one digital resources to another, by its properties such as pav:createdBy, pav:version and pav:retrievedFrom. Therefore, each digital resource will be unique.

Though PAV describes versioning, its ontology does not cover the provenance process where value flows from one process to another in the execution, which is needed for reproducibility.

Provenir Ontology (PO) is an ontology to represent provenance metadata and is defined by OWL-DL. Provenir is taken from a French word meaning *"to come from"*, and describes the history of an entity. Provenir's main components are the three classes:

- **Data:** Entities that are mutable, which are still undergoing changes or immutable, that is permanent. For example, the temperature value that is measured in a bath such as 50°C.
- **Process:** Actions that may result in the creation of new entities (data). For example, the measuring process that produces the temperature value as output.
- **Agent:** Agent causally controls the process. For example, a temperature sensor is an agent that controls the temperature measuring process.

In addition to the three classes above, the data class is expanded into two sub-classes. It has sub-classes of data collection class that represents datasets used and modified during the execution, and parameter class that represents parameters that influence the execution; both classes are generated from the execution of experiments as shown in Figure 2.3. The parameter class has three sub-classes namely spatial parameter that captures spatial metadata such as the graphical location, domain parameter that captures domain specific parameter such as tolerable level and the temporal parameter which is to capture temporal details such as timestamp and duration associated with data, agent and process. In showing dependencies between these classes, eleven properties are adapted from the Relation Ontology (RO) that are:

- part_of represents the inclusion relation between entities.
- contained_in represents the containment relation between entities.
- adjacent_to represents the association with the agent that may have an effect on data values.
- transformation_of represents the two entities that undergo the transformation stages.

- derives_from describes the derivation history of data entities.
- preceded_by decribes the ordering between processes.
- has_participant links data to process where the instance of data class participates in a process.
- has_agent links agent to process where agent affects the change in state of the process.
- has_parameter links the instance parameter class to instance of data collection, agent and process classes.
- has_temporal_value represents the time and duration of the data collection, process and agent.
- located_in represents one instance of data or agent associated with one location.

These dependencies are shown as edges in Figure 2.3 [2].

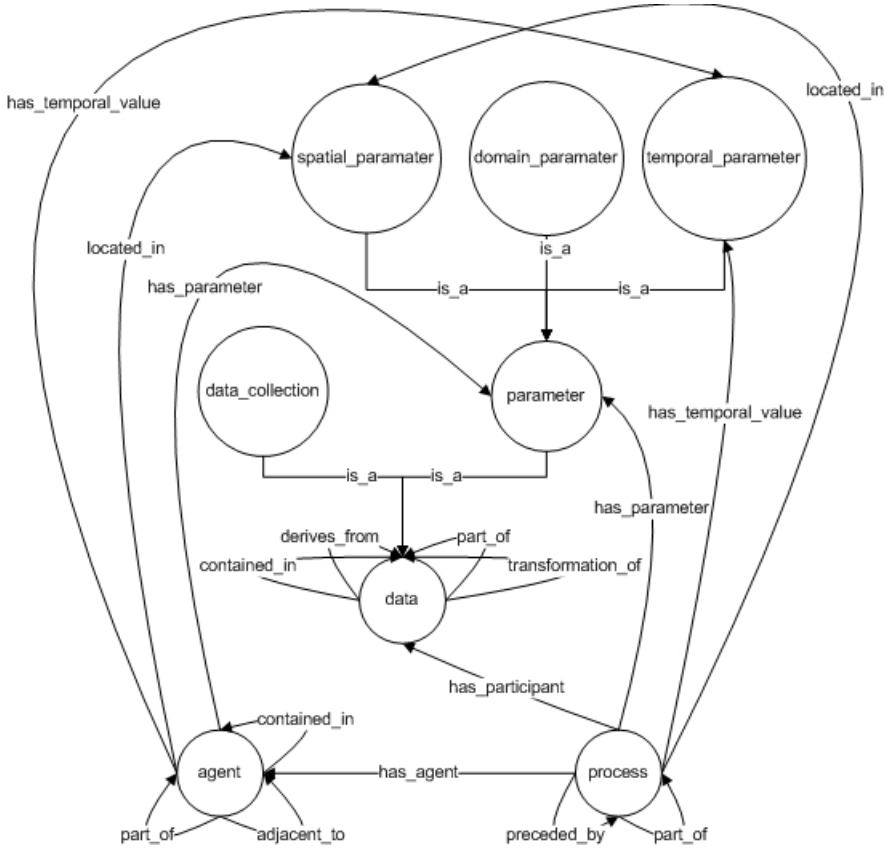


Figure 2.3: The classes and relationships in the Provenir ontology [2]

Provenir has been used to describe applications in the proteomics domain, which has led to the creation of ProPreo ontology [46]. In addition to Proteomics, Sahoo et al. [47] have demonstrated the ability of Provenir to model provenance metadata in parasite research and oceanography. In parasite research, the Provenir ontology's classes and relationship are extended and integrated with other existing ontologies in the domain, for example Parasite Experiment (PE) ontology to describe the parasite applications. In oceanography research, Provenir is also integrated with the Trident ontology as the research uses the Trident scientific workflow system. In these case studies, Sahoo [2] claims that one single provenance ontology may not be able to capture the provenance details from all different domains. This is because one provenance ontology is specific to a particular domain. The paper therefore proposes the need to integrate the Provenir

ontology with multiple ontologies to support interoperability and model provenance metadata specific to a particular domain. In carrying out this approach, Provenir has shown the extensibility of its provenance model by integrating its base model with other ontologies.

PROV data model is a general provenance model to describe provenance records that include three components that are entity, activity and agent, as depicted in Figure 2.4 below. PROV supports interoperability on the Web. PROV captures the provenance and traces the evolution of research object by describing its components with relations as follows:

- used: An activity used an entity to perform the activity.
- wasGeneratedBy: An entity was generated by an activity, therefore the entity becomes available or exist after the generation.
- wasDerivedFrom: An entity was derived from another entity, based on activity that caused the derivation.
- derives_from describes the derivation history of data entities.
- wasAttributedTo: An entity is assigned to an agent who is responsible to the entity existence.
- wasAssociatedWith: An activity is related to an agent where the agent has the role in the activity.

PROV expresses the description of research objects in the past, therefore also handling versioning using derivation subtypes wasRevisionOf. In PROV, new version relates to new entity. This differs to this work on expressing service versioning, where version can also relates to new activity version that is discussed in the next Chapter 3.

The Open Provenance Model (OPM) is a data model driven by community efforts towards provenance standardisation. OPM was originally created by Moreau et al. [4] in 2007 after the realisation of the importance of provenance in wide sections of the scientific community. This mainly came from the active participation of the provenance community in the Provenance Challenges [48]. In the third Provenance Challenge, all 15 provenance teams successfully mapped their models to OPM. Although each team had its own provenance model for capturing provenance information, the teams were

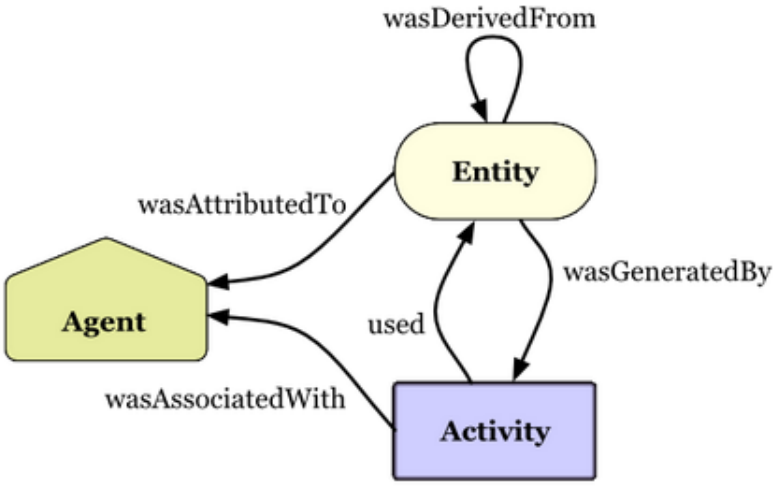


Figure 2.4: PROV Data Model [3]

able to map their provenance data model format to the OPM format. By doing so, they were able to share provenance information across different provenance models, through the use of OPM. Over time, the data model has been enhanced by including features from other existing provenance systems.
OPM has three main nodes and five edges. The three types of nodes, as depicted in Figure 2.5(a) are:

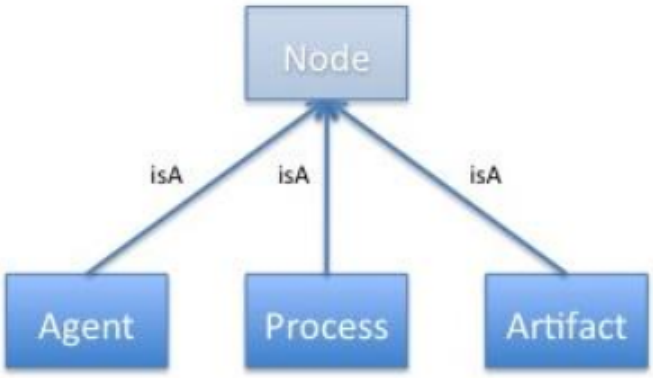
- **Artifact** (visually represented by ellipses): Entity that is used or generated by processes, and is an immutable piece of state. For example, the city code.
- **Process** (represented by rectangles): Activity or action that is performed using or producing artifacts; hence each process must have at least one artifact. For example, calculate the weather forecast for a particular city.
- **Agent** (represented by octagons): People, organisations or systems that perform and control activities or actions. This entity acts as a catalyst for a process. Weather sensors can be an example of an agent.

OPM has five types of edges, as shown in Figure 2.5(b), each edge represents the causal dependencies as following:

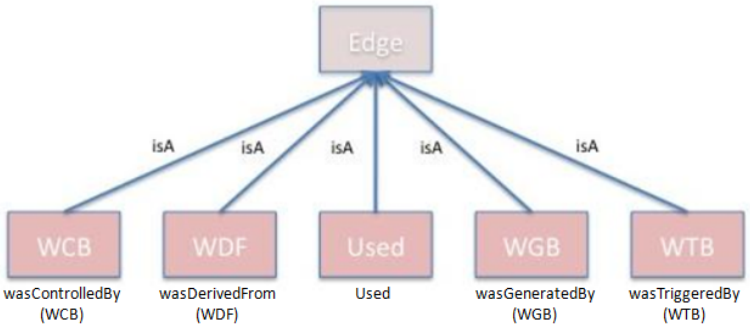
- used describes the relationship from a process to an artifact.

- wasGeneratedBy describes the relationship from an artifact to a process.
- wasTriggeredBy expresses the relationship From a process to another process.
- wasDerivedBy expresses the relationship from an artifact to another artifact.
- wasControlledBy expresses the relationship from a process to an agent.

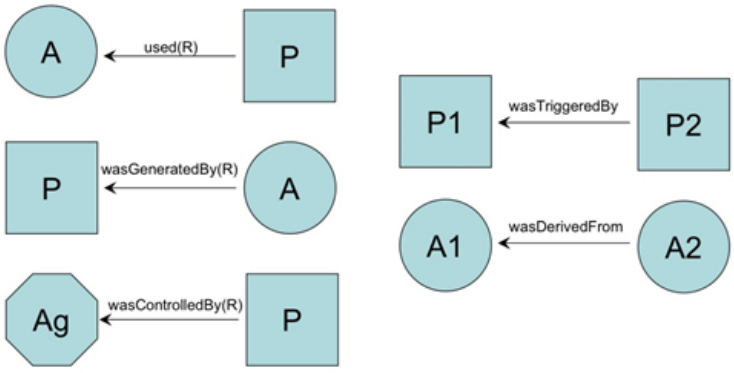
The three nodes and five different dependencies are all shown in Figure 2.5(c).



(a) Three types of nodes



(b) Five types of edges



(c) The nodes and edges (causal dependencies)

Figure 2.5: Nodes and edges in OPM specification [4]

An OPM example -"Atlas X Graphic" - taken from the First Provenance Challenge Workflow [49, 50] describes a scenario in which John Doe, as an agent, is executing the PC1 Workflow process to produce the artifact output, Atlas X Graphic. In order to do this, John uses four artifacts that are Anatomy Image1, Anatomy Header1, Reference Image and Reference Header. This example is illustrated using the OPM graph in Figure 2.6.

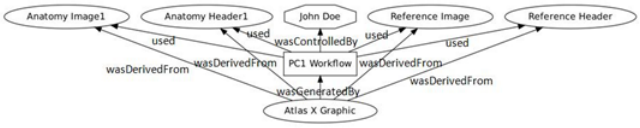


Figure 2.6: The Atlas X Graphic OPM Graph

Figure 2.6 shows the OPM graph generated from the example, while Listing 2.1 and Listing 2.2 shows a representation, in XML, produced from the OPM graph. The representation is a document that describes the OPM graph in Figure 2.6.

Listing 2.1: OPM XML describing the nodes

```
1    <processes>
2        <process id="p1">
3            <account ref="account1"/>
4            <label value="PC1 Workflow"/>
5        </process>
6    </processes>
7    <artifacts>
8        <artifact id="a1">
9            <account ref="account1"/>
10           <label value="Anatomy Image1"/>
11       </artifact>
```

Listing 2.2: OPM XML describing the edges

```
1        <wasGeneratedBy id="g_9">
2            <effect ref="a5"/>
3            <role id="r_8" value="x"/>
4            <cause ref="p1"/>
5            <account ref="account1"/>
6            <time exactlyAt="2010-10-18T22:02:34.883+01:00"/>
```

```
7          </wasGeneratedBy>
```

Listing 2.1 and Listing 2.2 shows a representation, in XML, produced from the OPM graph. The representation is a document that describes the OPM graph in Figure 2.6.

Listing 2.1 shows examples of two nodes that are processes and artifacts (Lines 1 to 11) while Listing 2.2 shows the edgeof wasGeneratedBy (Lines 1 to 7) in OPM model. However OPM does not express versioning of the evolution of its processes. Therefore this work has introduces a new edge that is needed to express service versioning. This is explained in Chapter 3.

OPM has been used in many applications such as to represent the World Wide Web and distributed systems. Freitas et al. [51] have found that OPM can be used as a foundation for the creation of the World Wide Web. The paper describes the Web provenance model which is built based on the identified provenance requirements for the Web, which is based on the OPM concepts. In addition to the Web, Groth and Moreau [52] use OPM to track provenance within distributed systems. A set of distributed systems' patterns for locality, failures and attribution are able to be addressed by the OPM model. By having the distributed systems described by OPM, provenance from different systems can be integrated cohesively, thus achieving interoperability.

A mapping between PAV, PO, PROV data model and OPM was created by the W3C Provenance Incubator Group to study the similarities and differences between the two models, and also other provenance models [53]. Both models aim to create a model for the representation of provenance. In this thesis, OPM is the data model used to describe a class of experiment. OPM was chosen as a data model for representing provenance in this work for the following reasons:

- **Simplicity:** Readable and easily understandable OPM notation
- **Structural validity:** Consistency with the way a data model defines and organises information
- **Interoperability:** The Third Provenance Challenge workshop has demonstrated that all fifteen provenance teams successfully mapped their models to OPM. This represents the provenance community's effort towards the realisation of OPM.

e-Science experiments were executed in different environments, therefore using a standard provenance data model would be an advantage, allowing experiments to generate the data and services and other provenance metadata although the environments are different. Chapter 3 will show that the OPM data model is sufficient to represent the experimental information, and OPM can be extended to support versioning. However, neither the provenance modelling directly addresses service versioning that enables the reproduction of an experiment. Therefore allowing OPM to support versioning is a contribution of this thesis.

2.2.4 Provenance Support in Workflow Systems

In recent years, provenance has become increasingly important in scientific applications. Systems have become more automated, capturing information about when, where, why and how experiments are carried out and by whom. Recording and using provenance information has become possible more easily. There are various provenance systems in many fields such as physics, bioinformatics, engineering and geographical sciences.

In scientific workflow system, the execution of computational processes captures data sets and services and the causal dependencies, supported by provenance is described in Taverna [5] Vistrails [31], Trident EVF [28] [18] and e-Science Central [21] [32]. Table **??** compares these workflow systems in terms of provenance features that are important to the work of this thesis. These features include: type of provenance information (data and services); workflow versioning and service versioning. Each of these features is now considered in turn, discussing in more detail how the four major provenance systems match up to them.

Table 2.1: Features of existing workflow systems with provenance support

Workflow Systems / Features	Taverna	Vistrails	Trident EVF	e-Science Central
Record Data Sets	Yes	Yes	Yes	Yes
Record Service	Yes	Yes	Yes	Yes
Support Workflow Versioning	No	Yes	Yes	Yes
Support Service Versioning	No	No	No	Yes

- **Type of Provenance Information (Data Sets and Services)**

In Taverna [37] each experimental run is considered as a workflow. Semantic Web technologies are used to record four levels of provenance: process, data, organization and knowledge. Process provenance records each service invocation, the inputs and outputs with time-stamps while data provenance records the data derivation paths which have generated the final data products. As for the organization provenance level, this records information on the experiment such as the creator of the data and service. Knowledge level links to the other three provenance levels, and records more implicit information such as from users' interpretation and understanding regarding the experiment run. Vistrails [31] provenance management component captures input and output datasets, parameter settings, library versions, workflow structures and also code (services). Trident EVF [18] Workflow workbench captures datasets, services and workflow activities. During a workflow run, e-Science Central [21] records input dataset and computational tasks that can be through third-party services. The artifacts (datasets), computational tasks (services) or actors are recorded in a provenance trace based on the OPM.

- **Workflow Versioning**

Vistrails, Trident and e-Science Central support workflow versioning by tracking the versioning of workflow design and execution information over time. Therefore users can track the changes of the workflows and refer to a specific version of a workflow that has been run in the past. They may also run different versions of the same workflow and try using the same dataset to see if it makes a difference. The versioning information in these systems is collected at the development time. Vistrails has an automatic workflow upgrade mechanism to support the workflow versioning. The comprehensive provenance support in Vistrails maintains detailed history information including the versions of a workflow. Trident uses direct evolution and contributions to keep workflow versions, as described earlier in this chapter. e-Science Central records versioning via its integral storage feature that handles versioning. On the other hand, Taverna has no workflow versioning feature supported during development time. However, Taverna is using myExperiment to keep track of the workflow version. In contrast to the other three systems, Taverna is using myExperiment that keeps workflow versions at the user's publishing time.

- **Service Versioning**

As well as workflow versioning, if reproducibility is to be achieved, it is important to be able track service versioning. Users should be able to examine the differences that occur if different versions of a service are used in a workflow. Taverna, Vistrails and Trident EVF do not fully address the needs for service provenance. This may be due to any service changed from the past execution is considered under workflow versioning. However, there is no description that captures information on the version of the service invoked. The service functionality may have changed after the provenance was captured, and if version changes are not captured and managed, an attempt at reproduction may not be successful.

e-Science Central has an integral storage that supports versioning of data, service and workflow through a virtual file store driver. When there is any change in previous service, a new one is automatically created. The user can choose any service and any version of a service prior to running a workflow. However, e-Science Central only handles services within the control of itself, therefore does not handle external services that are not in their control. If the external services has no supporting mechanism to making sure they are accessible, there is no guarantee that the past provenance trace can be reproducible.

The concept of service versioning on third-party web services than is not within the control of workflow executions has therefore been lacking in the provenance literature, and in the design of existing systems. This includes the standard mechanism to record service versioning, how to find the correct version of a service when it is called during reproduction, nor how to keep old versions of services available. The aim of this thesis is to address this shortcoming.

2.3 Service Versioning

Provenance is particularly important when an e-experiment is to be reproduced and re-run. Provenance provides the ability to reproduce all the steps leading to a scientific e-experimental result. This means provenance can describe how the result was generated, thus illustrating how the experiment was carried out. Provenance enables the recording of the data and services, including the data parameters used, and also timestamps of service invocations. If we are to look inside each of these services, there are also service

metadata that may be significant and need to be recorded in provenance; for example when a particular service was created and which version it is. It is often the case that a service will be changed after its initial deployment to fix bugs, improve the algorithm, or meet new requirements. This evolution of a service is likely to result in different versions being used in different workflow executions made at different times. Therefore service versioning should be supported by a reproducibility infrastructure to ensure that: a) even after new versions of a service are deployed, the old version still remains available and b) that the exact version is recorded in the provenance trace. Therefore, it is possible to know if the currently available version is the same that identified in the provenance trace. In this thesis the focus is on services using Web Services technology. Before considering versioning in more detail, Web Services architecture is discussed.

2.3.1 Adopting a Service-Oriented Architecture (SOA)

Taking past experiments and making them reproducible needs a way to connect the components. Service-oriented architectures provide a way to achieve this. Before we go further, we need to understand what a service is, and how services can be connected to build applications.

"A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services" [54]. A service takes input and produces output by executing tasks. The analogy of a service exists in the real world. Almost every interaction in daily life may be treated as the execution of a service, such as paying a bill through online banking, and sending messages through mobile phones. These tasks are provided by services that process input to generate output. However, the concept of a service is still evolving in the world of information technology.

A service-oriented architecture is designed to support the implementation of a collection of services. These services can work together in applications, if there is a means of connecting them together. Web Services provide technologies that allow this.

Web Services use standard technologies for communication including eXtended Markup Language (XML) [55], Uniform Resource Locators (URLs)[56], the Web Services Definition Language (WSDL) [57], the Simple Object Access protocol (SOAP) [58], and Universal Description Discovery and Integration (UDDI) [59]. These are important for this thesis, and so are now discussed in more detail.

XML is a W3C (World Wide Web Consortium) specification that defines the structure of documents for describing data. XML syntax consists of text-based mark-up that is machine (and human) readable. For this reason, XML is accepted widely as the standard format for data interchanged between systems. WSDL, SOAP and UDDI all use XML.

In the context of Web Services, a service's interface can be expressed in WSDL, which includes information on the location of the service, its available operations, and the communication protocols. A WSDL file is an XML document with six main elements to describe a Web Service; 1) port type that describes the operations input and output performed by the service, 2) port defines the individual endpoint with a single address for a binding, 3) message describes the inputs and outputs between web service providers and consumers, 4) types defines all data types used by the web service, 5) binding describes how to interact with the service and 6) service which specifies the address (URL) that is the location of the service.

Communication with services uses the common transport protocol SOAP. This is a protocol for exchanging data over HTTP, or other lower-level protocols. It provides a standard way for sending XML messages between applications, for example from clients to services. SOAP messages are also an XML document containing some or all of the following elements; 1) envelope indicates the start and the end of the message 2) header (optional) contains information relevant to the message and where additional new features and functionality can be added, 3) body contains the data being exchanged in the SOAP message and 4) fault (optional) that carries error messages within a SOAP message.

UDDI is defined by the Organisation for the Advancement of Structured Information Standards (OASIS). It is an XML-based registry which publishes information on available Web Services. UDDI allows organisations to publish their services, and provides information to help other organisations discover which services are available. UDDI describes services using WSDL. The information on services can include versioning information, which is important to this thesis, and so is discussed further in the next chapter.

Figure 2.7 illustrates a basic service-oriented architecture that shows how Web Services relate to SOA.

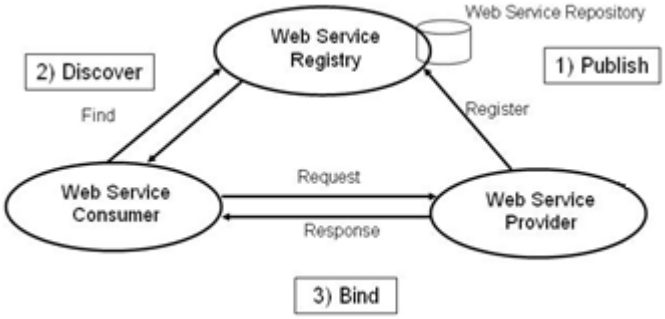


Figure 2.7: Service-oriented architecture basics

Figure 2.7 shows the interactions between the Web Service Provider, Web Service Registry and Web Service Consumer. Firstly Web Services are written by developers, who provide WSDL that describes the location of the service and the operations that the service provides. Next, the service is published by the Web Service Provider to a UDDI registry. Then, Web Service consumers can search the registry to discover the services that a client needs via application such as UDDI browser. Thirdly, the binding process takes place, based on information in the registry, clients use the services' WSDL to construct SOAP request messages for exchanging data with the service, usually via HTTP.

Groth [35] has described the use of the SOA in building the multi-institutional systems application. A variety of domains have adopted SOA including bioinformatics, physics and astronomy. In this work, SOA is adopted to facilitate the creation of a reproducibility framework.

2.3.2 Service Versioning Approaches

Although there is no standard mechanism for this at the present time, there are best practices which can offer some suggestions with regard to incorporating Web Service versioning in provenance. There are several approaches available [17] [16], however two web service versioning approaches are now considered that are using XML Namespaces and using tModels in the UDDI registry.

The first approach is using XML Namespaces. This approach creates an entirely new Web Service with a new WSDL file and namespace for each version. This means supporting the versioning of WSDL documents. Different namespaces (each showing

different versions) are used to achieve this. The drawbacks of this approach are that it requires, after each service update, changing all client applications so that they now call the new service, and the collection of services may become unmanageable as new versions are created, as it is not possible to categorise services into collections.

The second approach uses UDDI's tModel structure, specifically tModel instanceDetails which carries information about a service, such as the URL of the related WSDL document. A service version element can be added to the tModel. The version element is added in the keyedReference under the categoryBag in the tModel structure. By adding this, the version information will be available along with other existing service description in the UDDI registry. When calling a service, a client can use the UDDI APIs (for example using UDDIBrowser) to discover the service's access point and which versions are available. More details of this approach will be described in Chapter 3.

Both service versioning approaches take WSDL documents as important documents in managing versions of multiple services. Fang et al. [60, 61] extended WSDL and UDDI to manage version information. They designed a proxy to dynamically update a client application if a new version of the same service is created. Frank et al. [62] use a service interface proxy as a router to provide a service selection whenever a new version is available. However, this work will not make any extension to WSDL and UDDI. Instead, it uses the tModel service versioning approach where one tModel corresponds to one WSDL.

2.3.3 Workflow-Based Systems

Workflow-based systems are used to choreograph the execution of an experiment from data and services. The enactment of a workflow can be used to invoke computational services. Some workflow systems provide a graphical user interface for building workflows [5, 63, 64]. An example of a weather forecast using the Taverna workflow system is shown in Figure 2.8. Taverna has been used to orchestrate communication between web services into a workflow and to be able to support how services interact within the workflow. Figure 2.8 shows there are two inputs *CountryName* and *CityName* that are required to execute the process of *GetWeather*. The output of this workflow is the *WeatherInfo*. The other boxes of *CountryAndCity* and *GetWeatherResult* are the input and output XML splitter required by Taverna for each service. The Taverna splitter

services are for separating elements out or combining together elements in the XML document, which will be described in Chapter 4.

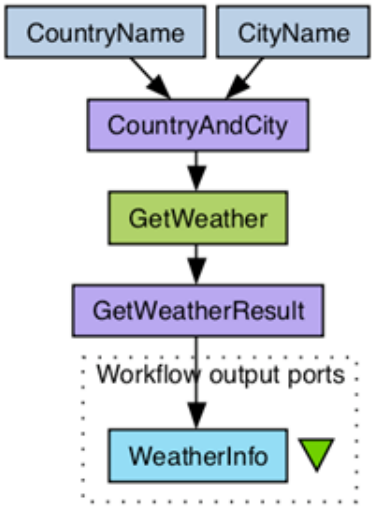


Figure 2.8: An example of a simple workflow that retrieves a weather forecast for the specified city [5]

In myGrid [65], researchers can create workflows directly using the Taverna workbench. This provides ways to capture the parameter settings, inputs and any relevant information.

This thesis adopts the Taverna Workflow System as the tool for a reproducibility solution as it is widely used and has community support for sharing through myExperiment.org. Workflow systems are also considered to be more user friendly for developers who do not have to worry about the technology being used to enact the workflow nor to call the services: they can instead focus on expressing graphically, in a high-level way, the solutions to problems in their domains.

Despite the advantages of workflow systems, existing workflow systems fall short in providing the basis for reproducibility. For reproducibility, it is not sufficient to only store the artifacts (input) values, the final result and service name or endpoint. It is also necessary to store information on the version of any services executed in a workflow. This is largely due to the problem of service versioning. Overcoming this problem is the focus for this thesis, and will be discussed in more detail in Chapter 3.

OPM should be able to identify and record the version of the services. However, an external service versioning mechanism is required to allow those services to be used. The service versioning design must be able to support two things:

- Preserving old versions of services
- Being able to call old versions of services

2.4 Reproducibility Taxonomy

It is not immediately obvious that we can achieve reproducibility unless we have a clear definition of what reproducibility is, and be able to organise and manage the information needed to achieve it. Lambe [66] has an interesting description of what taxonomy is, in the context of knowledge management. He defines three key attributes of an effective taxonomy: a classification scheme, a semantic map and a knowledge map. A taxonomy classifies related terms together so that it is easy to find the related terms in the category. A taxonomy also describes the relationships between terms in the taxonomy. For example, in the taxonomy of a computer system, Computer: Keyboard would imply the relationship *"is a part of"* between Keyboard and Computer. A good taxonomy is a type of knowledge map with regard to a particular knowledge domain. For example, in the provenance community, there is a Provenance Taxonomy introduced by Simmhans et al. [6] as shown in Figure 2.9. This paper provides a comprehensive description on provenance in terms of usage, subject, representation, storage and dissemination. These descriptions present, at a conceptual level, what a provenance system is; why a provenance system is needed in e-Science; and what is described as provenance and how that provenance is captured, represented and stored. The survey can be used as a starting point to understand existing provenance system designs. However, as will be seen, some service related provenance issues covered in this paper are missing from the taxonomy.

The purpose of contributing a reproducibility taxonomy in this work is to help readers to understand the things we need to be able to do if we are to achieve reproducibility. The taxonomy does not only classify, describe and map the knowledge domain, but a reproducibility taxonomy is made up of the things we must do to achieve that outcome:

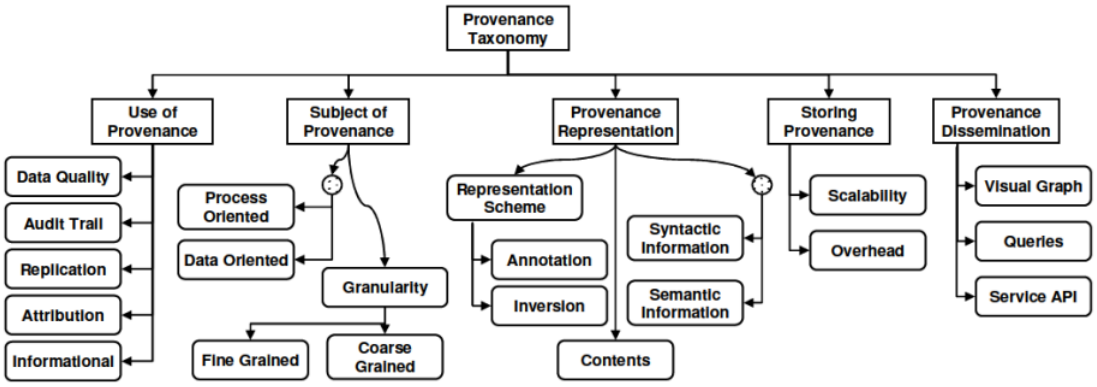


Figure 2.9: Provenance Taxonomy by Simmhans et al. [6]

preparing a collection (aggregation) that addresses service versioning, creating documentation (representing the data, services and dependencies), and using a workflow system. This taxonomy is based on the knowledge and findings derived from the work leading to this thesis work, and is therefore described in Chapter 5.

2.5 Discussion

This chapter has described a number of systems for representing and generating provenance. Many e-science systems are based on workflows, in which communication between services is choreographed in order to produce a result. It has shown that while there is a wide variety of previous work in the areas of workflows and provenance, there is a major gap in being able to handle service versioning. This is key to reproducing e-experiments that involve services that may be updated over time, for example to fix bugs or improve efficiency. In web services technologies, ways have been proposed for handling service versioning, but in itself this does not resolve the problem, as there has been no integration with provenance systems.

As described in this chapter, with the ability to interoperate with other provenance models, OPM was chosen as a data model for representing provenance in this work. Therefore, Chapter 3 will further review the entities in OPM and describe how it can be the basis for capturing the provenance trace of the execution of an experiment in the presence of services that may be updated over time.

This chapter has also investigated reproducibility taxonomies. These are important for describing and relating the information needed to achieve reproducibility. Existing

taxonomies do not handle versioning, and so the work to achieve this is described in Chapter 5.

Chapter 3

Achieving Reproducibility by Incorporating Service Versioning into Provenance

Over the years, the research community has realised that a major problem in sharing its research experiments with others, is the inability to reproduce past experiments. This problem is caused by 1) insufficient information describing the experiment and 2) research (experimental) artifacts and processes (services) that are not available.

This reproducibility process therefore needs provenance information to describe the execution of the experiment in a way that can allow reproduction. In addition, the experimental artifacts and services should be made accessible for later use. Therefore, the essential concepts underlying the reproducibility of experimental results are capturing the computation, along with the data on which it operates. In service-based e-science, the fundamentals of a computation are processes that take inputs and transform them into outputs. Therefore, the processes and all the datasets that are involved must be captured in order to allow reproduction.

As Open Provenance Model (OPM) is an emerging standard, this work explores whether the OPM is able to describe an experiment sufficiently precisely so as to support reproducibility. The work also addresses the issue of how to ensure that the versions of services involved in the experiment can remain available, as service versioning is part of essential requirements in reproducibility.

The contributions of this chapter are therefore:

- To describe how the Open Provenance Model (OPM) can describe a class of experiments, so forming the basis for reproducibility.
- To introduce service versioning into provenance.

3.1 Capturing Experiments Using the Open Provenance Model (OPM)

Capturing experiments involves recording information on experimental components, procedures and versions. Referring to the experimental elements introduced in Figure 1.1, this section focuses on the Components - Data and Services - as highlighted in 3.1.

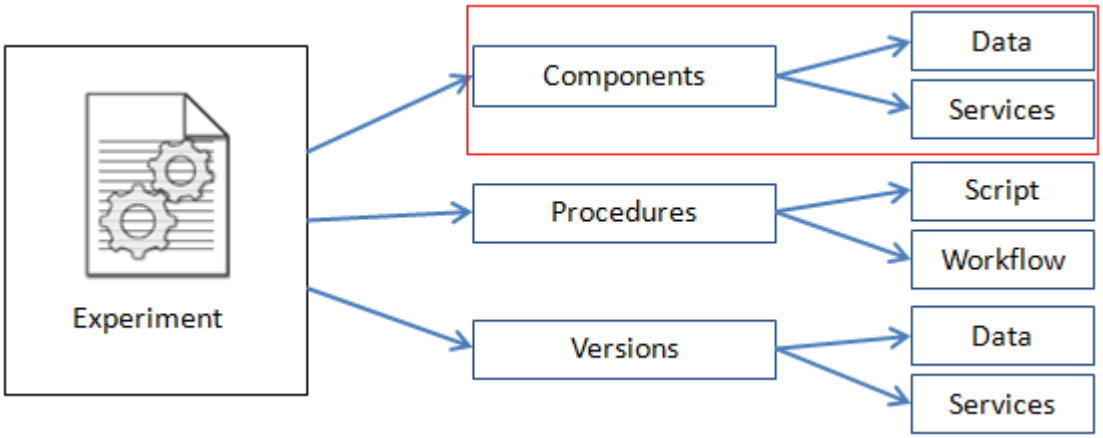


Figure 3.1: The components of an experiment

As described in Chapter 2, OPM is a way to describe digital experiments [4]. It has been used in many application domains (e.g. Provenance Challenge [48]) and can describe the components of an experiment. There are two main aspects of OPM: content and structure. Content refers to the components embedded in the data model, while structure reflects the organisation of the components in the model. The content of the OPM model captures the meaning of specific entities in the data model. It contains nodes encompassing artifacts (data inputs and outputs of fixed value), processes (services) and agents (a catalyst or controller of a service), that reflect an experiment's execution. Along with these entities are the edges, also known as causal

dependencies that make the connections between the entities. There are five types of causal dependencies in OPM; opm:used, opm:wasGeneratedBy, opm:wasTriggeredBy, opm:wasDerivedBy and opm:wasControlledBy. Causal dependencies are essential in reproducibility, which requires identifying the cause and effect in the experiment (X was caused by Y) and the linkage between them. For example, this OPM model structure allows an OPM model to describe how an output was derived from an input.

To illustrate the use and limitations of OPM for capturing e-experiments, the next section introduces what will be a running example and the OPM graph it generates.

3.1.1 An Exercise Advisor Example

To illustrate the use of OPM, an example of consuming multiple services was created. This uses an experiment to recommend exercise activities based on a person's body mass index. There are three services (processes) involved in this application, namely *Calculate BMI* to calculate a person's Body Mass Index (BMI) based on their height and weight, *Check BMI Category* to categorise a person body classification, and *Recommend Exercise Activity* to advise the appropriate exercise activities. Examining the description of an Exercise Advisor yields a list of the execution activities in the experiment:

1. The value of *Height* and *Weight* are filled in at the input interface by users (Input1 and Input2).

2. A process that takes both *Height* and *Weight* produces an output, a *BMI Score*. A service called *Calculate BMI* is used to compute this.

3. The value of *BMI Score* is taken as an input for *Check BMI Category* service. The output of this process is the *BMI Category*.

4. The value of *BMI Category* is taken as an input for *Recommend Exercise Activity* service. The output of this process is the *Exercise Activity*.

5. The sequence of tasks in this application: Firstly, *Height* and *Weight* are used for the service *Calculate BMI* and a *BMI Score* is generated by this service. Secondly, the *BMI Score* is used for the service *Check BMI Category* and a *BMI Category* value is generated. Thirdly, the *BMI Category* is used for the service *Recommend*

Exercise Activity and generates the recommended *Exercise Activity* which is the final result of the computation.

A systematic analysis of the list of execution activities above suggests the following list of possible artifacts, services (processes) and dependencies, as shown in Table 3.1:

Table 3.1: The artifacts, service and dependencies involved in the Exercise Advisor experiment.

Artifacts	Processes(Services)	Dependencies
Height	Service 1 (S1) Calculate BMI	Used
Weight		
BMI Score		wasGeneratedBy
BMI Score	Service 2 (S2) Check BMI Category	Used
BMI Category		wasGeneratedBy
BMI Category	Service 3 (S3) Recommend Exercise Activity	Used
Exercise Activity		wasGeneratedBy

There are five artifacts, three services and two types of dependencies involved in the experiment. The activities 1-5 are illustrated in the OPM diagram as in Figure 3.2.

Figure 3.2 illustrates the OPM graph of the Exercise Advisor example which depicts the inputs, services and outputs. The round shapes are the artifacts, the square shapes are the services (processes), while types of edges are *used* and *wasGeneratedBy*. This graph will generate a document which is called a provenance trace. This will be described in Section 3.2.

After the Exercise Advisor is used by the public, consider a scenario where the users have noticed that the recommended *Exercise Activity* is not providing a suitable activity. Some users suffer knee pain, and some users are suffering from asthma after following the recommended exercises. This leads to an improvement to the current *Recommend Exercise Activity* service to include new parameter of *Body Condition* before recommending an activity. This service update is due to some activities are not suitable if a person is suffering from some complications such as asthma, knee pain, heart problems, pregnant and many more. Therefore *Body Condition* will take into account these complications prior to recommend a suitable exercise activity.

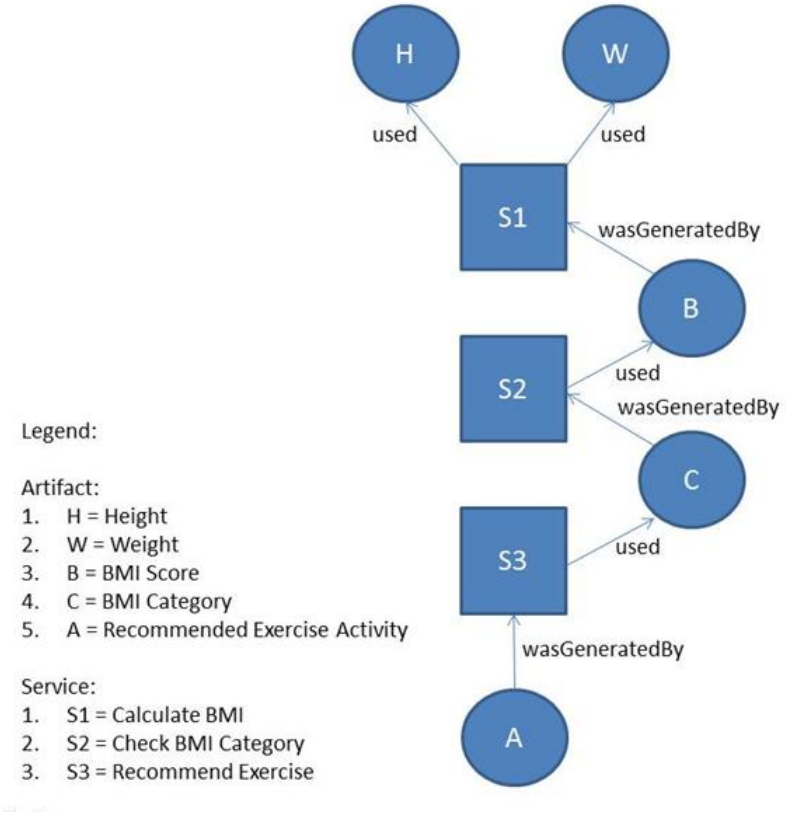


Figure 3.2: OPM representing the experiment to compute Exercise Advisor

An additional scenario is to include a person's daily free time as requested by the users due to their daily tight schedule that prevent them from doing the recommended activities. Therefore, by adding another new input parameter *Daily Free Time* to the existing service forces the service to have another service update.

From the above scenarios, the *Recommend Exercise Activity* has changed from initial version, to a second version and third version of service update as shown in Figure 3.3.

Service 3 (S3) evolves from version 1 till version 3: S3v1 -> S3v2 -> S3v3 and the last is the latest version of the service.

3.2 Capturing the Provenance Trace

When all the entities in the experiment have been identified, they must be captured and recorded as a provenance trace. The provenance trace gives information about the

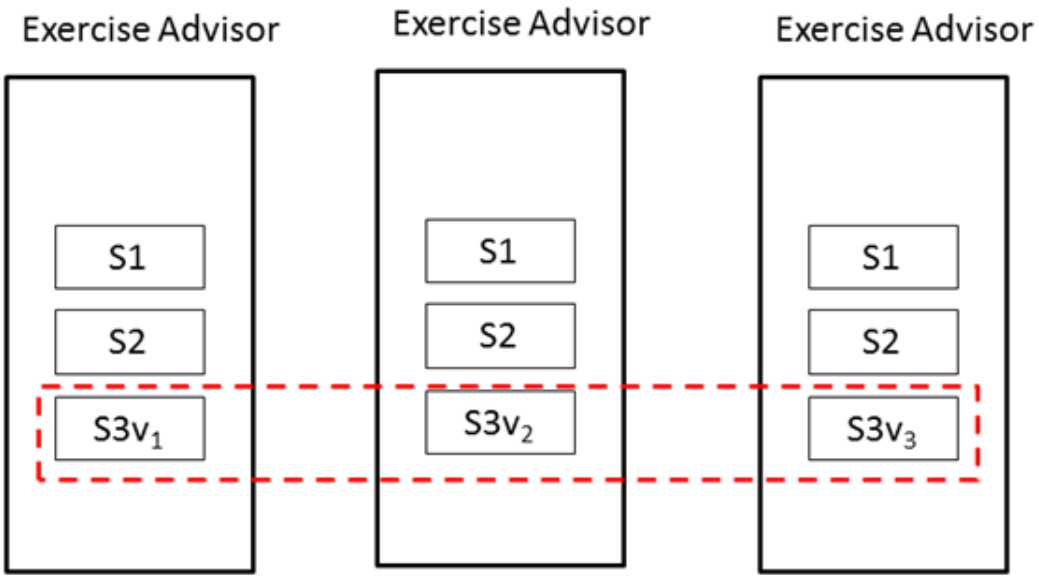


Figure 3.3: Service 3 (S3) evolves from version 1 till version 3

actual execution of an experiment. Therefore, in running an experiment, the creation of the final results that are derived from the input data are documented in a provenance trace. A provenance trace captures execution activities. Taking an idea from [67], Figure 3.4 presents a concept map of a provenance trace based on the work of this thesis.

This thesis uses OPM to represent the components in the experiment. The OPM provenance content and structure is therefore now described. In the document, OPM is represented as an XML document conforming to an OPM schema. The document shows how input data (artifacts) are transformed into output results (an artifact) through a sequence of services (processes), with causal dependencies that clearly show the causes and effects to the outputs. The OPM XML provenance trace for the Exercise Advisor from Figure 3.2, in Section 3.1.1 is shown in Listing 3.1.

Listing 3.1: The content and structure of the OPM XML for an Exercise Advisor

```
1       <opm:opmGraph xmlns="http://openprovenance.org/model/opmx#"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd=
            "http://www.w3.org/2001/XMLSchema" xmlns:ns4="http://example.com
            /" id="gr_16">
2        <opm:accounts>
```

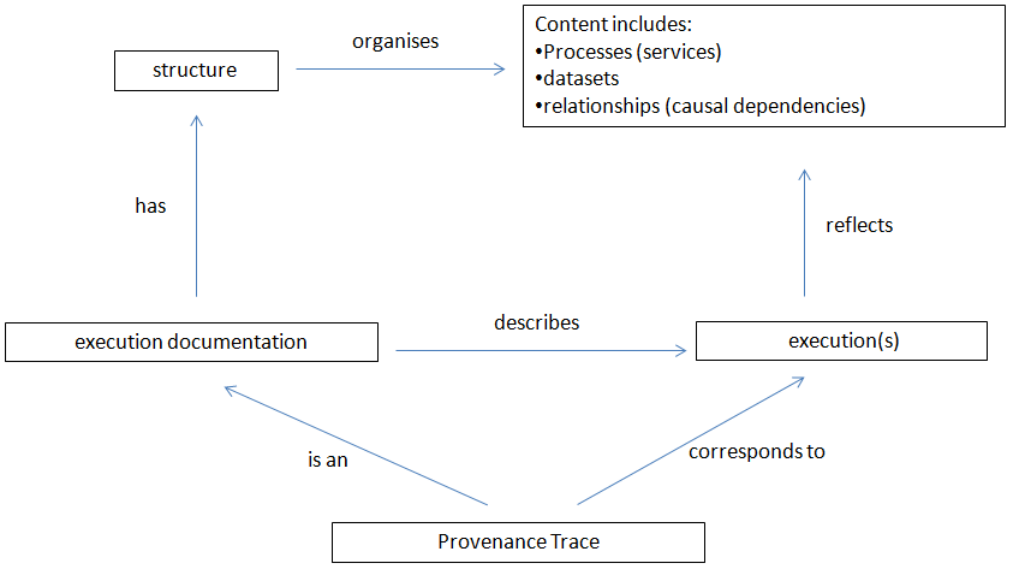


Figure 3.4: Concept map describing provenance trace

```
3           <opm:account id="account1"/>
4         </opm:accounts>
5         <opm:processes>
6          <opm:process id="myBmi">
7           <opm:value xsi:type="xsd:string">"http://localhost:8080/eabmi10
                /eabmi10?wsdl"</opm:value>
8           <opm:time exactlyAt="2014-10-06T04-46-27.447Z"/>
9          </opm:process>
10         <opm:process id="myCategory">
11          <opm:value xsi:type="xsd:string">"http://localhost:8080/
                eacategory10/eacategory10?wsdl"</opm:value>
12          <opm:time exactlyAt="2014-10-06T04-46-28.068Z"/>
13         </opm:process>
14         <opm:process id="myActivity1">
15          <opm:value xsi:type="xsd:string">"http://localhost:8080/
                eaactivity10/eaactivity10?wsdl"</opm:value>
16          <opm:time exactlyAt="2014-10-06T04-46-29.922Z"/>
17         </opm:process>
18        </opm:processes>
19        <opm:artifacts>
20         <opm:artifact id="weight">
21          <opm:value xsi:type="xsd:double">"44.0"</opm:value>
22         </opm:artifact>
23         <opm:artifact id="height">
24          <opm:value xsi:type="xsd:double">"1.58"</opm:value>
```

```
25          </opm:artifact>
26          <opm:artifact id="bmiscore">
27              <opm:value xsi:type="xsd:double">"17.63"</opm:value>
28          </opm:artifact>
29          <opm:artifact id="category">
30              <opm:value xsi:type="xsd:string">"UNDERWEIGHT"</opm:value>
31          </opm:artifact>
32          <opm:artifact id="activity1">
33              <opm:value xsi:type="xsd:string">"Proper dieting and Walking"</
                    opm:value>
34          </opm:artifact>
35        </opm:artifacts>
36        <opm:causaldependencies>
37          <opm:used id="u_1">
38              <opm:effect id="myBmi"/>
39              <opm:role id="r_1" value="1"/>
40              <opm:cause id="height"/>
41              <opm:account id="account1"/>
42              <opm:time exactlyAt="2014-10-06 12:46:30:PM"/>
43          </opm:used>
44          <opm:used id="u_2">
45              <opm:effect id="myBmi"/>
46              <opm:role id="r_2" value="2"/>
47              <opm:cause id="weight"/>
48              <opm:account id="account1"/>
49              <opm:time exactlyAt="2014-10-06 12:46:30:PM"/>
50          </opm:used>
51          <opm:used id="u_3">
52              <opm:effect id="myCategory"/>
53              <opm:role id="r_3" value="3"/>
54              <opm:cause id="bmiscore"/>
55              <opm:account id="account1"/>
56              <opm:time exactlyAt="2014-10-06 12:46:30:PM"/>
57          </opm:used>
58          <opm:used id="u_4">
59              <opm:effect id="myActivity1"/>
60              <opm:role id="r_4" value="4"/>
61              <opm:cause id="category"/>
62              <opm:account id="account1"/>
63              <opm:time exactlyAt="2014-10-06 12:46:30:PM"/>
64          </opm:used>
65          <opm:wasGeneratedBy id="g_1">
66              <opm:effect id="bmiscore"/>
67              <opm:role id="r_1" value=""/>
68              <opm:cause id="myBmi"/>
```

```
69              <opm:account id="account1"/>
70              <opm:time exactlyAt="2014-10-06 12:46:30:PM"/>
71          </opm:wasGeneratedBy>
72          <opm:wasGeneratedBy id="g_2">
73              <opm:effect id="category"/>
74              <opm:role id="r_2" value=""/>
75              <opm:cause id="myCategory"/>
76              <opm:account id="account1"/>
77              <opm:time exactlyAt="2014-10-06 12:46:30:PM"/>
78          </opm:wasGeneratedBy>
79          <opm:wasGeneratedBy id="g_3">
80              <opm:effect id="activity1"/>
81              <opm:role id="r_3" value=""/>
82              <opm:cause id="myActivity1"/>
83              <opm:account id="account1"/>
84              <opm:time exactlyAt="2014-10-06 12:46:30:PM"/>
85          </opm:wasGeneratedBy>
86        </opm:causaldependencies>
87      </opm:opmGraph>
```

Listing 3.1 describes the OPM representation in XML. In this OPM trace, myBmi process is equivalent to Calculate BMI process, myCategory is equivalent to Check BMI Category and myActivity1 is equivalent to Recommend Exercise Activity. Processes and artifacts are identified by unique identifiers (lines 6, 10, 14, 20, 23, 26, 29 and 32). Dependencies are identified by the cause (source), effect (target) and role, as shown in Line 36 to Line 86. For instance, in Line 37 to Line 43, the used dependency depicts that process *myBmi* has used artifact *Height*. Likewise, process *myBmi* has also used artifact *Weight* (Line 47). The used causal dependency expresses that the process *myBmi* can complete only because the two artifacts, *Height* and *Weight* are available. The *wasGeneratedBy* dependency expresses that artifact *bmiscore* (Line 66) can only exist if the process *myBmi* (Line 68) has taken place. In Listing 3.1, the two types of causal dependencies used and *wasGeneratedBy* imply that the artifact *bmiscore* which is the output generated is caused by the process *myBmi*, while the process itself is caused by the two artifacts it used, which are *Height* and *Weight*. The cause and effect explicitly describe the flow of data and processes.

In OPM, the data values of the artifacts that are used (line 40 and Line 47) or generated (Line 66) by a process, and also an agent (not included in this example) that controlled a process, are associated with a role. Roles are required in edges used,

wasGeneratedBy and *wasControlledBy*. Role captures additional information about the dependencies to differentiate between several different use and generation relations. For example, Line 39 and Line 46 show that one role (Line 39) is for taking height as input, and the other role (Line 46) is for taking weight as another input. The artifact can also be used by more than one service, for other reasons. If this situation occurs, more than one account is needed. Each account can derive one path, for example to calculate a BMI index. If the artifact is used for another purpose such as to convert the unit of *Height* and *Weight*, this will require a different account in OPM. In this work, the structure is account-less so the artifacts do not belong to any *Account* specification (Account (Line 2 to Line 4) shown in the Listing 3.1 is only there as an example). In OPM, edges can be annotated by time information, however in this thesis, time is not discussed because time is not required to describe the ordering of the data and process precedence in this work, as it is already covered by the causal dependencies. However, time is proposed to be recorded in the versioning section, which will be described later in Section 3.3.2.

In summary, this experiment shows that OPM represents artifacts that send and receive data, processes (services) that change that data, and relationships between artifacts and processes.

3.2.1 A Gap in the Provenance Trace

OPM is sufficient to describe the components of experiments and also the execution orders of experiments. The previous sections show that achieving reproducibility requires a provenance trace which is described based on the provenance model, as illustrated in Figure 3.5.

However, service versioning information is needed. It is added here through OPM annotations and OPM causal dependencies, based on the rules specified in the OPM Annotation Framework as presented in Moreau et.al [4]. Annotations in OPM can be held independently as an annotation entity, or can be added to other OPM nodes and artifacts.

Even if information about versioning is available, this is not sufficient for reproducibility, as there is no automatic mechanism in provenance to ensure that all the

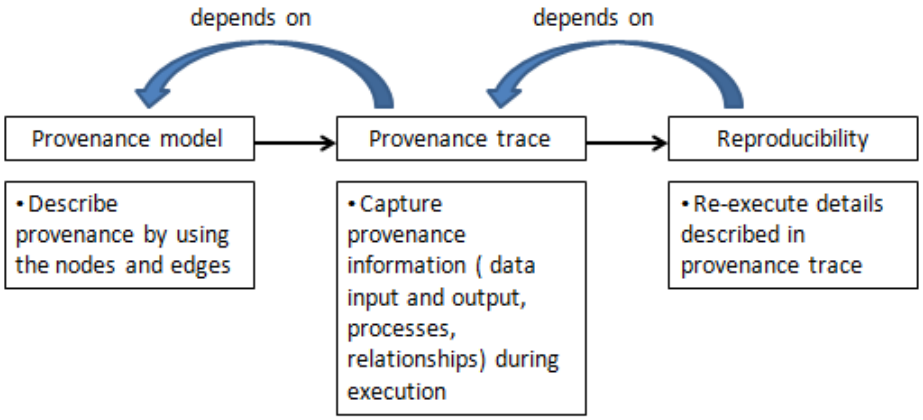


Figure 3.5: The dependency of provenance for reproducibility

multiple versions of the same service remain available.

Further, if multiple versions of services are preserved, the annotation information must link to the appropriate version so that it can be used in re-execution.

Therefore, the design of a system to allow the re-execution of experiments that include services that may have been updated must be able to support:

- Preserving old versions of services.
- Being able to call old versions of services.

3.3 Incorporating Service Versioning into a Web Service Architecture

Service versioning is essential in reproducibility. It also has other benefits. For example, in a research community, it is an advantage to be able to access multiple versions of the same service so that researchers can compare one version to another, and understand their effects on processing data.

Another reason to access multiple versions of the same service is so that any amendments and enhancements to an existing service do not affect the existing consumers of the previous version of the service, who may choose not to move to the new service (for example to keep consistency with previous results). In the future, we might imagine subscription services to inform the consumer that a new service version is available.

This will allow the consumer to choose whether to remain with the existing service or to upgrade to a new one.

Why web services are important in this work? Rather than adopting a specific programming, publishing algorithms as web services is an option for user. User can use the available web services through execution environments. The execution environments such as Taverna, provides user to take the web services and connect the services into workflows and execute them. Chapter 2 has described web services and its standard. WSDL is part of the standard and is well documented. WSDL provides a formalised and detailed input and output and this make it possible for user to use the web services in the workflow system. However, the web services need to be made available to public. The WSDL can be registered by the service provider (owner) to service registry to publish the location of available services. However, what happen if the services have been removed by their owners? The service may become inaccessible. Therefore, if service version is recorded, other alternative of same services can be recommended. This is described further in following sub-sections.

Web service exists from service provider or service owner. Therefore, it is recommended that service versioning is handled at the early stage of service creation by service provider or service owner. That means providing web services via different ports. Therefore, in order to incorporate service versioning, a service versioning convention scheme needs to be followed. In this work, the following service versioning convention is used:

Figure 3.6 describes the service convention that takes into account major and minor releases. If a service needs to add new service parameters, therefore major release is applied. If only minor code amendment such as fixing bugs, changes in algorithm may only apply minor release, and is backward compatible. Backward compatible means the new version is compatible with current version. Existing clients can use the new version. Also in this work, all service clients have the same compatibility contract: WSDL and XML Schema. Figure 3.7 illustrates the minor and major service releases. Refer to example S3v2, in which the service version is a minor release from S3v1, and is also backward compatible. Client 1 application still can use the new service version. However for another service update S3v3, the service version update is considered as a major release. This is an incompatible change due to changes in ports to provide new parameters, with new additional new label, as illustrated in Figure 3.7.

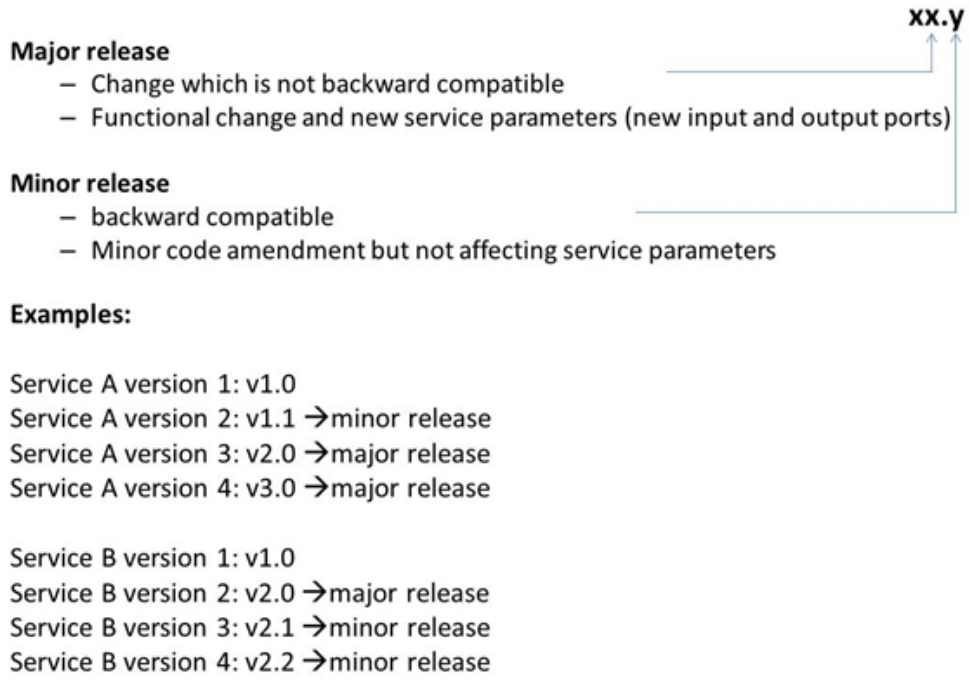


Figure 3.6: Service versioning convention scheme

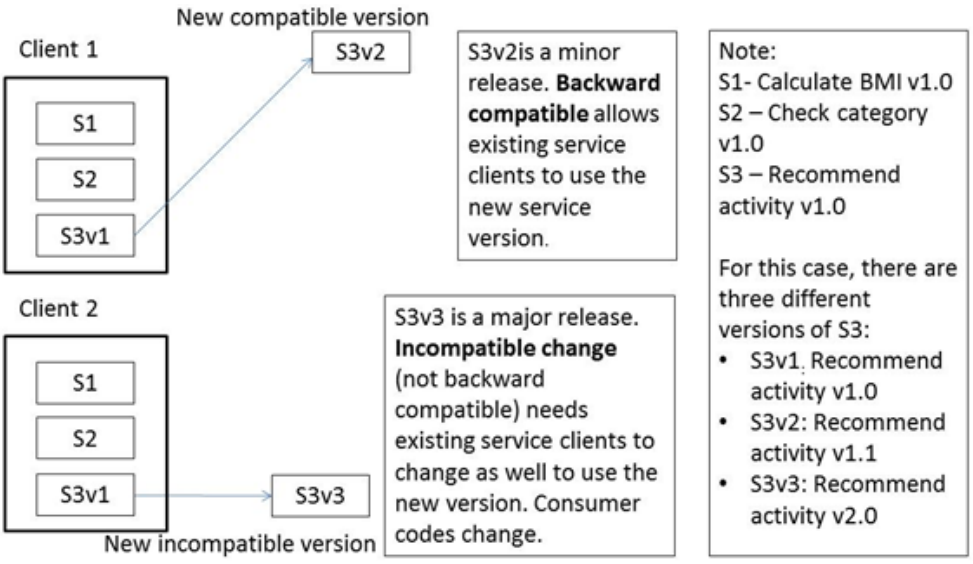


Figure 3.7: Compatible and Incompatible changes in service update

Consider a scenario in which a service is created and published to a server. A WSDL file is created and is used to describe a web service. In order to ensure there is sufficient

information to invoke the service, the WSDL information must provide the following: service description; service abstract interfaces and service concrete implementation [68]. The Service Abstract Interface description describes the interface layout, for example the abstract data type definition (Line 1 to Line 5) and operation parameters with input and output message (Line 13 to Line 16) as shown in Listing 3.2 for a BMI calculating Web Service.

Listing 3.2: A WSDL interface description snippet

```
1  <types>
2   <xsd:schema>
3       <xsd:import namespace="http://ea/" schemaLocation="http://
            localhost:8080/eabmi10/eabmi10?xsd=1"/>
4   </xsd:schema>
5  </types>
6  <message name="myBmi">
7       <part name="parameters" element="tns:myBmi"/>
8  </message>
9  <message name="myBmiResponse">
10      <part name="parameters" element="tns:myBmiResponse"/>
11 </message>
12 <portType name="eabmi10">
13  <operation name="myBmi">
14      <input wsam:Action="http://ea/eabmi10/myBmiRequest" message="
            tns:myBmi"/>
15      <output wsam:Action="http://ea/eabmi10/myBmiResponse" message="
            tns:myBmiResponse"/>
16  </operation>
17 </portType>
```

The Service Concrete Implementation description binds the interface description to a network addresses and protocol, as shown in Listing 3.3.

Listing 3.3: A snippet of the WSDL describing a service

```
18
19    <binding name="eabmi10PortBinding" type="tns:eabmi10">
20      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style=
            "document"/>
21      <operation name="myBmi">
22       <soap:operation soapAction=""/>
23       <input>
24        <soap:body use="literal"/>
25       </input>
```

```
26        <output>
27          <soap:body use="literal"/>
28        </output>
29       </operation>
30      </binding>
31      <service name="eabmi10">
32       <port name="eabmi10Port" binding="tns:eabmi10PortBinding">
33        <soap:address location="http://localhost:8080/eabmi10/eabmi10"/>
34       </port>
35      </service>
```

By having the service interface and implementation (Listings 3.2 and 3.3), a consumer can have a clear understanding about a service's interface and also the network access point to which messages can be sent in order to invoke a service. The full details of the WSDL descriptions are not covered in this work, but can be found in [68] [57].

Once the WSDL has been created, the next step is to publish it to a UDDI service registry. The service registry is key to this reproducibility work. In the work of this thesis, the jUDDI registry is used and described, as this structure supports the provision of information on service versioning. jUDDI stands Java implementation of the Universal Description, Discovery, and Integration specification for Web Services [59]. It provides a Web Services directory platform. Through it, consumers may find information about businesses and organisations offering web services, descriptions of those web services, technical information that exposes location and access information, and also the web service interface information.

Consider a scenario in which a service is consumed by a client. After the service is initially deployed, it may be changed to meet new requirements, to improve its algorithm, or simply to fix bugs. Later, a consumer wishes to reproduce an experiment that used the service. The jUDDI service registry can be used to ensure that the correct version is utilised.

The approach taken here to service versioning takes advantage of the loosely coupled architecture provided by web services technologies. Service versioning is the approach that should be taken by the Service developer, which is the Web Service Provider in Figure 3.8. As highlighted by the red circle dashed line, the Provider who is in control of creating and updating the service should keep the versions of updated service available for consumption using the service versioning approach, which is discussed in the next section. Therefore, whenever a consumer sends a request for a particular version of a

service, the Provider will always be able to invoke the service. Figure 3.9 illustrates the concept of how multiple versions may exist (in this case ten years since the service is first deployed), and the diagram shows that two versions of the same service S2 are available, that are S2v1 and S2v2. In order to have these versions available for the consumer, this section will discuss how the web services architecture component, in particular UDDI Web Service Registry, is used, as highlighted by the blue circle dashed line. The multiple documents represent the multiple versions of the same service.

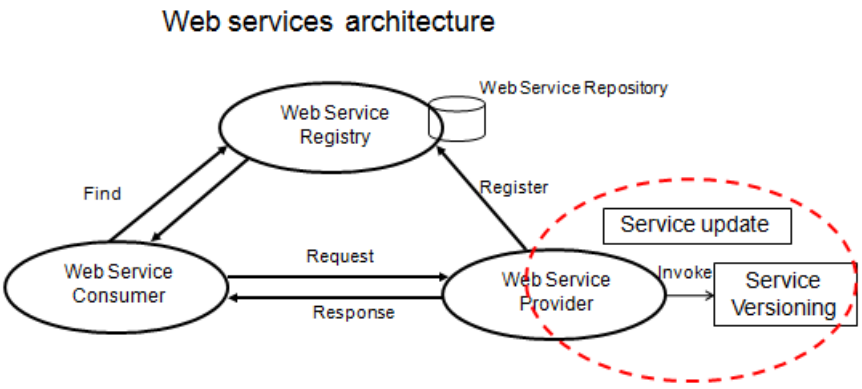


Figure 3.8: The Web Service Architecture extended with Service Update

The common practice is that only one version of a service is kept, and therefore all consumers only refer to the one and only version of the service. If there are new changes, the developers normally overwrite the earlier version. This gives a great advantage to consumers as only one fixed endpoint URL is maintained, thus, maintenance is greatly simplified. However, this is not a good practise as it makes the previous service versions become unavailable. The important issues are how to make versions of the same services available and how to call the appropriate endpoint URL based on the version number.

3.3.1 tModel Versioning Model

In order for experiments to be reproducible, a versioning model needs to consider how versions of services can remain available. Chapter 2 reviewed web service versioning approaches based on existing best practices. This section introduces the tModel concept

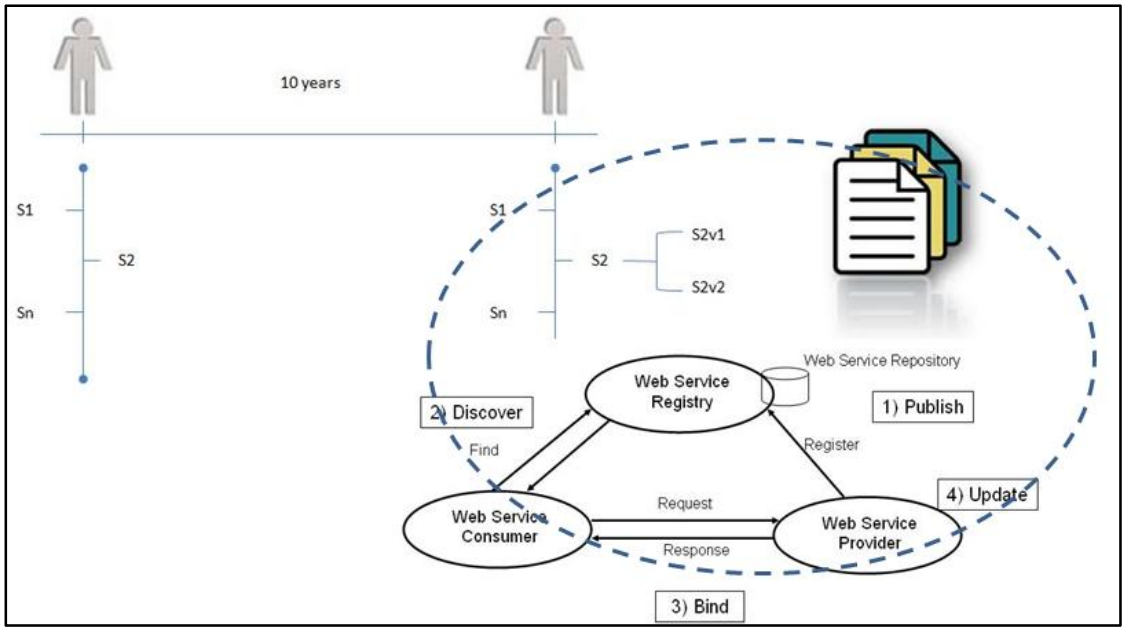


Figure 3.9: Two versions of S2; S2v1 and S2v2

and proposes how it can be used to create a service versioning model (and implementation) that describes how service versioning is kept in the jUDDI and how the multiple versions can be discovered and called. Recall the jUDDI concept described in Chapter 2. The idea of jUDDI is more or less similar to the concept of the common yellow pages where we can find information such as the telephone number and address of the service we want. In fact, jUDDI itself is known as an online yellow pages that is used by both service providers and service consumers. Therefore this section explains how the structure of tModel represents the interface of the Web Service. This will also include categorization information. The end of this section will discuss the advantages and disadvantages of using this tModel approach.

tModel is just like the XML namespace concept; each service interface in XML namespace is assigned to a unique name, whereas in tModel, each service interface is assigned to a unique tModel key [69]. XML namespaces were not used as they would create an entirely new Web Service with a new WSDL file and namespace for each version. This makes it hard to maintain a collection of versions of the same service.

Chapter 2 reviewed the relationship between jUDDI and WSDL, where WSDL is used to describe a web service and jUDDI is used to discover such web services. This

section draws attention to the jUDDI data types structure which service providers can register, and consumers can use to search for web services. There are four main data types in a jUDDI registry; businessEntity, businessService, bindingTemplate and tModel. Figure 3.10 shows the four jUDDI data types. jUDDI allows the publisher of a web service to register a businessEntity, which contains information on businessService, bindingTemplate and tModel.

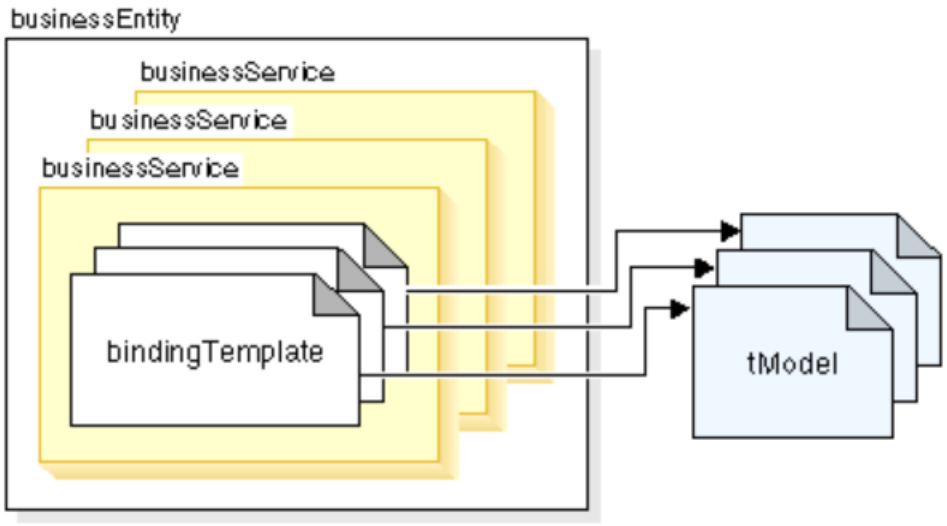


Figure 3.10: UDDI data types.

The first step when publishing to jUDDI is to create a single business description that can be linked to any number of service descriptions: this is called a business entity. This business entity relates to all services that are published under one specific business and organisation. In this example, it represents BMI. Having published a business entity for BMI, a unique key is generated for the business entity, together with a range of information including names, descriptions and discovery URL(s).

A businessEntity may contain zero or more businessServices that represent a specific service provided by the businessEntity. In each businessService instance, there are zero or more bindingTemplate data types that provide technical descriptions of the underlying service described. Each bindingTemplate must contain an accessPoint. The accessPoint in a bindingTemplate must specify the actual network address where the service can be accessed. The last data type in the jUDDI data structure is tModel. Each

tModel is uniquely referenced by a tModelKey, generated jUUID (Universal Unique Identifier) string. This key generation is automatically assigned by the jUDDI registry.

To understand exactly how the four data types are used to publish a service into jUDDI, recall the Exercise Advisor service presented in Section 3.1.1. It takes strings representing Height and Weight as inputs and returns a BMI Score. In order for the consumer to discover and use this service, the service must be published in the jUDDI first. To do this, the businessEntity, businessService and businessTemplate and tModel are created. In fact, when a service is published in the registry, a tModel is created to represent it. Listing 3.4 shows the tModel for the Calculate BMI Web Service.

Listing 3.4: The tModel representing the Calculate BMI Web Service

```
1  <tModel tModelKey="uddi:www.myeabmi10.com:keygenerator" deleted="false"
       xmlns="urn:uddi-org:api_v3" xmlns:ns2="http://www.w3.org/2000/09/
       xmldsig#">
2    <name xml:lang="en">Calculate BMI</name>
3    <description>This service is used to calculate Body Mass Index value.<
         /description>
4    <overviewDoc>
5      <overviewURL useType="text">http://localhost:8080/eabmi10/eabmi10?
           WSDL</overviewURL>
6    </overviewDoc>
7   <categoryBag>
8    <keyedReference tModelKey="uddi:uddi.org:categorization:types" keyName
         ="uddi-org:keyGenerator" keyValue="keyGenerator"/>
9   </categoryBag>
10 </tModel>
```

The tModel in jUDDI provides a reference to the corresponding WSDL document as illustrated in Figure 3.11 below. The figure shows the interface WSDL to interface tModel that means one WSDL corresponds to one tModel, unique to a particular version. WSDL is the description of a web service interface. A WSDL service portType and each operation within the WSDL service are captured using tModels in jUDDI. Once all this information has been added to the registry, the web service is ready to be discovered and can be consumed by web service consumers.

Let us imagine that years later, the same Exercise Advisor is updated from version 1 to version 2. Therefore, a new bindingTemplate should be created that contains a

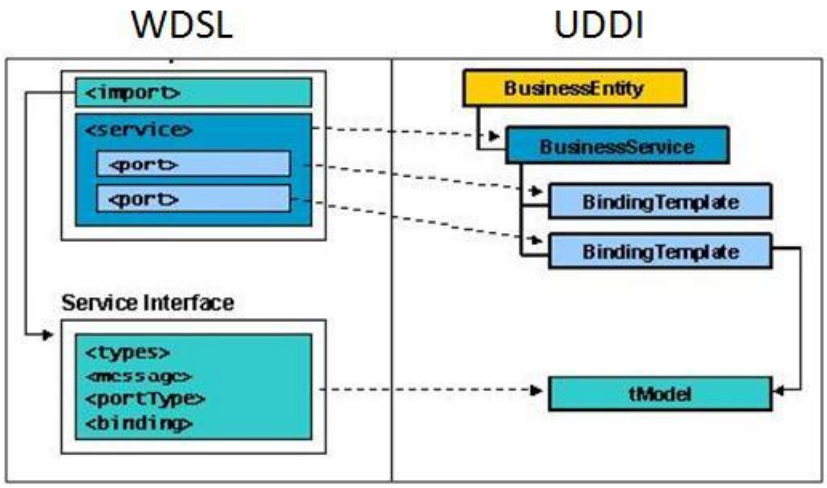


Figure 3.11: Relationship between WSDL and UDDI [7].

different tModel instance but still links with the same businessService. This feature is important as it allows multiple versions of a service to be grouped together under one businessService. The advantage is that otherwise consumers would not have any information on whether a service has several versions and which one to choose. Therefore searching for a specific version can be difficult. However, adding categorization information into the tModel can greatly improve the search functionality in UDDI.

A tModel categoryBag is like a bag that holds together all the interface tModel of the same businessService. Therefore when a service is updated and saved as a different version, all the existing versions can now be preserved and organised under the tModel. Therefore, the procedure for publishing and discovering versions of services is the same as described above.

How a service is published and discovered in jUDDI has been described here; another question arises over where to record such version information in OPM and how OPM can be referenced to jUDDI and WSDL.

3.3.2 Capturing Versioning in OPM

In this section, the focus is extending the current OPM to support versioning of web services. According to Vinoski [70], versioning is important because web services evolve over time due to many reasons. The reasons of evolvement are as mentioned in Section 2.2.1 (e.g. debugging, validity checking, updating).

In Section 2.2.3, an OPM model has been described as a model consisting of three main nodes and five types of edges representing the causal dependencies. The nodes as illustrated in Figure 2.5 denote the occurrences; artifact, process and agent. The edges are used to describe the causal relationship between the occurrences, for example how X is caused by Y. In this thesis, the focus is on web services, thus an extension of edges to incorporate the services versioning issues is proposed to be included in an OPM model. To recall, the OPM process node can also represent a service. Process and service have the same meaning, where both take input (artifact) and produce output (artifact). This extension is expressed by the attribution service metadata, for example when a particular service is created, what the version is and how the multiple versions of the same service are linked together as one collection.

In order to extend the current OPM edges is by taking the similar concept of an opm:wasDerivedBy edge that expresses the relationship from an artifact to another artifact. It describes an update of an artifact resulting to a new artifact. The derivation between the artifacts exists after performing or going through a process. This work is dealing with the derivation of services, an update of one service resulting to a new service.

Another edge type in OPM that involves process is opm:wasTriggeredBy edge that expresses the relationship between processes (services), where Service 1 is required to have started and completed in order to start Service 2. This condition differs from versioning, as the two different services may not hav been related to each other and may not have been referred to the same original service. Therefore, opm:wasTriggeredBy edge is not applicable for the case of versioning.

In web services, the services can develop from one service to another service. The two services refer to two different services which distinguished from each other but came from the original same service. Unfortunately, the representation of how the service was changed from one service version to the other version of service is not available. No current relation in OPM is defined to link the service versions, thus an extension of the edges type in OPM is required. This work introduces an extension of the edges type in causal dependencies with opm:wasVersionOf. It is believed that if there is a relationship that shows the dependency of the versions of a service, this will allow for future tracing.

The extension structure that incorporates versioning has three characteristics that describe the derivation for multiple versions of services of the original service. The

characteristics are described as follows:

- Each version is an enhancement that requires changes to a previous version of the same service.
- The next version of service is different from the previous service version, the expanding to the original service. This leads to the chain of services: Sv1 -> Sv2 -> Sv3 -> Sv4, the last is the latest version of the service as shown in Figure 3.12 below.
- A set of services, thus a collection. Extension of attribution of a causal relationship to provide further information on how one occurrence relate to previous occurrence.

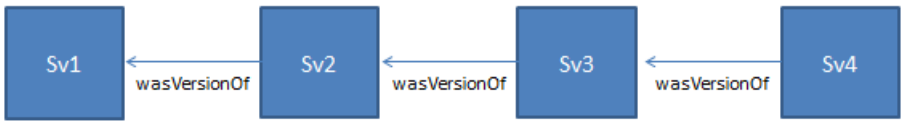


Figure 3.12: The model wasVersionOf edge

Each service can changed from time to time, thus we present it as different versions of that particular service. In this work, an OPM generator integrates with Service repository and Experiment repository as shown in Figure 3.13. Service repository contains information on wsdl and tModel that include service version information. The service version information includes date of service creation and service versioning naming that supports minor and major releases as described in Figure 3.6. Upon an execution run in a SOA system, the input and output data parameters are stored in Experiment repository.

By using the data from these two repositories, OPM Generator generates an OPM provenance trace. To generate wasVersionOf causal dependency in OPM trace, OPM Generator takes the service versioning naming and service creation date information from service repository to recommend the appropriate version of a service to be used.

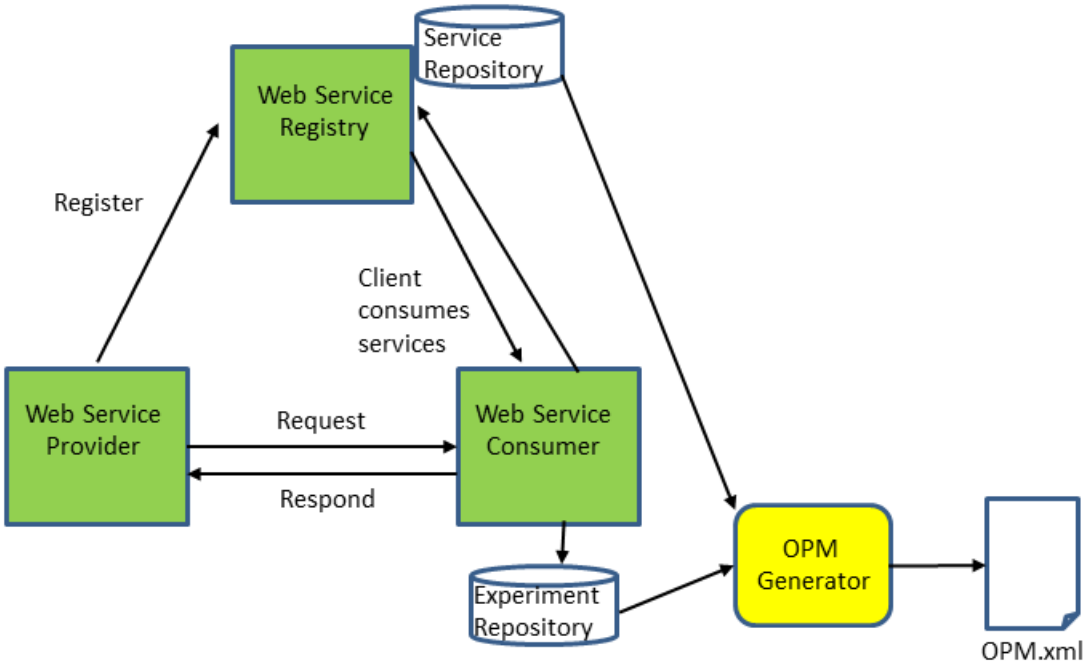


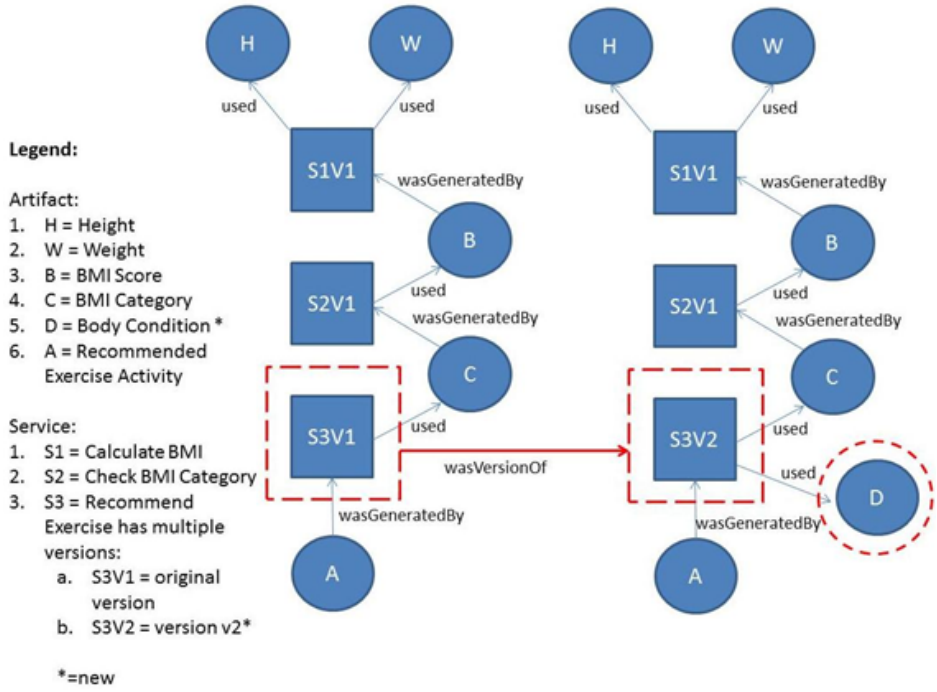
Figure 3.13: OPM Generator

OPM Generator will take alternate service that created prior to the services used during the execution run. If the service used is the first version, thus no prior version, therefore OPM Generator will take a service with the date of service creation greater than the service is used. The example of the OPM extension opm:wasVersionOf is described in the Figure 3.14(a).

In Figure 3.14 (b) is where wasVersionOf is shown, using the Exercise Advisor from in Figure 3.2, Section 3.1.1 as a running example. The example consists of using three services to calculate a person's Body Mass Index (BMI)(S1), check the category (S2) and recommend exercise activity (S3). The existing service, S3 is updated to a new version with added parameters. The S3 now has an updated version of S3v2. The OPM trace to illustrate the model of wasVersionOf for the S3 version 1 and the new S3 version 2 is presented in Listing 3.5. The wasVersionOf edge describes the derivation of two versions of the same service, namely myActivity1a is a newer version of myActivity1. The cause and effect explicitly describe the link between the two services based on the date of service creation. This information is essential to provide alternative service

- **Constraints:** No existing OPM edge of expressing the versioning relationship of one service to another service.
- **Proposed Approach:** An extension to have a new opm:wasVersionOf edge to express the link of service versions.
- **Description:** A service occurred and the service has changed from one service version to the other version of service.
- **Example:** The Service3V1 is opm:**wasVersionOf** Service3V2, thus the next version of service (Service3V2) is different from the previous service version (Service3V1). In other words, Service3V1 preceded or exist first beforeService3V2.

(a) The need for wasVersionOf edge



(b) A description of an execution run that shows the versioning relationship from one service S3v1 to another service S3v2

Figure 3.14: OPM description

which is the nearest version in case the current service is not available or missing. Thus, myActivity1a is an alternate service with the date of service creation greater than myActivity1. The details on how wasVersionOf is generated and how alternative service is selected are described in Chapter 4.

Listing 3.5: wasVersionOf in OPM trace

```
1       <opm:wasVersionedOf>
2           <opm:effect id="myActivity1a" />
3            <opm:role value="serviceVersion />
4       <opm:cause id="myActivity1" />
5         <opm:account id="account1" />
6       </opm:wasVersionOf>
```

However, the OPM extension of wasVersionOf approach is not yet supported in a workflow. However the mechanism for parallel experiment execution can be implemented in a workflow to allow user to compare the experiment results side by side, that will be explored further in future work.

Similarly, PROV model has also addressed a revision of an entity which the later entity is a revised version of an original entity. However, a revision attribute in PROV (wasRevisionOf) must be expressed as a subtype under Derivation, wasDerivedFrom relation. In other words, a revision is a kind of Derivation. In addition, the description of Derivation is only applies to entity, not the activity in PROV.

The provenance trace must describe the version of the service used in the execution. Using the tModel approach described in previous Section 3.3.1, one WSDL corresponds to one tModel. This means that the WSDL location in OPM trace uniquely indicates the specific version of the service used in the execution. A unique WSDL location is recorded that indicates a particular version of a service. Additionally, execution information providing a timestamp of each call to a service is recorded in OPM trace. As in jUDDI Registry, the timestamp of each service created is recorded. These time properties are essential as additional information to work out which version of the service was in used at the time of the service execution.

The features of the tModel have not previously been fully exploited in supporting provenance. Therefore, it is recommended that to achieve reproducibility, the service developer should register every new web service interface with jUDDI using the service versioning convention. By using tModel, the developer can now preserve the multiple versions of the same service. In addition to this work, there are other works that propose extensions on both WSDL and UDDI for version support in web services [60] Fang2007b Frank2008 Juric2009. In their works, they introduced an extension to WSDL structure to hold version information.

The main benefits of the tModel approach to supporting service versioning are:

- The tModel approach exploits the existing jUDDI registry standards and implementations.
- The tModel and its categorization feature facilitates the discovery of versions of a service.

3.4 Discussion

This chapter discussed how the Open Provenance Model is able to describe experiments. It has described the provenance content and structure of OPM using a provenance trace. This provenance trace is able to explain and reason about an experiment. Each experimental result has a provenance trace showing how the results were derived. A gap was noted in existing provenance systems in addressing the issue of service versioning. Additional information on versioning is needed to be recorded in OPM that is "wasVersionOf" for a comprehensive description of which version of services that the experiment used.

The tModel approach is described in detail to facilitate service publishing and discovery. Including the categorization information in tModel helps to preserve all versions of the same service and making it easier to discover and call the version of services accordingly. However, that is only possible if we are in control of creating and updating the services. For somebody on the consumer side, this is not possible. Therefore tModel name and time properties are introduced in OPM trace to make comparison of time at execution with time service created, can facilitate a service version discovery.

The following chapter describes the transformation from OPM model to a SCUFL model so that experiments can be reproduced.

Chapter 4

Transforming OPM to SCUFL

This chapter discusses the design of a framework to support the reproduction of experiments. During an execution of an experiment, a provenance trace records the processes that take place involving data and services. Chapter 3 has shown how a provenance trace is generated. In this chapter, the provenance trace is used to generate a workflow description that can reproduce a past experiment. This is to overcome the problem mentioned in Chapter 2 where the past experiment from the original experiment execution was executed in a different workflow environment from the one where provenance trace was reproduced. Provenance is captured in OPM and transformed into a workflow trace. The Taverna Simple Conceptual Unified Flow Language (SCUFL) model is chosen to be the target of the transformation. To achieve this, all information that is required for the execution of the experiment must be kept available, including datasets and versions of services. Chapter 3 described OPM, and it was shown that OPM can be used to capture the required provenance, so forming the basis for reproducibility.

In making the provenance trace executable, the reproducibility process includes taking a past experimental result captured in OPM, converting the OPM trace into the workflow format, in this case SCUFL, and then being able to re-execute the transformed file in Taverna. In this way, a past execution can be reproduced. The aims of this chapter are therefore to:

- Describe the mapping between the OPM (source) and SCUFL (target) models that allows reproducibility.
- Discuss how an algorithm to perform this mapping was designed and implemented.

It is noted that though Taverna is moving to t2flow schema, which is the updated version of SCUFL, the work in this thesis is still using the earlier version of SCUFL as this was current when the work began that is SCUFL or XSCUFL format in XML. SCUFL is used as the workflow structure is readable and understandable compared to t2flow. In t2flow, more complex information is covered. For the case of describing the concept in the work of this thesis, SCUFL schema is sufficient to be used as Taverna workflow schema for this work. The work on mapping SCUFL to OPM has been done during the Provenance Challenge workshop [48]. Missier and Goble [71] have successfully presented the two ways mapping from an execution of a Taverna workflow to OPM graph, W2G, and the generation of a Taverna workflow from OPM graph, G2W. In the paper, the authors provide the algorithm as mentioned above and highlight the two types of annotations in round-trip translation to prevent loses of information in the execution trace. Their works are in line to address the Third Provenance Challenge [48] which is to generate a Taverna workflow from an OPM graph. The work of this thesis share similar concept with [71] on provenance to workflow, however this thesis uses non-workflow experiment that developed in SOA system. On the contrary, this work focuses on the service versioning aspect of Web Services and uses OPM trace to capture the experiment run, handles service versioning and re-execute in Taverna Workflow Systems.

4.1 Generating SCUFL from OPM

Achieving reproducibility through the framework requires a transformation process from an OPM source document that undergo a mapping using rules that create the extraction and transformation algorithm that generates the SCUFL target document. The following sections of this chapter discuss techniques to enable the reproduction of an experiment captured in OPM format, which will be re-executed in the Taverna Workflow System as shown in Figure 4.1.

The four processes will be discussed in turn in the following sections. Before an OPM source document can be transformed to a SCUFL document for the Taverna Workflow System, it is essential to understand the SCUFL format for Taverna workflows. To do this, a comparison between entities in both target and source documents is drawn in the next section.

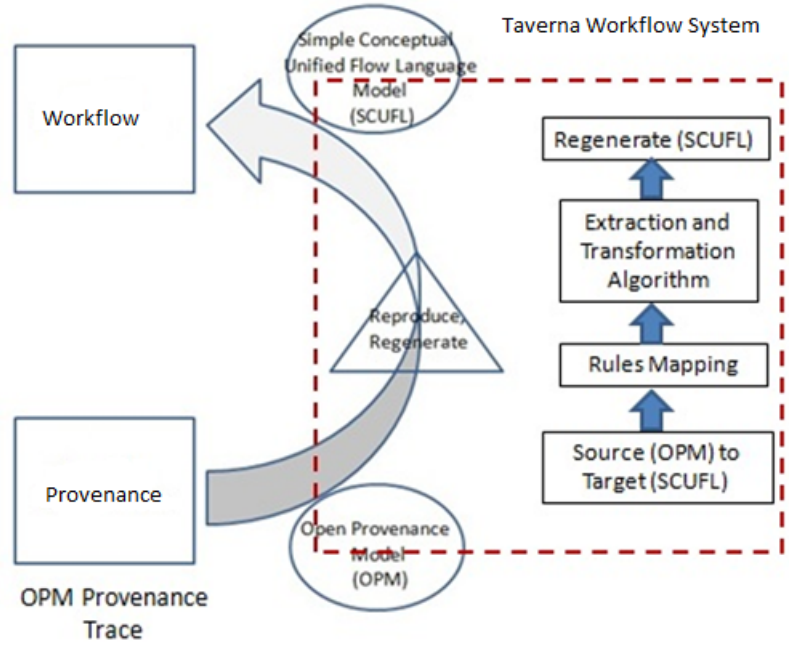


Figure 4.1: From OPM source document to SCUFL target document.

4.2 Comparing SCUFL to the Open Provenance Model (OPM)

In general, an experiment consists of data and service components, particularly input, process and output. Therefore both SCUFL and OPM models need to represent input(s), process(s) and output(s) in their data models. As described in Chapter 3, OPM is capable of representing an experimental run, modelling how input data is transformed to output results and recording the services that cause this to happen. As has been described, OPM does this by capturing the artifacts, processes, agents, and causal dependencies. In SCUFL, there are data sources, data sinks, processors, and data links. Table 4.1 shows the components that can describe an experiment, and their representation in the two models.

In OPM, artifacts fulfil the same purpose as data sources and data sinks in Taverna. The main distinction between SCUFL and OPM is in the way they each define relationships. In SCUFL, the relationship is defined by data links that have the following characteristics:

Table 4.1: The comparison between OPM and SCUFL entities

Component	SCUFL Entity	OPM Entity
Input	Data Sources	Cause: Artifacts, Processes, Agents
Process	Processes	Processes
Output	Data Sinks	Effect: Artifacts, Processes, Agents
Relationship	Data Links	Causal Dependencies

- A data source can be a workflow input or processor output port. For example, Height and Weight are workflow inputs.
- A data sink can be a workflow output or processor input port (which may be used by other processes). For example, BMI Score is a process output, and also used as input by another process Check Category.
- For each data link in SCUFL, data source(s) to data sink(s) are used to indicate the edge and links. However there is no causality relation in SCUFL that indicates the relations between data, or process to another process.

An example is given of how both SCUFL and OPM are used in the BMI Calculation example of Figure 3.2 in Section 3.1.1. Based on the comparison of both models, it is clear that the relationships representing data links in SCUFL convey the same meaning as causal dependencies in OPM, as depicted in Table 4.2. However SCUFL does not provide the causality relation that could give the meanings of the links for a better understanding when compared to OPM.

Table 4.2: A comparison between SCUFL data links and OPM causal dependencies.

SCUFL Data Links	**OPM Causal Dependencies Equivalent**	**Example Usage from the Exercise Advisor example**
Data source can be a workflow input	Type 1: used A process used an artifact	SCUFL: <s:link **source="Height"** sink="CalculateBMI" /> OPM: CalculateBMI used Height
Data sink can be a processor input port	Type 1:used A process used an artifact	SCUFL: <s:link source="BMI Score" **sink="CheckCategory"** /> OPM: CheckCategory used BMI Score
Data source can be a processor output port	Type 2: wasGeneratedBy An artifact was generated by a process	SCUFL: <s:link **source= "CalculateBMI"** sink="BMI Score" /> OPM: BMI Score was generated by CalculateBMI
Data sink can be a workflow output	Type 2: wasGeneratedBy An artifact was generated by a process	SCUFL: <s:link source= "CheckCategory" **sink="Category"**/> OPM: Category was generated by CheckCategory
Link from a process to another process	Type 3: wasTriggeredBy A process was triggered by another process.	SCUFL: <s:link **source= "CalculateBMI"** **sink=" CheckCategory"**/> OPM: CheckCategory process was triggered by CalculateBMI process.

4.3 Rules for mapping from OPM to SCUFL

When transforming from OPM to SCUFL, the same concepts are modelled in different ways. Understanding the two data models, as described in the Section 4.2, enables conversion of the execution description captured in the OPM format to generate a workflow description in the SCUFL format.

A set of mapping rules that maps the OPM description into SCUFL was designed based on entity mapping. The mapping uses semantic interpretation to bind the entities to a similar concept that has the same meaning. Semantics is the interpretation of an object according to a human understanding of the real world [72]. For example, "client" from a source schema is equivalent to "customer" from a target schema. Therefore instances of "client" in the source can be reused as instances for "customer" in the target. Therefore a mapping is required to perform such matching. This is used to achieve the extraction and transformation of information related to data and services: input, process, outputs and relationships.

In order to transform an OPM schema intoa Taverna Workflow schema, the following rules are used. There are two types of rules:

- **Entity Rules:** are rules based on the semantic interpretation, focusing on the meaning of entities in OPM XML schema and SCUFL XML schema. Therefore, the mapping is compatible if each entity from OPM carries the same meaning as the entity in SCUFL to which it is transformed
- **Relation Rules:** are rules based on relationships between the entities.

These rules are described in the following sections.

4.3.1 Entity Rules OPM to SCUFL

In entity rules, there are two things to highlight; each entity involved in the OPM, and the value of the entity.

The Entity Rules (ER) are defined as following:

ER1: For each entity in OPM, create an instance of a class representing the entity of the corresponding entity in SCUFL.

ER2: The value in an OPM entity is mapped to the value of an assigned entity in SCUFL.

Based on Entity Rules, the replacements of the entities in the transformation from OPM to SCUFL are based on semantic interpretation. The following tables, Tables 4.3 and 4.4 describe the production of SCUFL from entities and values.

Table 4.3: Generating SCUFL entities based on Rule ER1

Entities from source schema (OPM XML)	**Entities to target schema (SCUFL XML)**	**Example**
Artifactid	SourceName,SinkName	OPM: <opm:artifact id="Height">, <opm:artifact id="BMI Score"> SCUFL: <s:source name="Height">, <s:sink name="BMI Score">
Processid	ProcessorName	OPM: <opm:process id="CalculateBMI"> SCUFL: <s:processor name="CalculateBMI">

Table 4.4: Generating SCUFL entity values (ER2)

Values in entities from source schema (OPM XML)	Values in entities to target schema (SCUFL XML)	Example
Processvalue	ProcessorValue	OPM: <opm:value xsi:type="xsd:string">"http://localhost:8080 /eabmi10/eabmi10?wsdl"</opm:value> SCUFL: <s:wsdl>http://localhost:8080/eabmi10 /eabmi10?wsdl</s:wsdl>

Related to the above Tables 4.3 and 4.4 , the example of the generated SCUFL file is shown in Listing 4.1 and 4.2. In this OPM trace, myBmi process is equivalent to Calculate BMI process, myCategory is equivalent to Check BMI Category and myActivity1 is equivalent to Recommend Exercise Activity.

Listing 4.1: Process in OPM is replaced by

```
1  <opm:processes>
2   <opm:process id="myBmi">
3    <opm:value xsi:type="xsd:string">"http://localhost:8080/eabmi10/eabmi10
         ?wsdl"</opm:value>
4    <opm:time exactlyAt="2014-10-06T04-46-27.447Z"/>
5   </opm:process>
6   <opm:process id="myCategory">
7    <opm:value xsi:type="xsd:string">"http://localhost:8080/eacategory10/
         eacategory10?wsdl"</opm:value>
8    <opm:time exactlyAt="2014-10-06T04-46-28.068Z"/>
9   </opm:process>
10  <opm:process id="myActivity1">
11   <opm:value xsi:type="xsd:string">"http://localhost:8080/eaactivity10/
         eaactivity10?wsdl"</opm:value>
12   <opm:time exactlyAt="2014-10-06T04-46-29.922Z"/>
13  </opm:process>
14 </opm:processes>
```

Listing 4.2: Processor in SCUFL

```
1  <s:processor name="myBmi">
2   <s:arbitrarywsdl>
3    <s:wsdl>
4     http://localhost:8080/eabmi10/eabmi10?wsdl
5    </s:wsdl>
6    <s:operation>myBmi</s:operation>
7   </s:arbitrarywsdl>
8  </s:processor>
9  <s:processor name="myCategory">
10  <s:arbitrarywsdl>
11   <s:wsdl>
12    http://localhost:8080/eacategory10/eacategory10?wsdl
13   </s:wsdl>
14   <s:operation>myCategory</s:operation>
15  </s:arbitrarywsdl>
16 </s:processor>
17 <s:processor name="myActivity1">
18  <s:arbitrarywsdl>
19   <s:wsdl>
20    http://localhost:8080/eaactivity10/eaactivity10?wsdl
21   </s:wsdl>
22   <s:operation>myActivity1</s:operation>
23  </s:arbitrarywsdl>
24 </s:processor>
```

In Listing 4.1, the Process in OPM (Line 1 to Line 5) is transformed to Processor in SCUFL (Line 1 to Line 8) in Listing 4.2. This transformation is based on the Entity Rules, ER1 and ER2 described earlier.

4.3.2 Relation Rules OPM to SCUFL

After entity rules are processed, relation rules translate the relationships from OPM to SCUFL. Relationships are represented differently in both schemas. OPM relations are given the name causal dependencies, whereas in SCUFL they are data links. The Relation Rules process involves identifying the cause and effect relation in OPM and the linkage between them. Hence, the SCUFL XML needs to be analysed to identify all causal dependency relationships from the OPM file. To translate the cause and effect relations appropriately, the rules are described in Listings 4.3 to 4.5.

Listing 4.3: Relation Rule 1 (RR1) that generates data links in SCUFL from causal dependency used in OPM

```
RR1:
 for all used <P1,a1> do
   Check Causal Dependency
   If effect in OPM is Processid then
       Processid becomes data sink in SCUFL
   If cause in OPM is Artifactid then
      Artifactid becomes data source in SCUFL
   endif
endfor
```

Based on the rules in Listing 4.3, OPM can be translated into SCUFL. Table 4.5 describes the production of SCUFL from the causal dependency *used* in OPM.

Table 4.5: Generate Data Links in SCUFL from the Causal Dependency used in OPM

Causal Dependencies source schema (OPM)	**Data links target schema(SCUFL)**		**Example**
	Data source (SCUFL)	**Data sink (SCUFL)**	
1) used Cause: Artifactid Effect: Processid	Artifactid	Processid	OPM: <opm:effect id="CalculateBMI"/> <opm:cause id="Height"/> SCUFL: <s:link source="Height" sink="CalculateBMI" />

In a *used* relation, a process *used* an artifact to perform a computation. This means the artifact(s) are required for the process to complete. In SCUFL data links, there is the data source and data sink. A used relation will take an element *artifactid* (cause) from OPM which becomes a data source in a SCUFL data link, while the element *processid* (effect) in OPM becomes a data sink in a SCUFL data link. This is based on the first Relation Rule (RR1).
The second Relation Rule is as depicted in Listing 4.4.

Listing 4.4: Relation Rule 2 (RR2) that generates data links in SCUFL from causal dependency wasGeneratedBy in OPM

```
RR2:
 for all wasGeneratedBy <a1,P1> do
   Check Causal Dependency
   If effect in OPM is Artifactid then
       Artifactid becomes data sink in SCUFL
   If cause in OPM is Processid then
      Processid becomes data source in SCUFL
   endif
endfor
```

Based on the RR2 in Listing 4.4, Table 4.6 describes the production of data links in SCUFL from causal dependency *wasGeneratedBy* in OPM.

Table 4.6: Generate Data Links in SCUFL from Causal Dependency wasGeneratedBy in OPM

Causal Dependencies source schema (OPM)	**Data links target schema(SCUFL)**		**Example**
	Data source (SCUFL)	**Data sink (SCUFL)**	
2) wasGenerated By Cause: Processid Effect: ArtifactId	Processed	Artifactid	OPM: <opm:effect id="BMI"/> <opm:cause id="CalculateBMI"/> SCUFL: <s:link source="CalculateBMI" sink="BMI Score" />

In a *wasGeneratedBy* relation, an artifact *wasGeneratedBy* a process. The artifact can be understood as the data output of the process. In SCUFL data links, this relation will take an element *processid* (cause) from OPM : it becomes a data source for a SCUFL data link, while an element *artifactid* (effect) from OPM becomes a data sink in a SCUFL data link. The second Relation Rule (RR2) is applied here.
The third Relation Rule focuses on wasTriggeredBy as shown in Listing 4.5.

Listing 4.5: Relation Rule 3 (RR3) that generates data links in SCUFL from causal dependency used in OPM

```
RR3:
 for all wasTriggeredBy <P1,P2> do
   Check Causal Dependency
   If effect in OPM is Processid then
       Processid becomes data sink in SCUFL
   If cause in OPM is Processid then
      Processid becomes data source in SCUFL
   endif
endfor
```

Referring to Listing 4.5, Table 4.7 describes the production of data links in SCUFL from causal dependency *wasTriggeredBy* in OPM.

Table 4.7: Generate Data Links in SCUFL from Causal Dependency wasTriggeredBy in OPM

Causal Dependencies source schema (OPM)	**Data links target schema(SCUFL)**		**Example**
	Data source (SCUFL)	**Data sink (SCUFL)**	
3) wasTriggeredBy Cause: Processid Effect: Processid	Processed	Processed	OPM: <opm:effect id="CheckCategory"/> <opm:cause id="CalculateBMI"/> SCUFL: <s:link source="CalculateBMI" sink=" CheckCategory " />

In a *wasTriggeredBy* relation, a process *wasTriggeredBy* another process. The relationship shows that one process (P1) can activate another process (P2). P1 needs to start in order for P2 to start or to be completed. In a SCUFL data link, this relation will take an element *processid* (cause) from OPM, it becomes a data source in a SCUFL data link, while an element *proccessid* (effect) in OPM becomes a data sink in a SCUFL data link.

For the above mentioned Relation Rule, one example of how a SCUFL file is generated based on the causal dependency in OPM is shown in Listing 4.6.

Listing 4.6: Causal Dependency used in OPM

```
<opm:causaldependencies>
  <opm:used id="u_1">
     <opm:effect id="CalculateBmi"/>
     <opm:role id="r_1" value="1"/>
     <opm:cause id="height"/>
     <opm:account id="account1"/>
     <opm:time exactlyAt="2014-10-06 12:46:30:PM"/>
  </opm:used>
  <opm:used id="u_2">
     <opm:effect id="CalculateBmi"/>
     <opm:role id="r_2" value="2"/>
     <opm:cause id="weight"/>
     <opm:account id="account1"/>
     <opm:time exactlyAt="2014-10-06 12:46:30:PM"/>
  </opm:used>
</opm:causaldependencies>
```

Listing 4.7: Data source and data sink in SCUFL

```
1    <s:link source="height" sink="parametersXML1:height" />
2    <s:link source="weight" sink="parametersXML1:weight" />
3    <s:link source="parametersXML1:output" sink="CalculateBmi:parameters" /
         >
```

Listing 4.6 and 4.7 show how cause and effect under *used* causal dependency are represented by data source and data sink in SCUFL. This is generated based on the first Relation Rule, RR1 as mentioned earlier.

4.4 Generation of OPM to Taverna Workflow

Workflow language schema is updated from time to time to cater for new workflow requirements. For example, Taverna Workflow has initially used SCUFL and moved to t2flow. Therefore, this section addressed a generic Taverna workflow schema and the algorithm that can be used to drive the generation of Taverna workflow from OPM that does not involved SCUFL, as described in previous section. A generic Taverna Workflow contains processors, each having input and output ports, and datalinks to connect from a port of processor to another port. This work shares similar concept with [71] on generation of provenance to workflow. The causal dependencies from OPM can be used to generate Taverna Workflow. Listing 4.8 shows the pseudo-code for generation of Taverna workflows from OPM.

Listing 4.8: Generation of generic Taverna workflow from OPM

```
1 for all used <P1,a1> do
2    P1 is added to the set of Processor and their input ports
3    Check Causal Dependency -> for identifying DataLink (inputPort,
         outputPort)
4    If effect in OPM is Processid then
5        Processid becomes outputPort
6    If cause in OPM is Artifactid then
7       Artifactid becomes inputPort
8    endif
9 endfor
10 for all wasGeneratedBy <a2,P1> do
11     P1 is added to the set of Processor and their output ports
12    Check Causal Dependency -> for identifying DataLink (inputPort,
         outputPort)
```

```
13    If effect in OPM is Artifactid then
14        Artifactid becomes outputPort
15    If cause in OPM is Processid then
16       Processid becomes inputPort
17    endif
18 endfor
```

In Listing 4.8 Line 2 and 11, an instance of P1 in a set of Processor will not be duplicated.

4.5 The ReProduX Extraction and Transformation Algorithm

ReProduX consists of OPM Generator, OPM2Taverna Generator and OPM Trace Browser as illustrated in Figure 4.2. OPM Generator as described in Section 3.3.2, connected to two repositories namely Service repository and Experiment repository. The service repository is used to store all versions of services, and the Experiment repository is used to store all the information that should be captured as part of an opm provenance trace. OPM Generator will access these two repositories and generate an OPM provenance trace. Then, the OPM2Taverna Generator that contains an algorithm will extract entities from OPM and transforms them to Taverna Workflow. From this point, OPM2Taverna Generator also represents any Taverna Workflow schema including the initial Taverna schema that is SCUFL. Thus, this generator is written based on SCUFL Entity Rules and Relation Rules described in Section 4.3 and Section 4.4.

ReProduX has three approaches to handle service versioning as follows:

- Approach 1: Reproduce exactly using the same version of service(s) as documented in past OPM provenance trace, which is the original experiment execution trace.
- Approach 2: For the inaccessible services due to service update and missing, ReProduX will recommend the nearest service available to enable reproducibility.
- Approach 3: ReProduX allows user to choose a selection of services of their choice. The user can select any version of that service before generating a new OPM trace that can be used to reproduce exactly as per Approach 1.

The OPM2Taverna pseudo-code is depicted as in Listing 4.9:

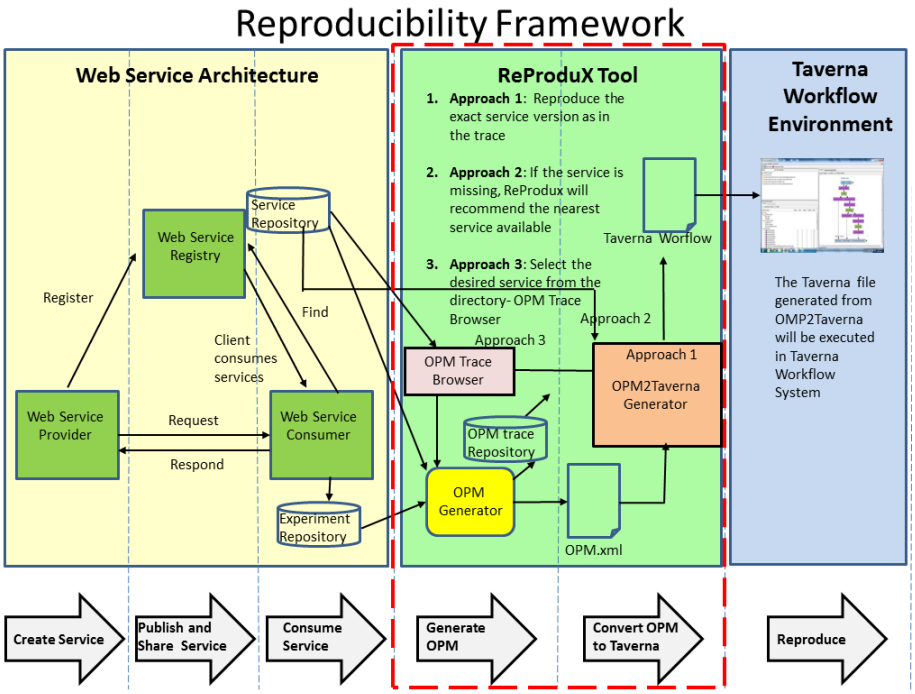


Figure 4.2: ReProduX Tool

Listing 4.9: OPM2Taverna Algorithm

```
1  Load OPM file
2  Parse the content of OPM file using DocumentBuilder
3  Store the parsed content to Document object
4  Extract OPM process parameters from Document object
5  Extract OPM used parameters from Document object
6  Extract OPM wasGeneratedBy parameters from Document object
7  Check OPM wasVersionOf parameters from Document object
8  Create SCUFL format and write in the parameters
9  If OPM processes are not available in Service Registry then
10   Find and Write in the process
11 endif
12 Write link source ans sink into SCUFL
```

In Line 4, 5 and 6, the rule in ER and RR are followed. In Listing 4.9 at Line 8, the SCUFL format is created. In this conversion work, Taverna requires XMLInputSplitter and XMLOutputSplitter in order for it to be able to process the input and provide output value. Therefore, a conversion of OPM trace into SCUFL workflow is to include these splitters as depicted in Figure 4.3.

```
private void writeScuflFormat() {

  int count = 0;
  String indent = "  ";
  String xmlInputSplitter = "org.embl.ebi.escience.scuflworkers.java.XMLInputSplitter";
  String xmlOutputSplitter = "org.embl.ebi.escience.scuflworkers.java.XMLOutputSplitter";
```

Figure 4.3: XML Splitters in SCUFL.

The splitters are added as processor whenever ReProduX encounters input, processes and output. The OPM trace is then used as an input of ReProduX which converts it into a SCUFL workflow containing the required information about the past experiment.

If there is wasVersionOf in the OPM trace, ReProduX algorithm will check whether the services are available in the Service repository. By default ReProduX will take the current service if it is available in the wasVersionOf dependency, for instance opm:effect id="myActivity1a". Otherwise ReProduX will use the alternate service version as shown in opm:case id="myActivity1" in Listing 3.5 in Chapter 3.

If both the services are not available or missing, then OPM2Taverna will recommend the alternative service by checking the Service repository as shown in Figure 4.4. In Listing 4.9 at Line 8-9, this is where the ReProduX Approach 2 is applied. OPM2Taverna will search for alternate service that created prior to the service used in the trace. However, if the service in the trace is the first version, therefore OPM2Taverna will select an alternate service with the date of service creation greater than the service in used. That is why service time creation is important in this algorithm.

OPM2Taverna will not recommend for the latest version, as this work believes the nearest version is the best candidate of service that is similar to the missing service. The latest service may differ a lot compared to the nearest service. Nevertheless, ReProduX also provides Approach 3, the ability to select specific service version of choice using a browser. In addition, user can also view the service browser if the user is interested to get the latest service and substitute the service with the current one.

In Listing 4.9 at Line 11, SCUFL file is successfully generated and can proceed to re-execution as described in next sub-section 4.5.1.

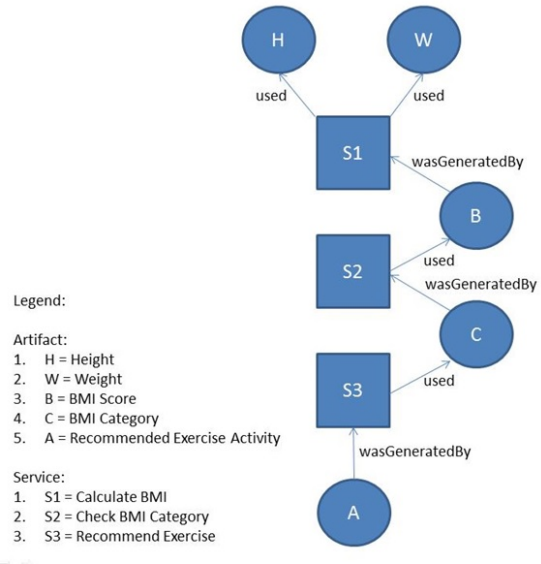
```
for(i = 0 ; i < vProcess.size(); ++i) {
    myOpmProcess = vProcess.elementAt(i);
    myOpmProcess.display();

    serverURL = myOpmProcess.getProcessValue();
    System.out.println("1) serverURL: " + serverURL);
    serverURLAvailable = searchUDDIRegistry(serverURL); // here will check service is
    availabe or not

    if(serverURLAvailable == 0) { // serverURL not available
        // suggest next possible service
        System.out.println(serverURL);
        System.out.println("2) serverURL pattern: " + serverURL.substring(0, serverURL.
        lastIndexOf("/") + 1));
        String nextAvailableVersion = searchNextAvailableVersion(serverURL);

        if (nextAvailableVersion.trim().length() > 0) {
            serverURL = nextAvailableVersion;
        } else {
            serverURL = "";
        }
```

Figure 4.4: ReProdux algorithm to check service availability.

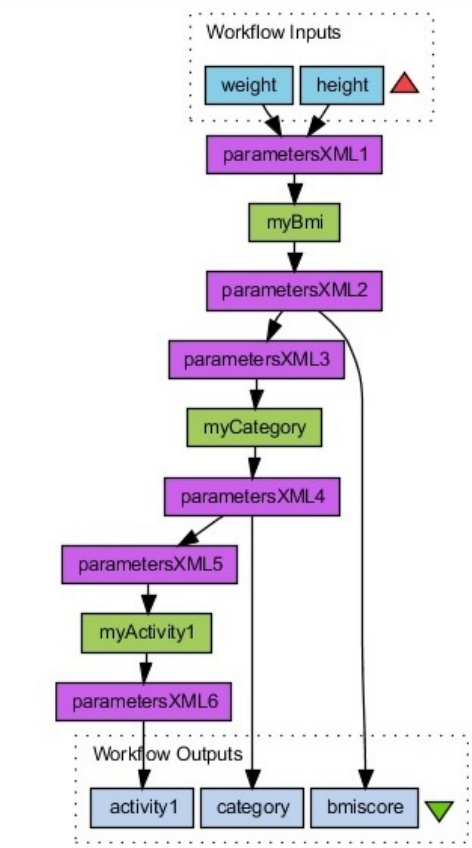
4.5.1 The Transformation and Execution

In ReProduX transformation, a set of OPM properties undergo a sequence of transformations through the OPM2Taverna mapping to produce a Taverna file. The generated SCUFL or Taverna file can then be loaded into the Taverna Workflow System and the past experiment successfully re-executed. Figure 4.5 (a) and (b) shows the graphical representation of the OPM and SCUFL in Taverna. In Taverna, the diagram shows the boxes that represent the components of SCUFL: Inputs, Outputs and Processes. The linkages are represented as lines with arrowheads.

Figure 4.6 shows how ReProduX that is built in this work will parse the source document OPM) and extract their content as well as their structure and then transform the source document into a target document (SCUFL). Then the SCUFL file is used to re-execute the workflow of past experiment using the Taverna Workflow System.



(a) Provenance for Exercise Advisor example in OPM



(b) SCUFL generated by ReProduX, visualised in the Taverna Workbench

Figure 4.5: Graphical representation of the OPM and SCUFL in Taverna

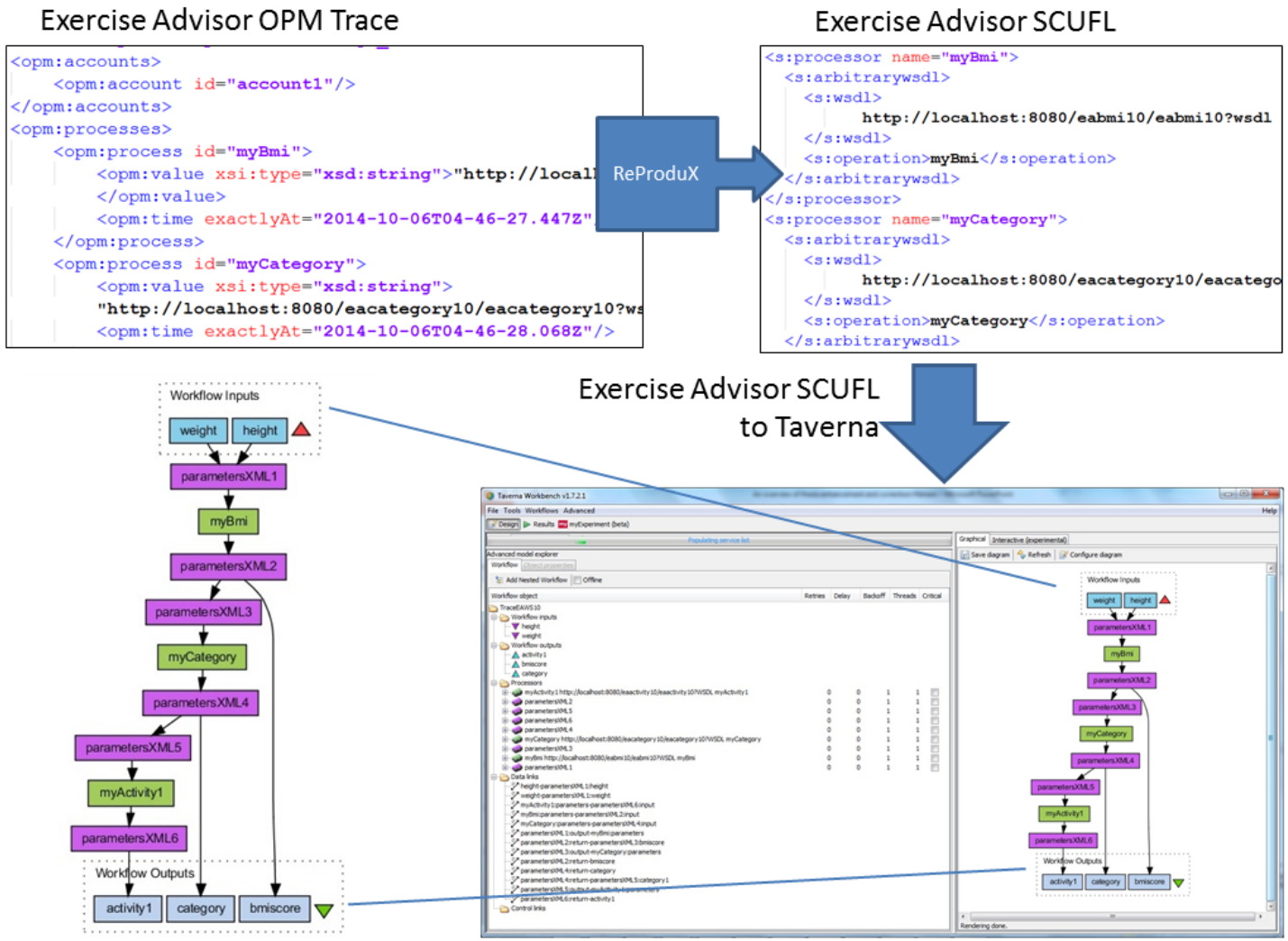


Figure 4.6: The re-execution in Taverna based on SCUFL file generated from OPM2Taverna.

4.6 Discussion

This chapter has investigated the transformation from OPM to SCUFL and OPM to Taverna Workflow (does not involve SCUFL) in order to support the reproduction of a previous experimental result. The contribution of this chapter is the rules for mapping OPM to SCUFL. This transformation can be applied to OPM XML documents and takes into account service versioning information captured in a previous workflow execution.

The transformation included both Entity Rules and Relation Rules with recommendation of services are keys for capturing experiments. The next chapter evaluates the success of the approach to re-execute past experiments by running a series of examples through the transformation algorithm.

Chapter 5

Evaluation of the Reproducibility Framework

This chapter evaluates the contributions introduced in Chapters 3 and 4. The goal of this research has been to design a reproducibility framework that is capable of reproducing e-experiments, in particular by being able to record the specific versions of services used in the experiments, and then call the correct version during reproduction. Chapter 2 showed that consideration of service versioning is missing from current studies.

To evaluate the ideas presented earlier, it was necessary to design and implement a system that can reproduce e-experiments. This chapter describes the design, implementation and the subsequent evaluation.

For this implementation, services are created in NetBeans , published in a jUDDI registry [73], and consumed by a Web Service Consumer in NetBeans. In this work, OPM Generator is created to generate a provenance of the execution (provenance trace) in OPM based format, transformed using ReProduX (an OPM to SCUFL Converter) into SCUFL format and loaded into the Taverna Workflow System to be reproduced. Figure 5.1 below illustrates the Reproducibility Framework evaluated in this work.

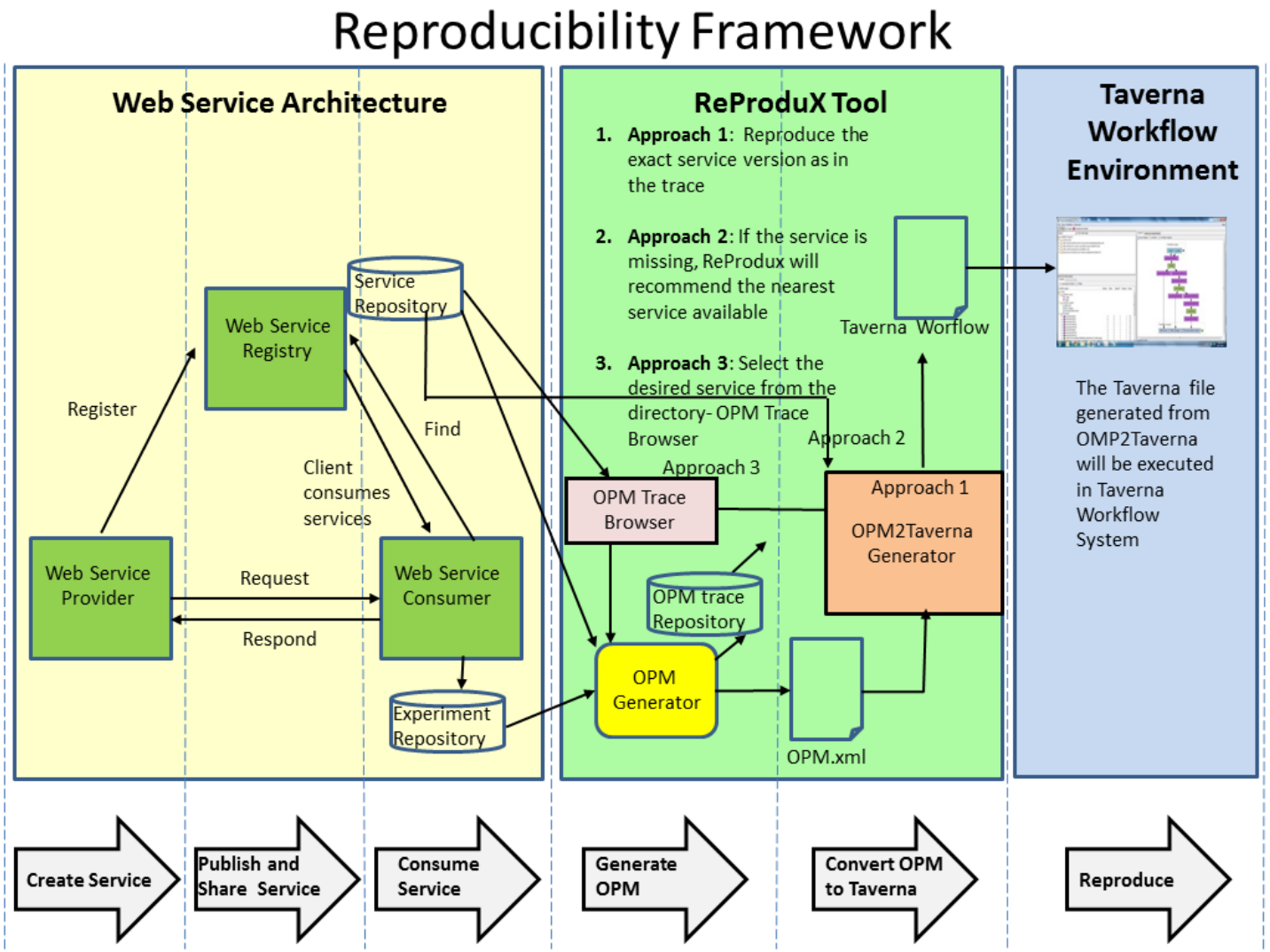


Figure 5.1: Reproducibility Framework

In order to demonstrate the proposed approach, two types of application are used. The first is the Exercise Advisor application with data and services built specifically to explore the reproducibility system. The second uses publically available web services from myExperiment repository, and so explores how the system works when the services are not in the control of the user attempting to run and reproduce the experiments.

5.1 Implementation of Reproducibility Framework

In this section, the implementation of the Reproducibility Framework concept that consists of three components; the web service architecture to compose, register and consume services, ReProduX tool to call the correct version of a service during reproduction and also to convert OPM to SCUFL, and Taverna Workflow Environment to re-execute the past experiment, is explained. The following sub-sections, will in turn describe the implementation steps that is depicted in the Figure 5.1, as follows:

Service Architecture

- Create new service version using service versioning convention
- Publish and share service into jUDDI registry
- Consume a new service version

ReProduX tool

- Generate OPM trace
- Convert OPM trace to SCUFL using ReProduX

Taverna Workflow Environment

- Reproduce in Taverna

5.1.1 Versioned Service Deployment, Publication and Consumption in a Web Service Architecture

The application scenario below describes the interactions between the Web Service Provider, Web Service Registry and Web Service Consumer as illustrated in the red dashed line in Figure 5.2.

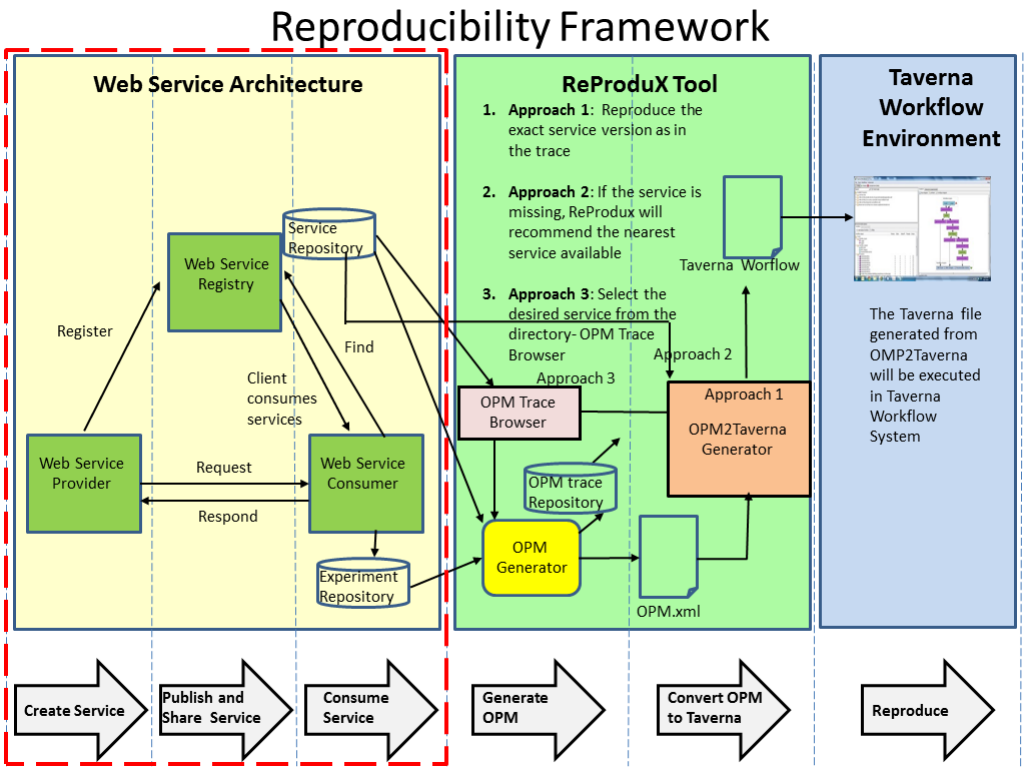


Figure 5.2: Web Service Architecture

The NetBeans IDE is used by Web Service Provider to create, edit, and deploy the Exercise Advisor web service to a GlassFish application server. GlassFish server is used to manage and expose web services over HTTP. It handles requests and responses to those services packaged in SOAP messages. As discussed in service versioning concept in Chapter 3, the provider will use the service versioning convention scheme when creating new version of the same service, taking into account minor and major release of a service, as shown in Figure 5.3. When the web services have been deployed, the WSDL description of the web services is generated.

Once the services have been deployed, the web service provider will then register the services into the jUDDI, a web service registry with a unique tModel, where each service interface is assigned to a unique tModel key. In jUDDI, there is no direct support of storing multiple versions of same service. Therefore, the service versioning convention is used and the version description of a service is stored as in Figure 5.4 and 5.5. Therefore,

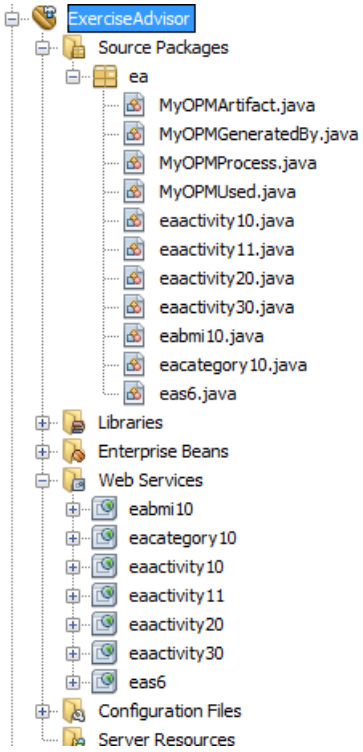


Figure 5.3: Web Services Project in NetBeans

in order to overcome the external services may become inaccessible, therefore in this work, all versions of services are automatically retained in a jUDDI service repository. In regards to the service versioning information, there are four tables used to capture version information. The required tables are depicted in Figure 5.5.

Versions of services in this database can be viewed from the following jUDDI Registry Browser in Figure 5.6. This browser is specifically created for this implementation work to enable the consumer to view the available services along with service information including service name, service description, service version, service wsdl path and when the service was created or last modified.

The consumer searches the jUDDI registry browser to discover the available web

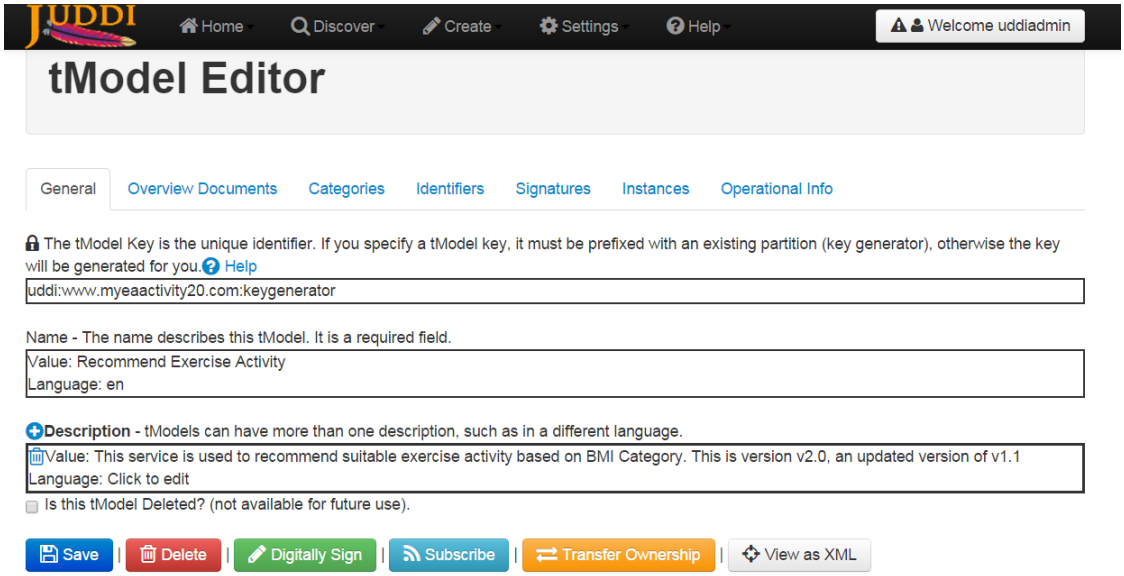


Figure 5.4: jUDDI tModel editor on service information

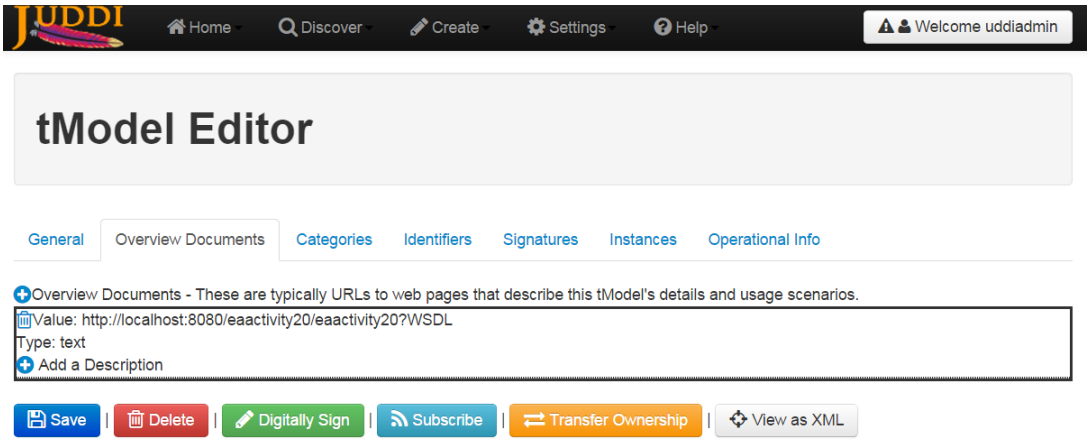


Figure 5.5: jUDDI tModel wsdl information

services of his or her choice. In this work, the specifically created Exercise Advisor web service will be consumed by a Web Service Consumer in order to generate the required result. As discussed in Section 5.1 Implementation, the created Exercise Advisor web services is published into the jUDDI repository and a GlassFish Web Server hosted the web service by listening to HTTP traffic [69].

In order to consume the web service, NetBeans is used to create a Web Service Consumer application, as shown in Figure 5.8.The WSDL document is then accessed

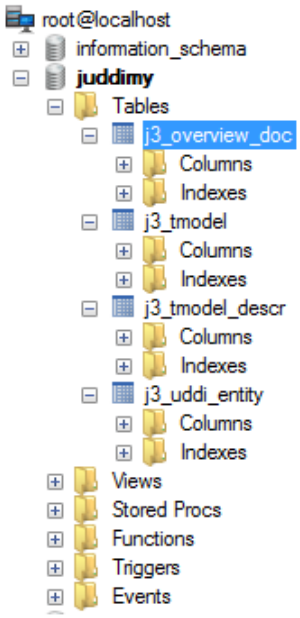


Figure 5.6: jUDDI Tables in MySQL

jUDDI Registry Browser

Service: Recommend Exercise Activity Search

No	Service Name	Service Description	Service Version	Service WSDL Path	Service Last Modified
1	Recommend Exercise Activity	This service is used to recommend suitable exercise activity based on BMI Category.	eaactivity10	http://localhost:8080/eaactivity10/eaactivity10?WSDL	2014-09-17 23:02:43
2	Recommend Exercise Activity	This service is used to recommend suitable exercise activity based on BMI Category. This is version v1.1, an updated version of v1.0	eaactivity11	http://localhost:8080/eaactivity11/eaactivity11?WSDL	2014-09-17 23:03:50
3	Recommend Exercise Activity	This service is used to recommend suitable exercise activity based on BMI Category. This is version v2.0, an updated version of v1.1	eaactivity20	http://localhost:8080/eaactivity20/eaactivity20?WSDL	2014-09-17 23:04:38
4	Recommend Exercise Activity	This service is used to recommend suitable exercise activity based on BMI Category. This is version v3.0, an updated version of v2.0	eaactivity30	http://localhost:8080/eaactivity30/eaactivity30?WSDL	2014-09-17 23:05:14

Figure 5.7: jUDDI Registry Browser displaying available services from Registry

by the Web Service Consumer and a SOAP request message is generated. The SOAP request is then received by the web service and a SOAP response is created and sent to the GlassFish Web Server. The Web Service result is then returned to the web service consumer via a HTTP response that includes the SOAP response.

Figure 5.8: Client Application Interface

```
1  Service Name:myBmi
2  height:1.6
3  weight:80.0
4  bmiscore:31.25
5
6  Service Name:myCategory
7  bmiscore:31.25
8  category:MODERATELY OBESE
9
10 Service Name:myActivity
11 category:MODERATELY OBESE
12 bodycondition:ASTHMA
13 dailyFreeTime:30
14 activity:Easy gym cyling (indoor) for 30 mins per day, 3 times per week
```

Figure 5.9: Input and Output Data Parameters of Exercise Advisor Execution

Upon the successful execution of the client application interface, the input and output data parameters are stored in the Experiment repository as shown in Figure

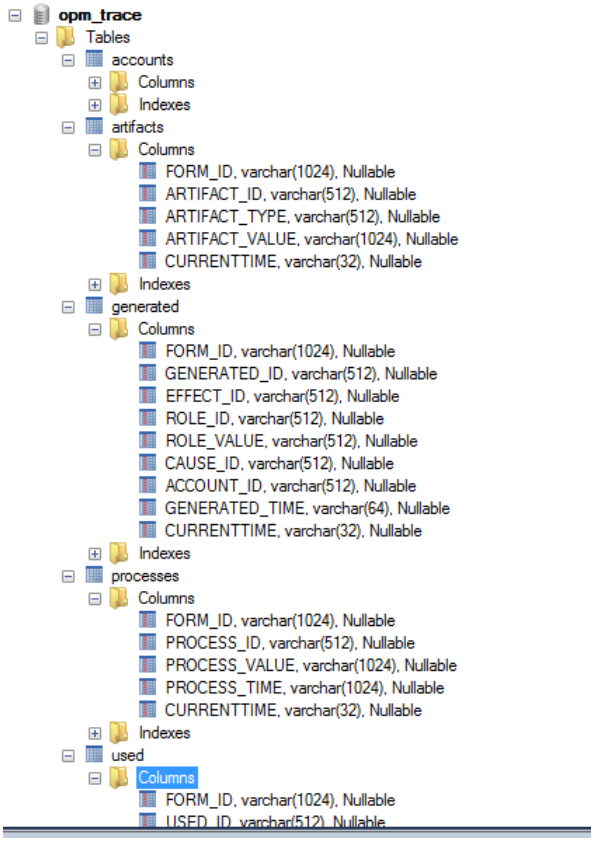


Figure 5.10: OPM Trace Tables in OPM Trace Repository

5.9 whereas Figure 5.10 shows the OPM trace tables in OPM Trace repository. This database is to be accessed by ReProduX tool that will be discussed in the next section, Section 5.1.2.

5.1.2 Versioning Support in OPM Generation and OPM2Taverna Generator in ReProduX

This section describes how the OPM generator creates an OPM provenance trace based on the data parameters stored in the mySQL Experiment repository. Once the provenance trace is generated, it is then transformed into an executable SCUFL workflow format by OPM2Taverna generator which has been discussed in Chapter 4. The square red dashed line in Figure 5.11 shows the components in ReProduX that are OPM Generator, OPM2Taverna Generator and OPM Trace Browser.

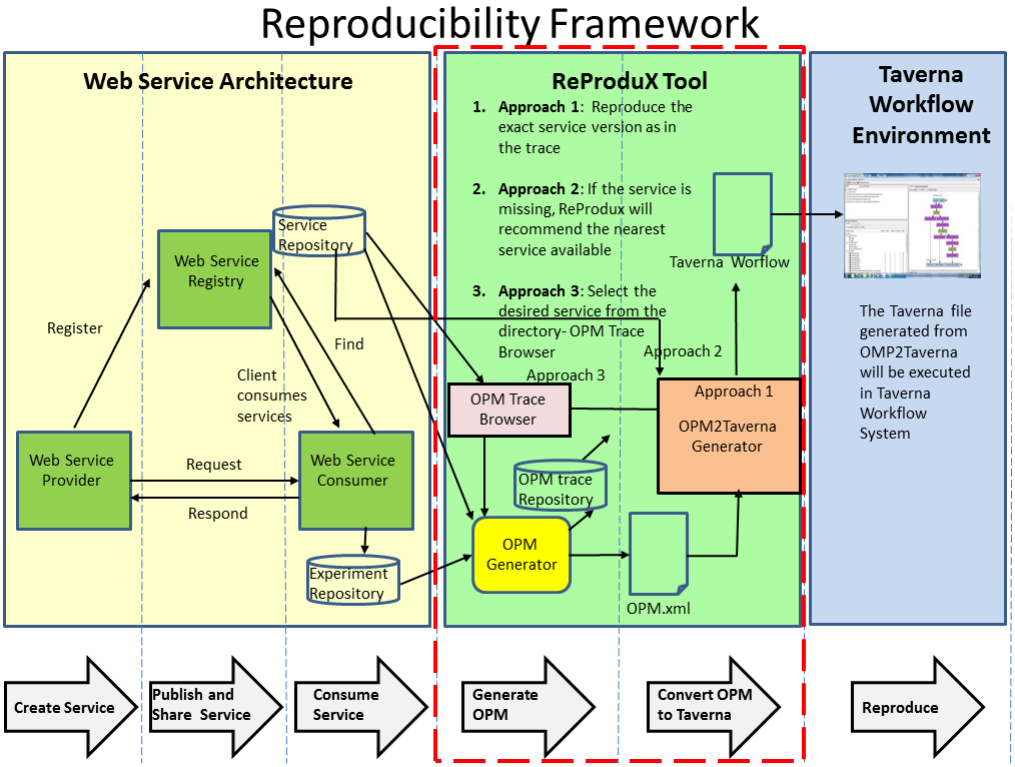


Figure 5.11: ReProduX Tool

The OPM provenance trace is generated from the OPM generator after the execution of one instance of a non-workflow system. From the webservice (Netbeans) client, all the information that should be captured as part of an OPM provenance trace during the execution run, is stored in the mySQL Experiment repository. In this work, real experiment input and output data parameters from Hand, Foot and Mouth Disease (HFMD) are used. OPM Generator is also integrated with a Service repository that contains the information of web service wsdl and versioning information. Then, the OPM generator will access both OPM Experiment repository and Service repository and generate an OPM provenance trace.

The OPM provenance trace includes wasVersionOf that is a versioning information to show the link between one service version to another service version. This versioning information is essential to provide alternative service which is the nearest versions in case the current service is not available or missing. OPM Generator will need to access the Service repository that contains versioning information and compare the relevant

service version naming convention and the date of service creation. Based on that, the OPM Generator will write the current service and the nearest version of the same service as shown in Listing 5.1.

Listing 5.1 shows the wasVersionOf in the OPM provenance trace in which myActivity has two service versions (myActivity1a wasVersionOf myActivity1).

Listing 5.1: wasVersionOf causal dependencies

```
1 <opm:wasVersionedOf>
2  <opm:effect id="myActivity1a" />
3  <opm:role value="serviceVersion />
4  <opm:cause id="myActivity1" />
5  <opm:account id="account1" />
6 </opm:wasVersionOf>
```

When the OPM trace has been generated, the user then can go directly to the OPM2Taverna Generator browser in ReProduX.

5.1.3 ReProduX Service Selection

The ReProduX uses three approaches to convert OPM file to SCUFL file considering the service version major or minor release wherever applicable, as shown in Table 5.1.

For Approach 1, newer version of service for both major and minor will not overwrite or replace the previous version. Therefore, ReProduX will take the exact service location recorded in the original provenance trace be able to reproduce exactly as the previous experiment execution. However, in some practices, if new version occurs, the service provider normally replaced the previous version. In this work, no versions of services are deleted or replaced.

For Approach 2, if the version of the service as in the OPM trace is not available or missing, the ReProduX by default will access from the Service repository and recommend the nearest available compatible service. ReProduX will substitute the missing version of a service to the recommended service version as described in Chapter 4. ReProduX will not recommend incompatible service as the content in the OPM trace is different, therefore direct service version substitution cannot be done.

For Approach 3, user can substitute the services in the OPM trace according to their preferences by choosing any versions of service that is under minor release that is

Table 5.1: ReProduX approaches

No	Approach	Major Release	Minor Release
1	**Approach 1:** Reproduce exactly, using the same version of service(s) as recorded in the original past OPM Trace.	Able	Able
2	**Approach 2:** If the service in the trace is missing, ReProduX will recommend the nearest service available.	Not able due to incompatible change	Able due to compatible change
3	**Approach 3:** User can select specific version of a service by using the OPM Trace Browser. After the selected desired version(s) of service (s) have been chosen, then OPM generator will generates the OPM Trace and repeat Approach 1.	Not able due to incompatible change	Able due to compatible change

compatible with the OPM trace, and re-generate a new OPM trace. If other versions of the same service are major release, the versions will not be displayed in the list as the service is incompatible to the current OPM trace. User can do the service selection in the OPM Trace Generator as shown in Figure 5.12.

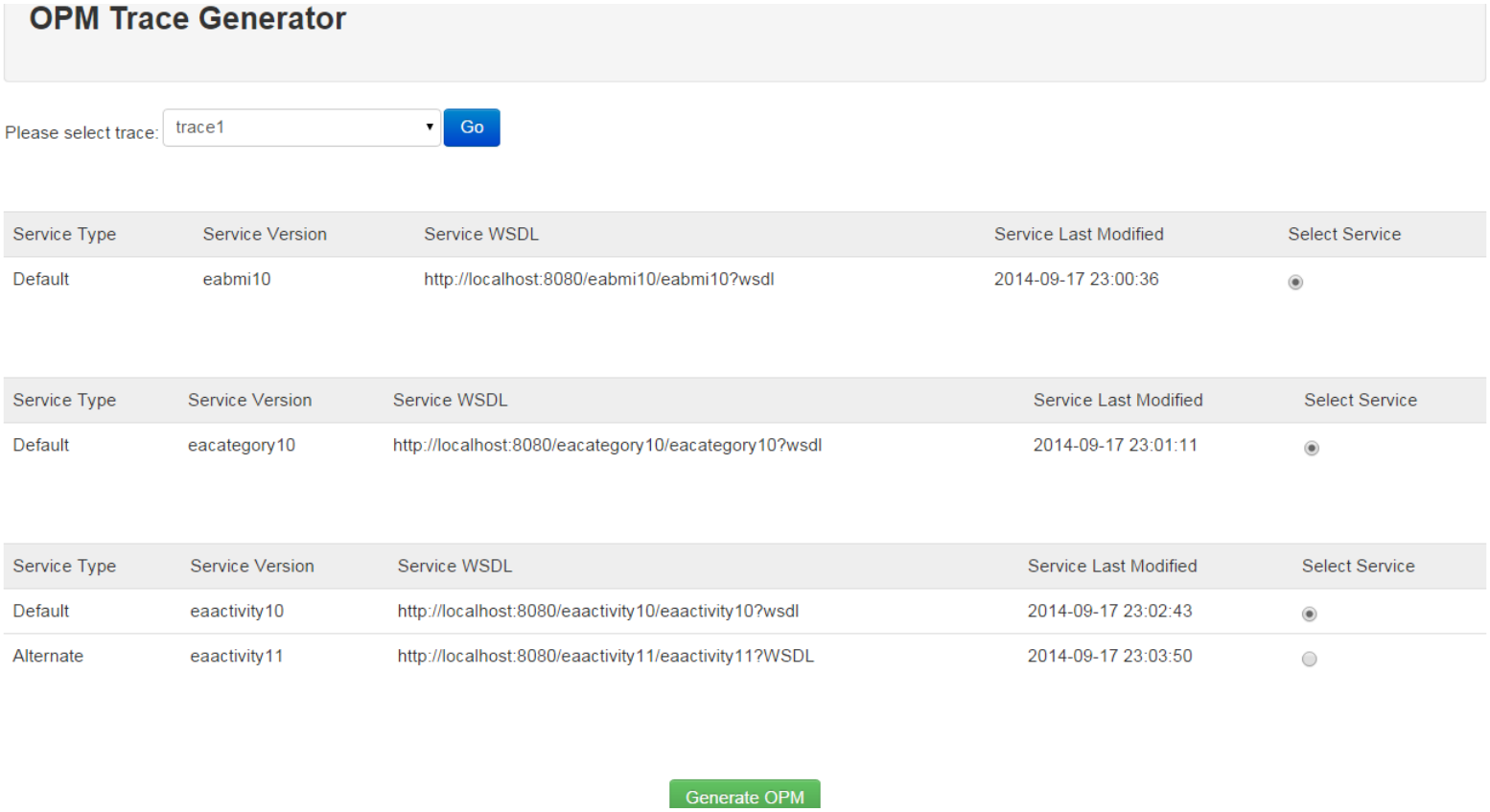


Figure 5.12: OPM Trace Generator

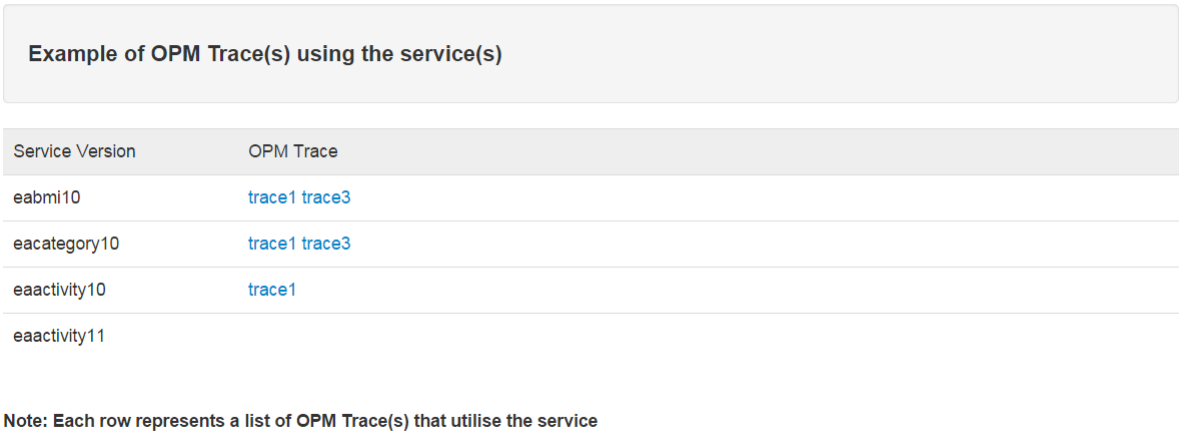


Figure 5.13: OPM Trace that utilise the service

Apart from the list of services in the OPM trace as shown in in Figure 5.12, there is also a section for the user to view other opm trace(s) that utilise the services in the OPM trace, as shown in Figure 5.13, as above. The Figure 5.13 shows that service version eabmi10 is utilised in trace1 and trace 3, whereas service version eaactivity11 only been used in the current trace with no other traces has been utilising it.

When the user has made the selection and re-generate the new OPM trace using the OPM generator, then the user can use the OPM2Taverna Generator in ReProduX.

5.1.4 Reproduce in Taverna Workflow Management System

This section describes how the SCUFL file that was generated by the ReProduX, is then loaded into the Taverna Workflow Management System to reproduce the experiment as shown in the red dashed square box in Figure 5.14.

By successfully enacting the SCUFL file workflow in the Taverna Workflow System, the implementation has succesfully shown as in Figure 5.15 that it is possible to re-execute an execution of a non-workflow into a workflow environment. This also shows that the OPM2Taverna mapping algorithm has transformed the past OPM original trace into a correct Taverna workflow.

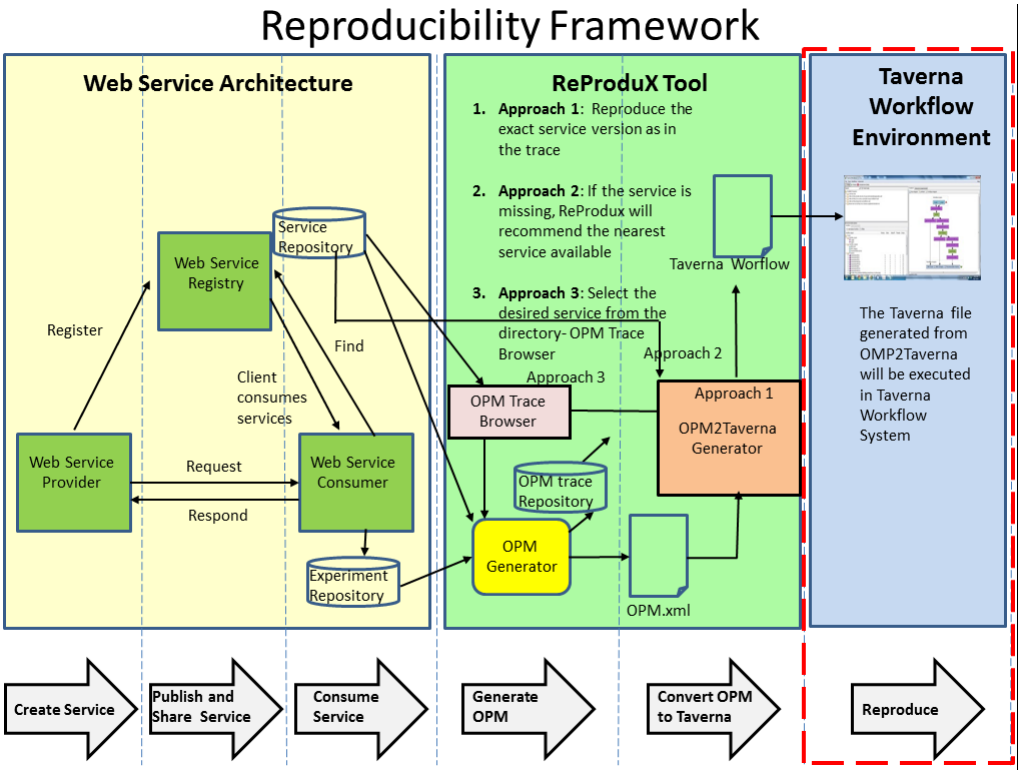


Figure 5.14: Reproduce in Taverna Workflow Environment

5.2 Evaluation

After being able to re-execute past experiments in the Taverna Workflow System, evaluations were made whether the past experiment is able to be reproduced correctly or not. There are three variations of test cases conducted in this work, as listed as follows:

- Verbatim Reproducibility: Be able to reproduce exactly, by calling the same service mentioned in past OPM trace.
- Non-verbatim Reproducibility: Be able to reproduce though one or more services mentioned in the OPM trace are no longer available, by calling other services that are compatible to the previous version of service.
- Comparative Analysis: Comparing ReProduX with Trident

The above test cases are in turn described in the following sub-sections.

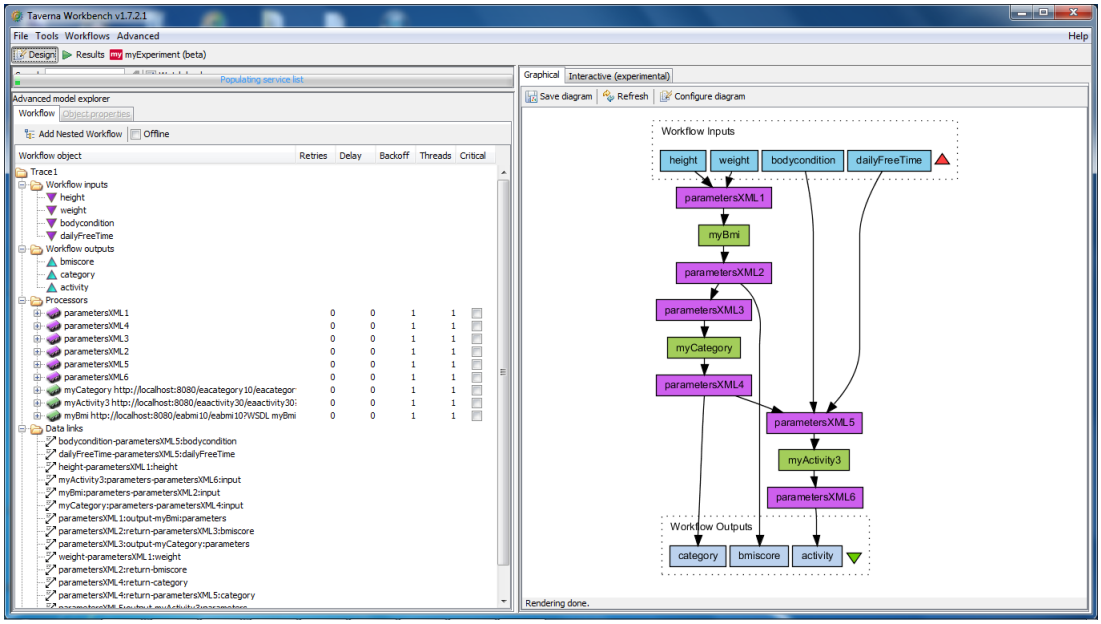


Figure 5.15: The experiment loaded into Taverna Workflow System

5.2.1 Verbatim Reproducibility

Two test cases are used to evaluate ReProduX, namely the Deterministic Model for the spread of Hand, Foot and Mouth Disease (HFMD) test case and test cases from myExperiment. The HFMD experiment is taken from a research cluster group in University Malaysia Sarawak. Currently, the HFMD experiment is conducted in silo and there is not platform to share the experiment. Therefore, it is not easily accessible unless the original owner of the experiment is willing to share the experiment. Another challenge in using the HFMD test case is to be able to create a link to call the Matlab's function. For this evaluation, the aim is to demonstrate that the OPM2Taverna mapping implemented in ReProduX generates correct and executable Taverna workflow. The expectation outcome from this evaluation is that the generated Taverna workflow will produce the same results as the original one.

- Deterministic Model for the spread of Hand, Foot and Mouth Disease in Sarawak Test Case

This mathematical HFMD model is to predict the spread of HFMD in Sarawak in terms of the number of infected persons. The HFMD model uses three parame-

ters namely susceptible, infected and recovered in which the original experiment was executed in a Matlab environment.

In this verbatim reproducibility the HFMD experiment has been chosen to demonstrate the ability of ReProduX to reproduce exactly as the original experiment's result. For this work, a web service is specifically created in SOA system that allows the service to call the Matlab's function to execute the HFMD experiment.

After the execution, the experiment's data is stored inside the mySQL Experiment repository. Then, the OPM Generator is used to generate an OPM provenance trace for the HFMD's experiment as shown in Listing 5.2 below. ReProdux then transforms the OPM into a Taverna workflow. The Taverna workflow file is successfully executed in Taverna 2.3 Workflow System. It is found that the ReProduX is able to reproduce exactly as the original HFMD experiment.

Listing 5.2 shows the generated OPM provenance trace.

Listing 5.2: HFMD Generated OPM Provenance Trace

```
1  <opm:accounts>
2   <opm:account id="account1"/>
3  </opm:accounts>
4  <opm:processes>
5   <opm:process id="hfmd">
6    <opm:value xsi:type="xsd:string">"http://localhost:8080/Eas1Matlab/
         Eas1Matlab?wsdl"</opm:value>
7    <opm:time exactlyAt="2016-01-01T03-02-28.618Z"/>
8   </opm:process>
9  </opm:processes>
10 <opm:artifacts>
11  <opm:artifact id="x1">
12   <opm:value xsi:type="xsd:double">"550700"</opm:value>
13  </opm:artifact>
14  <opm:artifact id="x2">
15   <opm:value xsi:type="xsd:double">"4"</opm:value>
16  </opm:artifact>
17  <opm:artifact id="x3">
18   <opm:value xsi:type="xsd:double">"0"</opm:value>
19  </opm:artifact>
20  <opm:artifact id="tStart">
21   <opm:value xsi:type="xsd:int">"0"</opm:value>
22  </opm:artifact>
23  <opm:artifact id="tEnd">
24   <opm:value xsi:type="xsd:int">"60"</opm:value>
```

```
25   </opm:artifact>
26   <opm:artifact id="tInterest">
27    <opm:value xsi:type="xsd:int">"10"</opm:value>
28   </opm:artifact>
29   <opm:artifact id="tOutput">
30    <opm:value xsi:type="xsd:double">"10"</opm:value>
31   </opm:artifact>
32   <opm:artifact id="xOuptut">
33    <opm:value xsi:type="xsd:string">"
          550707.342545949,4.2520525128916,0.00253834135001543"</opm:value>
34   </opm:artifact>
35  </opm:artifacts>
36  <opm:causaldependencies>
37   <opm:used id="u_1">
38    <opm:effect id="hfmd"/>
39    <opm:role id="r_1" value="1"/>
40    <opm:cause id="x1"/>
41    <opm:account id="account1"/>
42    <opm:time exactlyAt="2016-01-01 11:02:50:AM"/>
43   </opm:used>
44   <opm:used id="u_2">
45    <opm:effect id="hfmd"/>
46    <opm:role id="r_2" value="2"/>
47    <opm:cause id="x2"/>
48    <opm:account id="account1"/>
49    <opm:time exactlyAt="2016-01-01 11:02:50:AM"/>
50   </opm:used>
51   <opm:used id="u_3">
52    <opm:effect id="hfmd"/>
53    <opm:role id="r_3" value="3"/>
54    <opm:cause id="x3"/>
55    <opm:account id="account1"/>
56    <opm:time exactlyAt="2016-01-01 11:02:50:AM"/>
57   </opm:used>
58   <opm:used id="u_4">
59    <opm:effect id="hfmd"/>
60    <opm:role id="r_4" value="4"/>
61    <opm:cause id="tStart"/>
62    <opm:account id="account1"/>
63    <opm:time exactlyAt="2016-01-01 11:02:50:AM"/>
64   </opm:used>
65   <opm:used id="u_5">
66    <opm:effect id="hfmd"/>
67    <opm:role id="r_5" value="5"/>
68    <opm:cause id="tEnd"/>
```

```
69     <opm:account id="account1"/>
70     <opm:time exactlyAt="2016-01-01 11:02:50:AM"/>
71    </opm:used>
72    <opm:wasGeneratedBy id="g_1">
73     <opm:effect id="hfmdReadings"/>
74     <opm:role id="r_1" value=""/>
75     <opm:cause id="hfmd"/>
76     <opm:account id="account1"/>
77     <opm:time exactlyAt="2016-01-01 11:02:50:AM"/>
78    </opm:wasGeneratedBy>
79   </opm:causaldependencies>
80 </opm:opmGraph>
```

In the listing above, wasVersionOf causal dependency is not generated in the OPM provenance trace because the service has only one version to call function in Matlab.

Listing 5.3 shows the transformed SCUFL workflow file.

Listing 5.3: HFMD Taverna workflow file

```
1      <s:processor name="parametersXML1">
2        <s:local>
3          org.embl.ebi.escience.scuflworkers.java.XMLInputSplitter
4          <s:extensions>
5            <s:complextype optional="false" unbounded="false" typename="hfmd
                " name="parameters" qname="{http://x2eav1/}hfmd">
6              <s:elements>
7                <s:basetype optional="false" unbounded="false" typename="
                     double" name="x1" qname="hfmd:x1"
8                <s:basetype optional="false" unbounded="false" typename="
                     double" name="x2" qname="hfmd:x2"
9                <s:basetype optional="false" unbounded="false" typename="
                     double" name="x3" qname="hfmd:x3"
10               <s:basetype optional="false" unbounded="false" typename="
                     double" name="tStart" qname="hfmdtStart"
11               <s:basetype optional="false" unbounded="false" typename="
                     double" name="tEnd" qname="hfmdtEnd"
12             </s:elements>
13           </s:complextype>
14         </s:extensions>
15       </s:local>
16     </s:processor>
17     <s:processor name="parametersXML2">
18       <s:local>
19       org.embl.ebi.escience.scuflworkers.java.XMLOutputSplitter
20       <s:extensions>
```

```
21            <s:complextype optional="false" unbounded="false" typename="
                  hfmdResponse" name="parameters" qname="{http://x2eav1/}
                  hfmdResponse">
22              <s:elements>
23                <s:basetype optional="true" unbounded="false" typename="
                      double" name="return" qname="hfmdResponse:return"
24              </s:elements>
25            </s:complextype>
26        </s:extensions>
27        </s:local>
28      </s:processor>
29    <s:processor name="hfmd">
30      <s:arbitrarywsdl>
31        <s:wsdl>
32    http://localhost:8080/Eas1Matlab/Eas1Matlab?wsdl
33        </s:wsdl>
34        <s:operation>hfmd</s:operation>
35      </s:arbitrarywsdl>
36    </s:processor>
37    <s:link source="x1" sink="parametersXML1:x1" />
38    <s:link source="x2" sink="parametersXML1:x2" />
39    <s:link source="x3" sink="parametersXML1:x3" />
40    <s:link source="tStart" sink="parametersXML1:tStart" />
41    <s:link source="tEnd" sink="parametersXML1:tEnd" />
42    <s:link source="hfmd:parameters" sink="parametersXML2:input" />
43    <s:link source="parametersXML2:return" sink="hfmdReadings" />
44    <s:link source="parametersXML1:output" sink="hfmd:parameters" />
45    <s:source name="x1" />
46    <s:source name="x2" />
47    <s:source name="x3" />
48    <s:source name="tStart" />
49    <s:source name="tEnd" />
50    <s:sink name="hfmdReadings" />
51  </s:scufl>
```

Figure 5.16 shows the re-execution of HFMD experiment in Taverna Workflow System.

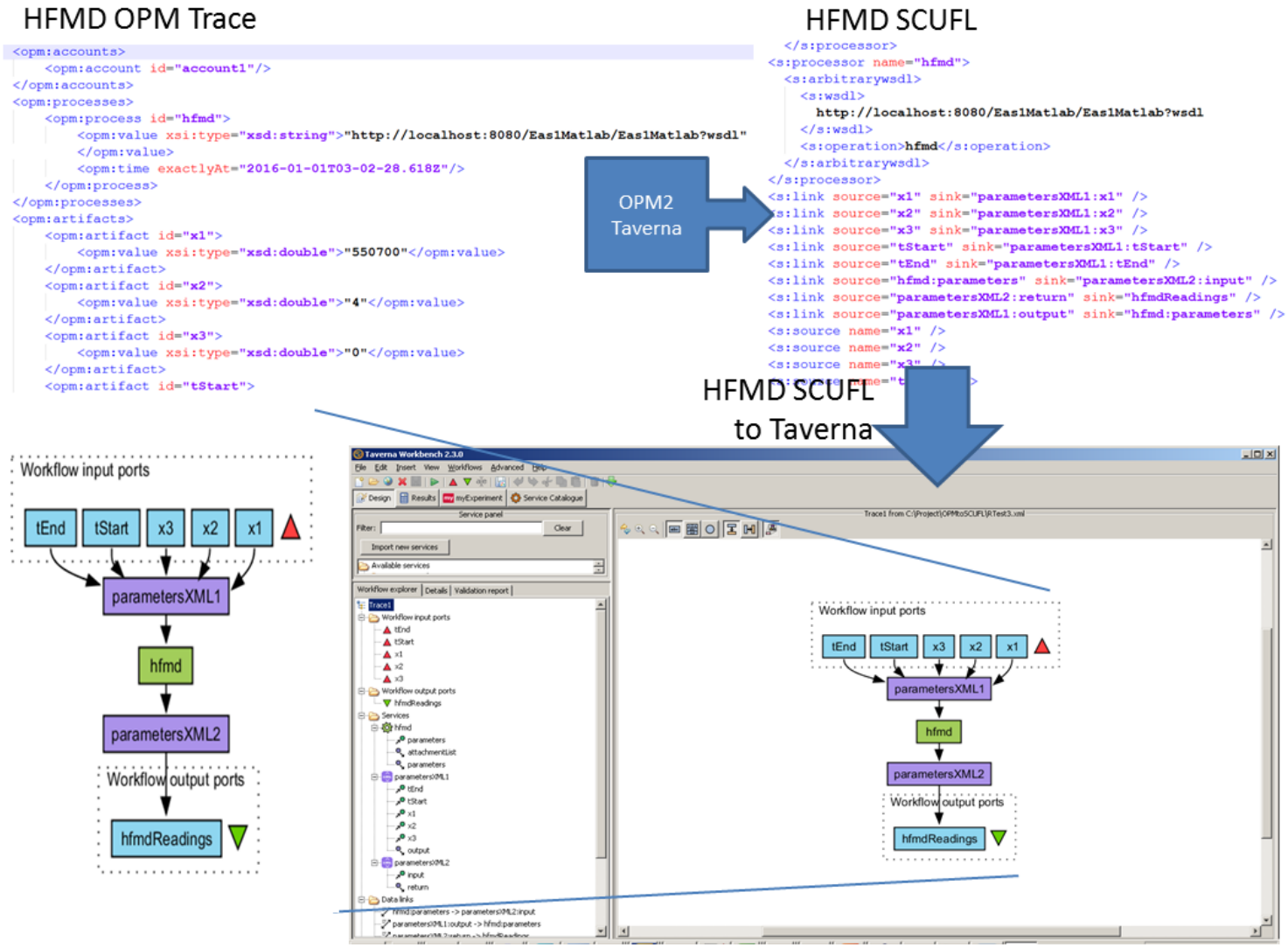


Figure 5.16: The re-execution of HFMD experiment in Taverna Workflow System.

Even though the HFMD experiment is successfully reproduced, the ReProduX has no control on versioning of services from the original experiment owner. For example, when the HFMD algorithm has been changed to a new version, ReProduX will not be able to access the other available version of the same services unless the experiment owner inform and update on the availability of the new version. This is why in this work, it is proposed for service owners to follow a versioning standard to allow sharing purposes.

- Workflows from myExperiment Test Case

Twenty experiments that utilise wsdl web services from myExperiment repository have been evaluated and it has been found that about eighty percent of the experiments workflows cannot be re-executed in Taverna Workflow Systems due to inability to access the external web services. Upon opening the workflows, Taverna returns error message such as depicted in Figure 5.17 as below.

Figure 5.17: Example of Taverna error message from Entrez Gene to KEGG Pathway

This is one of the problems that ReProduX aims to overcome where external web services becomes inaccessible. Despite the initial challenges of finding experiments workflows to be re-executed, the remaining twenty percent can be re-executed. In this evaluation, one example is demonstrated to show the ReProduX ability to reproduce via the Taverna round trip where the OPM trace of the experiment is generated by the execution of the Taverna workflow and goes into ReProduX to be converted back to SCUFL and re-executed in Taverna.

5.2.2 Non-Verbatim Reproducibility

Non-Verbatim reproducibility evaluation is to test the ReProdux algorithm that was discussed in Chapter 4 when one or more of the services mentioned in the OPM trace are no longer available. Previously, when this scenario happened, the experiments cannot be re-executed and will hinder reproducibility. In this case, ReProduX is designed to automatically deal with this scenario where ReProduX will recommend the nearest services according to the service creation or its last modified date.

In contrast to the Verbatim Reproducibility, the evaluation for the workflows from myExperiment or other real life existing experiments cannot be implemented for the Non-Verbatim Reproducibility evaluation due to the fact that the external web services are not within the control of the user and there are no available versioning of web services as specified in the Chapter 4. For this case, this work proposes that web services providers or owners to provide multiple versions of the same web services at the early stage of service creation. Service provider plays a vital role to manage versioning of web services.

Thus for this test case, a user needs to be the service owner in order to have a control over the creation of services that incorporates versioning information into the development. Hence, BMI test case is used as a proof of concept in order to implement the Reproducibility Framework. The BMI test case is used to show that service selection is possible for instance if multiple versions of the same service are available. For this evaluation, the aim is to find out the effect on the reproduced experiment results if different versions of one or more of the services in the trace are used.

- Exercise Advisor Test Case

For this evaluation, the following scenario is chosen to demonstrate the ReProduX capability. Consider that an OPM trace was originally generated using the following services namely:

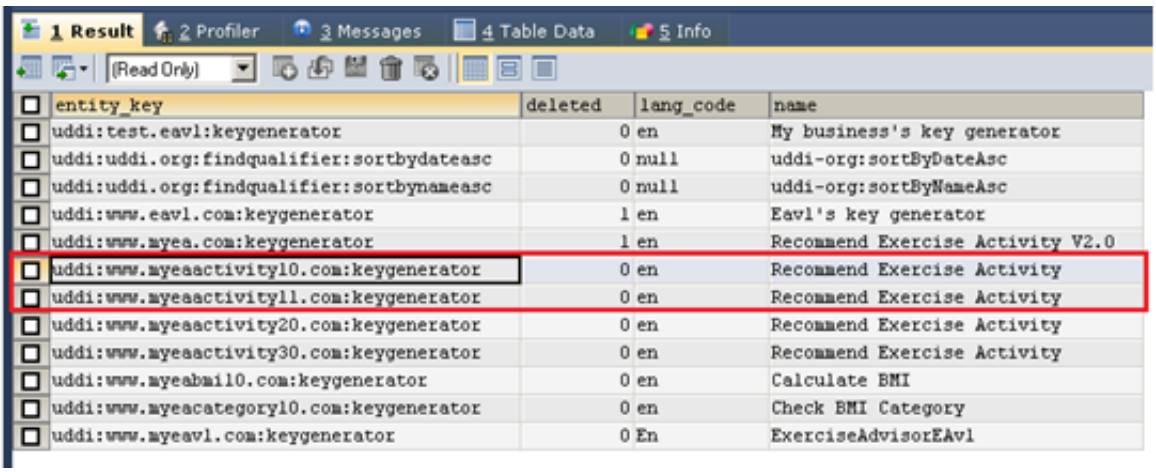
- http://localhost:8080/eabmi10/eabmi10?WSDL
- http://localhost:8080/eacategory10/eacategory10?WSDL
- http://localhost:8080/eaactivity10/eaactivity10?WSDL

A user wishes to reproduce past experiment by using the ReProduX. At that particular time, the third service eaactivity10 became inaccessible due to accidental removal by the service owner. The owner has updated the service to improve its functionality algorithm (minor release). The new service version is as follow:

- http://localhost:8080/eaactivity11/eaactivity11?WSDL

In this case, OPM2Taverna will check for the availability of services contained within the original OPM trace. If ReProduX found that any service is missing, it will recommend and replace the missing service with the nearest service available as discussed in Chapter 4.

Figures 5.18 to 5.20 are provided to demonstrate this evaluation.

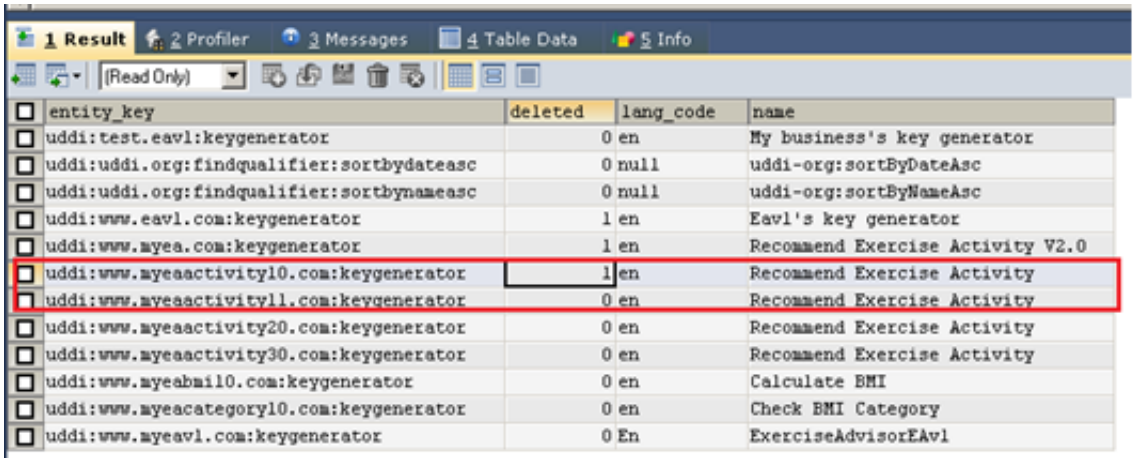


1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info

(Read Only)

entity_key	deleted	lang_code	name
uddi:test.eavl:keygenerator	0	en	My business's key generator
uddi:uddi.org:findqualifier:sortbydateasc	0	null	uddi-org:sortByDateAsc
uddi:uddi.org:findqualifier:sortbynameasc	0	null	uddi-org:sortByNameAsc
uddi:www.eavl.com:keygenerator	1	en	Eavl's key generator
uddi:www.myea.com:keygenerator	1	en	Recommend Exercise Activity V2.0
uddi:www.myeaactivity10.com:keygenerator	0	en	Recommend Exercise Activity
uddi:www.myeaactivity11.com:keygenerator	0	en	Recommend Exercise Activity
uddi:www.myeaactivity20.com:keygenerator	0	en	Recommend Exercise Activity
uddi:www.myeaactivity30.com:keygenerator	0	en	Recommend Exercise Activity
uddi:www.myeabmi10.com:keygenerator	0	en	Calculate BMI
uddi:www.myeacategory10.com:keygenerator	0	en	Check BMI Category
uddi:www.myeavl.com:keygenerator	0	En	ExerciseAdvisorEAvl

Figure 5.18: jUDDI Service Repository that depicts both eactivity10 and eactivity11 services are available. The "deleted"İ column with value 0 = service is available, value 1= service is unavailable

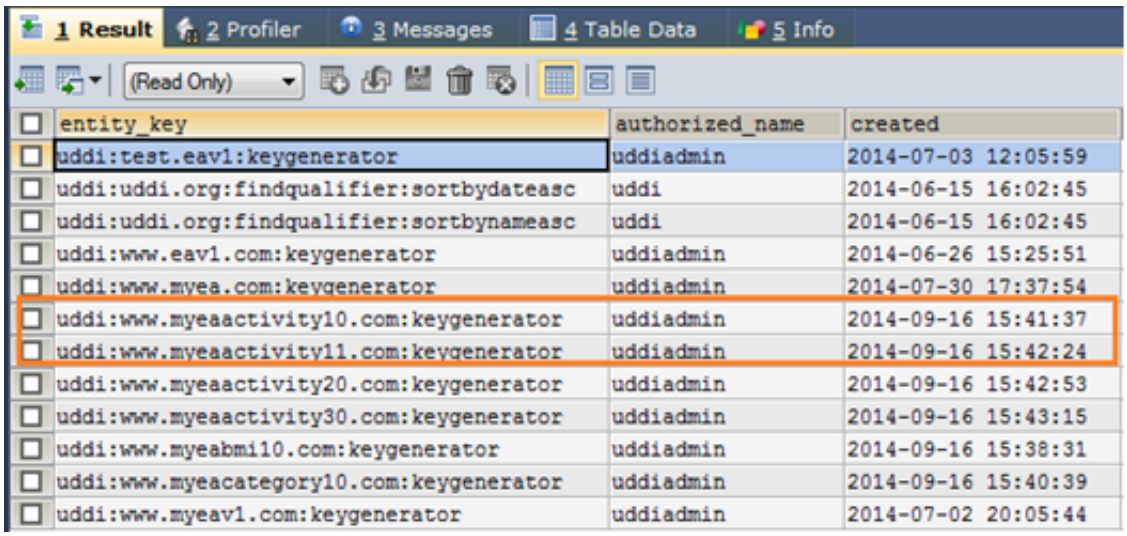


1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info

(Read Only)

entity_key	deleted	lang_code	name
uddi:test.eavl:keygenerator	0	en	My business's key generator
uddi:uddi.org:findqualifier:sortbydateasc	0	null	uddi-org:sortByDateAsc
uddi:uddi.org:findqualifier:sortbynameasc	0	null	uddi-org:sortByNameAsc
uddi:www.eavl.com:keygenerator	1	en	Eavl's key generator
uddi:www.myea.com:keygenerator	1	en	Recommend Exercise Activity V2.0
uddi:www.myeaactivity10.com:keygenerator	1	en	Recommend Exercise Activity
uddi:www.myeaactivity11.com:keygenerator	0	en	Recommend Exercise Activity
uddi:www.myeaactivity20.com:keygenerator	0	en	Recommend Exercise Activity
uddi:www.myeaactivity30.com:keygenerator	0	en	Recommend Exercise Activity
uddi:www.myeabmi10.com:keygenerator	0	en	Calculate BMI
uddi:www.myeacategory10.com:keygenerator	0	en	Check BMI Category
uddi:www.myeavl.com:keygenerator	0	En	ExerciseAdvisorEAvl

Figure 5.19: jUDDI Service Repository that depicts eactivity10 service is unavailable, with "deleted"İ column value = 1



1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info

(Read Only)

entity_key	authorized_name	created
uddi:test.eavl:keygenerator	uddiadmin	2014-07-03 12:05:59
uddi:uddi.org:findqualifier:sortbydateasc	uddi	2014-06-15 16:02:45
uddi:uddi.org:findqualifier:sortbynameasc	uddi	2014-06-15 16:02:45
uddi:www.eavl.com:keygenerator	uddiadmin	2014-06-26 15:25:51
uddi:www.myea.com:keygenerator	uddiadmin	2014-07-30 17:37:54
uddi:www.myeaactivity10.com:keygenerator	uddiadmin	2014-09-16 15:41:37
uddi:www.myeaactivity11.com:keygenerator	uddiadmin	2014-09-16 15:42:24
uddi:www.myeaactivity20.com:keygenerator	uddiadmin	2014-09-16 15:42:53
uddi:www.myeaactivity30.com:keygenerator	uddiadmin	2014-09-16 15:43:15
uddi:www.myeabmi10.com:keygenerator	uddiadmin	2014-09-16 15:38:31
uddi:www.myeacategory10.com:keygenerator	uddiadmin	2014-09-16 15:40:39
uddi:www.myeavl.com:keygenerator	uddiadmin	2014-07-02 20:05:44

Figure 5.20: jUDDI Service Repository that depicts the service creation date

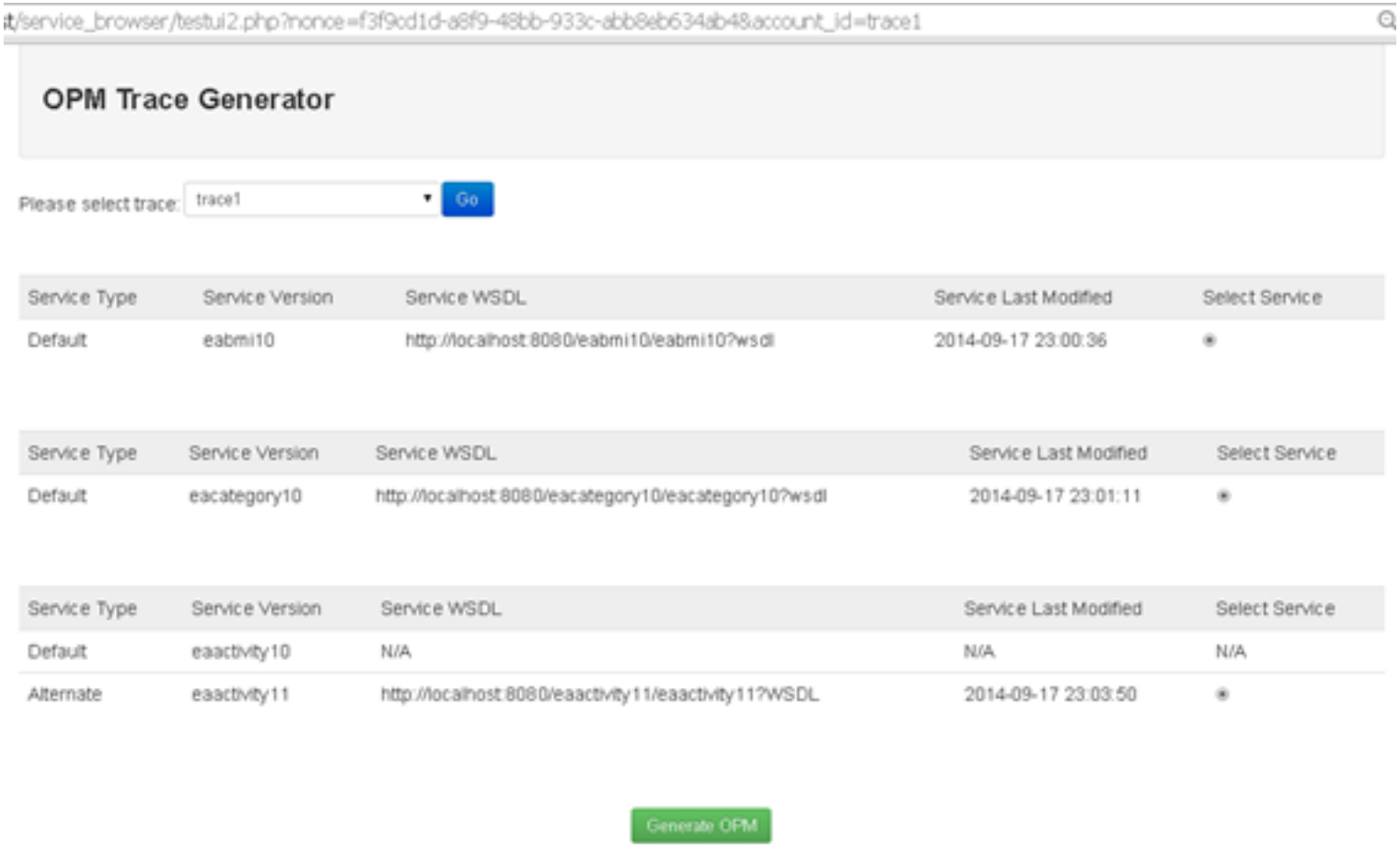


Figure 5.21: OPM trace browser that depicts eactivity10 service is not available

Listing 5.4: Original OPM trace showing eactivity10 process was used in the experiment execution

```
<opm:accounts>
 <opm:account id="account1"/>
</opm:accounts>
<opm:processes>
 <opm:process id="myBmi">
  <opm:value xsi:type="xsd:string">"http://localhost:8080/eabmi10/eabmi10
      ?wsdl"</opm:value>
  <opm:time exactlyAt="2014-09-24T02-18-35.737Z"/>
 </opm:process>
 <opm:process id="myCategory">
  <opm:value xsi:type="xsd:string">"http://localhost:8080/eacategory10/
      eacategory10?wsdl"</opm:value>
  <opm:time exactlyAt="2014-09-24T02-18-37.967Z"/>
 </opm:process>
 <opm:process id="myActivity1">
  <opm:value xsi:type="xsd:string">"http://localhost:8080/eaactivity10/
      eaactivity10?wsdl"</opm:value>
  <opm:time exactlyAt="2014-09-24T02-18-41.213Z"/>
 </opm:process>
</opm:processes>
```

Listing 5.5: Generated SCUFL workflow that shows the service has been replaced to eaactivity11

```
<s:processor name="myBmi">
    <s:arbitrarywsdl>
      <s:wsdl>
 http://localhost:8080/eabmi10/eabmi10?wsdl
      </s:wsdl>
      <s:operation>myBmi</s:operation>
    </s:arbitrarywsdl>
 </s:processor>
 <s:processor name="myCategory">
    <s:arbitrarywsdl>
      <s:wsdl>
 http://localhost:8080/eacategory10/eacategory10?wsdl
      </s:wsdl>
      <s:operation>myCategory</s:operation>
    </s:arbitrarywsdl>
 </s:processor>
 <s:processor name="myActivity1">
```

```
18      <s:arbitrarywsdl>
19        <s:wsdl>
20  http://localhost:8080/eaactivity11/eaactivity11?wsdl
21        </s:wsdl>
22        <s:operation>myActivity1</s:operation>
23      </s:arbitrarywsdl>
24    </s:processor>
```

Listing 5.4 and Listing 5.5 above demonstrated that the OPM2Taverna recommended the nearest service eaactivity11 to replace the missing service eaactivity10. Both services are similar, with onnly differences in their algorithms. Previously, both are available to be used, however the service owner has accidentally removed the earlier version, and became unavailable.

From the above evaluation, some services failed due to the unavailability of services. If the version of the service at that point of time is not available or missing, thus the workflow is incomplete. This is where ReProduX be able to recommend another version of the same service which is avaialable by accessing the Service repository. ReProduX proposed a selection of services to be used and this is possible if service versioning standard is incorporated at the earlier phase of service creation. Thus, this can assist users to still be able to reproduce even though existing service is not longer available.

5.2.3 Comparative Analysis

This evaluation compares ReProduX to Trident EVF on experiment reproducibility. This comparison is made based on requirements to support the creation of reproducible experiments on the basis described in reproducibility framework. This evaluation is based on the concepts of both tools as there is a constraint on accessing Trident evolution framework. It is found that there are differences between Trident EVF and ReProduX. This is summarised in Table 5.4 below.

Overall, Trident EVF work is more comprehensive than this work. Trident EVF is focuses on workflow evolution framework with data intensive experiments. On the other hand, ReProduX is dealing with external web service evolution services that are not under a workflow environment control. However Trident EVF does not address the issue with inaccessible external web services that are not in their control whereas ReProduX is providing accessibility in recommending alternate version of external web services if the original web services become inaccessible. ReProduX also provides the

Table 5.2: Comparison between Trident and ReProduX in handling versioning in reproducibility - Part 1 of 2

No	Requirements	Trident	ReProduX
1	Addressing the accessibility issue on external web services that are not under a workflow environment control.	Store and preserve external web services within Trident environment control. All information is stored in Trident Registry	Store and preserve all versions of external web services that are not under a workflow environment control. ReProduX proposes service provider or owner should provide multiple versions of services at early stage of service creation, using service versioning convention scheme. Publish and share service into jUDDI registry.
2	Automatically stores a provenance trace that detailed the execution that includes data, services and workflows	Trident has full control over the workflow execution. All information in execution is easily captured using Trident provenance data model (similar to OPM provenance model).	Generate OPM provenance trace that detailed the information of an execution.

ability to select specific version of service that enable user to reproduce and promote new discoveries in analysing different service version.

5.3 Limitations and Constraints

Limitations of the ReProduX are described as follows:

- The ReProduX only deals with WSDL-specified web services. This work does not cover other type of web services that is RESTful web services and scripts that are beyond the scope of this work.
- The experiments chosen in evaluation are to provide the proof of concept of ReProduX. However, in future, the experiment can be made as complex as needed

Table 5.3: Comparison between Trident and ReProduX in handling versioning in reproducibility - Part 2 of 2

No	Requirements	Trident	ReProduX
3	Capture the provenance of the external web services derivation.	If current service used in a workflow needs to be updated, the service update is associated with workflow versioning. Trident allows user to choose a choice of workflow version.	Service versioning is supported. Introduce *wasVersionOf* to describe the relation from one version of a service to another version of a service. ReProduX allows user to choose a choice of service version.
4	Capture the provenance of the workflows derivation.	Workflow versioning is supported via direct evolution and contributions.	OPM Traces (workflows) are stored in OPM Traces repository.
5	Automatically support the portability to new workflow environment.	Not required for Trident users as users dependent on Trident and reproducibility is done within Trident environment.	Support for portability to Taverna Workflow Environment. ReProduX automatically transform OPM trace into SCUFL that is a Taverna workflow schema.

for specific tasks and integration requirements.

- This work is utilising Taverna 1 SCUFL format. It is recognised Taverna has t2flow and SCUFL2 schema. The ReProduX in future can be improved to handle SCUFL2 schema in order to take advantage of the Taverna Workbench 3.
- In dealing with OPM traces, the ReProduX at this stage only takes the latest OPM trace, no versioning on traces yet been applied. The focus is on the service versioning.
- In this work, the service provider must follow the service versioning convention to save and retain multiple versions of services.

Apart from the above limitations, there is a constraint to access to Trident EVF. The Trident has been archived and Trident EVF is not available to download. There-

fore the comparative analysis between Trident EVF and ReProduX is based on the understanding of the concept.

5.4 Reproducibility Taxonomy

The reproducibility taxonomy describes the requirements for facilitating the reproduction of experiment. As shown in Figure 5.18, it consists of three requirements for reproducible experiments: (a) collection of multiple versions of services (b) documentation of provenance traces and (c) reproduction into workflow systems. In this section, each requirement is described in turn.

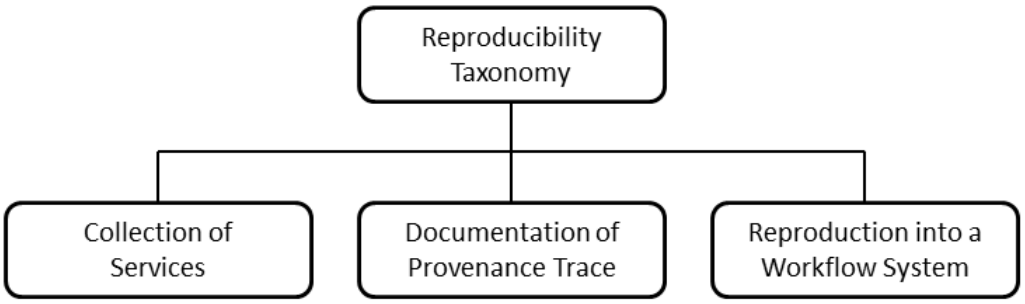


Figure 5.22: Reproducibility Taxonomy

Collection of multiple versions of services: Preparing a collection or aggregation of multiple versions of services is essential in this work in order to ensure service availability and accessibility. When a scientist is interested to re-execute past experiment, he must refer to the same versions of the services that were originally used. Therefore, an appropriate service versioning approach is needed to be able to preserve old versions of services and call old versions of services. In this work, service versioning convention scheme should be followed by the service provider or owner at the early stage of service creation. Additional relation or causal dependency regarding service versioning needs to be asserted in the provenance trace to describe the relation from one version of a service to another version of a service. In this work, wasVersionOf relation is introduced and the service creation time information. The annotation on timing information of service creation can be used in re-execution of the experiment.

With the availability of multiple versions of the same service, the scientist can explore with different variations to see the difference from existing work.

Documentation of provenance traces: Creating documentation of an experiment execution is a key of reproducibility as it contains all the details about the artifacts, versions of services and causal dependencies that is used to re-execute past experiment. Causal dependencies are essential in reproducibility, which requires identifying the cause and effect in the experiment (X was caused by Y) and the linkage between them. When all of the entities in the experiment have been identified, they must be captured and recorded as a provenance trace. The provenance trace is structured in a way to construct a record of how execution takes place. In order to ensure the collection of content is completely recorded, a provenance data model is needed.

Reproduction into workflow systems: It may not be possible to directly use an original provenance trace that is generated from an environment that is different from the re-execution workflow environment. In this work, ReProduX has the ability to reproduce past experiment on a different environment, thus providing experiment portability.

Workflow systems are used to choreograph the execution of an experiment from data and services. The enactment of a workflow can be used to invoke computational services. Workflow systems provide a graphical user interface for building workflows. Another feature that a workflow system can provide is visualisation that allow representation of execution in a workflow graph [5]. In addition, exploratory visualisation can be incorporated allowing different versions of services to be viewed and compared side by side to see the differences [74, 75].

5.5 Discussion of Results

In this chapter, a discussion of the implementation is provided, followed by an explanation of the reproducibility framework used for the evaluation. This chapter also presented that the evaluation of the concepts proposed is able to reproduce experiments, even when new versions of services had become available. It also showed that this provenance trace generated under one environment (workflow environment or development environment that producing a provenance trace) could be transformed using

ReProduX, an OPM to SCUFL converter to allow reproduction in a different workflow environment (Taverna).

Based on the evaluation and findings derived from this work, the ReProduX is able to demonstrate that the OPM-SCUFL(Taverna) mapping generates correct Taverna workflows. By using the workflows from myExperiment repository, ReProduX is also be able to generate OPM trace from the original execution of a Taverna workflow and re-generate the SCUFL, the new Taverna workflow and produce the same results as the original experiment, using the exact web services. This is possible because the external web services from the original SCUFL (Taverna) are available and accessible. However, there is also a case where this is not possible to re-execute when the external web services are missing or inaccessible. In this case, this work proposes a solution that service versioning design should be tackled at early stage during service creation by the service provider or service owner.

In addition, no versions are deleted or overwriten in this work, therefore the multiple versions of services can be retrieved by ReProduX at any time. If there is a case where one or more of the services that appeared in the original OPM trace are missing or no longer available, ReProduX is able to recommend and replace the missing services with other compatible services which are the nearest services according to their service creation date.

A Reproducibility Taxonomy is also presented in this evaluation chapter. In the taxonomy, three elements are identified as requirements for reproducible elements that are preparing a collection (aggregation) to support multiple versions of services; generating provenance trace documentation that represents the data, services and dependencies; and transform a provenance trace into a workflow system (with visualisation).

In the final chapter, the thesis will discuss the overall findings of the research and possible future work.

Chapter 6

Conclusion and Future Work

In this concluding chapter of the thesis a summary of the research and its contributions are presented. There is then a discussion of the opportunities for further work, and possible extensions in the area of reproducibility.

6.1 Research Summary and Contributions

In this thesis, the lifecycle for the reproducibility of experiments is discussed from the stage when the experiment is created and captured to the stage when the execution is reproduced. The first stage is mainly focussed on provenance infrastructure and processes. A comparison between existing provenance models was made to show how they capture information on data and services used in an experiment. However, this highlighted the deficiencies in terms of what is required to reproduce an experiment.

In Chapter 3, the OPM model is adopted as a provenance model to represent the experimental execution, encompassing data and services. A contribution was made to show how service versioning could be incorporated into provenance to address deficiencies in the existing schemes. Furthermore, a new method was shown in which versioning of web services, to support experimental reproduction, could be introduced through the tModel mechanism of jUDDI. An OPM Trace Generator is also implemented in this work to incorporate the service versioning information.

Chapter 4 introduced ReProduX, an approach to allow the provenance trace to be used as the basis for reproducing an experiment at a later time. This included mapping rules that transform entities and relations from the OPM source to target files in SCUFL

in order to allow reproduction in the Taverna workbench. The ability to recommend an alternate service version if the original service is unavailable and select specific version of service allow user to reproduce and promote new discoveries in analysing different service version.

In Chapter 5, the implementation and evaluation of the reproducibility framework is described and analysed. A reproducibility taxonomy is proposed to describe the categories that is essential to consider to reproduce experiments.

6.2 Contributions

The following list summarises the contribution towards knowledge made by this thesis:

- In Chapter 3, a service versioning approach is introduced to provide an alternative service in case the current service is not available or missing. To achieve this, a service with versioning information needs to be registered in Service Registry. This includes the service versioning naming convention and the date and time of service creation. Having all this information, OPM Generator be able to generate an additional wasVersionOf causal dependency in OPM trace. However, in order to achieve this, service versioning need to be in place at initial stage of service creation. Therefore, only the service owner has a control over the services.

- In Chapter 4, a ReProduX tool technique is developed to allow the reproduction of past experiments. ReProduX tool consists of three components. Firstly, OPM Generator that generates OPM provenance trace from an execution run and include wasVersionOf dependency in the trace. Secondly, OPM2Taverna Generator that extracts information from OPM trace to Taverna trace. OPM2Taverna has additional functions where it can check whether the services are available to be executed, and recommends the alternative service if the service is not available or missing. This is to ensure that the correct version of a service is called during reproduction. Thirdly, OPM Trace Browser that provides user to select specific service version of choice. ReProduX has three approaches in dealing with service versioning as described in Chapter 4.

- In Chapter 5, a Reproducibility Framework is implemented and evaluated. The three ReProduX approaches are tested using sample of experiments. It is found

that ReProduX tool is possible to assist in reproducing past experiments if only services are available and accessible. From the findings, reproducibility taxonomy is derived to identify the requirements for facilitating the reproduction of experiment. The reproducibility taxonomy is made up of three categories: collection (aggregation) to support multiple versions of services; documentation of provenance trace that represents the data, services and dependencies involved; and reproduction that transforms a provenance trace into a workflow system (with visualisation). The taxonomy is hoped to give an overview of things that we need to be able to do if we are to achieve reproducibility.

6.3 Future Works

This work has focussed on reproducibility, in particular on service versioning. The following list potential research that can be conducted as an extension of works from this research.

- Since this research realised that service versioning needs to be initiated at the first stage of service creation by service provider or service owner, therefore a further work on creating a standard mechanism or template to record service versioning is an advantage. In this template, service owner must store the service versioning throughout the development. Thus, this will keep old versions of services available. This template will also incorporate subscription services to inform consumer that a new service is available.

- In this work, ReProduX tool is evaluated through the Reproducibility Framework. A mechanism to compare the two provenance traces; original trace and reproduced trace is essential to validate reproducibility. Therefore, an extension to study on the differences between two executions through the traces would be an advantage, but focusing on ReProduX environment.

- ReProduX currently deals with service minor release that supports backward compatibility to call alternate service version. This work can be extended to also include major release in the future.

Realising the work to incorporate service versioning in reproducing past e-experiment requires further work to reassure the reproducibility lifecycle is achieved and making reproducibility result more reliable and accurate, thus not losing any vital information.

Bibliography

[1] Luc Moreau, Paul Groth, Simon Miles, Javier Vázquez-salceda, John Ibbotson, Sheng Jiang, Steve Munroe, Omer Rana, Andreas Schreiber, Victor Tan, and Laszlo Varga. **The Provenance of Electronic Data**. *Communications of the ACM*, **2007**, 2007. viii, 17, 18

[2] Satya S. Sahoo and Amit Sheth. **Provenir ontology: Towards a Framework for eScience Provenance Management**. *Microsoft eScience Workshop*, 2009. viii, 7, 18, 21

[3] Paolo Missier, Khalid Belhajjame, and James Cheney. **The W3C PROV Family of Specifications for Modelling Provenance Metadata**. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 773–776, New York, NY, USA, 2013. ACM. viii, 18, 23

[4] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul T. Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric G. Stephan, and Jan Van den Bussche. **The Open Provenance Model core specification (v1.1)**. *Future Generation Computer Systems*, **In Press**:743–756, 2010. viii, 7, 18, 22, 25, 40, 48

[5] **Taverna Workflow Management System Website**. Available online at `http://www.taverna.org.uk/`. viii, 28, 34, 35, 118

[6] Yogesh Simmhan, Beth Plale, and Dennis Gannon. **A Survey of Data Provenance in e-Science**. *Sigmod Record*, **34**:31–36, 2005. viii, 6, 36, 37

[7] Dave Ehnebuske Peter Brittenham, Francisco Cubera and Steve Graham. **Understanding WSDL in a UDDI registry, Part 1**, Sep 2001. Available online at `http://www.ibm.com/developerworks/webservices/library/ws-wsdl/`. ix, 58

[8] **European Bio-Informatics Institute (EBI)**, 2013. Available online at `http://www.ebi.ac.uk/escience/`. 1

[9] **myGrid Homepage**, 2013. Available online at `http://www.mygrid.org.uk`. 1

[10] D. Kolb. *Experiential Learning: experience as the source of learning and development.* Prentis-Hall, New Jersey, 1984. 5

[11] Geoffrey Bowker. **THE NEW KNOWLEDGE ECONOMY AND SCIENCE AND TECHNOLOGY POLICY**. *3rd Annual MIT/UCI Knowledge and Organizations Conference, CA*, 2004. 5

[12] **Merriam-Webster Online Dictionary**, Jan 2013. Available online at `http://www.merriam-webster.com/dictionary/provenance`. 6

[13] Jun Zhao, Chris Wroe, Carole Goble, Robert Stevens, Dennis Quan, and Mark Greenwood. **Using Semantic Web Technologies for Representing E-science Provenance**. In SheilaA. McIlraith, Dimitris Plexousakis, and Frank Harmelen, editors, *The Semantic Web Ű ISWC 2004*, **3298** of *Lecture Notes in Computer Science*, pages 92–106. Springer Berlin Heidelberg, 2004. 6, 15

[14] Wang chiew Tan. **Research problems in data provenance**. *IEEE Data Engineering Bulletin*, **27**:45–52, 2004. 6

[15] Peter Buneman, Sanjeev Khanna, and Wang chiew Tan. **Why and Where: A Characterization of Data Provenance**. In *In ICDT*, pages 316–330. Springer, 2001. 6

[16] John Evdemon. **Principles of Service Design: Service Versioning**, 2005. Available online at `http://msdn.microsoft.com/en-us/library/ms954726.aspx`. 7, 33

[17] Kyle Brown and Michael Ellis. **Best practices for Web services versioning**, Jan 2004. Available online at `http://www.ibm.com/developerworks/webservices/library/ws-version/`. 7, 33

[18] Roger S. Barga, Yogesh Simmhan, Eran Chinthaka Withana, SatyaSahoo, Jared Jackson, and Nelson Araujo. **Provenance for Scientific Workflows: Towards Reproducible Research**. **33**:50–59, 2010. 10, 11, 28, 29

[19] S. Fomel and G. Hennenfent. **Reproducible Computational Experiments using Scons**. In *International Conference on Acoustics, Speech, and Signal Processing*, **4**, 2007. 10

[20] J. P. Mesirov. **Accessible Reproducible Research**. *Science*, **327**:415–416, 2010. 10, 14

[21] Simon Woodman, Hugo Hiden, Paul Watson, and Paolo Missier. **Achieving reproducibility by combining provenance with service and workflow versioning**. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, WORKS '11, pages 127–136, New York, NY, USA, 2011. ACM. 10, 11, 17, 28, 29

[22] Jon Claerbout and Martin Karrenbach. **Electronic documents give reproducible research a new meaning**. In *Proc. 62nd Ann. Int. Meeting of the Soc. of Exploration Geophysics*, pages 601–604, 1992. 11

[23] **Stanford Exploration Project Website**. Available online `http://sepwww.stanford.edu/doku.php`. 11

[24] R. J. LeVeque. **Python Tools for Reproducible Research on Hyperbolic Problems**. *Computing in Science and Engineering*, **11**:19–27, 2009. 11

[25] Jonathan B. Buckheit, Jonathan B. Buckheit, David L. Donoho, and David L. Donoho. **WaveLab and Reproducible Research**, 1995. 11

[26] R.D. Peng and S.P. Eckel. **Distributed Reproducible Research Using Cached Computations**. *Computing in Science Engineering*, **11**(1):28 –34, jan.-feb. 2009. 11

[27] Louis Bavoil, Steven P. Callahan, Carlos Eduardo Scheidegger, Huy T. Vo, Patricia Crossno, Cláudio T. Silva, and Juliana Freire. **VisTrails: Enabling Interactive Multiple-View Visualizations**. In *Visualisation, VIS, 2008 International Conference*, 2005. 11

[28] Eran Chinthaka Withana, Beth Plale, Roger Barga, and Nelson Araujo. **Versioning for Workflow Evolution**. In *Proceedings of The Third International Workshop on Data Intensive Distributed Computing*. Association for Computing Machinery, Inc., June 2010. 11, 28

[29] Sean Bechhofer, David De Roure, Matthew Gamble, Carole Goble, and Iain Buchan. **Research objects: Towards exchange and reuse of digital knowledge**. *The Future of the Web for Collaborative Science*, 2010. 11

[30] Khalid Belhajjame, Oscar Corcho, Daniel Garijo, Jun Zhao, Paolo Missier, David Newman, Raul Palma, Sean Bechhofer, Esteban García-Cuesta, and et al. **Workflow-Centric Research Objects: First Class Citizens in Scholarly Discourse**. 11

[31] Juliana Freire and Claudio T. Silva. **Making Computations and Publications Reproducible with VisTrails**. *Computing in Science Engineering*, **14**(4):18–25, July 2012. 11, 28, 29

[32] Paolo Missier, Simon Woodman, Hugo Hiden, and Paul Watson. **Provenance and data differencing for workflow reproducibility analysis**. *CoRR*, **abs/1406.0905**, 2014. 11, 28

[33] **Reproducible Research Website**, 2013. Available online at `http://reproducibleresearch.net/index.php/Main_Page`. 14

[34] Roger D. Peng. **Reproducible Research in Computational Science**. *Science*, **334**(6060):1226–1227, 2011. 14

[35] Paul Groth. *The Origin of Data: Enabling the Determination of Provenance in Multi-institutional Scientific Systems through the Documentation of Processes*. PhD thesis, University of Southampton, October 2007. 14, 33

[36] Paul Groth, Michael Luck, and Luc Moreau. **A Protocol for Recording Provenance in Service-Oriented Grids**. In Teruo Higashino, editor, *Principles of Distributed Systems*, **3544** of *Lecture Notes in Computer Science*, pages 124–139. Springer Berlin Heidelberg, 2005. 14

[37] Robert Stevens, Jun Zhao, and Carole A. Goble. **Using provenance to manage knowledge of In Silico experiments**. *Briefings in Bioinformatics*, **8**:183–194, 2007. 15, 29

[38] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T. Silva. **Provenance for Computational Tasks: A Survey**. *Computing in Science and Engineering*, **10**:11–21, 2008. 16

[39] **Genealogy - Wikipedia**. Available online `http://en.wikipedia.org/wiki/Genealogy`. 17

[40] Simon Miles, Ewa Deelman, Paul Groth, Karan Vahi, Gaurang Mehta, and Luc Moreau. **Connecting Scientific Data to Scientific Experiments with Provenance**. 17

[41] Ilkay Altintas, Oscar Barney, and Efrat Jaeger-frank. **Provenance Collection Support in the Kepler Scientific Workflow System**. pages 118–132, 2006. 17

[42] Paul T. Groth and Luc Moreau. **Recording Process Documentation for Provenance**. *IEEE Transactions on Parallel and Distributed Systems*, **20**:1246–1259, 2009. 17

[43] James Cheney, Umut A. Acar, and Amal Ahmed. **Provenance Traces**. *CoRR*, **abs/0812.0564**, 2008. 17

[44] Simon Miles, Paul T. Groth, Steve Munroe, Sheng Jiang, Thibaut Assandri, and Luc Moreau. **Extracting causal graphs from an open provenance data model**. *Concurrency and Computation: Practice and Experience*, **20**:577–586, 2008. 18

[45] Paolo Ciccarese, Stian Soiland-Reyes, Khalid Belhajjame, Alasdair J. G. Gray, Carole A. Goble, and Tim Clark. **PAV ontology: Provenance, Authoring and Versioning**. *CoRR*, **abs/1304.7224**, 2013. 18

[46] Satya S. Sahoo, Christopher Thomas, and Amit Sheth. **Knowledge modeling and its application in life sciences: A tale of two ontologies**. In *In Proceedings of WWW*, 2006. 21

[47] Satya S. Sahoo, D. Brent Weatherly, Raghava Mutharaju, Pramod Anantharam, Amit Sheth, and Rick L. Tarleton. **Ontology-Driven Provenance Management in eScience: An Application in Parasite Research**. In *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part II*, OTM '09, pages 992–1009, Berlin, Heidelberg, 2009. Springer-Verlag. 21

[48] **OPM Provenance Challenge**. Available online at `http://twiki.ipaw.info/bin/view/Challenge/OPM`. 22, 40, 67

[49] Luc Moreau, Bertram Ludäscher, Ilkay Altintas, Roger S. Barga, Shawn Bowers, Steven Callahan, Carole Goble, Jennifer Golbeck, Paul Groth, A. Holl, Sheng Jiang, Jihie Kim, David Koop, Ales Krenek, Timothy Mcphillips, Gaurang Mehta, Simon Miles, Dominic Metzger, Steve Munroe, Beth Plale, Norbert Podhorszki, Emanuele Santos, Carlos Scheidegger, Karen Schuchardt, Margo Seltzer, Yogesh L, Claudio Silva, Peter Slaughter, Robert Stevens, Daniele Turi, Huy Vo, Mike Wilde, Jun Zhao, Yong Zhao, and et al. **The First Provenance Challenge**, 2000. 26

[50] **The OPM Provenance Model (OPM) - Open Provenance Model Website**. Available online at `http://openprovenance.org/`. 26

[51] André Freitas, Tomas Knap, Seán O'Riain, and Edward Curry. **W3P: Building an OPM based provenance model for the Web**. *Future Generation Comp. Syst.*, **27**(6):766–774, 2011. 27

[52] Paul Groth and Luc Moreau. **Representing distributed systems using the Open Provenance Model**. *Future Gener. Comput. Syst.*, **27**(6):757–765, June 2011. 27

[53] **Provenance Vocabulary Mappings - W3C**. Available online at `http://www.w3.org/2005/Incubator/prov/wiki/Provenance_Vocabulary_Mappings`. 27

[54] Douglas K. Barry and Patrick J. Gannon. *Web Services and Service-Oriented Architecture: The Savvy Manager's Guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003. 31

[55] **Extensible Markup Language (XML) 1.0 (Fifth Edition) - W3C Website**. Available online at `http://www.w3.org/TR/REC-xml/`. 31

[56] **Uniform Resource Locators (URL) Specification**. Available online at `http://www.w3.org/Addressing/URL/url-spec.txt`. 31

[57] **Web Services Description Language (WSDL) 1.1**. Available online at `http://www.w3.org/TR/wsdl`. 31, 53

[58] **SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)**. Available online at `http://www.w3.org/TR/soap12-part1/`. 31

[59] **UDDI Version 3.0.2 - OASIS Website**. Available online at `http://uddi.org/pubs/uddi_v3.htm`. 31, 53

[60] Ru Fang, Linh Lam, Liana Fong, D. Frank, C. Vignola, Ying Chen, and Nan Du. **A Version-aware Approach for Web Service Directory**. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 406 –413, july 2007. 34, 64

[61] Ru Fang, Ying Chen, Liana Fong, L. Lam, D. Frank, C. Vignola, and Nan Du. **A Version-aware Approach for Web Service Client Application**. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 401 –409, 21 2007-yearly 25 2007. 34

[62] D. Frank, Linh Lam, Liana Fong, Ru Fang, and M. Khangaonkar. **Using an Interface Proxy to Host Versioned Web Services**. In *Services Computing, 2008. SCC '08. IEEE International Conference on*, **2**, pages 325 –332, july 2008. 34

[63] **VisTrails Website**. Available online at `http://www.vistrails.org/index.php/Main_Page`. 34

[64] **The Kepler Project Website**. `https://kepler-project.org/`. 34

[65] Tom Oinn, Mark Greenwood, Matthew Addis, M. Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter Li, Phillip Lord, Matthew R. Pocock, Martin Senger, Robert Stevens, Anil Wipat, and Chris Wroe. **Taverna: lessons in creating a workflow environment for the life sciences: Research Articles**. *Concurr. Comput. : Pract. Exper.*, **18**(10):1067–1100, August 2006. 35

[66] P. Lambe. *Organizing Knowledge: Taxonomies, Knowledge and Organization Effectiveness*. Chandos Knowledge Management Series. Chandos, 2007. 36

[67] Paul Groth, Steve Munroe, Simon Miles, and Luc Moreau. **Applying the Provenance Data Model to a Bioinformatics Case**. 44

[68] O. Zimmermann, M. Tomlinson, and S. Peuser. *Perspectives on Web Services: Applying SOAP, WSDL and UDDI to Real-World Projects*. Springer Professional Computing Series. Springer, 2003. 52, 53

[69] L. Yu. *Introduction to the Semantic Web and Semantic Web Services*. Taylor & Francis, 2007. 55, 92

[70] S Vinoski. **The more things changed**. *Internet Computing*, **8(1)**:87–89, 2004. 58

[71] Paolo Missier and Carole Goble. **Workflows to open provenance graphs, round-trip**. *Future Gener. Comput. Syst.*, **27**(6):812–819, June 2011. 67, 79

[72] A. Boukottaya, C. Vanoirbeek, F. Paganelli, and O. Abou Khaled. **Automating XML documents transformations: a conceptual modelling based approach**. In *Proceedings of the first Asian-Pacific conference on Conceptual modelling - Volume 31*, APCCM '04, pages 81–90, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc. 71

[73] **Apache jUDDI Project Website**. Available online at `http://juddi.apache.org/`. 87

[74] Bill Howe, Peter Lawson, Renee Bellinger, Erik Anderson, Emanuele Santos, Juliana Freire, Carlos Scheidegger, António Baptista, and Cláudio Silva. **End-to-End eScience: Integrating Workflow, Query, Visualization, and Provenance at an Ocean Observatory**. 118

[75] C.T. Silva, J. Freire, and S.P. Callahan. **Provenance for Visualizations: Reproducibility and Beyond**. *Computing in Science Engineering*, **9**(5):82 –89, sept.-oct. 2007. 118