Achieving Reproducibility Incorporating Service Versioning into Provenance Model

Dayang Hanani Abang Ibrahim[1], Nadianatra Musa[1], Chiew Kang Leng[2], Jane Labadin[2],
Johari Abdullah[3], Sarina Sulaiman[4]
[1]Department of Information Systems,
[2]Department of Computational Science and Mathematics,
[3]Department of Computer System and Communication Technology,
Faculty of Computer Science and Information Technology,
Universiti Malaysia Sarawak, Kota Samarahan, 94300 Sarawak, Malaysia.
[4]UTM Big Data Centre, Universiti Teknologi Malaysia, Skudai, 81310 Johor, Malaysia.
hananii@unimas.my

Abstract—**Reproducibility has long been a cornerstone of science. Underpinning reproducibility is provenance, which has the potential to provide scientists with a complete understanding of data generated in e-experiments, including the services that were produced and consumed. This paper explores the issues of service versioning in provenance to achieve reproducibility. Current provenance model does not directly support service versioning. Therefore, this paper introduces an enhancement of a provenance model to incorporate service versioning mechanism that provides a way to access multiple versions of the same service so that researcher can compare one version to another, and understand their effects on processing data. The enhanced provenance model is able to track the changes of the same service (versions of the same service) over time and correlates versioned services with the results they generate.**

Index Terms—**Reproducibility; Provenance; Provenance Model; Service Versioning; Web Services Architecture;**

I. INTRODUCTION

Provenance is particularly important when a scientific e-experiment is to be reproduced and re-run. Provenance provides the ability to reproduce all the steps leading to a scientific e-experiment result. This means provenance can show how the result was generated, thus illustrating how the experiment was done before. Pizzi et al. [1] uses directed acyclic graphs to track the provenance of data and calculations in computational science to ensure reproducibility. A service is a unit of work that performs a computation that can be consumed by clients or consumers in applications or experiments. When a workflow is executed, a sequence of services is invoked. Provenance enables the recording of these services, including the data parameters used, and also timestamps of service invocations. Looking inside each of these services, there are also service metadata that may also be significant and therefore needs to be recorded in provenance; for example, when a particular service was created and which version it is. In existing provenance literature, versioning has not been directly supported in provenance model. It is often the case that a service will need to change after its initial deployment to fix bugs, improve the algorithm, or meet new requirements. Therefore, service versioning should be supported to ensure that even after new versions of a service are deployed; the old version still remains available. This evolution of services will eventually lead to multiple versions of a service, starting with the current version, and leading back to older versions that have in the past been used to generate data that may still be in use. This piece of service metadata is important for reproducibility. Therefore, reproducibility not only gives relevant information to permit the re-running of the experiment but also to look at the versions of a service that have been invoked in the experiment. This approach opens up the opportunity for discovery in examining the history of the service. 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 [2][3][4]. These works discuss the motivation for reproducibility, as well as describing infrastructure to support it.

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 may 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 paper we argue that versioning is an essential mechanism needed to support experimental reproducibility.

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 i) insufficient information describing the experiment and ii) 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.

II. BACKGROUND

Reproducibility is a cornerstone of science and is 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 recognize 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 [5] experiential learning theory, experience from the past can be taken as the source of learning for the future.

However, how can Scientist B reproduce the experiment? Is there a database where he can download all the required microbial protein sequence data? Bowker [6] 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. However, achieving this is not straightforward, and is therefore the key focus of the work described in this paper. 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: Virtuous cycle of reproducibility

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 [7]. The need to have a provenance trace of the experiment 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 [8], Open Provenance Model (OPM) [9], PROV [10], ProvONE [11] and Prov2ONE [12]. This paper 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.

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 directly supported in provenance model. 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.

The objective of this paper is to explore the issues of service versioning in provenance to achieve reproducibility. This paper introduces an enhancement of a provenance model to incorporate service versioning mechanism that provides a way to access multiple versions of the same service so that researcher can compare one version to another, and understand their effects on processing data.

III. PROPOSED SERVICE VERSIONING AND ITS APPROACHES

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. 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.

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: i) even after new versions of a service are deployed, the old version still remains available and ii) 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 paper the focus is on services using Web Services technology.

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, however two web service versioning approaches are now considered that are using XML Namespaces and using tModels in the Universal Description, Discovery, and Integration (UDDI) registry. UDDI is an XML-based standard for describing, publishing, and finding web services [13].

The first approach is using XML Namespaces. This approach creates an entirely new Web Service with a new Web Service Definition Language (WSDL) [14] 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.

Both service versioning approaches take WSDL documents as important documents in managing versions of multiple services. Fang et al. [15] 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. [16] 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.

IV. INCORPORATE 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 [17], provides user to take the web services and connect the services into workflows and execute them. 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. The service versioning convention is as shown in Figure 2.
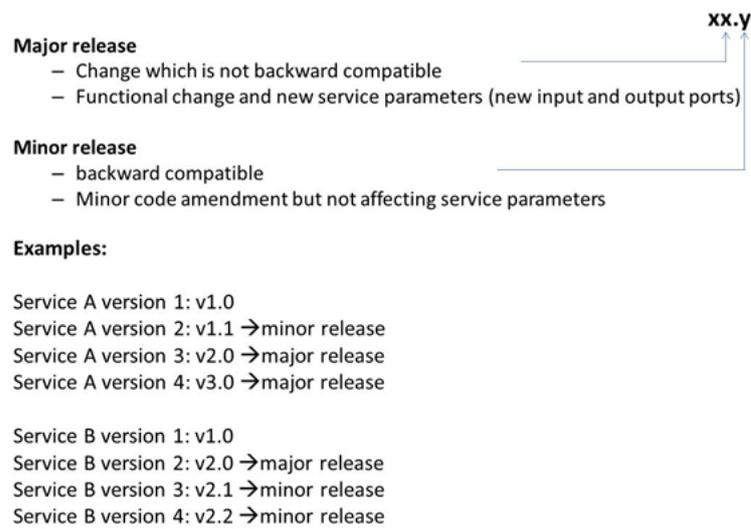
Major release
- Change which is not backward compatible
- Functional change and new service parameters (new input and output ports)

Minor release
- backward compatible
- Minor code amendment but not affecting service parameters

Examples:

Service A version 1: v1.0
Service A version 2: v1.1 →minor release
Service A version 3: v2.0 →major release
Service A version 4: v3.0 →major release

Service B version 1: v1.0
Service B version 2: v2.0 →major release
Service B version 3: v2.1 →minor release
Service B version 4: v2.2 →minor release

Figure 2: Service versioning convention scheme
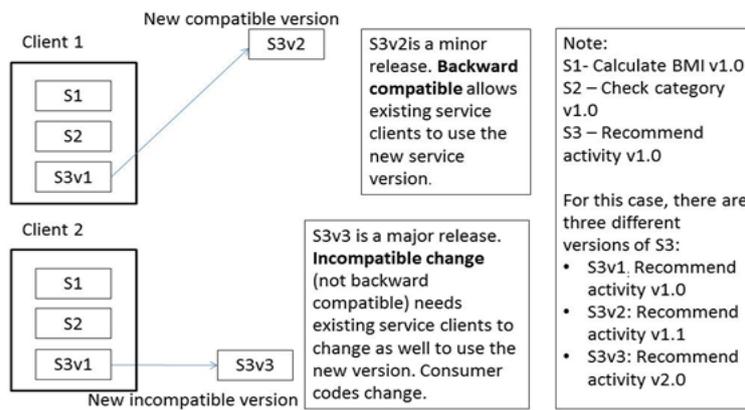
Figure 3: Compatible and incompatible changes in service update

Figure 3 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 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.

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. 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.

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 paper, 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. 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 4. 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.
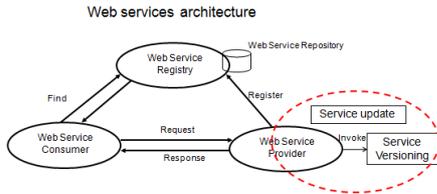

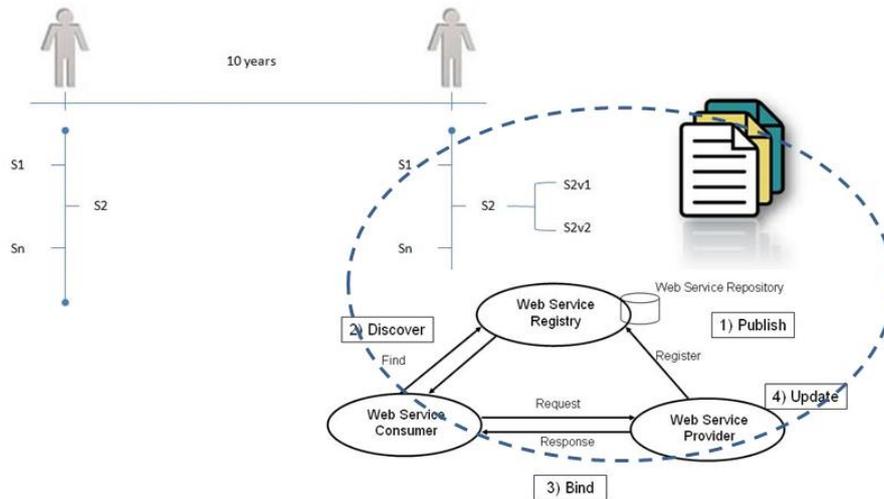Figure 4: The Web Service Architecture extended with service update

Figure 5: Two versions of S2; S2v1 and S2v2

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 practice 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.

V. CAPTURING VERSIONING IN OPM

In this section, the focus is extending the current OPM to support versioning of web services. Versioning is important because web services evolve over time due to many reasons. An OPM model has three main nodes and five types of edges representing the causal dependencies. The nodes as illustrated in Figure 6 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 paper, 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

Figure 5 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.

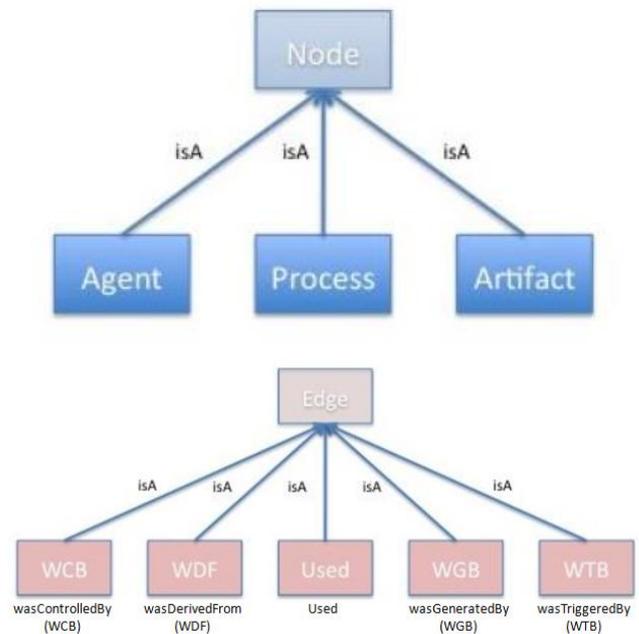how the multiple versions of the same service are linked together as one collection.

Figure 6: Open Provenance Model

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 have 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 paper introduces an extension of the edges type in causal dependencies with opm:wasVersionOf. Abang Ibrahim [18] 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 7 as 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 7: The model wasVersionOf edge

Each service can change 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 8. 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. Upon an execution run in a SOA system, the input and output data parameters are stored in Experiment repository.
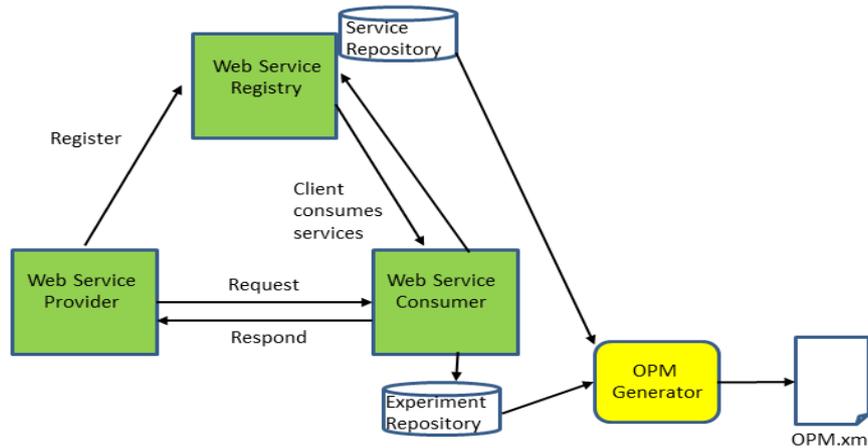

Figure 8: OPM Generator

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. 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 as follows:

- 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 before Service3V2.
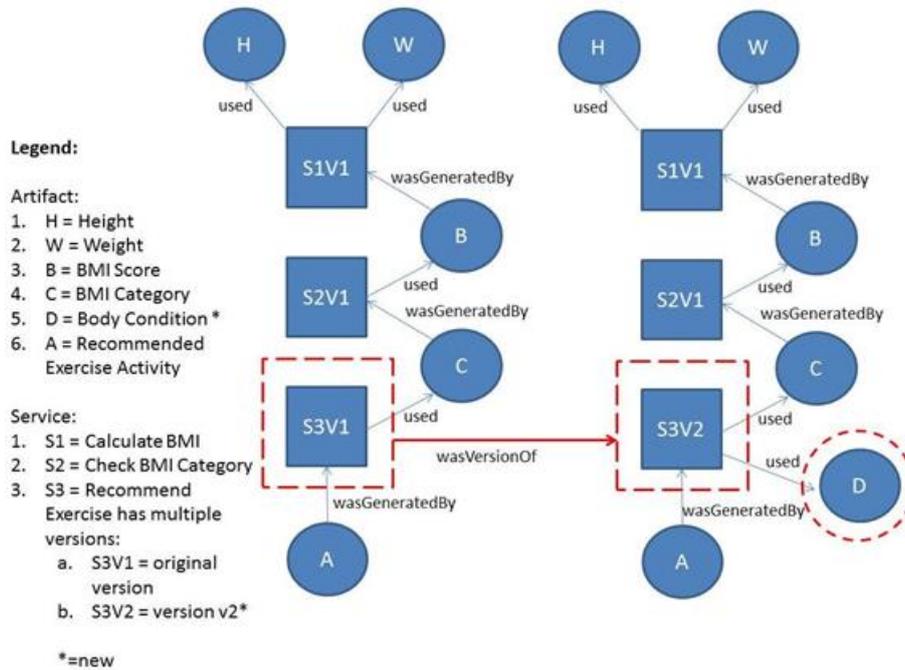
Figure 9: A description of an execution run that shows the versioning relationship from one service S3v1 to another service

Figure 9 illustrates an execution run that shows the versioning relationship from one service S3v1 to another service. 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 Figure 10. 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 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.

```
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>
```

Figure 10: wasVersionOf in OPM trace

The provenance trace must describe the version of the service used in the execution. Using the tModel approach, 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 [15, 16]. 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 facilitate the discovery of versions of a service.

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 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.

VI. CONCLUSION

In conclusion, the OPM model has been extended to represent the experimental execution, encompassing services, by introducing wasVersionOf causal dependency in OPM trace. Thus, service versioning can be incorporated into provenance to address deficiencies in the existing provenance model.

Service versioning mechanism provides a way to access multiple versions of the same service so that researcher can compare one version to another, or has an option to access another version of service if the current service is not available. This research has the potential to provide advantage over existing provenance model in incorporating versioning in service provenance. Since this paper 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. This template will also incorporate subscription services to inform consumer that a new service is available.

REFERENCES

[1] G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, and B. Kozinsky, "AiiDA: automated interactive infrastructure and database for computational science," Comput. Mater. Sci., vol. 111, pp. 218–230, Jan. 2016.

[2] R. S. Barga, Y. L. Simmhan, E. Chinthaka, S. S. Sahoo, J. Jackson, and N. Araujo, "Provenance for Scientific Workflows Towards Reproducible Research.," IEEE Data Eng Bull, vol. 33, no. 3, pp. 50–58, 2010.

[3] S. Fomel and G. Hennenfent, "Reproducible Computational Experiments using Scons," in 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07, 2007, vol. 4, p. IV-1257-IV-1260.

[4] S. Woodman, H. Hiden, P. Watson, and P. 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, New York, NY, USA, 2011, pp. 127–136.

[5] D. A. Kolb, Experiential Learning: Experience as the Source of Learning and Development. FT Press, 2014.

[6] G. C. Bowker, "The new knowledge economy and science and technology policy," in Science and Technology Policy - Volume I, vol. 1, 2004.

[7] Y. L. Simmhan, B. Plale, and D. Gannon, "A Survey of Data Provenance in e-Science," SIGMOD Rec, vol. 34, no. 3, pp. 31–36, Sep. 2005.

[8] S. Sahoo and A. Sheth, "Provenir Ontology: Towards a Framework for eScience Provenance Management," Knoesis Publ., Oct. 2009.

[9] L. Moreau et al., "The Open Provenance Model core specification (v1.1)," Future Gener. Comput. Syst., vol. 27, no. 6, pp. 743–756, Jun. 2011.

[10] P. Missier, K. Belhajjame, and J. Cheney, "The W3C PROV Family of Specifications for Modelling Provenance Metadata," in Proceedings of the 16th International Conference on Extending Database Technology, New York, NY, USA, 2013, pp. 773–776.

[11] "The ProvONE Data Model for Scientific Workflow Provenance." [Online]. Available: http://vcvcomputing.com/provone/provone.html. [Accessed: 29-Apr-2017].

[12] A. Prabhune, A. Zweig, R. Stotzka, M. Gertz, and J. Hesser, "Prov2ONE: An Algorithm for Automatically Constructing ProvONE Provenance Graphs," in Provenance and Annotation of Data and Processes, 2016, pp. 204–208.

[13] "The UDDI XML." [Online]. Available: http://uddi.xml.org/uddi-org. [Accessed: 29-Apr-2017].

[14] "XML WSDL." [Online]. Available: https://www.w3schools.com/xml/xml_wsdl.asp. [Accessed: 29-Apr-2017].

[15] R. Fang et al., "A Version-aware Approach for Web Service Directory," in IEEE International Conference on Web Services (ICWS 2007), 2007, pp. 406–413.

[16] D. Frank, L. Lam, L. Fong, R. Fang, and M. Khangaonkar, "Using an Interface Proxy to Host Versioned Web Services," in 2008 IEEE International Conference on Services Computing, 2008, vol. 2, pp. 325–332.

[17] K. Wolstencroft et al., "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud," Nucleic Acids Res., vol. 41, no. W1, pp. W557–W561, Jul. 2013.

[18] D. H. AbangIbrahim, "The Exploitation of Provenance and Versioning in the Reproduction of e-Experiments," PhD Thesis, University of Newcastle.United Kingdom, UK, 2016.