Capturing Service Versioning in Provenance Trace to Support Reproducibility

Dayang Hanani Abang Ibrahim, Chiew Kang Leng and Nadianatra Musa
Faculty of Computer Science and Information Technology,
Universiti Malaysia Sarawak, Kota Samarahan, 94300 Sarawak, 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. A key to reproducibility is the provenance model: a data model that structures information about an e-experiment. 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 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. This paper describes in greater detail the conceptualization of an experiment using the Open Provenance Model (OPM). As Open Provenance Model (OPM) is the provenance model standard, this paper explores whether the OPM is able to describe an experiment sufficiently precisely so as to support reproducibility. The paper 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.**

***Index Terms*—Provenance; Provenance Trace; Service Versioning;**

I. Introduction

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 the provenance model standard [1], this work explores whether the OPM is able to describe an experiment sufficiently precisely so as to support reproducibility. The paper 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 objectives of this paper 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.

II. Motivation

In this work, the motivation is as follows:

A. *Capturing Experiments Using Open Provenance Model (OPM)*

Capturing experiments involves recording information on experimental components, procedures and versions. 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.

B. *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 1:

Table 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)	Used
BMI Category	Check BMI Category	wasGeneratedBy
BMI Category	Service 3 (S3)	Used
Exercise Activity	Recommend 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 1.

Figure 1 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 next section.

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.

An additional scenario is to include a person's daily free time as requested by the users due to their daily tight schedule that prevents 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.

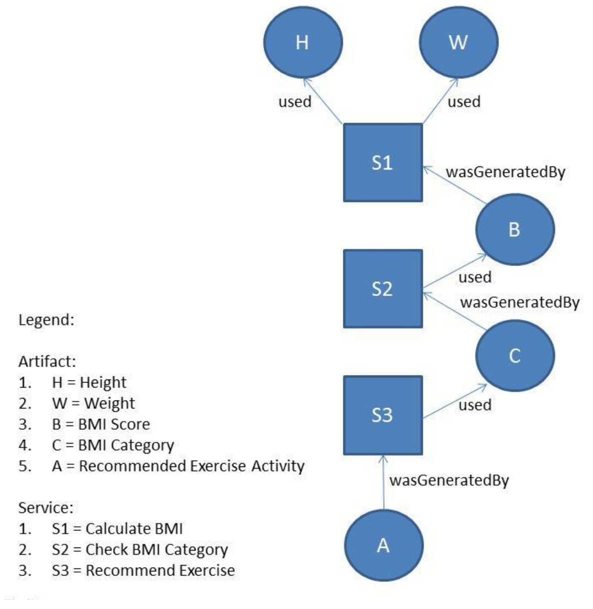

Figure 1: OPM representing the experiment to compute Exercise Advisor

C. *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 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 [2], this work 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.

D. *A gap in 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 2.

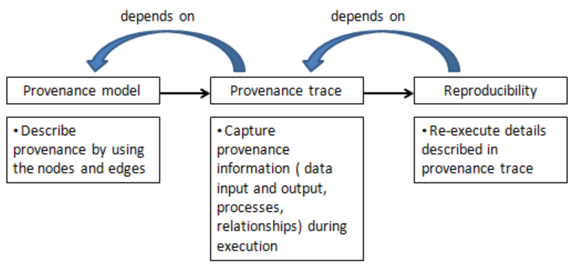
depends on
depends on
Provenance model
Provenance trace
Reproducibility
• Describe provenance by using the nodes and edges
• Capture provenance information (data input and output, processes, relationships) during execution
• Re-execute details described in provenance trace

Figure 2: The dependency of provenance for reproducibility

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 [3]. 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 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.

III. Method

In this section, the focus is extending the current OPM to support versioning of web services. According to [4], 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 3 denotes 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 how the multiple versions of the same service are linked together as one collection.

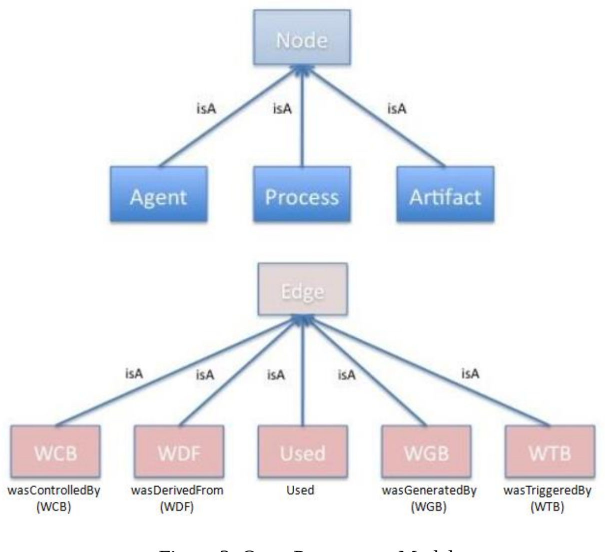

Figure 3: Open Provenance Model

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. [5] 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 4 as below.
- A set of services, thus a collection. Extension of attribution of a causal relationship to provide further information on how one occurrence relates to the previous occurrence.

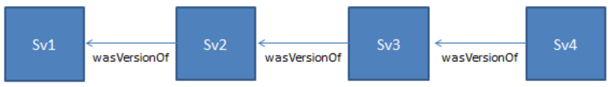

Figure 4: The model wasVersionOf edge

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 5. 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 Web Service Architecture system, the input and output data parameters are stored in Experiment repository.

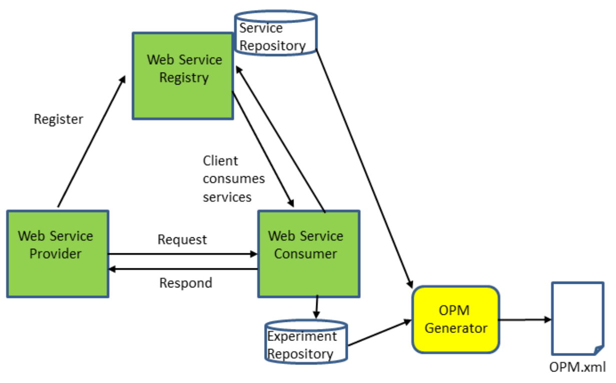

Figure 5: OPM Generator

Why are web services 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 WSDL can be registered by the service provider (owner) to service registry to publish the location of available services. However, what happens if the services have been removed by their owners? The service may become inaccessible. Therefore, if service version is recorded, another alternative of same services can be recommended. The tracing of these services is possible. It is recommended that service versioning is recorded at the early stage of service creation by the service provider (owner).

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.

For example, 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 6. 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.

```
 <opm:wasVersionedOf>
      <opm:effect id="myActivity1a" />
        <opm:role value="serviceVersion" />
<opm:cause id="myActivity1" />
   <opm:account id="account1" />
</opm:wasVersionOf>
```

Figure 6: 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.

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.

IV. Discussion

This paper 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. It is recommended that service versioning is handled at the early stage of service creation by service provider or service owner. 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.

ACKNOWLEDGEMENTS

The funding for this project is made possible through the research grant obtained from UNIMAS and the Ministry of Education, Malaysia under the Fundamental Research Grant Scheme 2/2013. [Grant No:FRGS/ICT01(01)/1073/ 2013(19)]. The authors would also like to thank Universiti Malaysia Sarawak for providing the resources used in the conduct of this study.

REFERENCES

[1] The OPM Provenance Model (OPM) – Open Provenance Model Website. Available online at http://openprovenance.org/.

[2] P. Groth, S. Munroe, S. Miles, and L. Moreau, “Applying the Provenance Data Model to a Bioinformatics Case,” In Grandinetti, Lucio (eds.) *High Performance Computing and Grids in Action*, IOS Press, Advances in Parallel Computing, 16, pp. 250-264, 2008.

[3] L. Moreau, B. Clifford, J. Freire, J. Futrelle, J. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simhan, E. G. Stephan, and J. V. D. Bussche, “The Open Provenance Model core specification (v1.1),” *Future Generation Computer System*, vol. 27, no. 6, pp. 743–756, Jun. 2011.

[4] S. Vinoski, “The more things changed,” *Internet Computing*, vol. 8, no. 1, pp. 87-89, 2004

[5] D. H.Abang Ibrahim, “The Exploitation of Provenance and Versioning in the Reproduction of e-Experiments,” *PhD Thesis*, Newcastle University, United Kingdom, 2016.