The Design and Implementation of a Provenance Generator

William Martin

August 2012

MSc Internet Technologies and Enterprise Computing

Abstract

In the field of Information Technology, the term “provenance” refers to specific types of metadata that provide a documented history of a piece of data or object. This facility is becoming an important issue since it can be used to assess quality, verify reliability and debate trustworthiness. Provenance traces can be very large and complex; for example, when recording the scientific process for future reproducibility. One interesting area of work in IT involves mining provenance graphs. However, there is currently a lack of sufficiently large traces with interesting characteristics available to test upon. In addition, those that do exist are domain specific and not easily customisable.

This dissertation describes the construction of a software generator that facilitates the creation of “synthetic” traces via the provision of “seed” traces. These traces, which may be expressed as graphs, expand into the final provenance trace, which is encoded in PROV-N, a newly designed, standard language written by the W3C Provenance Working Group. Ultimately, this will assist developers of data mining algorithms to create useful provenance traces with which they can test, which are readily accessible, and in which they can have confidence.

Acknowledgements

I would like to thank my supervisor Dr Paolo Missier for his advice, support and assistance throughout the year. I would also like to thank my family and friends who have helped me maintain a healthy balance between work and leisure.

Declaration

I declare that the work within this dissertation represents my own work unless otherwise stated.

Signed …….. Date……

Table of Contents

List of Figures

1 Introduction

Provenance is concerned with allowing assessments about the quality or reliability of a piece of information. Recently, provenance has been gaining significant traction within the Information Technology community [1], especially within the scientific domain. With this increased use has come a demand for technological support, however, current tooling for provenance is lacking.

The project attempts to make forays into provenance tools, considering and implementing solutions that will be desired by engineers as uptake continues to increase. Specifically, an application will be provided that will aid the creation of traces to test the data mining and querying of provenance. Within the final application however, there will be contained solutions for other, more generic provenance challenges.

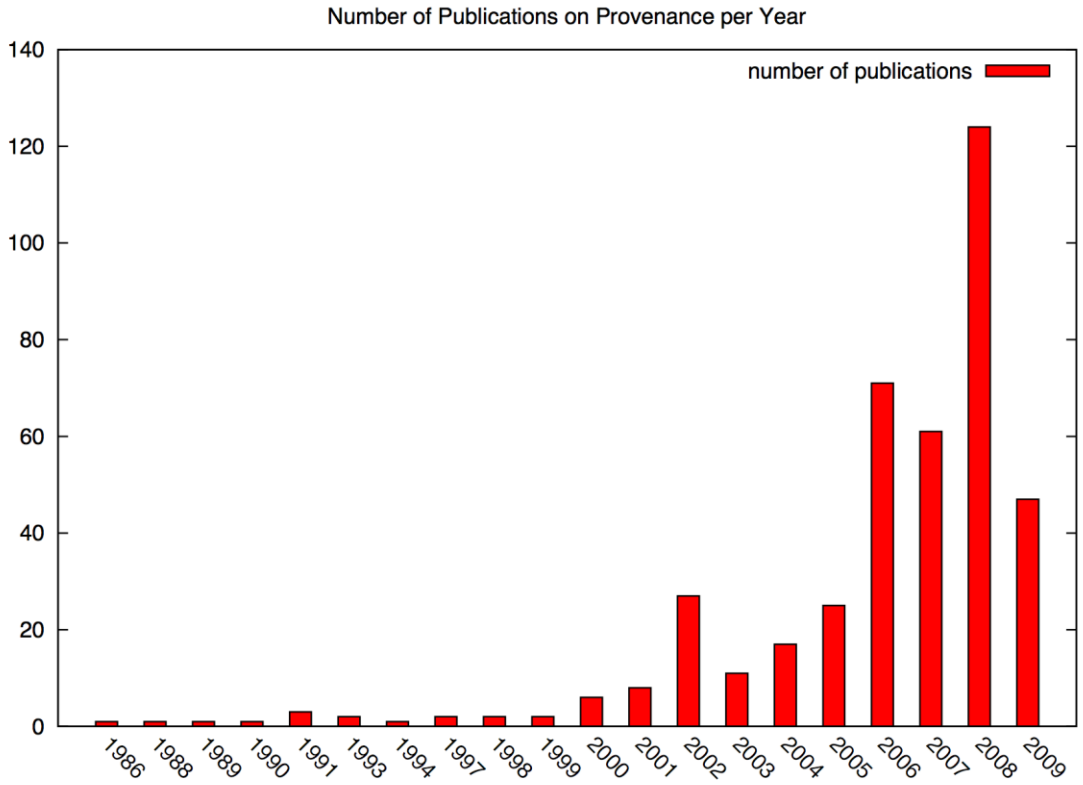


Figure 1: Number of Provenance Publications [1]

1.1 Overview of Provenance

The concept of provenance has existed for a considerable amount of time within the art community, providing information about the historicity of artefacts. In fact, the provenance of art objects is so vital that it is generally produced prior to auctions to establish value and help increase the price obtained. It is no wonder then, that provenance has found a home in I.T., particularly within the scientific community, where the authenticity and reproducibility of experimental data is of paramount importance [2].

Within I.T., provenance can have wide reaching applications, for example, businesses may utilise it in quality assurance processes or perhaps publication rights may be determined by the provenance of a photograph or document. Furthermore, in a scientific context, in each step of an experiment, the methods and tools employed may provide crucial insight into the reliability of results and act as guidance for reproducing the experiment in its entirety [3].

The structure and type of provenance provided about objects can vary significantly depending upon perspective; for example, one type of provenance may focus on who was involved in creating or manipulating an object whilst another may focus on the steps taken that result in that object being in its current state. Due to these disparities, an effort has been made within the provenance community to develop a standardised model that amalgamates these differing perspectives [4]. Since 2011, the W3C has developed a working draft named PROV, containing a conceptual data model (PROV-DM) [5] and formal notation (PROV-N) [6] among other additional works related to the PROV specification.

1.2 Motivation and Problem Solution

With this increased growth in the use of provenance, a number of new challenges and twists on old challenges have been exposed. One example of such a challenge is that as a large corpus of provenance traces becomes available, it is likely that there will be an effort to mine the information contained within. To efficiently mine data, developers require sample traces to test their algorithms upon. Unfortunately, the graphs currently available for this testing tend to fall under one of more of the following limiting categories:

- They do not conform to the standardised PROV Data Model
- They are domain specific and as such are not easily customisable
- They simply do not have sufficiently interesting characteristics

Within the current state of the art, developers requiring graphs that are suitable for algorithm training would need to either perform a tedious search for existing provenance traces that satisfy their criteria or alternatively, manually create the graphs [7], handwritten in PROV-N. To aid developers, the project aims to provide a way of easily generating “synthetic” graphs – graphs that are generic, e.g. not necessarily specific to any real object – that are customisable, conform to the PROV-DM and have interesting characteristics.

Data mining is a high level view of an area in which being able to generate these graphs would be useful. More specifically however, working on efficiently scaling storage solutions for these graphs would require the creation of very large graphs, which this project will allow. After storage, there is a requirement for scalable querying over provenance graphs and this requires performance testing, another field in which the project could help.

Finally, there are likely to be many orthogonal uses for the project that are not immediately obvious in the current state of the problem domain. As an example, at the time of writing, colleagues of the author were investigating visualising provenance graphs and indeed, the visualisation of very large graphs is a common research topic [8]. Having the ability to customise graphs easily for testing visualisation would have been very beneficial for their project.

1.3 Aims and Objectives

The chief aim of this project is to investigate and implement a method of generating easily modifiable, large provenance traces that will be encoded in the PROV-N formal language.

Objective 1

Explore and review the role and usage of provenance, graph theory and data mining literature.

Objective 2

Define a method of succinctly and explicitly describing seed traces that will act as the basis for generating full traces.

Objective 2.1

Determine the required client-configurable parameters.

Objective 2.2

Provide a syntax for defining the seed traces, including the additional parameters.

Objective 2.3

Define strict semantics for the syntax provided in Objective 2.2.

Objective 3

Develop a tool that provides methods for the creation of seed traces

Objective 3.1

Create an object model that maps to the PROV Data Model.

Objective 3.2

Provide functionality to expand seed graphs, implementing the semantics defined in Objective 2.3.

Objective 3.3

Provide functionality to convert the object model defined in Objective 3.1 to PROV-N.

Objective 4

Develop a GUI that will allow users to intuitively create seed traces.

Objective 5

Give the generation tool back to the provenance community.

1.4 Dissertation Structure

The entirety of the dissertation is structured as follows:

Chapter 1: Scene Setting

The document begins by attempting to lay a foundation for the reader in terms of document structure, with an overview of the problem domain and project motivation and aims. This will provide context and understanding for later decisions.

Chapter 2: Background Research

Chapter 2 provides an insight into the state of the art, examining existing research and technologies in the problem domain.

Chapter 3: Requirements

Having considered the background research combined with aims and objectives, this chapter provides a set of requirements the system must satisfy.

Chapter 4: System Design

Having identified aims, objectives and requirements, this chapter will discuss possible solutions. A high level design of the system will be presented with mock-ups and examples.

Chapter 5: Implementation

Utilising the design and mock-ups in Chapter 4, the actual implementation of the application will be covered, including any relevant tools and technologies employed.

Chapter 6: Evaluation

Chapter 6 will provide some insight into the strengths and weaknesses of the final system.

Chapter 7: Conclusions

Finally, there will be a review of the relative successes and limitations of the project to determine sections in which improvements could have been made. In addition, there will be considerable discussion given to design plans for future work.

2 Background Research

This chapter will introduce central concepts surrounding the project's domain that need to be identified in order to provide an appropriate solution. To begin, there will be a discussion on Provenance, the W3C specification to support it on the web (PROV) and a review of the existing tooling to support PROV. This discussion will not be exhaustive but exists to provide a foundational understanding of provenance and PROV to help understand the motivation behind design and implementation decisions. Subsequently, there will be brief consideration given to mining graphs and provenance; whilst not being essential, this will aid in providing additional context to the project whole.

2.1 Provenance

Although previously mentioned that provenance can be applicable to many fields, throughout the rest of this dissertation, for relevance, it will only be considered from a digital perspective. Earlier, there were some cursory references to how provenance might be useful, however, no clear definition has been provided. Understanding how provenance can be applied in a real scenario is crucial to the project. As such, below is a summary of a more practical application and many more scenarios can be found in the Provenance Incubator Group's Final Report [3].

2.1.1 Twitter Provenance

The twitter platform allows users to post short, 140 character messages called tweets. One common occurrence on twitter is known as retweets, where users repost another's tweet as their own. Initially, it is clear where a tweet originated, as it can be contained within the character limit but a message propagates around the platform, being retweeted multiple times, this 'in-message' provenance can be lost [9]. Twitter has now been introducing new functionality to allow this tweet metadata to be included outside of the message itself, allowing the provenance information to be preserved.

2.1.2 State of Provenance

Provenance is by no means a new field within the technology community; indeed, with the growth of the subject, there are now many definitions of provenance that emphasize differing viewpoints. Just some of these perspectives are provenance as a process [10], provenance as a directed, acyclic graph [11], why and where provenance [12] among a host of others that can be found within Luc Moreau's extensive publication, 'The Foundations of Provenance on the Web' [1].

A report produced by the W3C Incubator Group, discovered that the existing infrastructures to support provenance was insufficient. As a result, in 2011, the W3C began developing the PROV specification, which aims to provide an encompassing model for provenance on the web. The effort behind PROV is likely to push it to the forefront of the field and it will almost certainly become the most widely accepted framework for representing provenance information. As such, the next section will go into some detail regarding the specification itself, as the project will rely on it heavily.

2.1.3 PROV Specification

PROV is a new specification that aims to define the necessary components required to achieve the successful interchange of provenance information on the web and other similar environments [4]. As noted previously, provenance has no universal definition, however, the PROV Working Group have offered the following description that for the remainder of this dissertation will be adopted as the working definition:

'Provenance of a resource is a record that describes entities and processes involved in producing and delivering or otherwise influencing that resource. Provenance provides a critical foundation for assessing authenticity, enabling trust, and allowing reproducibility. Provenance assertions are a form of contextual metadata and can themselves become important records with their own provenance.' [3]

The PROV specification has a number of parts, including a primer, a data model, semantics for that model and how to create various representations e.g. in XML or RDF of the data model among others. However, four specification fragments are particularly relevant to this project.

- PROV-DM, a data model for provenance [5]
- PROV-CONSTRAINTS, a set of constraints applying to the PROV-DM [13]
- PROV-N, a notation for provenance for human consumption [6]
- PROV-AQ, mechanisms for querying provenance [14]

2.1.3.1 PROV-DM

The PROV Data Model provides a structure for the translation of provenance into a generic model that can be interchanged between systems. The PROV-DM is divided into a number of components covering various aspects of provenance and these in turn are members of either '*core'* or *'extended'* structures of provenance, where the former is domain-agnostic and the latter allowing domain-specific extensions to be defined. Further classification can still be performed by dividing many of the central concepts into types and relations, which will be the categorisation outlined below.

Types

Entities

At the heart of PROV, there needs to be a method of representing a 'thing' one wishes to provide provenance about. A 'thing' is an abstract notion, representing many possibilities, for example, a physical or digital object, or even an intangible concept or idea. In PROVDM, 'things' are described by 'entities' where an entity might be a book or a webpage.

Activities

As a primary function of provenance is the provision of information on how entities are formed or created and used, to this end, PROV provides an 'activity' type. An activity has a very broad definition, simply being something that occurs over a period of time, acting upon or with entities. Just like entities, activities can represent both the digital (copying a file) and physical (burning a CD).

Agents

In many cases, determining the reliability and trustworthiness of something involves knowing who is responsible for its creation or current status. Clearly, a publication from a recognised scientist is likely to carry more weight than that of a student. To address this idea of responsibility, PROV provides the concept of 'agents'; for example, this dissertation's author is an agent responsible for

its creation. Interestingly, agents may also be a type of entity or activity, providing the ability to express provenance about the agent itself.

Relations

Having considered the fundamental types in PROV, it is now important to consider how these can be connected; entities, activities and agents can all be related to each other but how? To answer this question, PROV proposes relations of which there are seven in the *'core structures'.*

Generation and Usage

Activities and Entities can be related in two ways: Generation, where an entity is created by an activity or Usage, where an activity can utilise an entity. For example, when a new text document is created, it may have been generated by a 'Saving' activity.

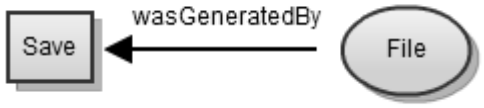


Figure 2: Generation of a File entity by a Save Activity

Likewise, a text document may be used by an 'Open' activity.

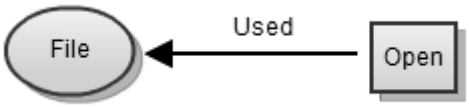


Figure 3: Usage of a File Entity by an Open Activity

Communication

An interesting combination of Generation and Usage occurs in the Communication relation. Communication occurs when an entity generated by one activity is then utilised by another. An example of this may be that when an application produces a file via saving, it is then opened by another application.

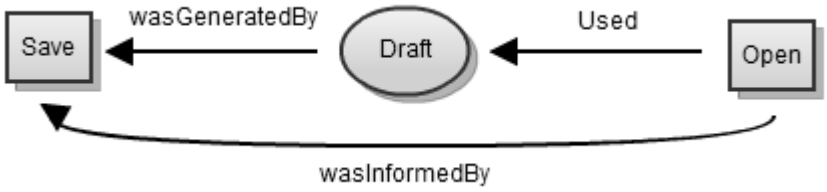


Figure 4: Communication between two activities

Derivation

Activities can both use and generate entities. In some cases, the act of using an entity may result in the generation of another. However, this relationship is more subtle than a simple generation and usage chain, requiring a level of influence between the utilised and generated entities. A good example of derivation is the revision of one file into a new one e.g. an edited version of a draft copy.

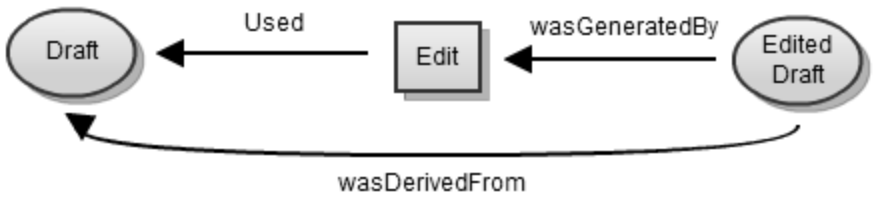


Figure 5: Derivation of an edited file from a draft

Responsibility Relations

Up to this point, there has only been discussion of the relationships that deal with how entities and activities interact. It is very important to have the ability to ascribe responsibility for an activity or ownership of an entity to an agent.

Attribution

The most obvious type of relation available is that of attribution or, to whom an entity should be credited. For example, a paper is attributed to its author.

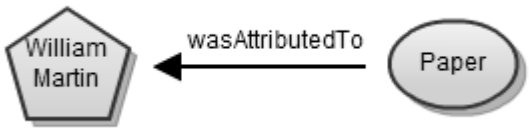


Figure 6: Attribution of paper to author

Association

In addition to being responsible for an entity, an agent may also be associated with an activity. The writing of a paper is one such example of an activity that the author, an agent, is responsible for.

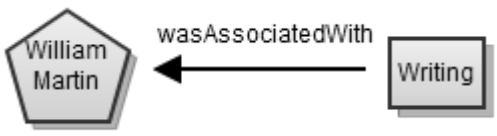


Figure 7: Association of writing activity with its author

Delegation

Finally, agents can also be related to other agents (or themselves) through delegation. Whilst a paper is attributed to the author, the author may in fact be working on behalf of an academic supervisor.

Figure 8: Delegation of student on behalf of supervisor

Extended PROV-DM Relations

So far, this dissertation has only covered the core of PROV-DM, however, there are many extended structures available. A good example of this is derivation; there are a number of subtypes such as Revision, Quotation and Primary Source. Whilst these extended structures are addressed within the project, they are not necessary for a fundamental understanding of provenance and therefore, to understand the project.

Extensibility Points

The PROV namespace, identified at the URI http://www.w3.org/ns/prov# affords a number of reserved attributes (type, role and location) to further facilitate the creation of application and domain specific provenance. For example, when describing attribution, the PROV-DM structures presented to date were an Agent, an Entity and an Attribution relation. However, not included in the model was the manner in which the entity was attributed to the agent. To solve this issue, the 'type' extensibility point can be used to state that the attribution was that of authorship. Additionally, it is possible to make use of other namespaces, such as Dublin Core or FOAF to provide extensibility without redundancy.

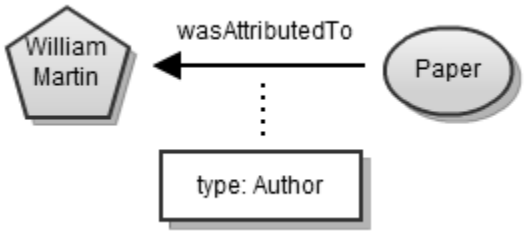


Figure 9: prov:type extensibility point

2.1.3.2 PROV-CONSTRAINTS

The PROV-DM specification defines a data model for expressing provenance on the web. Unfortunately, it is very possible to create provenance descriptions that do not make sense. This is particularly true when considering time in provenance, where one could, paradoxically, express that the starting time of an activity was after the creation time of an entity that it supposedly generated.

In general, PROV graphs are directed, acyclical graphs but there are some relations which allow cycles to occur such as in the following, where an activity was initiated by another:

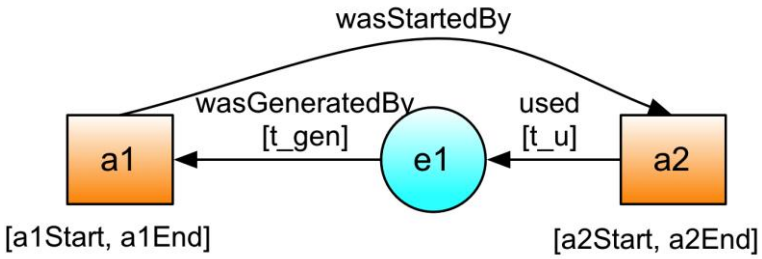


Figure 10: A legal cycle in PROV [15]

However, there are many instances that can occur which would result in illegal graphs such as entities being derived from one another in a cycle. The following graph does not make sense, as 'e0' was derived from 'e2' but 'e0' must have existed prior to the creation of 'e2'.

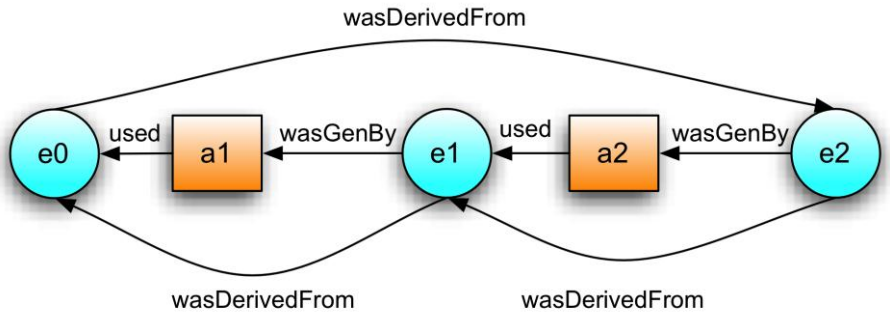


Figure 11: An illegal cycle in PROV [15]

As a result of these logic issues impacting consistency in PROV, the W3C Working Group released a specification called PROV-CONSTRAINTS to provide restrictive conditions that valid PROV models must satisfy. Whilst there are many rules conveyed by the specification, concerning uniqueness and ordering, they are not relevant to understanding the project. It is enough to understand that there is a standard list of limitations that can be imposed on PROV-DM models permitting applications to infer upon provenance and check its validity. This is crucial as valid models significantly improve the usefulness of provenance and the reliability of applications that process it.

2.1.3.3 PROV-N

Up to this point there has been discussion of how provenance information can be represented using the PROV-DM model with a graphical visualisation of its structure. However, it is also important to have both human and machine readable formats. There are currently a number of formats supported for serialising PROV including XML, RDF, JSON, Dot and Datalog with each of these encodings being useful for either inference rules, semantics or simply, developer preference for interoperability. A significant problem with these languages however, is their focus on machine readability. As a result, the PROV Working Group introduced a new language known as PROV-N for additional human accessibility.

PROV-N Syntax

To summarise the aforementioned, the strengths of the PROV-N syntax are threefold:

- Technological Independence. The syntax is simple and can be mapped to a number of other languages.
- Human Readability. The syntax of PROV-N is simple and functional, allowing use in illustrative examples.
- Machine Readability. The syntax of PROV-N is defined through formal grammar, allowing to be parsed by machines.

As PROV-N is a direct serialisation of PROV-DM, it is not beneficial to repeat all the available constructs or grammar. Instead, it is sufficient to provide some short examples of how a PROV-DM model would be represented in PROV-N.

PROV-N Examples

All PROV data model types must have an associated identifier. To create an entity with the identifier ‘e1’ in PROV-N:

```
entity(e1)
```

Figure 12: Defining an entity in PROV-N

Many constructs in PROV-N allow additional elements, for example, the creation of an activity, ‘a1’, which occurred between 2011-11-16T16:00:00 and 2011-11-16T16:00:01:

```
activity(a1, 2011-11-16T16:00:00, 2011-11-16T16:00:01)
```

Figure 13: Defining an activity with additional elements

Relations expressed in PROV-N provide a subject and an object and similarly to types, may provide additional elements.

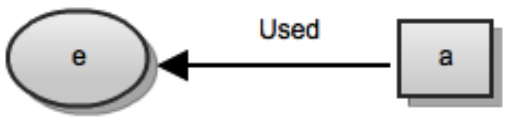


Figure 14: An example provenance graph with relation

The graph visualised above could be expressed in PROV-N in the following way:

```
activity(a)
entity(e)
used(a,e)
```

Figure 15: Figure 14 graph expressed in PROV-N

As mentioned previously, many constructs have optional elements. Indeed, an activity has already been defined that has taken advantage of these. Typically, if no optional elements are used, they are omitted entirely. However, occasionally it is desirable to utilise only some of these optional elements. In this case the '-' character is used as a placeholder. For example, if we had only the end time for an activity 'a1', it could be expressed as follows:

```
activity(a1, -, 2011-11-16T16:00:01)
```

Figure 16: Expressing optional elements in PROV-N

Type and Relation attributes are expressed using a square bracket notation at the end of a PROV-N expression:

```
agent(ex:William, [ prov:type='prov:person' ])
```

Figure 17: Expressing attributes in PROV-N

The PROV-N examples offered above only scratch the surface of available functionality. However, those provided should allow insight into the application of this technology.

2.1.4 PROV Tooling

Up to this point the focus has been centred on provenance and the PROV specification at a theoretical level. It is now appropriate to give significant consideration to the practical applications of PROV and the technology that exists to support it. This section will attempt to cover the severely limited tooling available for PROV with some consideration given to areas in which it is lacking.

There are two main repositories of code for assisting with PROV development, a ProvToolbox [16], containing interoperability tools and a Python implementation of PROV-DM with support for JSON representations [17]. Other than these, the community and tooling is very sparse.

ProvToolbox

This repository is managed by Professor Luc Moreau from the University of Southampton. It is the most comprehensive set of technologies available for supporting development in the PROV model. It acts mainly as an interoperability framework, providing ways to convert between the different encodings of PROV e.g. JSON, Datalog, PROV-N. All code in the repository is written in Java and additionally, it contains an XML schema for representing PROV in XML. Dependencies within the

repository are managed by maven and the aforementioned schema is utilised to generate Java class files, allowing for an object representation.

Limitations

There are a number of key limitations regarding this repository. Firstly, the documentation provided is very poor; in fact, the 'readme' presentation discusses the old Open Provenance Model, rather than PROV-DM which is very out of date.

Secondly and very importantly, the Java object model generated from the XML schemas has some significant disadvantages. The principal of these is that there is no easy way to traverse PROV graphs which is a major obstacle to its usefulness. Furthermore, as the model is generated from XML, it also ties developers into the JAXB libraries. Finally, as it follows the XML schema, the object model is not particularly intuitive, requiring references to types to be attached to relations rather than the types themselves.

Finally, there is some delay between updating the various encoding representations and the PROV-DM model so they can be out of sync.

Strengths

The repository provides very good interoperability between formats and provides a method for both parsing and generating PROV-N. Additionally, there are good test cases for each of the maven modules, demonstrating how to use them correctly.

TrungDong Repository

This repository is managed by Dr. Trung Dong Huynh, also from the University of Southampton. This repository contains a python representation of PROV-DM and a procedure to convert PROV into a JSON representation.

Limitations

The repository provides no support encodings other than JSON; specifically, not having support for PROV-N is an issue. Furthermore, on an supplementary note, the author is not familiar with the Python programming language.

Strengths

This repository has a very strong Python Object model to represent PROV-DM. It offers methods for traversing provenance graphs and for querying those graphs. Finally, there is some support for inference, employing simple rules from PROV-CONSTRAINTS.

Provenance Visualisation [18]

There is one other repository worthy of note, managed by Dr. Alan Eckhardt. This repository provides some javascript libraries to aid in visualising provenance in a graph like manner; it provides labelled nodes and edges on a canvas. This type of view is a very intuitive method of visualising graph structured provenance and indeed, similar methods are used in many non-specific graph visualisation libraries [19] [20]. Unfortunately, it currently only supports the old Open Provenance Model but the intention is to provide support for PROV-O soon.

2.2 Data Mining

For centuries, analysis has been conducted manually on data sets in an attempt to identify patterns. With the proliferation and ubiquity of computing technology, the rate at which data can be collected, stored and manipulated has increased significantly. At a high level, data mining involves searching data for correlations and discovering new useful meanings. These patterns can facilitate many applications, especially in the fields of science, engineering and commercial activities.

2.2.1 Mining Provenance Graphs

This section will not probe too deeply into how provenance graphs are mined as this is beyond the scope of the project. It will however, attempt to make a case for why mining provenance is beneficial and why it will be important in the future. Moreover, it will identify why the project's purpose is motivated by the need for mining provenance.

In the 2011 Data Mining Survey [21], one of the chief challenges faced in the field was the availability of useful data to mine. Although provenance is currently a very small consideration within the province of data mining, as more graphs become available, it is clear that they will be harvested by the data mining community.

Only a limited amount of work has been conducted on mining provenance graphs so there is a shortage of background research to consider. One key paper in this area, 'Using Provenance for Personalized Quality Ranking of Scientific Datasets' [2], provides a method of ranking search results based upon quality. In the conclusion to this paper, there is a clear cut case, motivating the need for a provenance generator. It states, garnered

'The accuracy of the provenance-based quality score can improve significantly as more provenance samples are available for training classifiers'

The provenance traces that are currently available for researchers are insufficient for mining. Indeed, a number of other papers [7] [22] necessitated or will require the creation of provenance graphs from a separate source for interrogation. Crucially, it seems that there is almost no research available regarding mining provenance graphs that adhere to the PROV model.

PROV-AQ

The W3C PROV Working Group has produced a specification PROV-AQ which provides some insight into accessing and querying provenance structures following the PROV-DM model. It recommends using SPARQL, a query language over RDF stores to traverse provenance graphs, an orthogonal issue to this project. The PROV-AQ specification is in an early draft and does not present much useful information relevant to generating graphs or the mining of them. However, as the specification develops, this is likely to be where relevant information will emerge, specifically with regard to techniques for mining provenance data.

2.3 Scalable Provenance Storage

In keeping with the previous section, there will not be too much discussion into provenance storage, merely evaluating why it is an important issue and how it motivates this project. As mentioned briefly in the motivation section, a significant portion of data mining involves constructing efficient storage solutions. In the conclusion of the paper 'PASSing the Provenance challenge' [22], the author states that one of the crucial problems they faced was failing to plan their schema and data representation with the query model in mind. One might imagine that if a large corpus of example traces were present to utilise, this would have been less of an issue.

There have been a number of provenance solutions proposed in literature in the past few years [23] [24]. Whilst being good solutions for simplicity and ease of use, they suffer from a lack of scalability. A recent paper, published in July 2012 focused on providing a scalable solution, stating that for each 100kb of data, there was 1Mb of provenance [25] [26] . Clearly, these numbers have the ability to increase rapidly as the size of the base data increases. Unfortunately, the framework developed in this paper was for the Open Provenance Model, an old version of PROV. In the coming years, as developers move towards the new specification, much like data mining, they will require an easy way to generate suitable solutions with which to test storage systems.

2.4 Graphs and Generating Graphs

The final section of background research will briefly touch on the nature of synthetic graphs and how generate them. In their rigorous graph survey [27] D. Chakrabarti and C. Faloutsos discuss the distinguishing characteristics of graphs and how synthetically generated graphs can be created to appear like real-world, naturally occurring instances. They provide some research that can be applied to provenance graphs when considering specific types of graph generator, such as those of random graph models and preferential attachment models (connected nodes become more connected). However, in general, the generators reviewed do not fit well with the provenance model.

2.5 Research Conclusion

This chapter raised a number of research issues worthy of note:

- There are many definitions and structures for provenance. These are in the process of incorporation into the PROV family of specifications. It is likely that the majority of future provenance on the web will utilise PROV-DM.
- There is currently very limited technological backing for PROV. What exists is good but lacks documentation and support.
- PROV-N is a good serialised representation of PROV-DM, providing human readability in addition to the advantages offered by other encodings.
- Data Mining is a growing field that has begun to consider provenance. However, there is a dearth of existing provenance traces for mining.

3 Requirements

3.1 Requirements Specification

The requirements identified in this section specify at a high level, the system and the functions provided to the user.

3.1.1 Functional Requirements

R1 – The application must provide a way to create and maintain provenance structures using the PROV-DM model.

R2 – The application must provide a way to add client configurable parameters to the provenance model.

R3 – The application must be able to expand seed graphs following R2, into final provenance traces.

R4 – The application must be able to create PROV-N from the output of R3.

R5 – The application must provide the user with an easy to use GUI for the creation of seed graphs.

3.1.2 Non Functional Requirements

R7 – The application should load immediately when opened.

R8 – The expansion of seed graphs into final graphs must not take an unreasonable amount of time

R9 – The GUI should be easy to navigate and quick to user friendly

R10 – Any messages or errors presented to the user must be non-technical in nature.

3.1.3 Hardware and Software Requirements

R11 – The application will be supported on Windows, Linux and Mac operating systems.

R12 – The application will be created using the Java programming language.

4 Design

This chapter will detail the conception of the system, identifying the key requirements for development. Ideas generated in this chapter will be used as a first draft for implementation of the final application. The design attempt to avoid tying down the application to any particular implementation, specifying only the high level concepts required to achieve functionality.

4.1 Graph Expansion

Consider requirements R2, *'The application must provide a way to add client configurable parameters to the provenance model'* and R3, *'The application must be able to expand seed graphs following R2, into final provenance traces'*.

When considering how to permit the generation of provenance graphs that are useful to the user, a number of issues need to be addressed. A first option, replicated in the data mining paper [27], could be the automatic creation of the graphs. However, without detailed research into the distinguishing characteristics of provenance graphs, this method would be unlikely to achieve any useful results. It is a much better idea to give the user more control over the final graph. This is the function of the aforementioned seed traces. These are small graphs that have a number of client configurable parameters which can be used to expand them into the final result.

4.1.1 Simple Parameters

Having decided upon a parameter based scheme, it is then important to discover what types of parameters are desirable in expanding graphs. The most intuitive method of increasing graph size is simply to create more nodes. To achieve this, the application should provide cardinality on nodes. Consider a simple provenance graph, where an entity is used by an activity; it is then trivial to imagine providing simple integer cardinalities on each node.

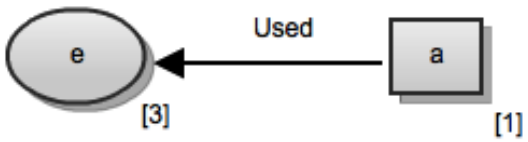


Figure 18: Cardinality on nodes

If this graph were to be expanded, the result is intuitive; each node gets multiplied by its corresponding cardinality, as seen in Figure 19. There are two important issues to note here. Firstly, the PROV-CONSTRAINTS specification states that types must be unique; the identifier and attributes of two records must not both be equal. Therefore, it is important that we increment the identifier of each entity to enforce this constraint. Secondly, in Figure 19, the relations have also been multiplied with the current semantics and this is correct but will need further analysis later.

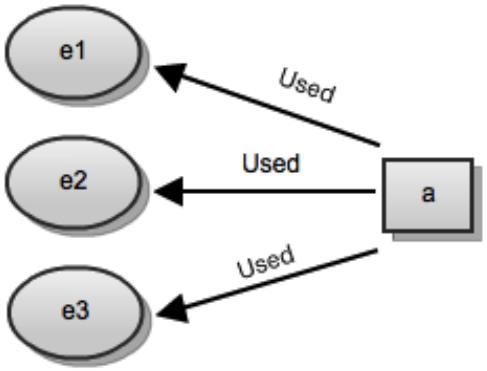


Figure 19: Expanded graph from node cardinalities

Having considered simple cardinalities, the application should also be able to supply an element of randomness in the graph, allowing for creation of unexpected graphs. This is particularly useful as developers requiring provenance graphs for testing may find that knowing how a graph will behave hinders the usefulness their tests. Consequently, the application will permit cardinality ranges to be associated with nodes:

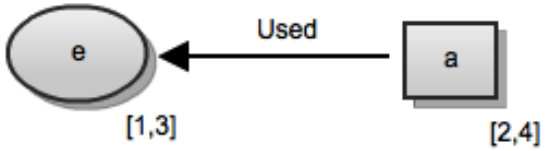


Figure 20: Cardinality ranges for randomness

When the graph in figure 20 is expanded, it could generate any combination of one and three entities and two and four activities. For example, in figure 21, we see a possible result of this expansion, with one entity and two activities. Again, the relations have also been multiplied.

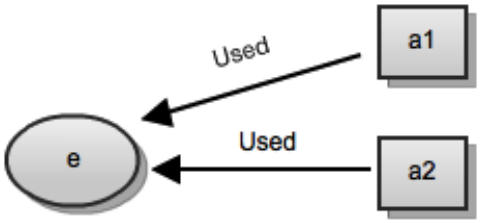


Figure 21: A generated graph from node cardinalities with randomness

Having reviewed how the number of nodes in a graph may be increased, the application should also provide a way to manage how nodes are connected via edges. Earlier, there was comment on how edges should be connected when nodes are multiplied – that every node on one side should be connected to every node on the other. The result would be that any expanded graph would have many edges throughout, a pattern that is not common in a provenance trace. As such, the application should introduce the concept of edge saturation (not to be confused with those in flow networks), which allows the user to determine the number of edges to be connected.

Figure 22 demonstrates saturation on a 'Usage' relation. The value corresponds to the fraction of edges to keep, where 1 is all edges and 0 is none, thus 0.5 is half. To clarify, in a graph with M cardinality on one side, and N cardinality on the other, a saturation of 1 will yield M*N edges; in the case of figure 23, a saturation of 1 would have 6 edges.

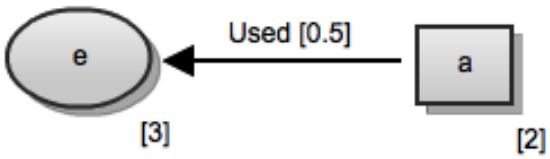


Figure 22: Saturation of edges in a seed graph

Semantically, the only implication of this is that we should get half saturation on the edges; how edges are selected or removed from the graph is an implementation concern. One potential execution can be found in figure 23. Additionally, the exact formula for the number of edges is:

*ceil((M * N) * Saturation)*

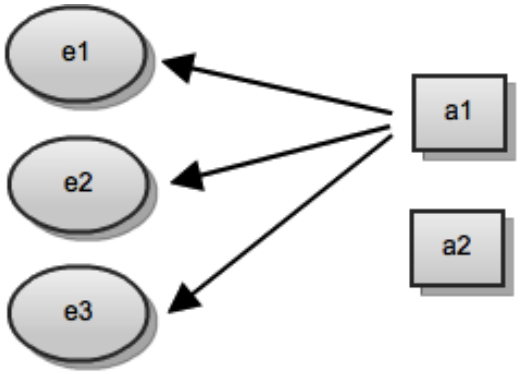


Figure 23: A possible implementation of edge saturation semantics

Moreover, it will be possible in the application to apply randomness to edge saturation in the same method as it is applied to nodes e.g. [0.3, 0.7] will saturate randomly between 0.3 and 0.7 of the total edges.

4.1.2 Patterns and Compositions

The simple cases provided in the previous section suggest a good framework for creating seed graphs. It is not easy however, to specify very large graphs with interesting characteristics. An attractive method to obtain this functionality is to regard a seed graph as a 'pattern' to which parameters can also be applied. Consider the simple seed graph in figure 24, specifying one entity and one activity, connected by a 'Usage' relation. If this is thought of as a pattern, then it can be used without redundancy throughout the seed graph in multiple places. Furthermore, it permits some interesting semantics reviewed in the following sections.

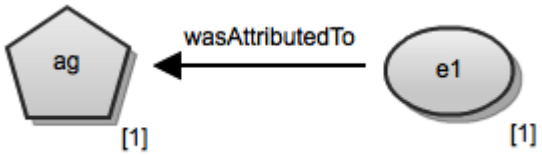


Figure 24: A simple seed graph

4.1.2.1 Connections

Figure 25 demonstrates a simple new seed graph in which one entity is attributed to one agent. If both this and figure 24 are considered to be 'patterns' that can be utilised throughout the seed graph creation process, one interesting application might be creating a linkage via a new relation.

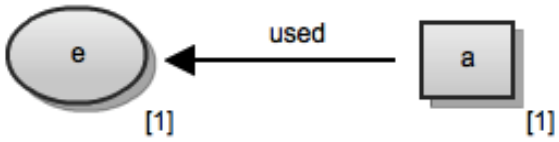


Figure 25: A simple seed graph pattern

If figure 24 is termed P1, and figure 25 is termed P2, a connection via a 'Generation' relation may be formed as seen in figure 26. It follows the same method of edge creation as previously noted, with the addition of specifying the id of the starting node and end node.

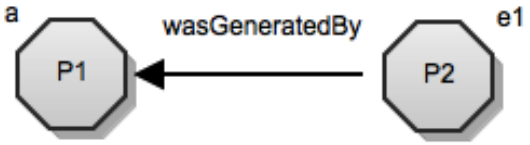


Figure 26: Connection of patterns

If this seed graph were expanded, the resulting graph would appear as in figure 27.

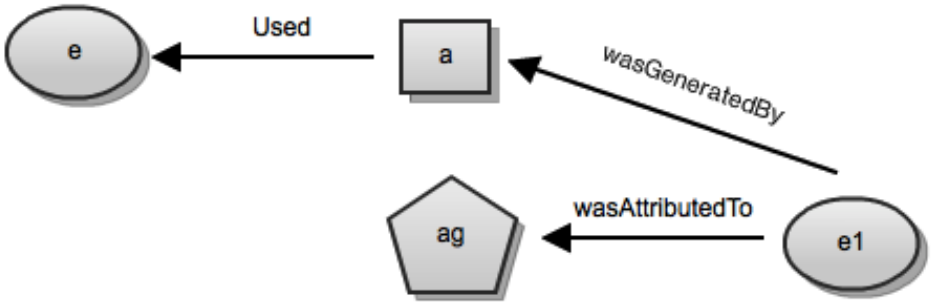


Figure 27: An expanded graph utilising connections

4.1.2.2 Merges

Figure 28 demonstrates a simple new seed graph in which one entity was generated by one activity. However, another approach is suggested and is similar to this use of connecting patterns and is designated “a merge”. If instead of defining a new relation between patterns, one specified that a node from each intrinsically represents the same thing, a particular operation may be performed.

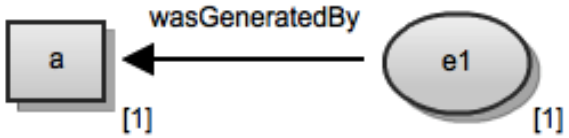


Figure 28: A simple seed graph pattern

Figure 29 exhibits a merge operation, which resembles that of an SQL JOIN in which figure 24 is P1 and figure 28 is P2. It is similar to connections in the previous section, with the exception of the same node being specified on either side of the equality relation. Importantly, an additional constraint is placed on node cardinality modifiers, that they must also be of equal value.

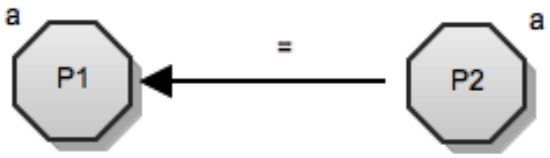


Figure 29: Merging of patterns

If this seed graph were expanded, the resulting graph would appear as in figure 30.

Figure 30: An expanded seed graph utilising merge

4.1.2.3 Repetition

A final application of patterns is a specialised case of the merge operation. In some cases, such as when creating very long, lean graphs, it would be valuable to have the potential to utilise repeating patterns. In such a case, cardinality can be applied to a pattern in a similar fashion to nodes. Additionally however, there must be two nodes at the edges of the pattern which can then be considered to be the same object as shown in figure 31.

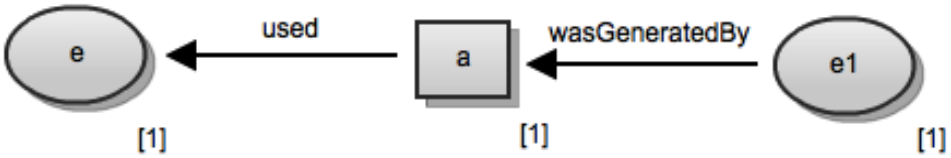


Figure 31: Repetition of a simple seed graph

If a cardinality of 3 was applied to this pattern and a specified merge occurred on e and e1, the result would be a long lean graph of ‘Usage’ and ‘Generation’ relations.

4.1.3 Syntax

As discussed earlier, PROV-N is a good language for expressing PROV structures but careful consideration must be given to representing these extra parameters for seed graphs. It turns out that it is possible to utilise PROV-N's extensibility through attributes which permit application specificity. Any attributes for an application can be qualified with any namespace prefix. This allows seed graphs to maintain their status as valid PROV graphs while including special attributes that the application can interpret.

To aid in this, a simple schema has been defined providing elements fundamental to the application functionality:

```
<?xml version="1.0" encoding="UTF-8"?>


<xs:schema targetNamespace="http://www.example.com/gen/"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:gen="http://www.example.com/gen/"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">


<xs:element name="minCardinality" type="xs:integer"/>
<xs:element name="maxCardinality" type="xs:integer"/>
<xs:element name="minSaturation" type="xs:decimal"/>
<xs:element name="maxSaturation" type="xs:decimal"/>


</xs:schema>
```

Figure 32: Schema for seed graph extensions to PROV-N

This schema can then be made use of by records in PROV-N. For example, figure 32 below can be represented by the following PROV-N:

```
activity(ex:a, -, -, [ gen:minCardinality='2', gen:maxCardinality='4' ])
entity(ex:e, [ gen:minCardinality='1', gen:maxCardinality='3' ])
used(ex:a, ex:e, [ gen:minSaturation='0.3', gen:maxSaturation='0.7' ])
```

Figure 33: Extended PROV-N representation of a seed graph

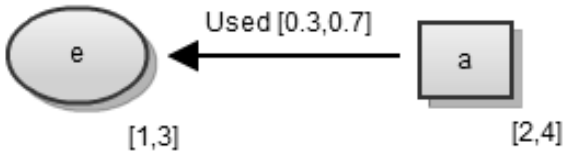


Figure 34: A Seed Graph that can be represented in PROV-N

4.1.3.1 Pattern Composition and Repetition Syntax

As shown in the previous section, syntax for simple node cardinalities and edge saturation is relatively simple as PROV-N provides obvious extensibility points. When considering how to provide patterns and associated functionality, the syntax becomes more convoluted. There are two pivotal steps in planning for this functionality. Firstly, there must be a way to determine which PROV constructs belong to a pattern and one potential solution to achieve this is adding another element to the schema:

```
<xs:element name="pattern" type="xs:string"/>
```

Using this extra attribute, applications can determine whether or not records and relations belong to patterns. This can then be utilised by applications to determine semantics for the extra pattern functionality. For example, to express the graph in figure 34 as a single pattern:

```
activity(ex:a, -, -, [ gen:minCardinality='2', gen:maxCardinality='4',
gen:pattern='p1' ])
entity(ex:e, [ gen:minCardinality='1', gen:maxCardinality='3',
gen:pattern='p1' ])
used(ex:a, ex:e, [ gen:minSaturation='0.3', gen:maxSaturation='0.7',
gen:pattern='p1' ])
```

Figure 35: A pattern expressed in extended PROV-N

The second problem regarding composition syntax is more complex. With the pattern element defined above, it can be determined which constructs operations should be performed upon. However, there is no clear process within PROV-N for specifying these operations; in fact, an undesirable potential solution could be very unintuitive, relying on the addition of many new schema elements that are not immediately obvious and something which should be avoided at all costs within languages. It seems that the best solution is in fact, to propose an extension to PROV-N outside the boundaries of the defined extensibility points. This has a disadvantage in that it violates somewhat the Principle of Single Responsibility, in that it obfuscates the language by including meta-level syntax within PROV-N. However, it also allows developers to take advantage of existing tooling and support and so appears to be a necessary evil.

Abstract Syntax for Patterns

Although it is entirely possible to keep the aforementioned schema element for patterns, at this point it may be more judicious to suggest a further method of defining patterns for consistency with latter constructs. To make sense, a number of extra EBNF (Extended Backus-Naur Form) declarations are required, however, for clarity the full pattern expression will be specified first, followed by any additional EBNF necessary. To define a pattern in this extended syntax an id is obligatory, followed by a list of all PROV-DM constructs included in the pattern:

```
pattern(p1, [ a, e, u ])
```

Figure 36: New Pattern Syntax in PROV-N

This expression states that there is a pattern “p1” that contains an activity, entity and usage relation with the ids “a”, “e” and “u”. Following the existing PROV-N definitions, this construct can be specified in EBNF as:

```
patternExpression ::= “pattern” “(“ pIdentifier identifiers “)”
```

Figure 37: Abstract Syntax for Pattern Identifiers

For the above rule to make sense within the existing PROV-N framework, there are a number of extra definitions required. Firstly, we require the addition of patternExpression to the existing expression rule:

```
Expression ::=
( entityExpression | activityExpression | generationExpression | usageExpression | star
tExpression | endExpression | invalidationExpression | communicationExpression | agentE
xpression | associationExpression | attributionExpression |delegationExpression | deriv
ationExpression | influenceExpression | alternateExpression | specializationExpression
| mentionExpression | membershipExpression | extensibilityExpression |
patternExpression )
```

Figure 38: Adding Pattern Expression to existing EBNF

Secondly, we require the specification of pIdentifier, which is simply an identifier:

```
pIdentifier ::= identifier
```

Figure 39: Pattern Identifier in EBNF

Finally, we must specify a rule for a list of identifiers:

```
identifiers ::= ( | identifier ( “,” identifier)* )
```

Figure 40: EBNF for a list of identifiers

From the specified EBNF rules above, it would then be possible to create a pattern construct in PROV-N that contains any number of records and dependencies.

Abstract Syntax for Connections

The syntax required for making connections requires the creation of a new expression. It should take two patterns and the identifier of the relation that will connect them:

```
used(u, a, e)
connection(p1, p2, u)
```

Figure 41: Extended PROV-N for connections

The EBNF to represent this can be found below. Additionally, as with the patternExpression, connectionExpression would also need to be added to the expression rule.

```
connectionExpression ::= “connection” “(“ pIdentifier pIdentifier
identifier “)”
```

Figure 42: EBNF for connections

Abstract Syntax for Merges

The syntax required for making merges also demands the creation of a new expression. It should take two patterns and the identifiers of the records within for the merging process. An extended example is shown below, demonstrating the creation of two simple patterns and how they would be merged.

```
entity(e1)
activity(a1)
used(u1, a1, e1)
pattern(p1, [ e1, a1, u1 ]) # Create our first pattern

activity(a2)
entity(e2)
wasGeneratedBy(g2, e2, a2)
pattern(p2, [ a2, e2, g2 ]) # Create our second pattern

connection(p1, a1, p2, a2)
```

Figure 43: Extended PROV-N for merging

The PROV-N found in figure 44 would result in an equivalent graph with the exception of not having useful patterns defined for future use.

```
entity(e1)
activity(a1)
entity(e2)
used(u1, a1, e1)
wasGeneratedBy(g2, e2, a1)
```

Figure 44: PROV-N equivalent to figure 43

The EBNF required for this functionality can be found below. Additionally, like the previous rules mergeExpression would also need to be added to the expression rule.

```
mergeExpression ::= "merge" "(" pIdentifier identifier pIdentifier
identifier ")"
```

Figure 45: EBNF for merges

Abstract Syntax for Repetitions

As a specialisation of the merge, the repeater operation requires only slight modification to syntax. It should accept one pattern and the identifiers of the records to merge upon and additionally, should accept attributes to allow for cardinality. Following the previous example,

```
entity(e1)
activity(a1)
entity(e2)
used(u1, a1, e1)
wasGeneratedBy(g2, e2, a1)
pattern(p1, [ e1, a1, e2, u1, g2 ])

repeat(p1, [ minCardinality='2', maxCardinality='4' ])
```

Figure 46: Extended PROV-N for the repeater operation

The EBNF required for this functionality can be found below. Additionally, like the previous rules repeatExpression would also need to be added to the expression rule.

```
repeatExpression ::= "repeat" "(" pIdentifier optionalAttributeValuePairs ")"
```

Figure 47: EBNF for repeater operation

4.2 GUI

Consider requirement R5, that *'the application must provide the user with an easy to use GUI for the creation of seed graphs.'* For successful adoption, it is vital that applications provide simple and convenient operations. Modelled on existing technologies, including Alan Eckhardt's provenance visualisation libraries, the application will make use of a graphical map view. Users will be presented with a blank canvas on which they may add nodes and relations that can be dragged and dropped to aid in organisation. A mock-up of this design may be seen below in figure 48:

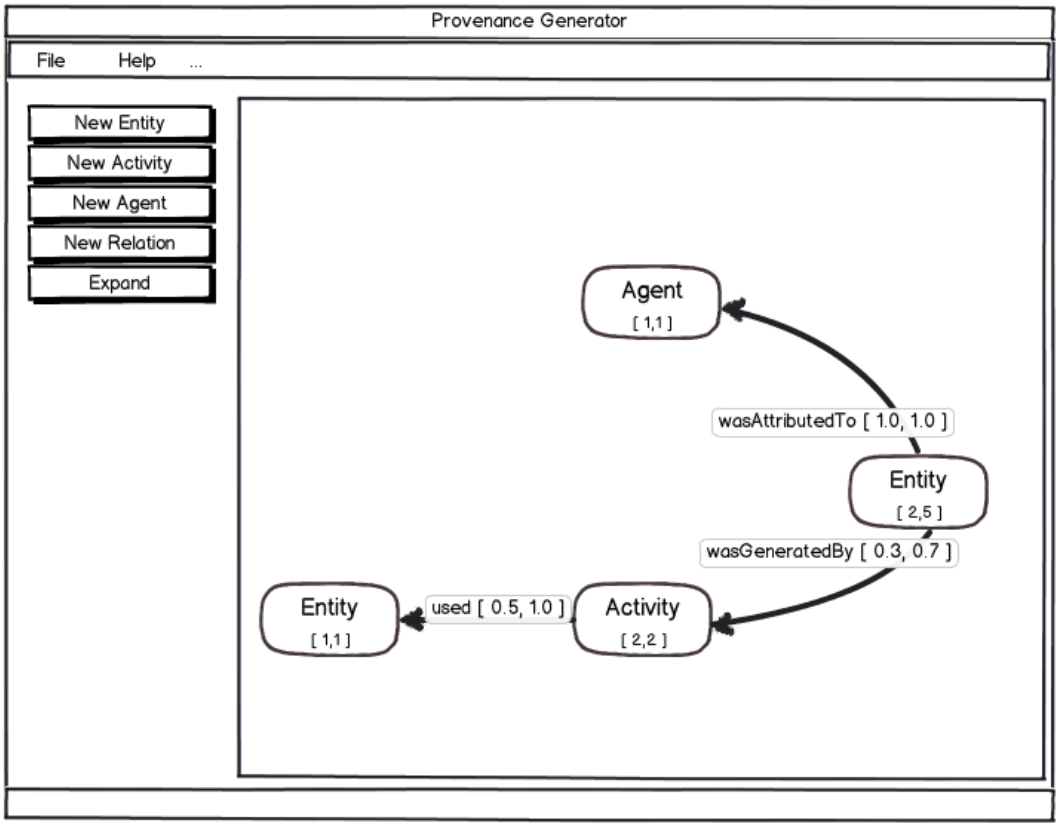


Figure 48: GUI Mock-up

Providing a design like this will allow users to very efficiently and intuitively create seed graphs, however, care must be taken with spacing to avoid incomprehensible connections.

4.3 Design Summary

In this chapter, a robust foundation has been presented, defining some key concepts that would facilitate keeping the application well-structured, ordered and maintainable. Furthermore, functionality has been enhanced, suggesting many ways in which different graph expansion operations could assist the user. From this design, an implementation model will be created capturing all the requisite functions.

5 Implementation

This section will explain the execution of the design, discussing in more depth how the application actually achieves functionality, rather than describing its theoretical objectives as discussed in the previous chapter. The implementation of the system was split into four major blocks which were identified as being fundamental to its intended purpose. Each has been rigorously selected and incorporates interdependent functionality across the parts. Furthermore, the modular design facilitates important abstractions and decisions which will be beneficial for future modifications. These identified blocks are: modelling PROV-DM in Java, providing parameters and graph expansion operations, generating PROV-N from the final graph and implementing a graphical user interface.

Language Choice

A brief note should be made regarding Requirement R13, referencing that the application should be programmed using the Java language. The reasons for this are threefold; firstly, the majority of tooling support available already exists in Java which permits the use of the interoperability tools. Furthermore, it is important to try and consolidate and extend the existing technologies to improve developer uptake. Secondly, on a less technical level, the author's experience is with Java, rather than Python and a more robust solution is anticipated by taking this into consideration. Finally, using Java satisfies Requirement 12 as applications are "write once, run anywhere".

5.1 JPROV

Consider requirement R1, *'The application must provide a way to create and maintain provenance structured using the PROV-DM model.'*

From the background research conducted, it is clear that the tooling needed to support PROV-DM is severely limited. Clearly, it is possible to create a strongly typed object model using the Java classes generated from XML schema in the ProvToolbox repository. However, as discussed earlier, this has a number of disadvantages, including tying developers into XML libraries and methodology but more crucially, there is a lack of support for traversing provenance graphs created in this model.

Due to these limitations, it is clear that a fresh solution was required. Consequently, a new Java library was created that is a direct mapping from PROV-DM to strongly typed Java objects. The class structure is intuitive, providing obvious methods to achieve the creation of a suitable provenance model. The library is independent of the rest of the project, maintaining a clean, modular design. This provides a significant advantage over the existing solution, permitting application within many other projects without dependencies on other libraries. Finally, it adheres to programming best practices in order to be easily extensible by the rest of the project, even more so in order to receive community uptake.

The JPROV library was split into three parts, a `'uk.ac.ncl.provenance.jprov'` package and sub-packages `'uk.ac.ncl.provenance.jprov.records'` and `'uk.ac.ncl.provenance.jprov.relations'`. The top level package contains a number of application wide interfaces, providing common PROV-DM attributes. Additionally, it includes a factory for all PROV constructs as JPROV is not designed to be extended via subclassing. The records are self-explanatory, providing the functionality for PROV elements and dependencies. A strong motivation for developing this library was to provide convenient methods of traversing PROV traces.

As such, each record implements the `'IsTraversable'` interface, which requires that all relations provide methods to access start and end nodes.

Class diagrams for the JPROV library can be found in Appendix A.

5.2 Graph Expansion

The previous section discussed how to model PROV-DM in Java but did not cite the extensions required to specify seed graphs. To achieve this, another library was created that composed upon records and relations, supplying extra attributes for weighting and saturation. Additionally, methods were then offered to expand seed graphs into their final state. Conceptually, the expansion process has three stages discussed below:

5.2.1 Node Multiplication

The first step in expanding the graph relies on multiplying nodes with the cardinality provided as a parameter. To achieve this, it is necessary to traverse all nodes in the graph in some way. It is important that this traversal is repeatable and obvious, allowing for pen and paper modelling of the process. As such, it is no good to simple iterate over a list of nodes as the ordering is normally determined by an arbitrary parameter such as when the node was added or by some natural comparison. Instead, the application executes a breadth-first traversal of the graph beginning from each root node. As PROV-DM graphs are unidirectional, root nodes may be discovered by finding nodes with no incoming edges[1].

Breadth-first traversal provides a predictable ordering of visited nodes. As an algorithm, it has two main operations: visit and inspect; and gain access to visit the neighbours of the current node. It begins at a root node and visits each neighbour in turn then, for each of its neighbours' neighbours, do the same. For example, in figure 49, the ordering of visited nodes would be: 'e1 -> a -> ag -> e' or 'e1 -> ag -> a -> e' as there is no constraint placed upon which neighbour to visit first. The algorithm can be viewed as interrogating a tier at a time.

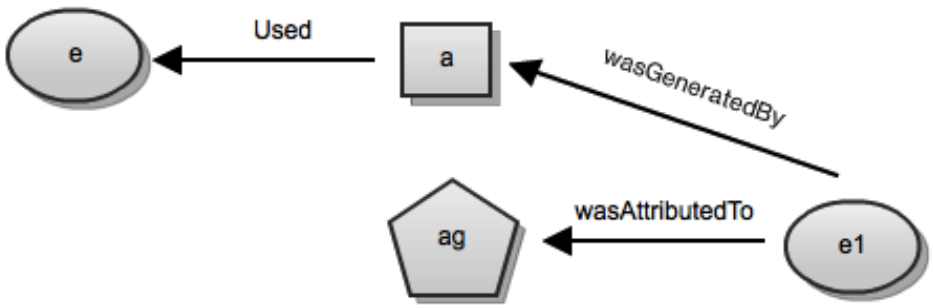


Figure 49: A PROV graph to be traversed

[1] It is important to note that while PROV-DM graphs are often acyclical, in some cases such as when using a 'wasStartedBy' relation (see figure 10), a root node may be absent. In such a case, a search must be performed to find a relationship that may cause a cycle and if no other incoming edges are found on the associated node, it may be assumed it is a root node. In graphs with cycles that do have a root node, we can simple ignore a node if previously visited.

At each node visited during the traversal, the node will be copied as many times as the cardinality specifies. These are then temporarily held within a 'Super Node' prior to edge saturation being performed.

It is worth noting that the complexity of a breadth-first traversal is O(|N| + |E|) where N is the number of nodes and E is the number of edges. If every node was connected to every other node, the worst case complexity of the algorithm would then be $O(|N|^2)$ however, whilst worth keeping in mind, the structure of provenance graphs makes this very unlikely for any useful application. Consequently, the complexity is likely to be closer to O(|N|), linear time, which is the best possible in this type of application where all nodes must be visited.

5.2.2 Edge Saturation

After expanding all nodes in the graph, it is then important to saturate the edges properly. As noted in the previous chapter, the semantics for edge saturation only state that a certain number of edges must be created or removed and not how the edges are chosen. In the application, a very simple edge saturation algorithm has been created which is defined in figure 50 below. Consider two super nodes containing M and N nodes respectively (where M < N) and a saturation level of S:

```
TOTAL_EDGES = ceiling((M * N) * S)

for each M AND while NUM_CONNECTED_EDGES < TOTAL_EDGES
    for each N
         connect M to N
         NUM_CONNECTED_EDGES++;
         if NUM_CONNECTED_EDGES >= TOTAL_EDGES
             BREAK;
         fi
    endfor
endfor
```

Figure 50: Pseudocode for edge saturation

This implementation results in each M being connected to all Ns until edge saturation is reached such as in figure 51:

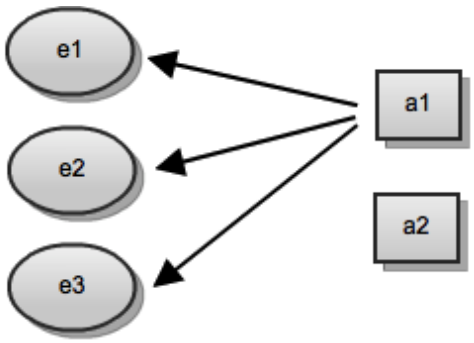


Figure 51: Edge saturation example

5.2.3 Node Elimination

The final step in the graph expansion process is the elimination of nodes that are unconnected. After edge saturation, a final traversal of the graph takes place, searching for any nodes with no incoming or outgoing edges and these are simply removed from the final graph. This occurs to prevent many islands causing an exceedingly disconnected provenance trace – a characteristic which is very unlikely to occur.

5.2.4 Expansion Issues

During expansion there are two important issues to recognise. Firstly, there has been no discussion of when random cardinalities and saturations are made concrete and when this takes place is a critical step. Consider a seed graph that has been generated into a final provenance trace. If a user decides that this graph is useful but they would like to add further nodes within the seed graph whilst maintaining the overall structure of the final graph, it is detrimental to generate a new set of random values each time as the desired final structure will likely be lost. As such, prior to any expansion algorithms, a traversal of the graph takes place consolidating these random values. Moreover, this allows the addition of a 'replay' operation, where a graph can be regenerated without revaluating randomness.

The second issue to note is that of identity within the graph. As discussed earlier, the PROV-CONSTRAINTS specification requires that no two PROV constructs can have the same identifier and attributes. As such, at the expansion phase, the identifier of constructs is incremented to confer uniqueness.

5.3 PROV-N Generation

Consider requirement R4, *'The application must be able to create PROV-N from the output of R3.'* It was noted earlier that PROV-N is a good choice for encoding provenance structured using the PROV-DM as it is both human and machine readable. Consequently, it is necessary to somehow serialise the final JPROV graph into PROV-N. Initially, it appeared as if this would need to be performed manually – a difficult task considering the nuances of language generation.

Fortunately, within the ProvToolbox, there are some helpful methods for serialising PROV. As mentioned in the background research chapter, there is a JAXB representation of PROV-DM. By utilising this, it is possible to convert to XML and then in turn to PROV-N via the Interoperability Framework.

The procedure for conversion is as follows:

- Perform a breadth-first traversal of the JPROV graph
    - At each node or relation, create an equivalent in the XML model
- Serialise the new model to XML
- Perform a conversion from XML to PROV-N using the InteropFramework available in ProvToolbox

5.4 GUI

The final step of implementation is that of providing a graphical user interface as an accessory to the existing work. In this section, only an overview of the application's interface will be reviewed as much of the functionality is merely simple form wrappers supplementing methods outlined in previous sections. The GUI follows the design principles laid out in Martin Fowler's Presentation Model [28]. This architecture is similar to the more common Model-View-Controller design, resulting in a clean separation of concerns between the logic of the application and the user facing view. As such, it is possible for the rest of the application to be utilised programmatically if future developers wanted to offer a different interface for the graph generator.

As stipulated in the design, the application provides a convenient storyboard visualisation of the graph, allowing for the easy addition of elements. The canvas is implemented using the Graphics2D library, drawing rounded rectangles with gradient fill for nodes. Alpha values are used to create drop shadows behind the nodes. An interesting and challenging problem is determining where to place a new node on the graph when it is already occupied by many constructs. For this, a simple algorithm was appended to find the closest unoccupied space to the top left. A better alternative may be clicking the intended location of the node on the canvas.

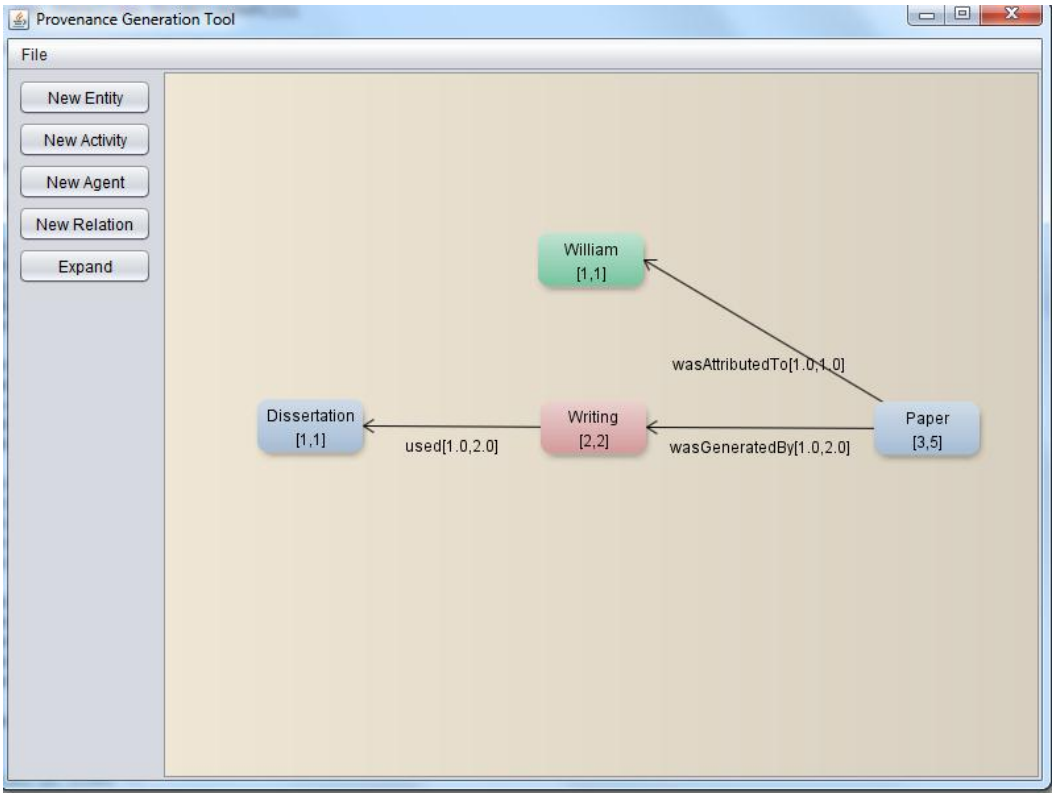


Figure 52: GUI graph canvas

On the canvas, elements are focusable and can be controlled via keyboard or mouse. Focused elements are highlighted with a black border for usability. The MouseListener and MouseMotionListener interfaces are implemented to enable dragging and dropping of components and to enable dragging edges to make new relations.

As seen in figure 53, creating a new node or relation generates a popup form, requiring an identifier and cardinality or saturation values for the construct. To add provenance specific attributes, it is necessary to double click on a node or relation as placing them on the central canvas at the outset create a cluttered interface.

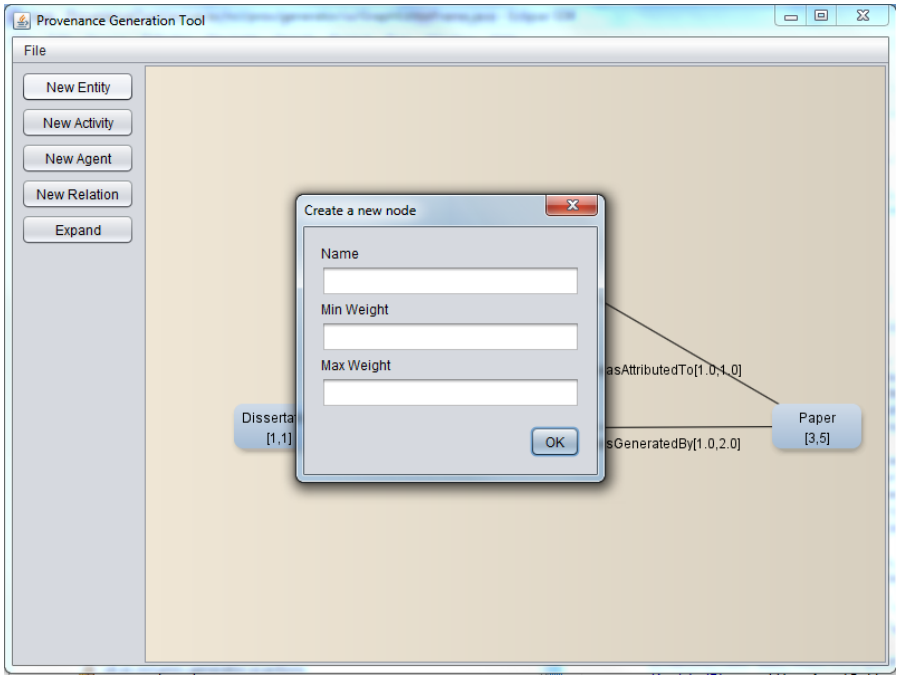


Figure 53: Adding a new node

In order to create a new edge, the user must click on the start node, click the 'New Relation' button and then drag the new edge across to the end node. An additional dialog appears similar to the node creation except requiring the edge type and saturation parameters instead of cardinality. Expanding the graph provides the ability to specify where the generated PROV-N should be stored.

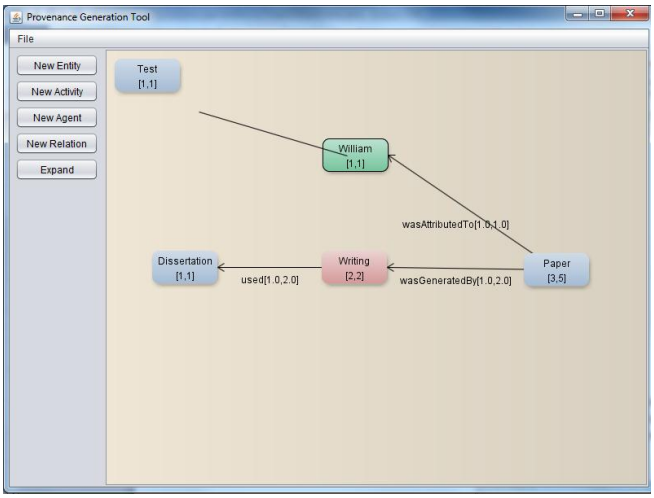


Figure 54: Dragging a new edge for creation

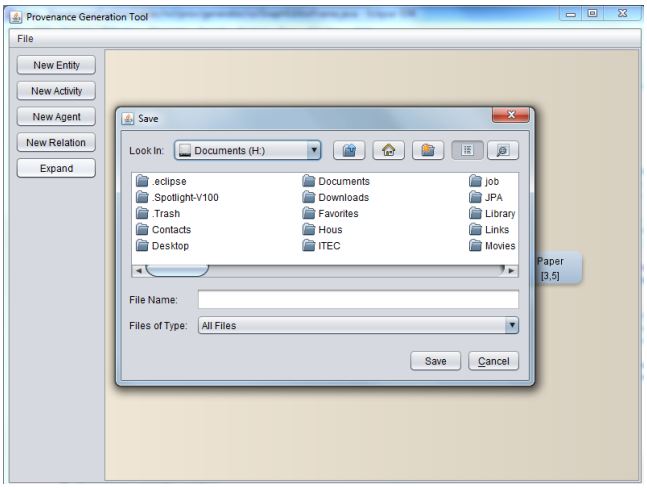


Figure 55: Saving PROV-N

5.5 Summary

This chapter has covered the practical implementation of core parts of the design. It should be noted at this point that due to time constraints, not all of the design was completed, specifically regarding pattern operations and extended syntax. However, the application produced does satisfy the central requirements and is still a useful tool, providing a number of interesting operations while exploring the limitations of the existing tooling to support PROV.

6 Evaluation

This section will analyse the project's activities and evaluate them against the original aims and objectives. Consequently, it will identify specific areas of success and also clarify the extent to which the intentions have been met.

6.1 Aims and Objectives Revisited

The priority of the project was to *'investigate and implement a method of generating easily modifiable, large provenance traces that will be encoded in the PROV-N formal language'*. Referencing the objectives set out in section 1.3, it can now be determined whether the project has achieved its purpose.

Objective 1

Explore and review the role and usage of provenance, graph theory and data mining literature.

In undertaking background research, many different interpretations of provenance were identified allowing for a broader understanding of the issues the assignment would help resolve. Furthermore, recognising the domains in which the project would have an effect aided in a greater understanding of the scope of the project and the functionality that would be desirable in the final application. As such, this objective can be considered completed.

Objective 2

Define a method of succinctly and explicitly describing seed traces that will act as the basis for generating full traces.

As the most demanding part of the project, this objective required a significant amount of deliberation. Determining useful client-configurable parameters and the semantics these provided was an intricate task as they needed to be simple enough to implement whilst remaining advanced enough to be useful. The parameters defined for node multipliers, edge saturation and pattern composition satisfy these criteria. Furthermore, the schema defined for the generator application attributes permit the sufficient extension of PROV-N to meet the final sub-objective. The work completed for this objective is satisfactory, however, it is possible that there are other more intuitive methods of expanding graphs that could be explored.

Objective 3

Develop a tool that provides methods for the creation of seed traces

This objective dealt with implementing the design architecture in Objective 2 and as such, was also a significant task. Within the sub-objectives, not all were completely satisfied. A successful model was created in Java as a direct mapping from PROV-DM and this was in turn, extended to allow the creation of seed graphs. However, these seed graphs did not implement all the functionality of the design, lacking graph patterns and composition. Finally, the application provided for conversion of the final graph to PROV-N, but did so by utilising existing work in an inelegant solution. Consequently, this objective can be considered only partially completed. It should be noted however, that the design of the application for this objective permitted easy extensibility for the remaining designs to be implemented in time.

Objective 4

Develop a GUI that will allow users to intuitively create seed traces.

A Java Swing GUI was created with an efficient and usable, visual graph editor. It permitted the creation of PROV types and relations and their properties with the additional functionality required for the generator. This objective was achieved fully, when measured against the functionality resulting from objective 3, rather than the criteria from objective 2 i.e. missing graph pattern compositions.

Objective 5

Give the generation tool back to the provenance community.

Due to time constraints, this objective was not met. It is crucial that any work delivered to the community be of a high standard, complete with documentation and well integrated with existing tooling otherwise it will struggle to be adopted. Whilst the ultimate goal of this objective has not changed, it is worth taking some time to ensure the work is assimilated appropriately and then deliver a tested and bug-free tool in the subsequent weeks.

Evaluation

When referencing the aim of the project, which is the investigation and implementation of a provenance generator and the creation of a final prototype application, it is fair to say that the project has been successful. However, there are some corollary issues which the project did not quite resolve. Nevertheless, via good architecture principles and careful design the author has endeavoured to make it easy for any future efforts to address.

6.2 Testing

An important aspect during the development of any system is testing to ensure correct execution and to identify bugs. Unfortunately, within this project, testing was limited due to time constraints. Three types of testing were briefly employed: unit, integration and system.

Utilising maven's built in execution of test cases made unit and integration testing relatively simple. A few test cases were performed with the priority of confirming that the maven environment was correctly installed. Within these tests, there were some creations of simple JPROV structures, graph expansions and PROV-N generation. A significant flaw however, is that they were only performed with cases that were expected to work correctly rather than edge or error cases. Although no automated tests were written to test error cases, it should be noted that defensive programming practices were employed to reduce potential bugs and throughout the development process, these were tested manually.

System testing was simply a matter of ensuring that the GUI provided worked as expected. In this case, some error cases were performed such as error values for node cardinality and edge saturation. This testing discovered some flaws in the system such as a poor implementation of requirement R10, that, *'any messages or errors presented to the user must be non-technical in nature'* where some messages were not helpful informing the user of issues.

In future, it would be ideal to create a large scale automated test suite and a guide to system testing. However, the tests that were written will continue to prove useful in future development, especially in relation to changes to PROV-DM affecting the JPROV model.

7 Conclusions

This chapter will draw the project to a close, examining some successes and failures and how these could be reinforced or avoided respectively. Additionally, there are recommendations for the future development of the presented framework. Moreover, there is a recognition that the project succeeded in its exploration of the generation of provenance as an aid to the development of the provenance community.

7.1 Achievements

The most important achievement in the project is that it satisfactorily solves the problem proposed by the high level aim. Combining a comprehensive design with a modular architecture allowed for the completion or partial completion of nearly all the objectives. In the cases where objectives were not entirely attained, significant effort has been made to ensure that research and design has been undertaken to allow for straightforward resolution.

The project has provided some key research opportunities into the PROV specification and the supporting tools, identifying some weaknesses and methods to resolve these problems. This research provided the foundations for a construction of a substantial design to help solve an issue that is likely to be key as provenance continues to grow. The modelling of the provenance generator was carefully considered, employing existing solutions where possible and providing extensions where necessary. Moreover, the design promoted extensibility, allowing the project to be easily progressed in the future.

The solution that has been provided by the project permits the creation of simple seed graphs which can be expanded into larger provenance traces. It has an intuitive and simple to use interface which will hopefully aid in adoption by developers using PROV in the future. Furthermore, the design of the application stressed the importance of modularity, allowing further enhancements by developers.

The research undertaken, the design created and the application implemented should all have a positive effect on the provenance and PROV communities. With the prior tooling being so uninspiring, it is hoped that the technologies developed in this project will find a home within the development of PROV. Apart from the primary motivation of the project of allowing developers to create synthetic graphs, it is anticipated, perhaps optimistically, that the JPROV library will be used by Java developers when debating how to make their applications PROV enabled.

7.2 Problems

Despite the project's successes, it has certainly not been without problems. The first challenge encountered was the lack of appropriate research in a number of fields. Originally, research was conducted into graph theory to aid in the creation of provenance graphs with interesting characteristics. However, it very quickly became clear that this domain is incredibly extensive and very technical. Additionally, despite the fact that publications on provenance have been increasing in the past few years, there is a scarcity of literature on storing and mining provenance that is relevant to the project.

In the same vein, the PROV specification is relatively new and consequently, has not found widespread acceptance and its use, as of yet reflects a dearth of community support. Anecdotally, this is evidenced by the fact the author could not ask a question related to provenance on the well-

known Q&A site “Stack Overflow” because the ‘Provenance’ tag simply does not exist. This lack of support made some aspects of the design quite challenging. In fairness, it should be acknowledged that despite the PROV specification documents being extensive, the volume and density of the information is often difficult to parse and this also creates a steep learning curve. This issue is further compounded by the specification drafts being modified midway through the project. Moreover, the lack of technological support for PROV resulted in a number of areas in which new functionality had to be invented where in a more established domain one would expect to find existing solutions. Finally, there were often discrepancies and time lags between the specification and the ProvToolbox code being updated, resulting in some confusion.

On a personal note, the author found it challenging to manage time constraints, maintaining a healthy balance between two jobs, the project and personal life. As such, not all the design was implemented, however, as stated previously, any area in which the solution was not completed, effort has been made to allow for straightforward continuation.

7.3 Learning Outcomes

The project has provided a number of opportunities to increase experience in research and development. Initially, managing such a large body of research with which the author had no experience was quite a daunting task. In performing this background investigation, it was important to effectively differentiate relevant from irrelevant information, while refining analytical and reviewing techniques. The research has introduced the author to an entirely new body of work which has proved to be of significant interest, so much so that continued involvement in this field is eagerly anticipated.

From a design perspective, the project has necessitated improvement in architectural skills. Emphasis has been placed throughout on providing a solution that is easily extensible for future purposes. Incorporating this modularity, especially in the presence of a Java Swing application with which the author has limited experience, proved to be a taxing but rewarding task.

The project represents a significant body of work demanding the development of organisational skills, persistence and energy. It was certainly a challenge to adapt from a largely taught semester to a self-motivated assignment of such length and complexity. The benefits of insights gained into the academic world of research were also considerable.

Finally, as the PROV community is currently quite small, the project has offered an opportunity for involvement within a new open source initiative by introducing a number of new technologies and techniques that were unfamiliar prior to the project. It is the hope of the author that participation in this community will be continued beyond this project.

7.4 Further Work

Due to time constraints, not all of the design was fully implemented in the final solution and therefore there are a number of areas in which development could be extended. In this regard, allowing seed graphs to be used as patterns and providing operations to be performed on these patterns, such as composition and repetition (see section 4.1.2) are likely to be the most useful features that could be integrated.

With current edge saturation semantics (see section 4.1.1), there is no constraint placed upon how edges are to be added or removed. As such, the solution provides an implementation (see section 5.2.2) that is both simple and reliable for graph expansions. It is very likely however, that different algorithms for edge saturation would be desirable, perhaps choosing every other edge to add, or providing an element of randomness by which edges are chosen. It would be useful to include a number of additional implementations to the application for user's convenience. This would require an extension of the Generator Schema defined in the design chapter. It would necessitate a supplementary application specific element on edges perhaps defined as follows:

```
<xs:element name="saturationImpl" type="xs:string"/>
```

This could then be used in conjunction with a number of standard terms such as 'everyOther' or 'random' to allow the generator to select the correct implementation type.

The Generator Schema provides extensibility to PROV-N, permitting for representations of seed graphs to be encoded. However, currently there is no convenient way to convert from JPROV to PROV-N, requiring an intermediate step through the ProvToolbox Interoperability Framework. It would be ideal if it were possible to both parse and generate PROV-N directly.

In the same vein, it was clear from the design chapter that there was no ideal solution to provide syntax for patterns and the more complex operations upon them. The potential resolutions were to use PROV-N's extensibility points in a very convoluted and unintuitive manner or to extend the PROV-N syntax directly, which isn't astute because it provides two purposes for the same language. Ideally, a meta-language could be developed that performs operations over PROV constructs, similar to the dot language. This would permit a clean separation of concerns for developers but unfortunately it is a significantly more complex undertaking.

Finally, as the PROV family of specifications are still drafts and undergoing changes, it is important to consider how JPROV and the options provided by the solution will be maintained and kept current.

7.5 Concluding Remarks

Overall, the project was successful in investigating and providing a way of generating synthetic provenance graphs. New, exploratory research was performed into a nascent field and a solution was designed and implemented that helped solve an emerging issue. The project was also beneficial to the author, providing exposure to new technical, academic and administrative issues. As a result, the project can be considered a successful endeavour.

Bibliography

[1] L. Moreau, “The Foundations for Provenance on the Web,” *Foundations and Trends in Web Science,* vol. 2, no. 2-3, pp. 99-241, 2010.

[2] Y. Simmhan and B. Plale, “Using Provenance for Personalized Quality Ranking of Scientific Datasets,” *I. J. Comput. Appl.,* vol. 18, no. 3, pp. 180-195, 2011.

[3] Y. Gil, J. Cheney, P. Groth, O. Hartig, S. Miles and L. P. d. S. P. Moreau, “Provenance XG Final Report,” 8 December 2010. [Online]. Available: http://www.w3.org/2005/incubator/prov/XGR-prov-20101214. [Accessed 29 August 2012].

[4] Y. Gil and S. Miles, “PROV Model Primer,” 24 July 2012. [Online]. Available: http://www.w3.org/TR/prov-primer. [Accessed 29 August 2012].

[5] L. Moreau and P. Missier, “PROV-DM: The PROV Data Model,” 24 July 2012. [Online]. Available: http://www.w3.org/TR/prov-dm. [Accessed 29 August 2012].

[6] L. Moreau and P. Missier, “PROV-N: The Provenance Notation,” 24 July 2012. [Online]. Available: http://www.w3.org/TR/prov-n. [Accessed 29 August 2012].

[7] J. Zhao, C. Goble, R. Stevens and D. Turi, “Mining Taverna's semantic web of provenance,” *Concurrency and Computation: Practice and Experience - The First Provenance Challenge,* vol. 20, no. 5, pp. 463-472, 2008.

[8] G. Ellis and A. Dix, “A taxonomy of clutter reduction for information visualastion,” *IEEE Transactions and Computer Graphics,* vol. 13, no. 6, pp. 1216-1223, 2007.

[9] P. Groth, “Use Case Retweets,” 11 January 2010. [Online]. Available: http://www.w3.org/2005/incubator/prov/wiki/Use_Case_Retweets. [Accessed 29 August 2012].

[10] P. Groth, S. Miles and M. Luc, “A model of process documentation to determine provenance in mash-ups,” *ACM Transactions on Internet Technology (TOIT),* vol. 9, no. 1, 2009.

[11] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Misser, J. Myers, B. Plale, Y. Simmhan, E. Stephan and J. Van den Bussche, “The Open Provenance Model core specification (v1.1),” *Future Generation Computer Systems,* vol. 27, no. 6, pp. 743-756, 2011.

[12] P. Buneman, S. Khanna and W. Chiew Tan, “Why and Where: A Characterization of Data Provenance,” in *8th International Conference on Database Theory*, 2001.

[13] J. Cheney, P. Missier and M. Luc, “Constraints of the Provenance Data Model,” 3 May 2012.

[Online]. Available: http://www.w3.org/TR/prov-constraints. [Accessed 29 August 2012].

[14] L. Moreau, O. Hartig, Y. Simmhan, J. Myers, T. Lebo, K. Belhajjame and S. Miles, "PROV-AQ: Provenance Access and Query," 19 June 2012. [Online]. Available: http://www.w3.org/TR/prov-aq. [Accessed 29 August 2012].

[15] P. Missier and K. Belhajjame, "A PROV encoding for provenance analysis using deductive rules," in *The International Provenance and Annotation Workshop Series*, Santa Barbara, 2012.

[16] L. Moreau, "ProvToolbox Github Repository," [Online]. Available: https://github.com/lucmoreau/ProvToolbox. [Accessed 29 August 2012].

[17] T. D. Huynh, "TrungDong Github Repository," [Online]. Available: https://github.com/trungdong/w3-prov. [Accessed 29 August 2012].

[18] A. Eckhardt, "Provenance Visualisation," 2012. [Online]. Available: https://github.com/alaneckhardt/Provenance-Visualisation. [Accessed 29 August 2012].

[19] Graphviz, "Graphviz," 2012. [Online]. Available: http://www.graphviz.org/. [Accessed 29 August 2012].

[20] Cytoscape, "Cytoscape: An Open Source Platform for Complex Network Analysis and Visualization," 2012. [Online]. Available: http://www.cytoscape.org/. [Accessed 29 August 2012].

[21] K. Rexer, H. Allen and P. Gearan, "2011 Data Miner Survey Summary," 2011. [Online]. Available: http://www.rexeranalytics.com/Data-Miner-Survey-Results-2011.html. [Accessed 29 August 2012].

[22] D. Leake and J. Kendall-Morwick, "Towards Case-Based Support for e-Science Workflow Generation by Mining Provenance," *Lecture Notes in Computer Science,* pp. 269-283, 2008.

[23] D. Holland, M. Seltzer, U. Braun and K.-K. Muniswamy-Reddy, "PASSing the provenance challenge," *Concurrency and Computation: Practice & Experience - The First Provenance Challenge,* vol. 20, no. 5, pp. 531-540, 2008.

[24] R. Barga and L. Digiampietri, "Automatic capture and efficient storage of eScience experiment provenance," *Concurrency and Computation: Practice and Experience,* vol. 20, no. 5, pp. 419-429, 2007.

[25] Y. Simmhan, B. Plale and D. Gannon, "Query capabilities of the Karma provenance framework," *Concurrency and Computation: Practice and Experience,* vol. 20, no. 5, 2008.

[26] J. H. Abawajy, S. I. Jami, Z. A. Shaikh and S. A. Hammad, "A framework for scalable distributed provenance storage system," *Computer Standard and Interfaces,* 20 July 2012.

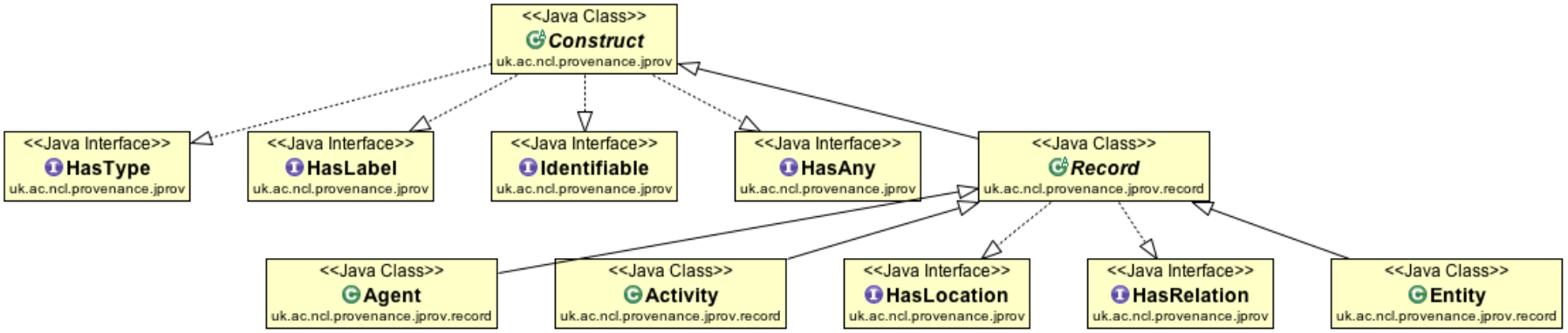
[27] A. Chapman, H. Jagadish and P. Ramanan, “Efficient Provenance Storage,” in *2008 ACM SIGMOD International Conferance of Data*, Vancouver, 2008.

[28] D. Chakrabarti and C. Faloutsos, “Graph mining: Laws, generators, and algorithms,” *ACM Computing Surveys (CSUR),* vol. 38, no. 1, 2006.

[29] M. Fowler, “Presentation Model,” [Online]. Available: http://martinfowler.com/eaaDev/PresentationModel.html. [Accessed 29 August 2012].

Appendix A

JPROV Record Class Diagram



JPROV Relation Class Diagram