The Context of Processes

Achieving thorough Documentation in Provenance Systems through Context Awareness

by

Ian Wootten

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

School of Computer Science

Cardiff University

June 2009

Abstract

To fully understand real world processes, having evidence which is as comprehensive as possible is essential. Comprehensive evidence enables the reviewer to have some confidence that they are aware of the nuances of a past scenario and can act appropriately upon them in the future. There are examples of this throughout everyday life; the outcome of a court case could be affected by available evidence; or an antique could be considered more valuable if certain facts about its history are known.

Similarly, in computer systems, evidence of processes allow users to make more informed decisions than if it were not captured. Where computer based experimentation has enabled scientists to perform complicated experiments quickly with ease, understanding the precise circumstances of the process which created a particular set of results is important. Significant recent research has sought to address the problem of understanding the *provenance* of an data item - the process which led to that data item. Increasingly, these experiments are being performed using systems which are distributed, large scale and open. Comprehensive evidence in these environments is achieved when both documentation of the actions performed *and* the circumstances in which they occur are captured. Therefore, in order for a user to achieve confidence in results, we argue the importance of documenting the *context* of a process. This thesis addresses the problem of how context may be suitably modeled, captured and queried to later answer questions concerning data origin.

We begin by defining context as any information describing a scenario which has some bearing on a process's outcome. Based on a number of use cases from a Functional Magnetic Resonance Imaging (fMRI) workflow, we present a model for representation of context. Our model treats each actor in a process as capable of progressing over a number of finite states as they perform actions. We show that each state can be encoded by using a set of monitored variables from an actor's host. Each transition between states therefore is a series of variable changes and this model is shown to be capable of measuring similarity of context when comparing multiple executions of the same process. It also allows us to consider future state changes for actors based on their past execution. We evaluate through the use of our own context capture system which allows common monitoring tools to be used as an indication of state change and recording of context transparently from stake holders. Our experimental findings suggest our approach to both be acceptable in terms of performance (with an overhead of 4-8% against a non context capturing approach) and use case satisfaction.

Acknowledgements

This thesis has just one author attributed to it, but I could not have completed the work without the help of a number of people.

Firstly, thanks to my supervisor Professor Omer Rana, for his guidance throughout my work and comments on drafts of this thesis. I would also like to thank the many researchers who have influenced my thoughts - I believe your words have shaped me to become a different person. Simon and Paul, thanks for helping me out when I've found things difficult (which honestly, was a lot of the time) and providing me with such a rich subject area to begin my work within. To all my colleagues at Cardiff - thanks for providing me with laughs when I haven't felt like laughing and putting up with all my talk of t-shirts.

Housemates of Inverness place, thanks for being such early risers and inspirational researchers. Thanks to all my family and friends for their constant nagging (incidentally you can all stop asking now). Jon and Cathy - thankyou for the use of your house as a cinema and becoming such valuable friends over my time as a postgraduate - as well as all that pizza.

I am grateful to my parents for laying the foundations that have ultimately got me to this point, along with their patience and love. Thanks also to my sister for not nagging me.

Finally, thanks to my wonderful wife Viv for your support - both emotional and financial over the last 3 years. Meeting you was not part of the plan for this research, but without you I could not have completed it. Thanks for believing in me.

Table of Contents

For Granny and Pop ...

Chapter 1

Introduction

Documenting the history of an entity in the real world plays a vital part in understanding an entity's origin and current state. There are exemplars of this throughout day-to-day life; the documentation of an antique's history may contribute toward its worth, documenting an accident using information from the people who observed it provides statements of what was seen, or documenting a patients medical history may allow understanding of whether correct procedures for their care have been adhered to. Documentation in these scenarios could occur in a number of forms including text, audio or video.

In order to interpret this body of information we make judgements based upon these past statements, which is often not an easy task. It requires being fully informed of actions and the circumstances in which those actions occur, as well as being skilled at logical interpretation of all of these details when they are put together. Interpretations of information where only the most basic evidence is available, or where erroneous reasoning is carried out, may result in ill-informed results being arrived at, which may ultimately end in unpredictable consequences in the cases of the above examples. Therefore where vital judgements are made against such data, having as much evidence available as is possible about a scenario is extremely important.

Similarly, within computer systems the documentation of the history of data items is very important. Knowledge of where a particular data item was originally obtained from and the transformations which were made upon it all contribute to understanding the final result of that process by proving its authenticity and the functional process it was a result of. The value of the final result therefore depends heavily upon the level of documentation which is collected about the processes which are performed upon that data. By being informed of this information, we say that we are *aware* of the provenance of that data.

The Compact Oxford English Dictionary describes the word *provenance* by *i) the origin or earliest known history of something* and *ii) a record of ownership of a work of art or an antique.* We use the definition throughout this thesis that **the provenance of some data is the process which led to that data** [55]. Such a definition differs from previous works which for example describe and model provenance as a data type [95], a type of database query [16] or a set of user interactions with a system [18].

Scientists have been more frequently making use of distributed computer systems to solve those problems that cannot be solved with local systems within their control. Computer-based representations of scientific processes make it possible to quickly execute experiments many times with modified hypothesis to gather extremely large amounts of data, with similar experiments possibly taking many weeks if conducted in a traditional lab. Where open systems are used as representations of experiments, entities which are involved in the process may not be under the control of the scientist executing them. Enforcing any particular type of mechanism for capturing evidence about the process which has occurred may therefore not be possible. It is increasingly difficult for scientists to understand how data products returned in such environments have been derived and by using open systems it becomes necessary to document the process which occurred to answer provenance based questions.

Examples of such questions include:

- Which resources were used in that experiment?
- Where were the resources which were used located?
- Why does one result differ from another result calculated from a previous run of the same experiment?

1.1 The Context of Processes

Consider the following real world example of a process: As consumer demand for ethically sourced goods increases, consumers desire to know more about the origins of the product which they desire to purchase. In the simple act of purchasing a t-shirt, a consumer may

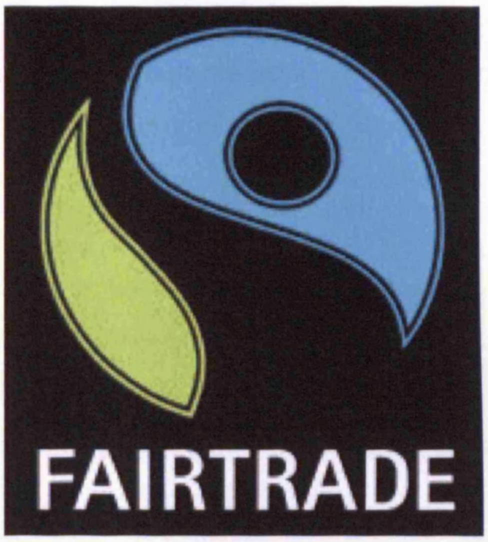


Figure 1.1: The Fairtrade mark describes (in-part) the provenance of the products on which it appears

want to know that the cotton used in its production is organic or otherwise and whether or not its materials have been traded fairly before they commit to making a purchase. The fairtrade symbol shown in figure 1.1 guarantees the consumer that the product on which it appears was made using goods according to conditions set out by the Fairtrade Foundation [*]. Seeing this logo and being aware of what it represents, the consumer purchases the product. The purchase of the t-shirt therefore is a decision based on the awareness of the provenance of the item, which in part is detailed by the symbol.

Although such a mark describes the source of the goods, the process which is performed upon it to transform it from its raw cotton state to a shop ready garment is not. A description of this process could be a record of the steps which were taken in the shirt's production (harvesting the cotton, weaving the shirt, dying the shirt etc). This evidence could equally instruct the consumer on how to repeat the same steps if they wanted to create a shirt themselves. Knowing that a process occurs however does not answer many other related queries about the nature of that process. The same steps may have been performed to create the t-shirt, but under much worse conditions for those who produced it, or the shirt in question could have been produced in an altogether different country. Being aware of the Fairtrade mark and the original source of the goods does not help the consumer

[*] http://www.fairtrade.org.uk

in answering such queries. Instead, the consumer needs to know the *context* in which the actions which comprise the process took place. This information allows the consumer to be aware of the circumstances or setting in which a process has taken place, which (along with the description of the process) allows a consumer to be better informed to make their purchase.

To be aware of the context of actions is a desirable quality for all processes in the real world. The above example illustrates its wider application. In computer systems, being aware of the conditions under which processes (or more accurately, the steps which they are comprised of) are performed equips an observer with additional knowledge when making queries about that process. Such information is beneficial when attempting to understand the results of computer based experiments. Often, these are repeated many times during experimentation with minor alterations to the process. Execution of a scientific process in an environment subject to high load for example, may allow a scientist to conclude why their results are unusual. Typically the context of the process might not be recorded, unavailable or difficult to interpret if recorded independently from documentation of the process. Without such information, the experimenter will gather the evidence they have available to them and form an interpretation of it, which ultimately may be incorrect or different from how they would interpret it when aware of context. By documenting context in such a situation they are enabled to make decisions not only against how the sequence of actions in a process were structured, but the circumstances in which those actions were carried out. When recording information regarding process execution, it is important to know about any differences to the environment which were observed. This ensures that any future judgements made across the evidence are as accurate as possible. Later in this thesis, we explore the approach of modelling context in computer based applications as a summary of the environmental conditions under which a process occurs.

1.2 A Problem of Confidence

Groth [32] explains that the concept of provenance has been introduced to ensure a greater confidence in the process by which physical or data items have been made. Specifically, this user confidence is concerned with the results generated by multi-institutional systems. In the t-shirt purchase scenario described, confidence is desired concerning the origins and manufacture of a garment when a consumer is looking to make a purchase. If a user is unaware of the process leading to some data, they will undoubtedly consider it at a value less than that of a similar data item that has associated evidence to describe its origin. By structuring documentation in agreed formats and sharing it with users of these systems, it is possible to gather and easily disseminate evidence amongst them. The same tasks can therefore be performed whilst instilling users with a greater confidence in the data created as a result of process execution.

As with our examples, the notion of a record of process occurrence is typically not enough to satisfy common provenance queries which a user may have. In its strictest sense, a record of process would only document the sequence of steps taken and the order in which they were performed. Without records of further detail (i.e the time at which this occurred, the user which started the process, etc), more specific provenance queries are unable to be completely answered.

However, recording evidence in this manner requires a user explicitly making the decision upon the level of documentation which they desire to record and its content prior to the process being carried out. To capture data at a high granularity is unnecessary all the time, but when this is required it is often only realised post-invocation - at which point it is too late for the data to be documented and repeating similar behaviour may take a long time or be difficult to reproduce. Complete confidence could therefore only truly ever be satisfied if a user is aware of the provenance of every single data item involved in a process, where provenance is the entire process of every data item's creation. This is an extremely grand vision to work towards; as to satisfy such a requirement, documentation leading right back to the beginning of time could be necessary to describe it [54].

Whilst current provenance solutions offer mechanisms for the capture of the evidence about steps in a process, few provide a way in which to document the context under which those steps are conducted or to interpret said documentation until after it is recorded. This context can prove to be of value when more specific provenance queries are outside the scope of a provenance capture system.

The huge variety of application domains to which documentation of process and the recording of context may be applied in part determines the type of data that needs to be collected as the state of an entity. Whereas provenance tools are generally created specifically for a single domain or modified to satisfy requirements for that domain, the

Provenance Aware Service Oriented Architecture (PASOA) project[†] aimed to provide tools which could be used to record and reason over evidence captured throughout open environments. Understanding the variety of information which should be able to be captured as context and creating a model for supporting its representation in distributed systems are therefore problems which need to be solved.

1.3 Context within Systems of Black Boxes

The use of highly specialist software at particular research institutions, often with common pieces of functionality has led to the adoption of service based systems to share functionality between these institutions. In this manner, it is possible for users of these systems to perform functionality outside of their domain and upon hardware which would outperform their own. These services have been termed as "black boxes", as little is known about their internal functionality by clients who regularly use them. Only the service's intended function and how it is called, is able to be ascertained from descriptions made publicly available. Within these situations (where implementation of a service may be hidden) a user may have no control over the documentation which is recorded upon a particular actor and is restricted to what is exposed by the actor's administrator. Typically, some remote monitoring of these systems occur at these actors, yet an experimenter may not be aware of this. As processes execute in these loosely coupled environments, the only actors which are able to assert contextual information about the process are those who are involved in the process. They are both aware of the environment (or situation) in which they execute, and the functionality which they expose. As the information collected at the actor may be extremely relevant to the experimenter (such as any monitoring over the period in which the process occurs), being able to capture this along with documentation of the process and providing references to it would be useful to anyone who may have to reason over it.

1.4 Thesis Statement, Aims and Contributions

The hypothesis of this thesis can be summarized as follows:

"The use of contextual information to augment records of causal processes (where the information holds a temporal relationship with the process) enables

[†]http://www.pasoa.org

comparison of results to be made in cases which documentation of the causal process alone would not do."

The main aims of the thesis are as follows:

1. To develop a contextually aware model for representing evidence about past processes. This model would be capable of relating documentation of context observed upon an actor to actions detailed in documented evidence. It should be able to represent the most plausible causal hypothesis based upon the body of knowledge and evidence about events and relationships which are currently available. The complete distributed process which caused events to be asserted or observed by an actor should be able to be determined through use of the model.

2. To describe how context originating from actors involved in a process may be structured and subsequently queried. This representation should be able to be used in the two possible alternative situations which are seen as the primary motivation for its capture. This should include (but is not limited to) the use of context for evaluation of action based evidence and the evaluation of context using evidence of actions.

Through exploration of these problems, this thesis provides the following research contributions:

- A distinction between *process* and *context* in open systems
- The indication of the failure of current provenance solutions to integrate appropriate information sources for context as a tool in process documentation.
- An understanding of the problems that originate in attempting to automate the capture of both environmental context and process documentation and the tradeoffs that occur between them as a result.
- A description of how observations of context can change over time and how *actor state* can represent this
- A formal definition for both the state of actors and the event types which trigger transitions between them. These definitions may also be used to describe states which occur outside of processes.

- A model which uses the definitions presented, to represent the evolution of an actors current or past state as a finite state automaton.
- A description of the 6 types of common cases regarding the availability of context in a service based system. Information provided by a service is available at varying degrees of detail.
- An understanding of the relationship between monitoring systems and the role they play in providing contextual information for process documentation.
- Conversely, how the collection of process documentation concerning a monitored actor in a process may be used to understand the origin of the state of that actor at any given point in time.

In addition, we also present software contributions in the form of:

- A prototype policy-based context collection framework for Web Services, the State Assertion Registry (StAR). This allows users to specify the contents of and how and when assertions of context are recorded according to the requirements of their own particular application domain, specifically focusing on that of environmental context.
- A prototype interpretation tool for environmental context. This uses the notion of distinct contexts generated from unique combinations of multi-variate time series variables to determine the conditions under which an actor has been executing functionality over a particular period of time.

1.5 Thesis Structure

This work is organised as follows: In Chapter 2 we perform a comprehensive literature review followed by an overview of current provenance solutions' inability to provide contextual information. Chapter 3 introduces a set of definitions and terminology appropriate for context, which is used in chapter 4 as the basis of a model for its representation. In Chapter 5 we present a solution which builds upon an existing architecture to create a system which is capable of documenting context as well as process. In Chapter 6 we evaluate our implementation of such an architecture and finally in Chapter 7 we conclude. This thesis builds upon and extends the research in our previously published works [87–90].

Chapter 2

Literature Review

Whilst the capture of sufficient documentation of provenance is universally considered paramount in distributed systems, the lack of a uniform description or standard for it has resulted in a number of differing interpretations of what it actually is. The difference in meanings across disciplines [60] has subsequently produced variations in what is recorded as part of each of these interpretations. More recently effort has been made for these projects to work together to produce a universal model for provenance throughout them [25]. Previously it has been presented that descriptions used in provenance terminology fall into two distinct categories, of *history* and *recipe* [92, 96]. Here, recipe focuses on the steps that were taken to create a data product and history details the events that occur relating to that data product. Provenance captured as "recipes" therefore are repeatable, but do not guarantee that the data product created as a result of performing them will be reproduced. A description of the history relating to a data product however does not mean that either the steps can be repeated or the data item reproduced. Our focus in this thesis is with that of the term recipe as a sequence of recorded steps; of most value when re-executing a process. Our choice of provenance terminology is that provenance is the *process* which was taken to create a particular data item [56]. Whereas provenance terminology and the data items captured between projects may differ, the ultimate aim of its capture is unique. Documentation describing provenance is desired to capture evidence of how data products or actors in systems came to be the way they currently are. We include a third set of terms about the *environment* which is used in provenance terminology to provide descriptions of the environment under which each of the documented steps takes place and is shown in

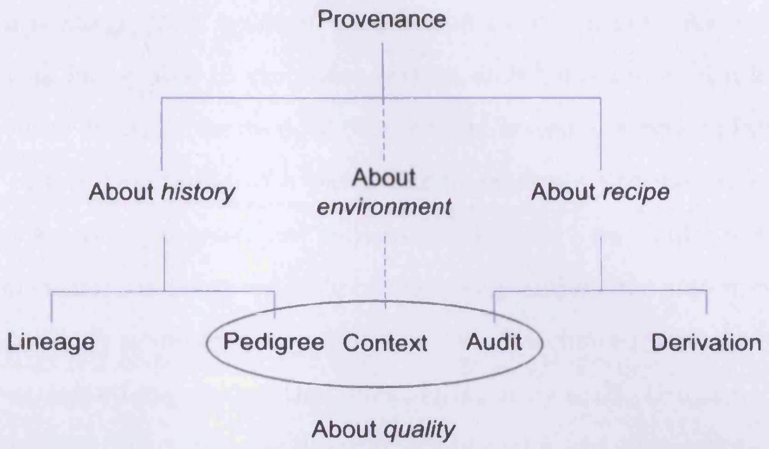


Figure 2.1: The relationships between different provenance "types", revised from [92]

figure 2.1. Whilst of no use in determining the origin or source of data products on its own, it complements recipe and history by providing additional details of the context a process was performed within. Context may be repeatable and doing so is more likely to reproduce data items as recipes are repeated.

This chapter explores related literature in the fields of provenance in multiple scientific domains and monitoring systems. We perform this review with the hypothesis that context is either not considered or misrepresented as an alternative type of provenance and that through a more appropriate capture technique it would be possible to repeat experiments whilst suitably documenting the environmental conditions under which they are taking place.

2.1 Context

Context has been investigated in the field of artificial intelligence (AI), as a means of solving the problem of "generality" - being able to restate a specific situation as a broader one [52]. Several works have focused on being able to formalise the concept of context for use in AI [7, 39]. Most of note in the literature they review are the role which has been given to context from its use in natural language. Clark and Carlson for example regard context as information available to a person for interaction with a particular process on a given occasion and they attempt to capture context which is potentially necessary for a process's success [22]. Leech [47] describes context's effect as narrowing down the communicative possibilities of a message as it exists in abstraction from context. Akman and Surav [7] describe context as being able to eliminate certain ambiguities or multiple meanings in a message. From these descriptions we note that context is seen as a type of information which is only capable of being recorded at a particular instant in a process. It may even be the case that context for some processes are necessary to achieve a particular outcome. The goal of being aware of context is being capable of narrowing ambiguous statements with several meanings to those with a single interpretation. This is achieved through "considering all related factors" to something, rather than considering it by itself. Our belief is that context enables the disambiguation of process descriptions in real world applications as well as these linguistic descriptions. By recording related factors to a particular process, those who reason over evidence are able to make more informed judgements on what has occured in the past. In the same paper, Akman and Surav [7] proceed to detail how context allows well defined queries of information to be made, which would enable better precision of documentation. These principles are also true of in scientific simulations, where vast amounts of processes are able to be quickly executed. Context could allow better, more precise queries to be made against available evidence.

2.2 Processes and their Context

We have already stated that provenance is captured in order to determine how an entity or data item came to be the way it currently is. We believe that the alternative definitions for provenance across different projects reviewed in this chapter have led to the inclusion of 4 common types of information within provenance descriptions for this purpose. *Action* describes the steps that have been taken in performing a particular process. This can be as historical observations or a 'recipe' for a set of actions. Using a record of actions, a scientist is able to find all the data items which have been used in, or produced as a result of, a sequence of steps. They are also able to find descriptions of the steps themselves and the order in which they were conducted. Workflow descriptions are typically used as a record of action. *Situation* describes the particular conditions under which an action or a sequence of actions (process) were conducted. These conditions are not limited to those systems which a process is implemented within. For example, situation could include descriptions of hardware or software upon which a set of actions are implemented, the time at which they began or the user who executed the workflow. In order to capture the reason for an experiment a record of *intention* is necessary within evidence. Although not commonly distinguished from other assertion types * intention is important to establish why an experiment may have been conducted in a particular way. Lastly, *interpretation* captures a hypothesis of the results or observations made within an experiment. This requires knowledge of the domain for which the records of actions are captured.

Out of the elements currently captured by pre existing provenance systems, the notion of action is treated as the fundamental type of information which is necessary to assert how a process has been conducted. It is important to scientists to be able to represent this information to ensure experiments are performed correctly and to repeat those steps that produce interesting results. Without these records of action, it is difficult to determine how results produced as part of a process have been derived, due to the many disparate components involved. Many solutions already exist for the capture of actions, commonly focusing on solutions suited for the requirements of a particular domain. The other three data types which we define here; situation, intention and interpretation do not describe the steps which have been performed, but the circumstances surrounding those steps - their *context*. This chapter shows that whilst context is not essential in describing each of the steps of how a process occurs, it is fundamental to understanding why these steps have occurred. Our particular focus within this thesis is with the much lesser explored field of the situation in which actions are conducted, due to the many tools which already exist capable of documenting it.

2.3 Approaches to Evidence Capture

As mentioned previously, many differing descriptions of what "provenance" is have resulted from the task of tracking the origin of data. In this section, we explore a variety of scientific environments which capture evidence about processes.

2.3.1 Provenance Environments

Scientific workflow systems allow the composition of a variety of distributed resources into chains to form processes to achieve some algorithm more complex than each of the indi-

*intentions may be captured as content within annotations to data vidual resources involved. Often this is facilitated in a visual manner, where each resource output forms the input to the next resource in that chain. The chains of resources would be described through use of a workflow specification language, such as the Business Process Execution Language (BPEL) or IBM's Web Services Flow Language (WSFL). Several workflow based systems exist which record process documentation in order to determine the provenance of a data item, resultant through executing a particular workflow. Typically within these systems, documentation concerning process is captured both about the operations which led to particular data items [50,67] (inputs, outputs, order of invocation, data parameters), and the evolution of the workflow (user modifications) [18,28]. This is achieved using an eager approach, that is, recording documentation as it is observed, rather than collecting it and recording it at a later date.

Applications which have requirements for the capture of the origin of data, typically record elements of action and context as meta-data. Due to its machine based nature, the automation of its capture is possible and means that scientists need not be concerned with documentation of process, instead focusing upon the experimentation they wish to perform. This automatic capture is needed where scientists may forget to document important steps with information which may be critical to understanding the process at a later date. Common practices include the wrapping of functions with scripts or classes to "provenance enable" them, as well as recording additional documentation in Web services. In this way documentation of process is able to be captured without having to be remembered by a user. However, attempting to capture all evidence automatically has been regarded as difficult, especially when rich semantics are to be included. Instead, the automatic creation of metadata and later enrichment through semantic annotations has been demonstrated as a means of providing a hybrid approach to a more complete process documentation set [20].

MyGrid

MyGrid[†] is an project focusing on facilitating scientific *in silico* experimentation within the bioinformatics domain. The Taverna workflow environment used in MyGrid provides a visual front end to the use of Grid services which provide elements of functionality a experimenter requires. Taverna has its own workflow description language which allows abstract workflow definition of which instances include data parameters and specific services.

[†]http://www.mygrid.org.uk

The Taverna workbench which was designed as part of the MyGrid project, allows the creation of computer based (in silico) experiments for bioinformatics [67]. Provenance is recorded automatically both eagerly and lazily through 4 different provenance log types, process, data, organisation and knowledge in order to satisfy the requirements of multiple user types [95]. Whilst process and data capture are common with many other provenance middleware, contextual information concerning the service user, experiment design or hypothesis a workflow execution is based on is not so. Knowledge provenance, that is, data concerning the creation and evolution of contextual information is able to be mined from the other 3 forms of documentation logs. The use of public resources however means that workflows are required to be repeatedly re-run in order to ensure that data in the workflow is up to date [94]. Comparisons against provenance graphs may be drawn from workflows which are the same but produce alternative results, or workflows which differ but produce the same results.

Provenance in this scenario is important to allow scientists to determine how or why results have been produced after a particular process has been executed in Taverna [95]. A more systematic, structured representation of this information is called for in this community to that of ad-hoc logging of events. Its capture has been separated out into a number of distinct levels (data, process organisation and knowledge), to cater for the wide variety of users who will be inspecting it at each of the different levels of detail that they require. Automatic logs of each of these provenance types are able to be recorded during a workflow invocation and is represented in a serialisable Resource Description Framework (RDF) format. These describe events which occurred in the system during a process, including the times services were accessed, intermediate/final workflow results and input data parameters or contextual information. Here, contextual information includes the user of the data, the design of the experiment or the larger project it is a part of. These automatic logs are often representative of information that would be captured manually to lab note books if the same experiment was performed by hand. The RDF represents a graph of data products and parameters, connected by the relationships they hold with one another [94]. Some information may easily be recorded automatically due to being process-oriented and determinable at runtime by MyGrids FreeFluo workflow engine, such as values of data parameters to services. To include further complicated semantic meta-data as part of these logs requires that manual annotations are made by a user or third party using free or controlled vocabularies [93, 95]. Context therefore is crucial to capture detail of a variety of types of information in MyGrid. Each of these hold a complex relationship with the experimental process which is executed, which according to our earlier definitions is either evidence describing the situation or original intention of executing the process.

Visualising provenance information in MyGrid is made possible with a number of tools which cater for RDF metadata visualisation. However, due to the vast amount of relationships represented in a provenance log it is difficult for users to interpret a full visualisation held within them and so filters to show the most relevant relationships to the current user or task are necessary [95].

CMCS

The Collaboratory for Multi-scale Chemical Science (CMCS) project, also describes provenance as meta-data, or "data relating to other data" [64]. CMCS has been demonstrated on a number of applications in group analysis of heterogeneous data captured during combustion research [63]. Meta data is viewed as the central concept behind capturing the pedigree (provenance) of data items. It was created to ensure that research results are properly documented with enough information to guide further research in the chemical sciences community [63, 64]. The project has attempted to construct a set of core representations for provenance, instead of standardising the format research communities are to use. CMCS uses a file representation mechanism to capture resource descriptions such as authors, relationships and date information [64, 69]. Many of these elements we would consider context using our own descriptions (such as modified and created dates, version number, platform, creator or any associated keywords). In one description of CMCS, a requirement is given as to "document the context and value of the data" [69], indicating the original motivation for the data elements capture was more than just the steps which led to an item. An extension to the project's generic schema for documenting provenance has been created to define those elements specific to the chemical sciences domain. This includes the species name, its chemical formula and its chemical properties, with the ability to specify additional data items. Provenance in CMCS is created on demand according to a set of relationships specifically defined for the user or group for whom it is intended [64]. This is an attempt to recognise the differences persons in the community will have in terms of their provenance requirements and the meaning which they automatically ascribe to it.

CMCS centres around the use of a portal for sharing provenance information, which provides a variety of tools for annotating, exploring and visualising recorded relationships. Using a middleware called Scientific Annotation Middleware (SAM), the tracking of provenance is automated [68]. The data pedigree is extracted from an XML based representation of a document before being placed as property files in a Distributed Authoring and Versioning System (DAV), along with the original document [68,69]. As DAV URL's correspond to property files and the documents with which they are associated, CMCS is able to update the pedigree of data without altering the original document through modifying the property file for a document. This is achieved through a portal based interface which allows researchers to have access to management, collaboration and productivity tools [64, 68]. These tools include a generic pedigree browser and graph portlet, which provide both the original metadata and relationships as well as a visual representation of the data set as a node-and-arc based graph [63, 64]. Also included is a mechanism to provide manual annotation of documents using a free text format with further analysis post publication [69]. A minimal schema is defined describing data pedigree meta-data which is seen as an extensible mechanism for other researchers to use to meet more specific needs. These tools are described as being independent of the chemical sciences research community and could be adopted elsewhere [68]. However, as with MyGrid, the chain of relationships can often be long and complex to understand, with a less detailed representation being insufficient for users in the long term. As a response, a graph tool is provided which is able to be configured to show a graph of specific relationships (such as processing history or association between refereed papers) for resources and filtering out all others [63,69].

In contrast to many other workflow based systems, there is no mechanism in CMCS in order to automate the capture of evidence from a workflow execution. Instead, this is achieved manually through portal submission or WebDAV-aware applications. For those without the programmatical skills to enable this submission, manual annotation would be a time consuming process.

CMCS shows that attempting to build systems which cater for such diverse application domains will always necessitate the use of extensions to fully meet user needs. Whether this is achieved through manual annotation of pre-recorded documentation or plug-ins to generic systems to capture all information relevant to the particular application domain, user modification to systems is required based on a projects specific use cases.

Geodise

The generation of semantic metadata for Grid enabled functions has been demonstrated in engineering design where its capture enables engineers to be advised on design improvements in problem solving environments [19,79,85]. In the GEODISE project [‡], the annotation of mathematical functions with meta data allows engineers to determine how to build work-flows which are appropriate for solving specific design problems. This data management infrastructure was created in response to solving the problems of large amounts of distributed data encoded within multiple formats [85]. The knowledge within GEODISE is constructed from detailed meta data captured both automatically and manually which describes the interfaces for available Matlab functions [19]. Requirements are later presented for a system capable of 'augmented' provenance capture, which is based upon assertions that provenance should be captured at different levels, across a number of differing categories - all of which are application dependent [20]. The capture of both documentation of the process which led to data, as well as the semantic meta data which describes it, is argued to be useful in the enhancement of process documentation [20]. This meta-data by our definition would be construed as the context for the process, detailing runtime environment settings (situation), decisions (intention) and conclusions (interpretation). The representation for recording evidence is through ontologies and semantic annotation. However, this method of annotation has not been applied to anything but the engineering application it was built for and is application dependant, meaning different annotations will be necessary for any others which choose to adopt it. GEODISE's query mechanism is based around identification of the particular workflow template, workflow invocation and associated generated metadata for a given query. Workflow templates describe semantic meta data to be captured during a process invocation and are assigned identifiers which are captured to a separate remote repository. The template, on invocation is also assigned a particular instance identifier and the relationship between the two is also captured. The repository enables the sharing and efficient search of data and due to the data and semantics which are captured, users have the option of performing a direct query, a semantics based query, or a combination of the two.

[‡]http://www.geodise.org

The Virtual Data Model

The Grid Physics Network (GriPhyN) project has worked towards creation of a language which is able to describe transformations performed upon data in application domains such as High-Energy Physics and Astronomy [26]. The Virtual Data System (VDS) created is capable of documenting associations between datasets which are derived through existing knowledge of transformations along with subsequent query of these relationships. These are stored in a 'virtual data' catalog which acts as a repository for the documentation of process concerning a data item [27,96] whilst relationships between data and procedures are expressed in a Virtual Data Language (VDL). This catalog is able to be defined by a user by extending a Virtual Data schema, with their own descriptions of data sets applicable in their particular scientific community through the VDL. This gives a selection of different data descriptors across the different projects which use the VDS. Through associating data with descriptions of the functional procedures which derive them (termed as *prospective* provenance [96]), it is possible to describe the complete derivation procedure for products which are generated during workflow invocation [26]. The virtual data language interpreter operates against the virtual data catalog in order to provide a querying mechanism. Queries against captured information allow experimenters to determine if the data they require exists and if it does not, it is created, so long as transformations detailing its creation process are available to do so. A query may also return a directed acyclic graph of the tasks which on execution will create a specified data product [26]. The model also includes contextual information concerning the state of hardware and software environment whilst transformations are executed, known as *retrospective* provenance, with Zhao et al. arguing that the capture of both prospective and retrospective provenance makes for a *complete* provenance record which allows a user to reason about any data or conclusions drawn from it [96]. VDS however requires that functional knowledge of the system is explicitly declared to be able to describe the transformations which are documented - whilst this is common in application programs, in an open system a client will typically have no knowledge of the underlying logic of actors which were involved in an experiment. It is therefore impossible for a client to make any assertion about "retrospective" provenance (unless it is described by the service) in an open system, which would make evidence about the process incomplete according to Zhao et al.

Triana

Triana [§], a workflow composition tool from the University of Cardiff, doesn't strictly have provenance tracking, but instead uses history tracking as its primary form of process documentation. History tracking dynamically builds a copy of a workflow with all parameters as the workflow is executed [50]. A data object records its path though any given workflow, storing information at every process it passes though. A saved history annotated workflow can be re-opened in Triana and re-run to give the same result. The history of data objects is tracked through the workflow, storing information at each of the processes it passes through. As provenance is unable to be handled in a generic manner by components, due to Triana knowing nothing about resources except for the interfaces which they use, "clip-ins" are used to pass metadata around with objects. A clip-in can be anything, such as axis labels with graphable data or passwords for secure data access and also the history of the data object as it passes though the workflow. If a component in Triana does not know how to handle a particular clip-in then it is passed through unchanged with the data. The Triana engine records the state of the data object before and after it has passed through a component inside a clip-in. As yet Triana only performs this tracking for Java based components built within it (functionality available locally to machine upon which Triana is installed) and not for Web Services. This is restrictive given that services are often used in similar problem solving environments (such as MyGrid's Taverna) and are capable of capturing evidence for remote service based actors. Also, storage of this history is captured local to the Triana installation, where the use of a central storage repository might be more appropriate for dissemination amongst experimenters.

Vistrails

The modeling of provenance as a set of user interactions in a workflow system allows the capture of provenance in a transparent, unobtrusive manner. This action-based provenance approach is used in the visualisation of data models such as the planning of radiation oncology treatment [18,28]. The workflow environment (VisTrails), maintains a record of user actions as scientists go about interacting with the system to explore hypotheses. Included is both information on the data products generated and the workflow evolution, which would

[§]http://www.trianacode.org

typically have to be documented by hand when using other existing visualisation approaches. As the workflow evolution is captured as a tree (a vistrail), where each node is modeled as a version of that workflow and each edge a transformation of one workflow to another, the experimenter can evaluate the exploratory process they have conducted to achieve a particular result. The division of the system into a workflow cache and player [18], ensures that only new combinations of parameters and operations are executed, and the rest are retrieved from cache achieving a prompt re-execution of similar workflows. A layer-based model for the system is used which captures the workflow definition, evolution and execution traces [76], with support for user annotations upon each. VisTrails is able to have every one of these either replaced or built upon with new layers. VisTrails does not however capture such information for any data items which are modeled outside of the workflow environment (for example temporary files). The passive approach to detailing this evolution of the workflow is unique, but only really necessary where the workflows which are in use are evolving. The creators actually state VisTrails is not intended to replace existing workflow systems, but instead enhance them [28]. In a workflow which is not subject to change, where the same experiments are repeatedly performed, data exploration is not a primary concern. Instead (as we have already described in earlier applications) the context of the actions being executed is.

Kepler

VisTrails algorithm for the reduction of workflow execution is later borrowed in the Kepler Scientific Workflow System ¶, which uses it to execute partial workflows in a multi-disciplinary service oriented environment [10]. The workflow evolution is a complete trace of how a workflow has evolved over a period and allows users to return to those versions of a workflow which produced interesting results. Kepler models the re-run of previous workflow through a Smart Re-run Manager (SRM) which allows partial workflow to be executed prior to a particular altered component. To avoid the time consuming task of re-executing the complete workflow, results are extracted from cache given that the same actors have previously been executed with the same parameters. Actors which produce an output which is dependent on both the inputs, parameters and when the actor is invoked, are deemed non-cacheable and can be specified to be re-executed every time a 'smart' run

¶http://www.kepler-project.org

is performed. Kepler is modeled as a number of separate 'concerns' which include actors, their composition, the computational implementation of the workflow system and provenance. This concern may be bound visually to the workflow in question, allowing users to observe if provenance is being recorded for a particular workflow run. A number of levels of detail are given to a user at a maximum level that offers complete re-creation of workflow results. Several different pieces of information are recorded, intermediate data products, workflow definition, evolution and inputs and outputs. Also collected is the *context* of each run, which is described as the who, what, where, when and why [10]. Although actor based, such information is manually associated through and recorded by the Kepler environment itself. Publicly available services would therefore only have records of context which had been invoked via Kepler, even if such information was documented in a public repository.

2.3.2 Passive Process Capture

We have earlier introduced how metadata may be captured for actors in a workflow environment without interaction from the user. The following systems try and collect *all* provenance data passively.

The Earth Systems Science Workbench (ESSW) is a data management infrastructure for the dissemination of Earth Science data products amongst research groups [29]. The interpretation of such data may not be achieved easily between groups, where important descriptions may be accidentally omitted when new data products are disseminated to others. Experimenters will commonly use generated scripts which capture meta data about products as part of the path to the location of the meta data file on a storage medium. Such a mechanism for detailing contents is vulnerable, where files are able to be moved and paths may no longer refer to the same file. Instead of providing a comprehensive environment for uniform capture of all necessary data, ESSW attempts to describe the existing scientific process and provides guidelines for lineage composition [13]. This is non-intrusive through its use of scripting wrappers to log meta data, rather than requiring a change to existing processing methods to enable users to create such logs. The capture of data lineage includes both meta data about the data products generated and the process which was used to generate them. Meta data templates for each object within an experiment are specified through use of XML Document Type Definitions (DTDs), which specify data to be collected each time an experiment is performed. These documents allow scientists to observe the source of errors as well as locate subsequent erroneous data products within workflows. The captured XML is stored in a remote MySQL database, where lineage documentation search and retrieval is efficiently enabled through a Web based interface [13]. The suggestion of using a more complex query mechanism than just the data product and its related child transformations draws on the use of the RDF as a means of representing more general properties of the meta data which is captured [13]. Such query extensions would allow more general questions to be answered, more conducive of the types of reasoning that scientists typically make.

Recording provenance can also be performed at an extremely low-level, for example when calls to libraries and the operating system are recorded as provenance. The Provenance Aware Storage System (PASS) operates at such a level, automatically recording user and program actions [14]. To record all events at this level represents a huge quantity of data which could be far too vast for a user to comprehend, so observations are able to be annotated with a coarser representation to aid its subsequent use during reasoning. Similar low level monitoring is employed in the Earth System Science Server (ES3) project [30] through call tracing at a Linux shell prompt, though PASS is able to collect it at a finer detail level from the host's operating system. Both systems however employ a similar *passive* collection of documentation to describe the process. This differs from solutions already presented, as information is collected from observations that can be made about the process without modification to the system. The level at which provenance should be represented however, firmly relies on its application with the imposition of default data items to be recorded - which does not offer a suitable solution in all cases.

2.3.3 Process Oriented Provenance

The Provenance Aware Service Oriented Architecture (PASOA) [||] and Grid Provenance [**] projects describe provenance as "the *process* which was taken to create a particular data item". Both groups are attempting to describe a computer-based documentation of process such that assertions detailing process in distributed systems may be represented. PASOA has developed the Provenance Recording Protocol (PReP) [34, 35], which describes how provenance may be captured within a service oriented system. This is primarily centred

[||] http://www.pasoa.org

[**] http://www.gridprovenance.org

around the concept of a *provenance store*, where documentation of process is sent by both client and services which are involved in an interaction. Recording in this manner gives two independent views of the process which has occurred between actors (from both clients and services), which may agree or be non-consensual. The PReP protocol is comprised of four phases [34]: negotiation, invocation, provenance recording and termination. The negotiation phase allows both client and service to agree upon a third party provenance store where process documentation shall be recorded. In the invocation phase, the service actor is invoked, returning its result to the client. During the provenance recording phase, provenance documentation is recorded to the provenance store. When all documentation has been recorded, the protocol ends with the termination phase. In later work, Groth et al. [37] formalise the types of data that process documentation should capture through introduction of the provenance assertion (p-assertion). Such assertions can describe the interaction between actors through capture of the messages that are exchanged between them, relationships between assertions and the state of an actor at a point during the interaction. Here, although its content is undefined by either of the projects, the actor state assertion provides a mechanism to capture valuable information on the context of an action at any given time in a process. Most recently the content captured in these particular assertions has been described as internal information which contributes to the process in some way, but how that information is generated is not of interest [32]. In fact the assertion has been built according to several use cases motivating context capture alongside documentation of process. The examples given are documentation of experiment configurations or the semantics of entities involved in a process [56]. This approach has been adopted for a variety of application scenarios, such as Aerospace Engineering [46] and Trust Assessment [72].

The Grid Provenance project adopts the p-assertion and provenance store as critical components of their Provenance Architecture [33], formally defining the p-assertion and taking into consideration the security aspects that need to be upheld at a provenance store. Due to the architecture's technologically independent nature, it is possible for different technologies to use the ideas and models presented. Both PASOA and the Grid Provenance projects present an interesting alternate definition of what provenance is; that rather than it being a particular category of data to describe a process, it is the process itself. The greater the amount of detail that is exposed about this process, the better enabled to make judgements on relationships those who reason over such data are. Context, as shown by the use of the actor state assertion, is an integral part of this solution, necessary in a multitude of applications. The choice to not adopt any formal structure for its content (instead leaving this to the applications which make use of the model) shows that the variety of different motivations for context capture (intention, interpretation and situation) are difficult to represent in a single structure. The Grid Provenance project has made attempts at trying to solve this problem, though due to context's application dependance, representation of it has never been formally specified.

2.3.4 The Open Provenance Model

In response to the wide variety of systems which have been employed to represent evidence, the provenance community has been working together on a model to describe causal processes to better understand differences/similarities in their applications. The Open Provenance Model (OPM) has been designed with a number of core aims including allowing provenance information to be exchanged between systems and to allow developers to build tools which operate on such a model [25]. It has been presented in a technology agnostic manner so that existing provenance applications may represent their own evidence within it using implementation suitable for their own application. In OPM processes are built from 3 elements: artifacts which represent the state of objects in the system, processes which represent the actions or series of actions making up a process and agents which control execution of a process. Processes are represented by graphs of these 3 elements which ultimately result in leading to an artifact. The arcs in each graph in OPM represent the causal dependencies between entities. In this model, contextual pieces of data could be considered artifacts as they are immutable pieces of state. Time is modeled through annotations which may be applied to an OPM graph, to indicate start and end time of processes. Although this approach does lend itself well to represent many of the application scenarios we have already covered in this chapter, it requires work on the part of the project to implement a solution to its representation. Currently, aside from allowing provenance information to be easily interpreted, adopting OPM may not benefit all those applications which have their own proprietary provenance representation - especially if they do not have an interest in sharing such information. Due to its representation of only the causal process, OPM does not include adequate information to enable re-execution of that process, such as the location of any services used. Therefore it could not be used as the only representation available for a provenance environment and would require either an extension of the generic schema or a more detailed secondary representation.

2.3.5 Provenance in Database Systems

Within database environments, provenance solutions have focused upon the data lineage problem. This can be described as the set of source data which produced a given data item in a materialised view, along with the processes which were performed upon that source data [23]. Woodruff and Stonebraker explain the lineage problem as the base set of data and the sequence of processes performed upon it to create a processed dataset. They introduce a lazy calculation method for fine grained lineage which does not rely on the creation of metadata but creates it upon request [86]. This is described as weak inversion and computes the function f^{-w} of a particular function f which attempts to map from the output of f to its input. This is demonstrated with respect to determining a set of tuples from a database containing attributes which during processing affected the resultant tuple set. No formal method is given to determine the weak inverse however, which also requires verification and can only offer guarantees about the lineage generated rather than a perfect inversion of the data. The best description of the output given by weak inversion is that it is either a subset of the original data, or a set which includes it. Ultimately this is quite vague and would not suffice in a situation where inspection of the data items was necessary at intermediate steps in a complete process. The view of provenance is conducive with a sequence of steps (functions) being performed upon some input data which ultimately create some final dataset. However, as the source data is structured as tables in a database, lineage does not include the attributes we describe which make up context.

Cui et al. [23] further explore the data lineage problem by presenting algorithms to identify the exact set of base data contributing to a particular view item. Methods for consistent lineage capture within a warehousing environment, comprised of a number of sources are also presented.

Buneman et al. expand on this idea [16] through distinguishing between two provenance types, "where" provenance which details the location(s) from which data was extracted within the database, while defining previous data lineage work as "why" provenance; why a particular tuple is within the database. In other work [15], Buneman et al. suggest a timestamp based approach for recording change descriptions for XML data, which merges versions based on keys in contrast to previous difference based approaches. Their research is motivated by the tradeoff between frequency of data collection and the space necessary for archival of each state of the database. For our notion of context, this is also a concern due to the wide amount of detail we have described it as covering as well as the frequency with which some of these elements may be updated.

2.3.6 Provenance Capture for Humans

The concept of provenance is not just applicable to tracing the origin of data items or physical objects in the real world. Bell introduces the idea that it is possible to determine the context of persons through the data which is exposed about them publicly upon the internet [12]. He describes the problem of trusting identity online and asserts that the provenance of a *person* can be aggregated from multiple public identities of that person which they freely choose to expose. As users of the world wide web expose more and more information about themselves online through easier to use and more available tools, it is possible to determine a profile for a person gathered from this data. These are available through various websites which expose elements of that person's life (images, events etc) in the world wide web. Such information describing the provenance of real persons allows a detailed picture of who someone is gathered from how they are represented upon the internet to be built, with Bell questioning whether the users of these systems are fully aware of the implications of publishing such data. The introduction of more structured semantic markup available upon each of the websites he uses as exemplars mean that it is relatively easy to make links between these disparate representations of identity (examples of which include the websites tumblr.com, flickr.com, delicious.com and upcoming.org). Data mining of these aggregated identities allows a user who desires to have such historical information for a person to make decisions based upon that persons current context online and how much they trust them. Similar work by Mindswap[††] explores using the provenance of a particular set of statements to infer trust relationships between persons [31]. The problem with ascertaining provenance for humans in such a manner is that it is only possible for a particular type of person. As with actors in a service based system, not everyone chooses to make such data readily available about themselves, or ensures that it is marked up in such

[††]http://www.mindswap.org

a way that links between relevant pieces of information can be found. It is entirely possible not to have such data available or data which only describes particular aspects of that person's life. Contextual information able to be derived in these scenarios may therefore be weighted by the type of information which is freely exposed.

2.3.7 Generic Provenance Solutions

There exists a tradeoff between creation of generic solutions to documenting processes and more specific use-case solutions built for a specific project.

Case for generic solutions: Whilst process documentation is able to be captured for query by experimenters in specific environments, where service-based middleware is used, this is captured purely by the client system in any particular scenario. Documentation in an open, service based system requires the use of a more generic solution to capture complete documentation of all actors involved in a process. This includes information which is able to be captured by the actor providing the service, so that all actors in a given interaction are able to make their own assertions of provenance at any given time. A more complete view of what has actually occurred in a system is then represented, due to the increased level of trust which may be placed in assertions which reinforce and confirm the same events. Although the PASOA project presents a model for documentation of processes in SOA as a solution, it does not include any formal method by which the context of interactions are able to be documented, nor considers how it may be used.

Case against generic solutions: The application for which process documentation is captured determines the type of data which is necessary for capture. Solutions which attempt to document the process occurring in all applications will always require extensions to the generic system as provenance use cases are realised within a particular domain. It has been highlighted that attempting to standardise both the meaning and representation which applications have used for provenance is unlikely to meet the needs of particular communities [64]. Although the open nature of many of the provenance solutions examined lends well to a more complete process documentation (through wrapping components with calls to additional functionality), until the user has made decisions on the types of queries which are to be made of the data, a sufficient solution to that domain most likely will not be achieved.

2.3.8 Revisiting Key Concerns

Here we revisit some of the key findings from our critical evaluation of the state of context in current approaches to capturing evidence about processes.

- *Context is integral to answering provenance queries* - In the many provenance environments which we have discussed, documentation is often captured to describe the context in which a process has executed. We described how context is used in language, in order to reduce the ambiguous nature of statements. Where several interpretations of evidence could occur, context needs to be examined to better understand it. Understanding this context is paramount to satisfying a variety of provenance queries, such as knowing who executed a process recorded in MyGrid, or runtime environment settings captured in Geodise. These exemplars show that context is necessary to answering these questions and that it is already being captured in the majority of projects.

- *Context is distinct from process* - The method by which many existing approaches choose to model the notion of context is entirely distinct from that of the steps which have been performed to achieve results. In Taverna for example, process organisation is modelled separately from data, giving multiple levels of detail for those who require it. In the PASOA project, the sequence of steps in a process and their relationships are represented by interaction and relationship assertions, whilst evidence describing context is modeled by an actor state assertion. Other approaches have not distinguished between the two, recording evidence about both as a single collection of data used to answer provenance queries. We believe that such a distinction is necessary in order to facilitate use in the maximum number of use cases possible.

- *Context is application specific* - As CMCS showed, building a generic solution in order to a answer provenance queries will always result in extension of that solution to satisfy application use cases. An all in one outline of what should form the content of context documentation is therefore inappropriate. The Grid Provenance project we reviewed has made attempts at a common structure for actor state, but it was never integrated into the model as it was not able to satisfy all participating projects requirements.

- *Context does not have a uniform solution* - We evaluated a number of environments that offer generic solutions which could be used in other domains. There are however, many solutions each built for a particular domain. Although work is continuing to find an uniform approach to modeling this evidence, in the form of the Open Provenance Model, the variety of differing applications, requirements and solutions mean that currently there is no single representation which is able to satisfy all the demands of the entire provenance community.

2.4 Approaches to Context Capture

In this section we explore common approaches to observation of contextual information in open systems. Our focus is with systems that are capable of capturing situation, as this is widely modeled through use of monitoring systems. We show that although a variety of alternatives exist, no single monitoring mechanism is currently able to satisfy all user requirements of provenance systems. We suggest a model for context which may be implemented in a variety of ways. This model should be able to be used upon a set of variables made available by the actor. In doing so, transitions between states which are overlooked in other provenance solutions are able to be highlighted. This is performed in open systems where little knowledge of actors or the domain is available. We show that such an alternative could be successfully integrated with provenance systems to better understand contextual history of each actor within the environment in which it resides. Alternatively, we show that documenting context in this manner also yields a better understanding of the process which led to a context.

2.4.1 Log File Analysis

Log file analysis involves the mining of interesting events and patterns from log files resident on a particular system or collection of systems. Typically, a user of a logging system will instrument their code with print statements in order to indicate the functionality which is being executed in a program. Complex logging systems exist to ease the process by which this is done, providing a variety of levels at which logging statements can be made, so they may be switched on/off simply [40]. The user executing the code is then able to observe the logged statements to analyse them and attempt to determine what has occurred based on their own understanding of the system. While such tasks can be performed manually by a system administrator, this is only realistic for smaller log files and is a time consuming process.

To inspect the large amounts of data that may be within logs maintained in busier systems, analysis tools can be used to mine events more quickly. Using a known string pattern of evidence for an event, logs can be processed by searching for that particular string. If any are found, alerts can be generated to warn system administrators so that appropriate actions may be taken in response. However, reporting back all events individually often fails to completely diagnose a situation. The simple event correlator (SEC), uses system knowledge to indicate which events require investigation. This reduces the reporting of false positives/negatives and groups events to indicate where further investigation may need to occur [74, 82]. Using regular expressions, SEC is able to quickly parse log files for known evidence of events, extracting data using expression matches into variables. Usually log files which SEC inspects are written by logging details of events of interest as a single line in the file. However, several different rule types cater for the evidence which may only be apparent in complicated scenarios when multiple lines have been logged [74]. In order to recognise and differentiate between events SEC attempts to fully describe the context under which abnormal events occur. In later work, Vaarandi (SEC's author) presents a number of algorithms for mining frequent patterns from log files using breadth-first search and clustering algorithms [83, 84].

SEC illustrates how contexts can be used to reduce the number of reports generated from observations gathered from monitoring data, though is limited in particular to log files made upon a single system. An open system may include a multitude of monitoring mechanisms, including those which may not capture contextual evidence in a log file format. The interpretation of a number of log files which are distributed across several different hosts would prove to be extremely difficult during analysis, particularly when attempting to synchronize those logs according to the order in which events occurred on different hosts.

2.4.2 Time Series Monitoring

Many distributed systems make use of time series monitoring tools in order to capture details of how particular metrics perform over extended periods of time. A common approach is by using a monitoring daemon located upon each of the systems which is to be monitored, where several system metrics are reported either to a central repository or to other systems upon the local network at pre-defined intervals. The Ganglia monitoring System [51] operates in such a manner, sending results to a network multi-cast address to keep all actors in the system aware of the history of metric values on all other actors. Using a web front-end hosted at one of the actors allows administrators to view the results of metric collection visually and determine if past behavior of metrics has been unusual or interesting. The quantity of data collected by such systems can be extremely large, and storage and query of metrics which are collected at small intervals can become difficult if left for long periods. For instance, given a polling interval of once a minute and 30 metrics each of 64-bit size, a weeks worth of data would be approximately 2MB. Given that time series data typically can be held for several years for systems and networks consisting of hundreds of nodes, a repository of monitoring data could hold several gigabytes of useful data. As a solution, the Round Robin Database Tool (RRDTool) was developed by Tobi Oetiker which creates a database of fixed size for data storage [65]. The way in which the databases created with RRDTool maintain their fixed size is by reducing the granularity of the oldest data items. This method of storage therefore assumes that the oldest data is least relevant to a user. As the majority of monitoring is concerned with more recent traits of the data, long term storage may be held at intervals which are summarised according to rule specifications made by the user. RRDTool has been adopted by a number of other monitoring systems as a means of reducing the size of logging databases, the time taken to query them and the ability to quickly visualise the stored data using a pre-built front-end [1, 3, 5, 51]. The flexible nature of the RRDTool repositories makes it a suitable choice for high performance systems such as Clusters and Grids [51].

While time series monitoring systems are a good mechanism for quickly investigating problems within a distributed environment, they merely only act as a means of documenting observations which have been made by an external entity. Any interpretation or reasoning is however the responsibility of the systems administrator, such as identifying and associating a cause event within the evidence which has been ascertained. The evidence which has been gathered will not warn of errors or unusual behavior, nor is it currently possible to be made aware of the functionality a system was executing - this must all be inferred by a user. Similarly, dependant on the particular metric set which is being used for monitoring, there may not be strong enough evidence for an administrator to make a judgement on the cause of an event.

2.4.3 Monitoring Standards

A variety of standards exist which are noteworthy in relation to achieving context capture. IBM has developed a standard to describe events used in enterprise management and business applications, to provide a common format which can be implemented in a variety of systems [66]. The common base event is included in the Web Services Distributed Management (WSDM) specification as a means of representing that information which is common across all events and should be captured. The OASIS Group have produced a Web Services specification for definition/structure of additional information pertaining to a service's execution, which is designed to facilitate organisation and sharing [49]. It is suggested that context may be propagated during application message transmission through use of SOAP message headers, but provides no means of being able to capture contexts for later queries by stakeholders. An additional service element is given as a means of determining the current status of a service activity, with a limited number of status types and transitions between them. Where use of services in processes are known, the activity types should always indicate that a service status is active between request and response messages - revealing little more than what is already known by a user. Although there exists attempts to implement it ‡‡ the standard has not been widely adopted in provenance environments due to the decision of what context should contain having already been made, rather than decided by each projects application domain.

2.4.4 Web Service Monitoring

In service oriented systems, service providers will often co-locate custom monitoring solutions with their services in order to satisfy their particular monitoring needs. Generally, these tools provide information at a far greater level of granularity than is necessary for those clients which make use of the services. However, they provide a useful way to determine if the actors which comprise a system have altered in any area across the period that they execute their functionality. Solutions have been developed in order to determine performance bottlenecks, such as message passing events [6], but metrics able to be observed in this manner are few. The IBM data collector (now part of the Tivoli Composite Application Manager for SOA) captures log data for Web Services by intercepting and instrumenting

‡‡http://sblim.wiki.sourceforge.net/

Web Service requests [2]. This is used with another tool called the Web Service Navigator which allows the execution of a set of services to be represented visually, in the hope of better understanding the behavior of the services [70]. Determining the cause for a particular actor's behavior requires administrators of these services to wrap them with tools to capture behavior, or at least be aware of the functionality which is within them, and to be able to recognise the relationships which exist between the changes in state and the process which is executed across the system. This is not a trivial task and requires significant effort by developers to expose these relationships. Also, such knowledge is not always immediately available and may require analysis after execution of services has completed. The method of monitoring services is intended for systems developers themselves, rather than clients who may have use cases for context.

2.4.5 Grid Monitoring Architecture

The Open Grid Forum (OGF) has presented an architecture for the capture of monitoring information in Grid systems known as the Grid Monitoring Architecture [81] (GMA). Although not a standard, one of its key aims was to suggest an architecture which would achieve creation of inter-operable tools for the Grid. As no single solution is given by OGF, it is suggested that poor implementations may be created and therefore suggestions for implementation are given (scalability, security etc). The main components in the GMA are *producers* and *consumers* of events of interest along with a *directory service* for registration (of producers and consumers), publication and searching. The directory service operates much like a registry in a service oriented architecture, registering producer and consumer locations and publicising them for later client query. All the monitored information represented in GMA are events. In a typical scenario, event data (available from multiple sources) will be collected by sensors and managed by a producer. The directory service is aware of the events due to previous registration of the producer with it. The directory service is searched by multiple consumer types who may register to subscribe to real-time monitoring updates from producers. The producer then sends event data to the consumer, who may carry out actions on other remote hosts. (Several implementations of GMA exist [11, 17, 21], each facing criticism such as being unable to cope with the volume and frequency of data updates or having been specifically created for the needs of in-house projects.) Where events being monitored are of critical importance to understanding the context of a complete process, losing any record of event occurrence may result in incorrect hypotheses being made. In order to ensure that this does not occur, all information which holds a relationship with a process should be reasoned over if it is available.

2.4.6 Monitoring and Discovery System

The Monitoring and Discovery Service (MDS) is the information services component of the Globus Toolkit 4.0, developed by the Globus Alliance [4]. It provides Web Services which monitor and discover resources in a Grid. Set across two different types of service, an *index service* which performs data collection and a subscription and/or query interface, and a *trigger service* which performs data collection and can be configured with actions to be carried out on particular pre-conditions being met. A third (as yet unimplemented) *archive service* is planned to allow temporal queries of data from an index service. The performance of a previous version of the MDS has been tested and proven to be scalable when data is cached within the system [91].

2.4.7 Relationships in Context

The use of performance data to obtain insights into the relationship between an application, and hardware/software has previously been explored [80], enabling automatic model generation through performance analysis. Within job-based execution environments, work has been performed to enable provenance recording with a minimal level of system intrusion [73]. Automatic instrumentation of such applications with performance monitoring code is possible due to direct availability of implementation. The trade-offs between the level of intrusion to both the application system and user, necessary to capture adequate provenance information has previously been likened to a cube [73] where intrusion to the system and user are modeled on the x and y axis and the amount of available information on the z axis. We have also previously discussed the other types of relationships which might exist between other factors in contextual information [87]. The most desirable system is described as one with no intrusion to system or user, but providing all information about the two. In service oriented systems instrumentation of actors with particular metrics decided by a client may not be possible, due to their loosely-coupled interaction with the querying actor. The level of intrusion which is possible in such environments is therefore minimal, and as such so is the level of information able to be captured. An alternative to these systems could be made through exploring how service oriented systems (where direct knowledge of implementation may be unknown) may record assertions of context using resources which are not necessarily part of the application system, alongside documentation of service interaction. The combined use of such assertions is possible in two ways, understanding how an actor performed within the context of an interaction and secondly understanding what an actor was doing when a particular performance pattern occurred.

Causal Structures in Context

In monitoring context within a system, we assume that the metrics which are chosen to be observed over time are good indicators of the effects of a given cause, allowing the documentation of a causal relationship between an effect and its cause. That is, an administrator should be able to determine the cause of the phenomenon which they observe in monitoring data from just viewing the monitoring data. In reality, the associations which administrators make are based on more than just a knowledge of such phenomenon, and draws from their knowledge and experience which they have about the underlying system to infer relationships which hold between observations and their given causes. They may have a great understanding of the systems and software which operate upon them, therefore are able to offer insights which a less knowledgeable administrator may not do.

Constructing causal structures from monitoring data requires that some sort of training is initially performed to ensure that future classifications of events are based on what is observed in the past as well as any knowledge initially provided. It is possible to classify a given cause event through representing any knowledge as a decision tree, where each node in the tree represents some condition of a particular attribute which is monitored. The final leaves in such a tree would hence be the events which are assumed to be a cause of the attributes specified. Classification using traditional data mining algorithms in this manner does not take into account the temporal order of the observations which are provided as input. This means that parent nodes in the derived tree for leaves may be based on observations which have occurred after the leaf observation. This is an important failing of current data mining algorithms where classification is not based on temporal knowledge of the data. To classify in such a manner goes against our own intuition of cause and effect, and the only common rule of definition for causation in systems that a given effect cannot precede its cause [55]. Any data analysis of observations of state which operate on a decision tree as a classification mechanism need to carefully consider erroneous relationships in any mined structures that may be produced if temporal order of the data is not taken into account.

To counter such inappropriate classification Karami presents an algorithm to preserve important attributes for temporally ordered data [43]. The algorithm works by flattening consecutive records into a single record, attempting to serve in the discovery of relationships between temporally distant values. Using this conversion technique means it is possible to reuse existing data mining tools to extract the relationships between these distant values. Karimi makes use of the existing C4.5 decision tree algorithm [71] to demonstrate this technique with sets of temporally ordered data. As C4.5 requires a single attribute to use as the decision attribute within a record, Timesleuth [44], the software which Karami developed, iterates over each attribute in a record attempting to find the best decision attribute. A visual representation of a structure which appropriately represents the temporal order of cause-effect observations can be thought of as traditional decision tree where each child nodes time of observation is after that of its parent node.

While finding causal structures between events in this manner would be desirable for actors in a system, it requires those who reason over the contextual data to be aware of the events which potentially caused them. The open nature of many provenance recording environments, means knowledge of the events which occur upon a actor may be extremely limited, with little or no data made publicly available. Analysis therefore in this manner may not always be possible, especially so for clients.

2.4.8 The Black Box Problem

When making observations about services we assume that some level of information is publicly available from the actor to be recorded as evidence. Where many workflow based provenance systems are able to make observations, workflow actors are either local to the experimenter or within the experimenters control. We refer to such actors as *white box*, as almost all knowledge of their internal functionality is exposed to a provenance system. Such scenarios are not indicative of how use of resources within open systems are performed in the main. Numerous disparate resources may be used within a workflow, where knowledge of internal functionality is limited (grey box) or completely unknown (black box). This "black-box" problem has previously been highlighted elsewhere [87, 92], indicating that reliable collection of process documentation cannot be achieved until it is solved. Assertions therefore in such environments need to either: be made by the actors involved in the workflow themselves (rather than being asserted by the enactor system); be metadata elements agreed upon between the collection system and the actor. In the first case, assertions may include information which may not be understood when being reasoned over. Hypotheses generated as a result may draw less on this information and more upon the previous behavior of a given actor (indicated by possible interaction with other services). From the second scenario, to get actors to agree to expose extra metadata about how they operate would prove to be difficult in a system where very little is known about the actor itself. It is likely that a tradeoff between the granularity of data and intrusion level to an actor will remain [87].

2.4.9 Revisiting Key Concerns

- *Context Interpretation is difficult* - Simple log file inspection might be sufficient for understanding the situation of a single actor, but we have noted this is time consuming when interpreting vast quantities of data. Although mechanisms such as SEC exist to reduce the amount of overall data a systems administrator has to reason over, ultimately the interpretation is carried out as causal relationships between events are able to be recognised. This approach would therefore not be appropriate for a administrator who wishes to quickly enable their services with context recording capabilities. Although causal relationships could be found automatically (using an algorithm such as C4.5), this would still require that potential causes to be initially hand-picked, again requiring the attention of a administrator who has system knowledge.

- *Existing Solutions are Inappropriate for Context* - As we noted in section 2.3.8, there is no one solution that is capable of meeting the variety of demands that each application will have for context. Similarly, currently no means of monitoring is capable of solving these problems either. Generic data structures (such as CIM) and monitoring standards will not be sufficient for the variety of data types of the projects reviewed in section 2.3. The designer of an experiment (as in MyGrid) could not be represented in CIM and much of the other data captured within such a schema would most likely be redundant for each of the evaluated projects own use cases.

2.5 Summary

We have shown in this chapter a number of points of note for documentation of context derived from our analysis. Firstly, we observe that the problem of context is common throughout current provenance applications. Due to the broad number of applications to which it may be applied, a wide variety of terminology and solutions exist for its capture. Each of the applications reviewed has a tight coupling between their use cases and the types of data which need to be documented to satisfy them. No one monitoring solution has been universally adopted for context capture for this reason, with scientists choosing to implement their own solutions for its capture. Context has been proven as necessary to answer many provenance queries, providing greater detail for the causal relationships which exist between events in a process.

Monitoring systems however have not been implemented with a view to capturing causal information from scenarios with provenance requirements, instead storing historic details of the same variable observations made at consistent intervals for an actor. Such information potentially could allow service administrators to understand how their actors, in a particular state, came to be in that state, and would provide a workflow operator with an understanding of the context under which assertions were made. Where documentation of interaction does not suffice in answering provenance related queries therefore, it may be possible that use of contextual information will do.

Automating context capture as much as possible is extremely desirable in provenance applications. Vistrails [18] has shown that by making the capture of provenance data entirely transparent to a user, it is possible to ease much of the burden of documentation which they may have endured previously. With open, actor-based systems, it is possible to instrument many of the components with provenance enabling libraries such as PReServ to provide this transparency. However, the black-box problem means that this can only be practiced where enough components have been instrumented and a complete chain of causality is able to be reconstructed after the fact.

Monitoring systems such as SEC illustrate that capture of context can be used to filter large collections of documentation based on specific criterion. Applied to repeated execution of a particular workflow for an application, this technique could also be used to filter information about single processes from large repositories of provenance information. This could present significant time saving benefits to users when investigating provenance evidence.

In the following chapter we continue by defining the terminology which will form the basis of modeling the notion of environmental context in distributed systems, motivated by the issues described here.

Chapter 3

Defining the Context of a Process

We have already shown in Chapter 2 that a large variety of solutions to the documentation of specific contexts for processes in distributed systems exist. We consider now that actors involved in such processes (the entities which work together to achieve a result), may progress through multiple differing 'states' during and as a result of their involvement as part of a process. These states may be documented to form part of a record of context. Our work builds upon the concepts presented in [32, 33] which describe how process may be documented in service based architectures. We present a collection of definitions which model the notions of *process* and *context* as distinct, separate concepts which both in part contribute to answering queries of the provenance of a data item. We propose a formal definition for describing context of actions in a process along with how changes to each of the distinct contexts are able to be measured and represented. The resulting definitions will form the basis for a structured model for documentation of context in distributed systems which follow in Chapter 4.

3.1 Introduction

Our work is based heavily on the definitions presented in [32,33] for documenting the process which led to data items in a service oriented system. Most of note in these works are the definition for the provenance of a piece of data as a process, how provenance is represented and stored by a third party. This definition contrasts with other definitions for provenance which treat it as a particular data type or set of data types which are able to be recorded. By distinguishing the information collected about the interactions which occur between actors and any additional data which is able to be collected upon an actor itself (such as context or relationships), a very versatile model is presented.

The additional value which variables recorded upon an actor provide to an overall provenance model over and above the core process description has not been considered throughout provenance work to date. This is despite providing valuable information on an actors behavior during execution of a process. We define the state of each of these actors to be represented by variables which are able to be recorded at an actor at a given point in time. This recording may be by a third party or the actor themselves. Therefore, we consider that an actor's state is a unique configuration of variables on an actor which can be determined by analysis of those variables able to be recorded during process execution. As the value of these variables fluctuate, so too does the states which the actor is in, which may or may not be documented. As we showed in section 2.3.8, alternative solutions exist to document those actions which comprise a process and also domain specific representations of context. However, lack of structure within the assertions of context are common, often making it only useful to the domain which originally created it. We have previously examined solutions to provenance in open systems which rely on instrumentation of actors to capture documentation. However, more realistically, exposure of provenance in this manner may not always be possible resulting in limited information from so called 'black-box' services. Where experimenters are restricted to the information provided by an administrator, determining how to best make use of the information which is publicly available for an actor is important as this is their only information source. If there is a greater number of variables capable of being recorded upon an actor than is exposed within actor state, then these should be captured as part of assertions of state to enable the maximum amount of reasoning with them. Depending on how descriptive such information is, an actor could progress through many states during process execution which may not all be relevant to the analysis it is being used within. A possible solution involves an administrator of an actor being able to filter results to exclusively include variables they are most interested in.

3.2 The Brain Atlas Experiment

We introduce an example process which will be used as a running example throughout this chapter, shown in figure 3.2. This workflow is used to create population-based brain atlases

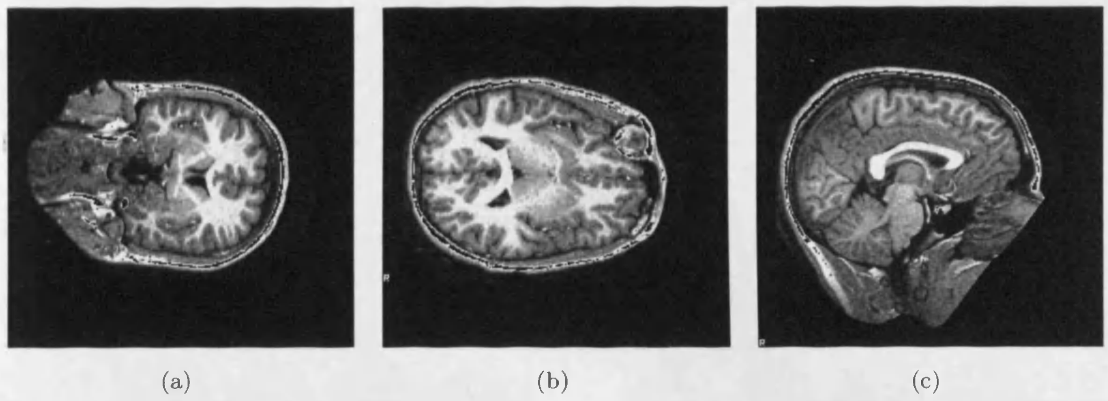


Figure 3.1: Output from the provenance challenge workflow across x, y and z axis

from high resolution anatomical data from the Functional Magnetic Resonance Imaging (fMRI) Data Center* and was used as an example at both the first [62] and second provenance challenges [53]. These challenges were constructed to achieve an understanding of the variety of alternative mechanisms and tools used in process documentation capture, as well as the scope of the topic of provenance. The image data has been published in the fMRI Data Center [41].

We concentrate here on the final part of the workflow, which converts an averaged brain image (determined by averaging the intensities of multiple MRI scans) gathered from a collection of high-resolution anatomical data into graphics files showing slices of the brain, such as those in figure 3.1. The provenance challenge workflow did not focus on a particular data representation or implementation of this process. However, for our representation of the workflow this is comprised of two web services, *slicer* and *convert*, shown in figure 3.3 where each service is represented by a box and the data items which are passed between the services as circles. The slicer service takes two data items, the averaged brain image (atlas image) and metadata about that image (atlas header) giving as output an atlas slice across a particular dimension (x, y, z). The atlas data set slice is then able to be converted using the convert service into a graphical image. Both services pass data to and from a third client entity which controls the entire workflow. We consider that the execution of each service does not yield an instantaneous response, i.e. it is possible to make a number of observations about the system as the service is being executed. We refer to each of the entities involved in the workflow as an *actor* and their act of declaring that something is true as an *assertion*. Assertions are able to be recorded about the interactions between the

*http://www.fmridc.org/

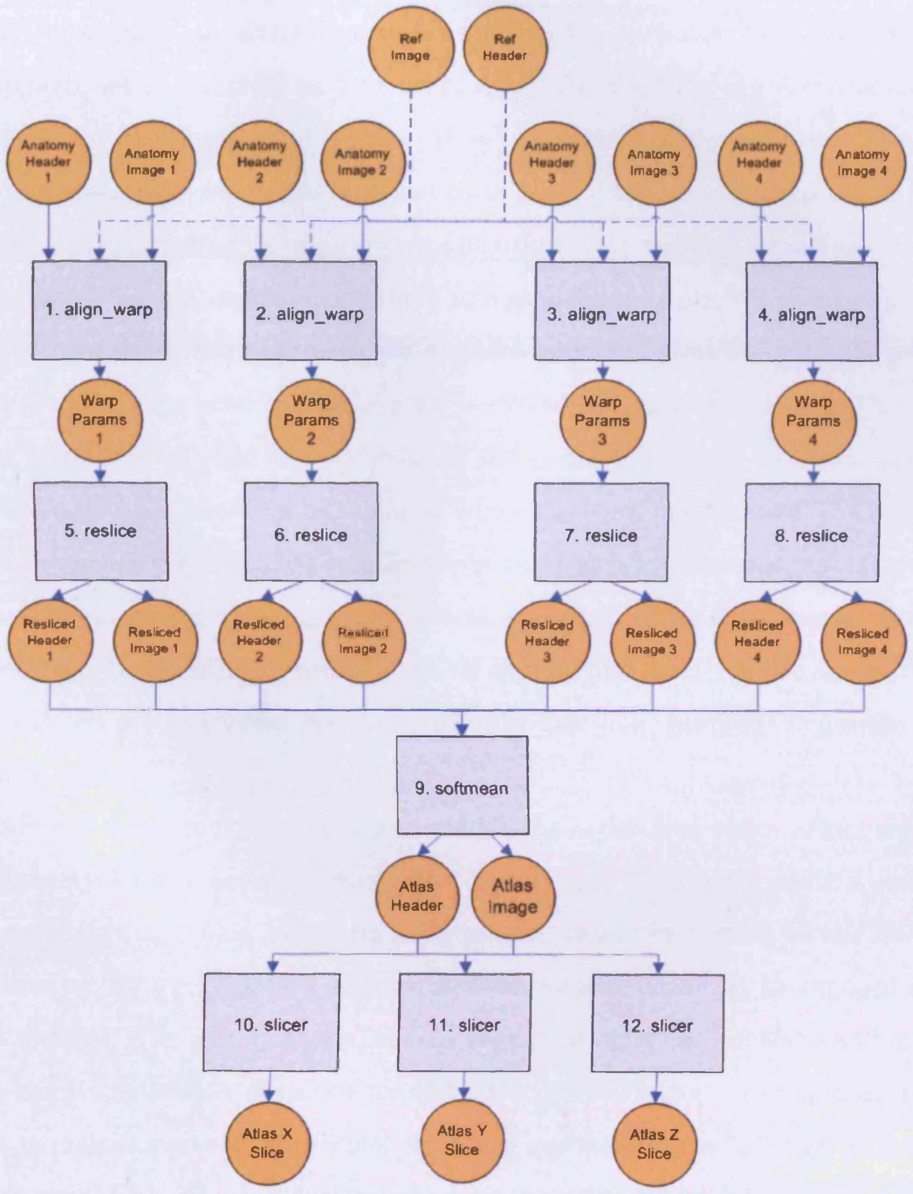


Figure 3.2: The provenance challenge process

actors involved, as well as a number of observations made by actors during those interactions. These observations may be comprised of information captured from monitoring tools co-located with actors (located upon the same host system), as well as information which may be captured by a tool created by a user (i.e. instrumented from actors themselves, version information for services, accuracy of records, etc). For the purposes of our example, we refer to this complete set of variables as β for each actor. An assertion representing the current configuration of β would represent all the values of variables in β at the time at which it is captured. All assertions which are recorded form part of the *process documentation* which describes what has occurred during process execution. It is possible for all assertions to be sent by the respective actors who made them to a *provenance store*, which is a repository for process documentation. Each actor in the scenario may communicate with the provenance store, in order that each actor's view of what occurred during invocation may be captured. An actor's assertions of what has occurred do not necessarily agree with one another and in such cases a decision needs to be made on which evidence can be used. This means that a third party service for instance could lie about their past interactions, as its open nature means users of that service have no administrative control of what it asserts. However, as each of the actors in our scenario are trusted, we assume that assertions do agree. Reasoning over all assertions are performed post invocation by querying recorded documentation from a provenance store.

We assume two stakeholders in the scenario who have different views. The *administrator* who has an unrestricted access to the actor decides how it is configured to capture and expose documentation. This would be a technician who is in control of the hardware or host environment for a service and decides those variables that might be exposed as part of context recordings. The *user* of the actor triggers execution of each of them and is therefore subject to the configuration decisions made by the administrator. If an administrator has chosen not to include particular variables as part of a monitoring installation for any reason, their values cannot be recorded during experiments and will later be unable to be queried from a provenance store. In our given scenario a user would be a scientist whom executes the entire process and collects its results from within their workflow client. We assume a worst case scenario for accessing information about an actor, but the approach we adopt can also be used for situations where access may be more relaxed. A scientist may, for example, be both a user and administrator for a service and therefore can make access and

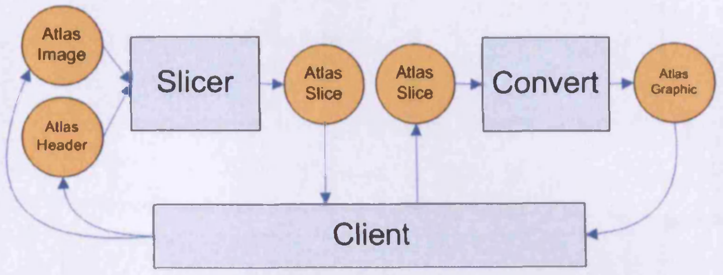


Figure 3.3: Example brain atlas imaging workflow

configuration decisions based on their own experimental requirements.

3.3 Motivating Requirements

We begin by clarifying our terminology and approach for provenance based upon our own observations of existing systems. Earlier in section 1.2 we noted the concept of provenance as a question asked to instill user confidence in the process used to create some data. We define an *action* to be the execution of the functionality made available by an actor, which is able to be invoked through well defined points of communication, such as through receiving or responding to messages. We use the word 'action' as opposed to 'function' as it implies a task being executed, rather than described or defined. A *process* is seen as a sequence of *actions* which are conducted in order to achieve some combined action. We adopt an actor-based model for representation of these processes, where processes are comprised of collections of *actors* each of whom is able to execute some functionality. If we consider our example in section 3.2, it is possible to represent each of the functions within the application as an actor and the complete process as a sequence of executing each of these actor's functionality. This actor model accurately represents the service-based representation which has been commonly adopted for provenance environments to date.

3.3.1 Documenting Process without Context

Where actor based systems are used in execution of a process, provenance requirements generally focus on building tools to document the interactions which occurred between components. We demonstrate a current approach to capture provenance for our scenario by applying the PreServ model for process documentation [37] to the Brain Atlas example.

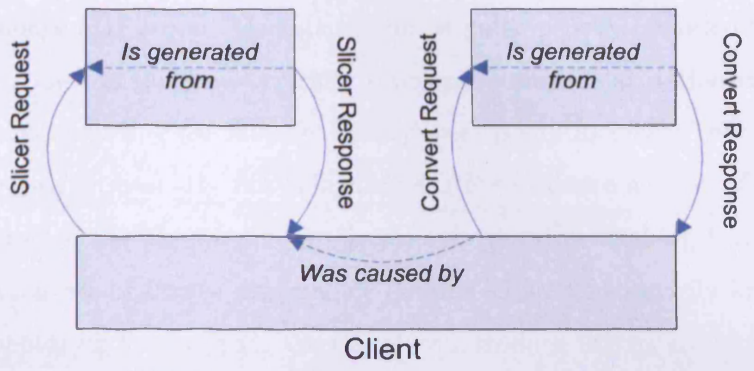


Figure 3.4: Causal dependencies as documented with PReServ

Using PReServ, the client and both services would each document interactions independently from the provenance store, according to those interactions they had observed using interaction assertions, possibly through just documenting message content in its entirety. Relationships would be expressed between each message to indicate where causal connections applied (i.e where one or more events or data items are a cause of one event or data item). Given these assertions were recorded to a universally agreed provenance store, any actor would then be able to query the entire selection of causal connections observed during processes execution. With this information, it is possible to reconstruct the actions which were carried out, the order in which they were carried out and the input or output data which were exchanged. These causal connections are common across applications with provenance requirements, meaning that their capture is often catered for in the first instance in systems built to document provenance. Using interaction and relationship assertions for our example means that a connection would exist between each interaction event, relating its input to its output and a further connection would link one interaction event to another, shown in figure 3.4. This makes it possible to identify a single trace within documentation for each process, by querying the documentation recorded by each actor. Internal information assertions in PReServ allow for further detail to be contributed by actors by describing the internal features of an actor immediately before or after an interaction occurs.

Process documentation is a type of *evidence* recorded about processes to allow provenance queries to be answered at a later date. Given a set of actors which have been enabled to capture it, it may be recorded by each of those actors who are involved in a particular process. We also adopt these views; that provenance is seen as a question answered by query of evidence and process documentation is preemptively recorded to answer these later queries. As each actor may typically represent a particular action in a sequence of steps for a process, recording the step that every actor performs means that it is possible to document the entire process. By not collecting context evidence as part of documentation describing actions, we enable maximum queries over the two. Making these separation of concerns allows answer of future provenance queries without necessarily knowing them at the time documentation is recorded. We therefore introduce this as an initial requirement.

Requirement 3.1. Evidence for actions in process documentation should be recorded separately from the circumstances for those actions.

PReServ allows assertions of actor state to be made about a process, but its applications tend to always consider assertions which may only be recorded at the instants when the actor triggered recording. Whilst it is possible to trace the interaction events that occur, the time intervals over which an actor is performing functionality are not well described. If for example the slicer and convert services each take 20 minutes to execute, then only documenting observations when messages are exchanged is not completely representative of the entire execution interval. Such an approach provides very little information on the systems behaviour over these intervals. Previously actors have been provenance enabled through wrapping sets of libraries or services with classes to trigger process documentation capture. It is therefore possible for assertions documenting causal connections to be easily captured when functionality is triggered by a client. In figure 3.5 we show an example of a clients interactions with a service over time. The time t_2-t_1 shows the lifetime interval of the service actor. In applications which have monitoring systems employed, it may be possible to observe variables of interest more often than at just these two instants in time, and values may fluctuate over the course of that actor's operation (such as time series values). It is important that this information is captured as it documents the period of activity of an actor and could be important for answering provenance queries.

Requirement 3.2. The provenance of an actor should include values of an actor's state variables at instants other than when the actor triggered recording.

We take the view that a record of context is a representation of a collection of properties able to be measured by an actor. These properties may alter over time and therefore may change throughout the time a process is executed. As with discrete points in a time series,

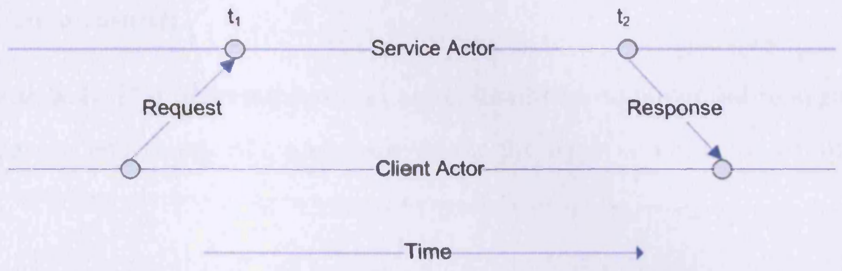


Figure 3.5: Actor invocation from a client does not imply an immediate response

measurements are assumed to be able to be made at regular intervals (through use of a daemon, for instance). By measuring a collection of these properties it is therefore possible to determine the points at which a particular context has changed. Context therefore would be a *dynamic* summary of an actor's properties which would be capable of capturing more than just a single value for any measured variable. If captured in this manner, subsequent queries of provenance would be able to include queries of patterns observed over the periods actions were executed, rather than single values. For systems which have actions which take a long time to complete, this approach is more favourable to recording single values for metrics, as it would ultimately result in more information being recorded over these periods which can be used for analysis. An alternative to recording all data is to summarise this it by averaging observed values, providing a better solution than an arbitrary point chosen for a variable during the time the actor executes.

Requirement 3.3. The provenance of an actor should include how an actor's state variables change over the period the actor performs an action, not just a set of static values of measured variables.

The challenge described in our example scenario demonstrates an attempt at how different models may be used to represent a single workflow. Typically, once the sequence of actions which make up a workflow have been agreed, that sequence may be performed multiple times in order to investigate how different settings may effect overall results. Whilst the documented actions will not change for the same workflow, the parameters used as input and the environment in which those actions are executed is likely to. Therefore, in order to be able to compare the context recorded whilst each action is executed, it would be desirable that it is documented in such a way that differences between contexts for the same action are measured.

Requirement 3.4. The provenance of an actor should be documented in such a way as to enable subsequent collections of state observed for the same action to be compared against one another.

Relationships can exist between events which may be observed in monitoring information. A significant body of work is available which focuses on deriving these relationships through trial and error experimentation [58, 77, 82]. A provenance system captures structured documentation concerning a process realised within a computer system. Such documentation is only recorded as a process is carried out and requires modification to the actor implementation in order to instrument it. However, as we have discussed in section 2.4, further contextual documentation is often made about the process in the form of monitoring information of the actors execution environment. These values are captured as collections of time series variables which are able to demonstrate the effect of some functionality which is invoked, but as both provenance and monitoring systems exist independently of one another, relationships which may occur between the two are often overlooked. An administrator may observe that an effect, documented as a peak in memory usage for example, is attributed to invocation of a service they provide. This causal relationship between the two pieces of information may be known by the administrator, but as instrumentation to capture causal relationships does not exist upon an actor, this is extremely difficult to assert and subsequently verify. If these two pieces of information are however disregarded as documentation sources by an actor, any causal relationships which could be discovered at a later date will be lost. An alternative approach would be to collect the information observed over the course of the process which may later be interpreted by a more knowledgeable actor, even if relationships were unknown. In this manner, relationships could be investigated post observation once such relations are more fully understood. As observed in section 2.4.9, a limitation with this approach would be that if no summary of observed data were made, the volume of data created could be high - making it difficult to interpret.

Requirement 3.5. The provenance of an actor should provide information relevant to the history of an event even when causal relationships which apply to it are unknown.

3.4 Context Use Cases

We now proceed by defining a number of different use cases where documented context observations are necessary to form hypotheses of events which have occurred. Each of these use cases demonstrate example usage of both documentation of process and context, which were not part of the original challenge. As these cases are based upon a workflow well established in the provenance community, they are able to be used as a direct comparison of the work already presented in the challenge which does not document context. The nature of the workflow in general is also demonstrative of the techniques used in the variety of processes performed in scientific applications. With each definition, we begin by describing the use case based upon our earlier original example based on the fMRI workflow in Section 3.2.

3.4.1 Context Analysis Using Action

Use Case 1. *The administrator of the convert service would like to understand why a particular state has been entered more often in the last day.*

Use Case 2. *A user of the workflow would like to understand how a particular actor arrived at a documented state.*

We refer here to the notion of actor state introduced earlier in section 3.1, that state reflects a unique configuration of variables on an actor - for which we give a more formal definition later in this chapter. If an actor is observed to have been in a state that is undesired, finding the root cause of it will enable an experimenter or administrator to take measures to avoid it in the future. We would like to describe a system where both action and context are documented consistently, from which it is possible to determine why a particular context may have been observed. Using a history of how past actions have caused particular transitions between states, we can determine the cause event which gave rise to a specific state. Query of process documentation therefore would allow actor owners to understand how a particular actor state came about given some initial starting state, which will prove useful when attempting to assess the root cause of any states which indicate undesired actor properties. Conversely, analysis of public provenance repositories should allow users to understand how previous actions have affected the actors states also. That is, it should be possible to inspect both evidence of context/process and determine if an observed action was the cause of an observation of state. If a service creates a number of images for example, was that action the sole reason why available disk space was reduced on execution?

3.4.2 Comparison of Past Actions

Use Case 3. *A user of the fMRI workflow would like to perform multiple invocations of the workflow and draw comparisons in the observations of state made for actors involved.*

Much scientific experimentation relies on varying the configuration parameters of actors to determine the differences which may occur in the processes which they are a part of. Such experimentation means that the same processes are executed many times as a result. Where actions in processes may be executed multiple times and each of the steps that they represent may be compared using provenance capture tools, so too should the context in which those actions are executed. As per our requirements, it should be possible to record the state in which an action was executed, and draw comparisons with the state recorded when the same action is executed again. This allows a distance measure to be calculated based upon the situation in which each of those actions occurred.

3.4.3 Comparison of Past Processes

Use Case 4. *The experimenter would like to ensure that differences in the execution time of each process are minimal over multiple invocations of the workflow.*

Comparison of a process is based upon comparisons of every one of the states in which an action is performed. These are then able to be compared against the states measured for the same actions in a second process. Many executions of the same functionality could yield a large amount of process documentation, something which is commonly performed in workflow environments. Being able to filter this information to those processes which have properties within a particular distance rating would enable swift navigation of these records, and facilitate easy comparison.

3.4.4 Future Prediction of Context

Use Case 5. *An experimenter would like to determine the most likely future conditions under which an actor will be operating.*

Given that an actor's state is measured (as its context), it is possible to predict how future events may affect that state. Based upon our notions of transition (described in section 3.5.2), it is possible to determine which events have caused specific transitions between states upon an actor in the past. Using this, we can calculate the probability of particular transitions occurring again, and a future state may be based on observations to date. A scientist therefore would be able to determine the likely states for an actor within a process given some knowledge of likely events that may occur.

3.5 A Basis for Solution

The PreServ model of representing provenance is an excellent basis for the descriptions we give for context. PreServ has focused on solving many of the more general concepts prevalent within the subject of provenance including satisfying several characteristics necessary for high-quality documentation. These characteristics describe documentation that is factual, attributable, autonomously creatable, process oriented, immutable and finalisable [32].

Through use of the PreServ models separation of representation of actions in a process from those assertions of any internal information upon an actor, we solve the problem of modeling context separately from actions, which has been given in requirement 3.1. We also adopt the six characteristics for high-quality process documentation given above when representing actions. Use of PreServ also advocates use of an actor-oriented model, where contributions detailing the provenance of data can be made by all those actors involved in a process.

3.5.1 Outstanding Requirements

A number of other requirements are however left unsatisfied though a standard adoption of PreServ's model. Firstly, requirements 3.2 and 3.3 which imply that assertions of actor state are documented consistently, rather than at particular instants. Such documentation is possible through use of the PReServ model, by combining data which is collected over periods of time (time intervals) into a single assertion - but this approach to date has not yet been thoroughly explored. Documenting every single value as separate assertions is also possible with PReServ, but where variable values fluctuate frequently, communication overheads would prohibit its use. Requirement 3.4 also places restrictions on how the context of a process should be structured, namely that it should be possible to measure the similarity of two measurements of context. Such a structure would ideally be able to be applied to sequences of actions, or processes, to enable comparison of a complete process trace using context.

3.5.2 Context

The collection of information about a particular actor involved in a process has been shown to contribute to understanding the evolution of the current situation of that actor or to interpret the process documentation which was captured. Further to our review in section 2.1, we define *context* as the situation or setting in which an action was executed which has some bearing upon that action's outcome. Context is viewed as a key component of provenance and also adopts some of its key features. For example, we define context (as has already been done with provenance) as a query which is able to be answered from query of evidence. Evidence to satisfy context questions may also be recorded as process documentation and answered as a result of a more complicated provenance question. We say therefore that context is a sub-query of provenance.

Observations which describe context can include a variety of data types. The variables which comprise it however are all considered to be primitive data types (integer, string etc). We consider that such observations may be either *static* or *dynamic* types (i.e data remains the same or changes over the time an actor is available). Over a number of records for an actor, this means a particular variable measured will remain the same for every observation. Static data includes items such as service identity, name, owner, version, capability, etc. Such information is similar to that published by an actor to a registry service in a service oriented architecture. Dynamic data examples include service execution time, uptime, availability, memory usage, etc. Such information needs to be recorded by the environment hosting the actor, and may be made available on demand. The accuracy of such dynamic data is dependent on the type of measurement tools being used.

Definition 3.1. *The context of a process is the set of conditions which hold true as a process executes and have some bearing upon the process outcome.*

Context however has been shown to be application dependant. Trying to define the data types which will bear use throughout the many existing provenance environments is a task which is extremely difficult to satisfy. Our concern also is not with how actions have been chosen to be represented in these systems. As the notion of an action is commonly modelled throughout distributed components in order to determine how a process was composed, we merely make an assumption that action *is* documented. We proceed by defining terminology for context data types able to be recorded upon actors and then explore how this data may be of benefit when integrated into an existing process documentation system.

In order to support the concept that context is measurable and dynamic we introduce the idea of a transition between two contexts which are observed as following one another over time. We assume that the primary cause for any transition between observations of context is an instantaneous 'event' in our model. It may be the case that these events cannot be measured due to being local to that actor, but it is possible to observe their effects as a changes in variables recorded as context. If a single variable is monitored as context and remains at a constant value, transitions between contexts which follow one another may be missed. Such transitions might be able to be observed if a greater number of variables were captured as a context record. Recording a second variable for instance, may indicate that there is a change in its value for the same period of time as the first. A single variable therefore may not give an accurate picture of the context of an actor over a period of time, as a further variable may indicate a difference resulting in two different contexts. Collection of additional variables will contribute to better understanding the context of an action, but may not completely and thoroughly describe it. We first introduce the event types which occur and are assumed to be the trigger of these transitions.

Definition 3.2. *A primitive event is a happening of interest which occurs at a particular point in time.*

Definition 3.3. *A composite event is a sequence of primitive events.*

Definition 3.4. *A latent event is a primitive event which has not been observed, even though it may have occurred.*

In our example a primitive event could be a message being sent from the client actor to the slicer service as this is a solitary event which occurs only once. A composite event could be the atlas slice request and atlas graphic response messages exchanged between the client and the convert service, as these are two primitive events observed in sequence from each actor's point of view. A latent event may be that the slicer service used another service,

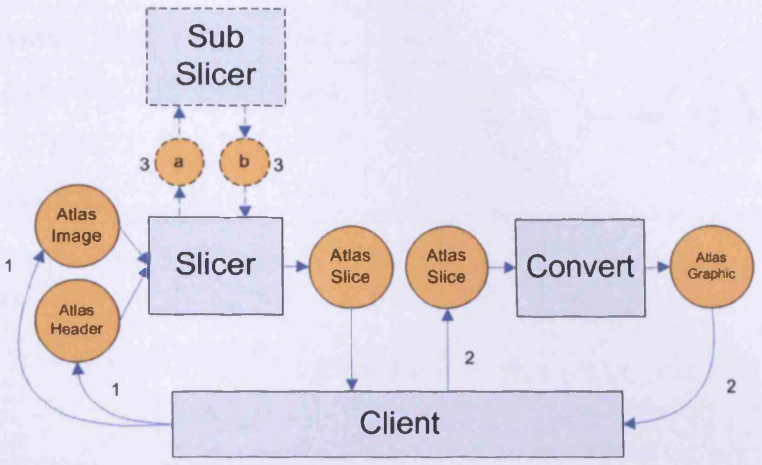


Figure 3.6: Example Event Types; 1: Primitive; 2: Composite; 3: Latent

sub slicer, in order to obtain its result, but does not declare this event to other actors or record it anywhere. Each of these types of event are shown in figure 3.6.

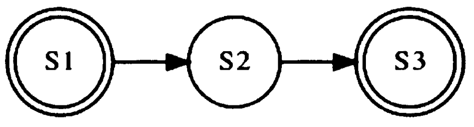
We suggest that a single representation can be made for the set of all variables able to be recorded upon an actor, known as *actor state*. An actor's state always indicates variable values for an actor at particular instance in time, however not all variables may be monitored as actor state. These variables, as discussed earlier are liable to fluctuate, so several state changes may occur during invocation of a service, thereby leading to multiple state transitions. We also propose that there also exist primitive, composite and latent states (i.e those states which occur at a particular time, are a sequence of primitive states and may not be observed, yet may have occurred respectively). It is possible for an actor to return to a previous state, i.e. the same configuration of variables is observed later in that actors lifetime. The notion of the *state of an actor* is independent of process, that is, we consider an actor has a state whether or not it is involved in a process. If states occur outside of a process they may not be categorised as latent, but may also provide important information on the behavior of actors once a process has been executed.

Definition 3.5. *The state of an actor is the set of all values associated with variables concerned with that actor at a particular time.*

In our example the state of each of the actors may fluctuate over the period which the workflow is executed, each representing a unique configuration for the variables represented by β. Therefore each of our actors may progress through multiple states over the course of a

State	Observation Time	*cpu*	*net*	*load*
s_1	1164277522	4.71	13084	2.56
s_2	1164282522	4.71	15698	2.56
s_3	1164287522	4.00	15698	2.56

(a) Derivation of unique states from variables, where β is unique for each state



(b) FSM of variable data

Figure 3.7

process invocation in which it plays a part. In figure 3.7(b) we show the slicer service having gone through each of the derived states $\{s_1, s_2, s_3\}$ during its execution. Monitoring systems provide a common source for similar sets of data which may not include all variables which are *able* to be observed. By including a number of extra variable values as elements of state over the time an action executed, we may reveal that several other states had occurred. Using such tools however means that states can be observed and asserted at a different time to when they actually occurred. For example, we may be able to collect observations from an actor as it is executing, but only assert them after the process has completed due to communication overheads they incur.

While capture of such information accurately describes how an actor may be performing during process execution, to consistently capture all possible variables on a system is not intuitive and quite unfeasible. We now consider *views* to provide a subset of the complete state information (β) to extract that data which is important to an end user. A view therefore essentially acts as a filter on the state of an actor, retrieving the values of data items of interest to a particular user and ignoring everything else. The administrator of a given service decides and defines the content of a view. This will typically be what is actually recorded as state, rather than all those available variables which are measured for it.

Definition 3.6. *A view of actor state is a subset of the set of the variables representing the state of an actor.*

Definition 3.7. *An observation of actor state is the action of recording of a view of actor state at a particular point in time.*

Referring back to our example again, the view of actor state here would be a subset of β for both services (we shall call this ψ, where $\psi \subset \beta$), whilst an observation of state would refer to the act of recording all the variables within that view (e.g. either to memory or file on disk). In table 3.7(a) we demonstrate how observation of ψ which is comprised of 3 variables (cpu usage, network traffic and load) corresponds to the states which are observed over the slicer services execution. It can be seen that each of the states holds a unique combination of the values of ψ. It is possible that a number of the states which the slicer service has gone through may be latent, if ψ does not contain a particular variable in β which distinguishes one state from the next. For instance, it may be possible to include an extra variable v within ψ which holds two different values between the time s_1 and s_2 were observed. Analysis of this would lead to a further state being recorded between these states.

3.5.3 State Observations

As an actor's state may change during process execution, describing and capturing all states may not be necessary and may even be counter-intuitive. Instead, using a description which refers to a "collection of states", as a single observation, reduces the states, grouping those states deemed similar together as one. For example, the variable *net* shown in figure 3.7(a) represents the number of packets being sent from the slicer service and were to frequently change between two values (or *oscillate*), it may not be important to report every transition of state. Instead the user of the information may prefer to group these states for the whole period into a single, collective observation representing the behavior for the entire period. We define a number of descriptions for state observations as follows:

Definition 3.8. *A primitive state observation is a single state instance.*

Definition 3.9. *A composite state observation is a collection of states.*

Definition 3.10. *A composite state interval observation is a sequence of primitive state observations each representing constant values for each measured variable over a time interval.*

There are also examples of data which may be recorded more frequently in some state observations than others. As tools which record observations may differ, so also may the frequency at which they are observed. Depending on the nature of the application, it may be necessary to record all observations, even though those data items comprising an observation may not have been observed at regular intervals. Additionally, there may be elements that need to be captured which do not necessarily 'fit' with the description of state. To illustrate such a situation consider an example where an experimenter desires that they are able to observe the overall execution time from one of the actor's point of view. This may be because the latency in communication between the client and services may seem unreasonably high to the actor. To assert this as an element of state would be inappropriate, due to the execution time constantly increasing. This would mean that an observation of execution time would be different each time it was made, causing a large number of different states to be observed. Analysis in such a situation would presumably yield little useful information due to no two states being the same. These types of data should only ever be asserted once by an actor, and require a separation from other elements which may be observed a number of times during process execution. Instead we define an element which should be ignored when analysing the variables which comprise state, but is still asserted as part of process documentation.

Definition 3.11. *An additional state element is a state variable which is asserted by an actor as actor state but is not part of a state observation.*

The time at which an observation of a value is made may not necessarily be the point at which it is analysed to form a state observation, or when it is asserted to storage. Where variable observations are made over a long period, analysis to determine state transitions may take a large amount of time, and waiting until a more appropriate time to conduct them may aid the performance of the process documentation system whilst the process is executing. As recording context should not impede how quickly the original process was carried out, it would be more appropriate in such situations to record evidence and later process it following execution. It may also be the case that an additional state element represents a static data type (such as the service's identity) and therefore only need be captured once over the entire lifetime of an actor.

3.5.4 The Relationship between Context and Process

The relationship which can be said to exist between context and process is temporal - that is, the data recorded as context is only captured during process execution. Whilst a more powerful relationship could exist, we assume the most basic one we can claim (a temporal one - describing a relationship in time) until a reasoning engine (human or otherwise) is used to make these more complex associations.

Definition 3.12. *The relationship which exists between a state transition observed upon an actor and a causal process which occurs over the same period of time is at its most basic level a temporal one.*

3.6 Evidence and Provenance Hypotheses

The collection of state information upon an actor can be interpreted in a variety of ways depending on the evidence which is known about a process. Important information may concern not only the functionality of actors and how they respond to events, but the hardware and software which is used within a process. Due to the huge variety of situations under which the state of actors may be evaluated, consideration of each of the levels of verboseness of process description needs to be accounted for.

Actor state can be recorded to support hypotheses about process documentation and vice versa. For example, an experimenter may choose to collect evidence about a process and support a hypothesis about that evidence through documenting actor state. Alternatively they may collect evidence about the context of an action and evaluate a hypothesis about it by looking at process documentation. Each may be collected in the same process documentation repository, or observations of state may be left within the repository within which they were originally recorded by the monitoring tool.

A logical interpretation of observations asserted by each of the actors who recorded evidence about a process will yield the best hypotheses about what has occurred. This is also true even if process documentation does not exist to explain the relationships asserted. Supporting evidence should be cited to explain why such a hypothesis is asserted in these scenarios. Forming hypotheses about the documentation relies on an inherent trust of the asserting actors. We do not consider the situation where actors may provide false information concerning their behavior as this problem is not exclusive to context and would exist with many approaches to answering provenance queries which we have already evaluated.

Given the variety of event types which we have defined in section 3.5.2, there are many cases where events may either be latent due to being unknown or poorly described. Events which are monitored may be used to infer what is assumed to have occurred in a particular scenario. To assert such information back as events which occurred would be incorrect, as the assertion is merely what is believed to be true, rather than an observation which has been made about the process. Therefore, we introduce a new definition to cater for asserting such hypotheses.

Definition 3.13. *A provenance hypothesis is an assertion about a process which is believed to have occurred based upon a body of available evidence.*

From our original example in section 3.2, a service invocation message being sent from the client to the slicer service will be documented differently by both actors. For the client, they observe that they have sent a request message, whilst the provider observes that they have received a request message. However, both pieces of evidence have a causal relationship with the single event that the message was sent. Identifying events in process documentation can lead to discovery of relationships which occur between actors. Analysis of all actor's assertions available from a process can yield evidence describing such events.

Events in our system are modeled as going through a number of different stages from their initial occurrence to finally forming a conclusive hypothesis about what has occurred. We believe there are 4 such stages; in the first *production* stage an event occurs upon a particular actor such as our slicer service receiving an invocation message. In the *observation* stage a monitoring tool will record that this event has occurred (possibly as part of an observation of state). These events and states are then correlated with one another in the *interpretation* stage in order to identify relationships which hold in the data, either by an automatic correlation mechanism (as with the C4.5 algorithm in section 2.4.7) or manually by a human actor (as with the majority of provenance environments, such as Kepler). In the optional *dissemination* and *verification* stages the causal hypothesis which has been calculated is published to all parties which desire to know what has occurred and then is verified by them to confirm that it is true, or they could conclude themselves when reasoning over the observations made. If these conclusions were to differ, then each party must decide which conclusion they deem more trustworthy.

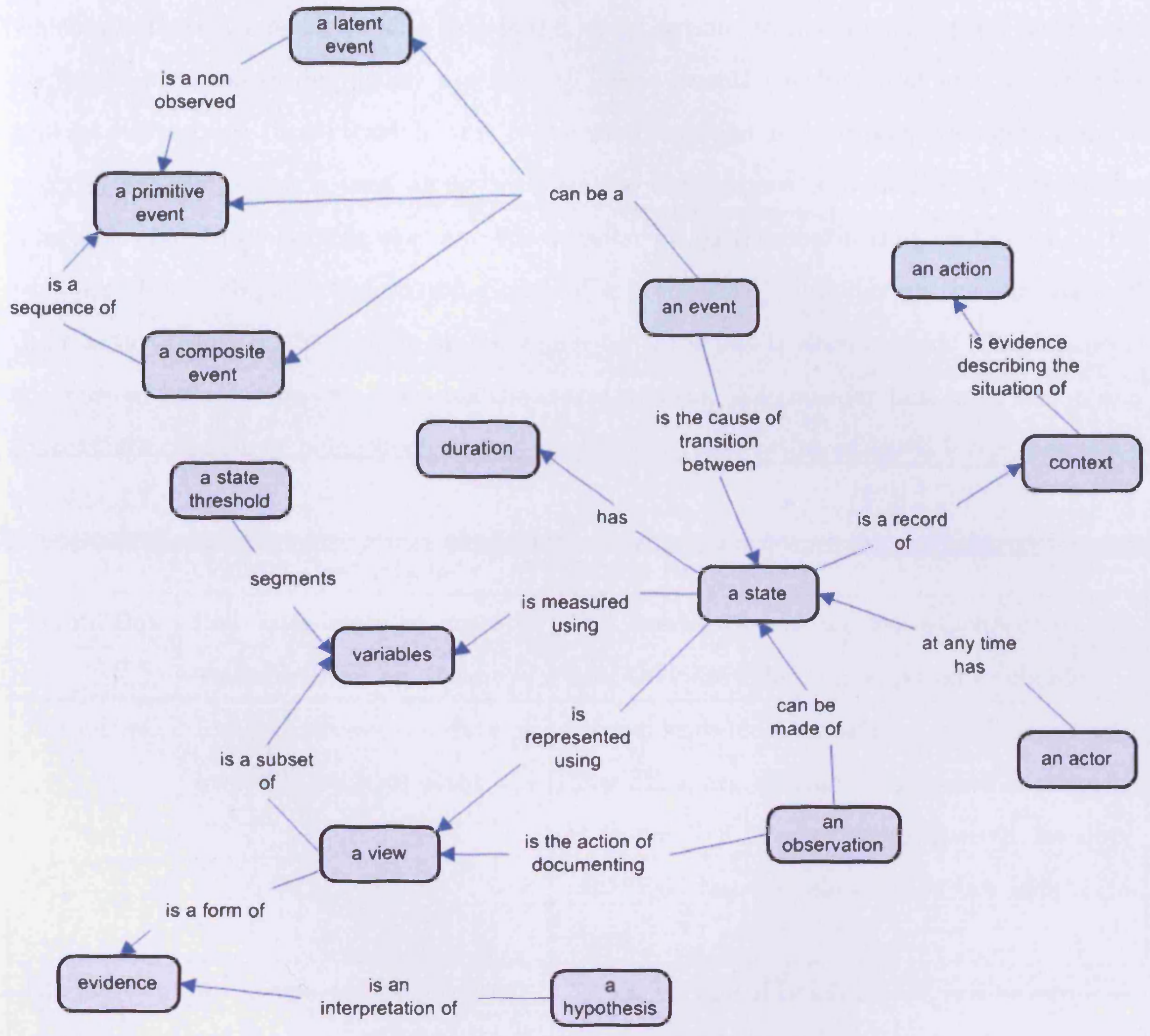


Figure 3.8: A concept map representing relationships between context terminology

We visualise the relationships between the terminology which has been presented so far through use of a concept map in figure 3.8.

3.7 Solving Use Cases within Restrictive Environments

Solutions to the earlier use cases we presented (page 50) depend on possible restrictions within the environment adopted to represent the process. We introduce a number of ways in which each use case may be solved making best use of information available when subject to these restrictions. The patterns are applicable to any actor based scenario, rather than just the one we present in this chapter. These patterns are based around two scenarios from which all others are derived. The first is the use of actions to interpret context, where any documentation concerning events are used to help a scientist to interpret how a particular context has arisen. In contrast to this is the use of context to interpret documentation of process, where context is used along with limited descriptions of action in an attempt to interpret what those actions may be. We consider for each scenario that we have an actor who performs a single action forming part of a process and that during the execution of these actions each of the unique states which an actor has is documented. These actions are invoked by an actor who executes the entire process. We consider that both action and context are capable of being documented on both of these actors at three levels, described in table 3.7.

	Context Restrictions	Action Restrictions
White Box	Full knowledge of state is available from an actor	Full knowledge of actions which occur and how they are related is exposed to clients.
Grey Box	Limited access to state is available from an actor	Partial knowledge of actions which occur and how they are related is exposed to clients. It is possible that relationships will be documented, but be latent from the view of a client.
Black Box	No access to state is available from an actor	No knowledge of relationships will be exposed by an actor to a client.

Table 3.1: Actor Access Level Descriptions

3.7.1 Using Actions to interpret Context

An ideal scenario for interpretation of state transitions is that both observations of state as well as process are collected. In such cases, we have complete collection of provenance information and are able to use the causal structure which is known to interpret observations of state which have been made. Typically, this information would be useful for actor owners to understand why their actors have performed in a particular way. Alternatively, as observations may only occur upon the actor in question, it is possible that there is no information gathered concerning state.

1. Causal Process Undefined - (Grey/Black Box) The causal information which is asserted is one sided, with only the client able to make assertions. State information is also only available for the client. We cannot make any assertions regarding state as access has been denied from repositories which contain state information by the actor owners. Any relationships concerning causal structure of the distributed process need to be made by the client. In a service based example, this would describe assertions of events indicating the service being invoked.

   The slicer and convert service administrators are unable to make available any observations of actor state which are able to be captured locally. This could be because they do not have access to their implementation and cannot modify them to capture state. Instead the experimenter has to capture observations local to the client actor. In order to ensure consistency across workflow runs, the experimenter defines a number of checkpoints at which messages were sent/recieved to/from the services are recorded. This allows the experimenter to determine whether the execution time of the service (as observed from the client) is as expected.

2. Causal Process Defined - (White Box) It is possible to understand each state of an actor in a process using known causal knowledge about that process. This can occur from alternative views, where access to particular information is available or restricted by the actor which produced it. We consider two users: *administrator* and *user*, who have such access permissions.

   (a) Administrator View - An administrator of an actor is able to determine why given states have been observed upon it, through inspecting collected process documentation and following the causal structure which is described. Ultimately, this documentation may reveal that a particular state (which is perhaps undesired), was caused by other actors due to a given pattern of interaction. For example, an administrator of an experiment is able to conclude that a given state upon an actor was related to a particular service invocation which in turn was related to a larger causal process.

   By looking at the process documentation which has been captured for it, the administrator is able to locate an assertion relating to the service being invoked and thereby identify a causal chain which is related to the assertion as a result.

The administrator concludes that the service has been involved in a larger causal process from the documentation describing it.

(b) User View - If a user has access to each of the service state repositories involved in a process, they are able to determine the set of states an actors has been in across the period that actor was invoked. This is dependent upon whether a given actor has exposed such information in order that a user is able to view it. The set of states allows a user to understand which states have arisen upon actors, during a process invocation and subsequently the differences in state between executions of the same process.

By reasoning over the monitoring data, the user is able to ascertain the unique states which were observed over the time the workflow was invoked. After several invocations, it is possible for the user to determine whether the behavior noted exhibits similarities across each invocation.

3.7.2 Using Context to interpret Actions

Complementary to the use of assertions of process to interpret state is the use of state information to interpret actions. This information is available within a provenance repository to be queried for process documentation.

1. Causal model is known - (White Box) If a relationship between events upon an actor is known, it is asserted to the actor's provenance store in order that on occurrence of the events an asserted relationship is able to be highlighted. Such relationships are able to be composed into hypotheses about a given pattern of interaction.

   Use Case 6. *A user of the slicer service desires to be made aware of all events related to the sequence of state changes on each actor which occur as they invoke the workflow.*

   Using known causal relationships from the provenance store, it is possible for the user to be informed of events which might otherwise not be highlighted.

2. Probabilistic model using states and events - (Black/Grey Box) Attempt to create a probabilistic model based on state and event knowledge. As observations of events occur, it is possible to construct a *probable* model of relationships between states using observations over time. From this, it would be possible to export a probability model

of the relationships between states, or indicate plausible causes of events with the most appropriate one being selected by a user, which could be asserted as a hypothesis:

Example 3.1. $p(e_2|e_1) = 0.45$, $p(e_3|e_1) = 0.55$

Here we describe the probability of events given that another event may have occurred. This is determined through event observation and primitive correlations which can be formed between events. In both of the above examples, knowledge of the specific relationship (i.e. Causal, Functional, Related to) between events may be unknown. We consider that the relationship is not just limited to expressing causal relationships only, with relationships able to be asserted without necessarily being described.

The user of the slicer and convert services has no information detailing related events to those which have occurred during execution of the fMRI workflow. In order to determine relationships between events, it is necessary for the user to build a probabilistic model of the most likely events to be observed as a result of other events occurring. A hypothesis, detailing the most likely sequence of events related to those observed, is able to be asserted as what is thought to have happened.

(a) Precise function definition - Where the relationship type between events is known through an explicit definition e.g. a causal relationship.

(b) Imprecise function definition - Where the relationship type between events is unknown, but it is known that a relationship between events exists.

3. Inferred causal model using monitoring - (Black/Grey Box) By tracking the way in which an actor performs over time and how previous events have affected system state, we can derive a causal structure between monitored events and states (based on temporal mechanisms). This causal model will become more indicative of the actual relationships at an actor as the body of causal evidence is improved. This is the model which is most relevant to the probabilistic approach we have introduced above. Whether these events are directly related or the relationship based upon other latent events is also a problem which needs to be solved.

   Through capturing previous event history and observations of system state, a model of the strongest causal relationships which exist between states and events is inferred. A hypothesis is able to be asserted for the information available.

3.8 Summary

We began this chapter by outlining motivating requirements and use cases for context capture based on an example scenario used in the provenance challenge. We chose to adopt terminology and a model which used an actor based approach to facilitate documentation capture. We showed that whilst some of these requirements were able to be satisfied by this model (requirement 3.1), others were not (requirements 3.2, 3.3, 3.4 and 3.5) and further investigation was prompted to better suit these original problems we introduced.

We continued by defining a number of terms for use in describing the context in which causal processes occur. These definitions do not depend on a particular architecture or implementation and therefore may be used in any actor based system. Each of them is based on aggregation of the context of actions for the same process executed multiple times. The results of these executions may then be used for analysis. We suggest that a record of context may only be used in support of a particular hypothesis once it has been aggregated together with other records, to form an average measure of normal conditions for an actor. A single record of context will be of no use on its own, even if records of action have been captured also, as context captured in the single instance could be unusual. Future and comparative analysis of context therefore requires that the processes are executed a number of times.

We proceed by using the definitions presented here as the basis for a model of environmental context which may then be applied to the domain of open, loosely coupled systems prevalent throughout provenance work to date.

Chapter 4

Modeling Context for Processes

Chapter 3 contained a discussion about the terminology appropriate for formalising actor state, and we believe there exist a variety of ways in which such terminology may be used to construct models representative of the changes of state and their associated causes that occur upon an actor within a distributed system. These include Hidden Markov Models, Directed Acyclic Graphs and Petri Nets. Rather than explore every type of model, we choose one model as an example - modeling through use of automaton theory [42], as a means of modeling the concepts introduced in Chapter 3 and present it in this chapter.

4.1 Motivation for a Model of the Context of Process in Provenance Systems

In our consumer example in Chapter 1, we demonstrated how it is possible that a person's decision making process is influenced by the detail they have of the situation or setting in which a set of actions take place. Similarly, for a scientist who conducts experiments in a problem solving environment, their thought process for determining what causes led to an experimental outcome may also change once they are aware of the environment of those activities, which contribute to generating that result. It may also be the case that the increased level of documentation about actions which were performed enable a scientist to provide further insight to the results.

Mechanisms to document situation do already exist - such as the variety of systems evaluated in section 2.4. However, we have also shown, due to their implementation outside of provenance systems, that much of the data available is not documented in a manner which is able to be queried as process documentation. Any relationships which do exist between the data in such systems cannot be found due to this separation. In such situations, interpretation based on both would have to be performed manually by the scientist through inspecting both sets of data and asserting relationships they think may occur.

Several domains have developed their own responses to capturing context, able to satisfy use cases for their own applications. As recent developments are leading toward more open, generic and shared models of provenance [32, 61], having a model of context which is able to be combined with these more generic models would be beneficial. We assert that the primary benefits of a formal method of modeling context will be as follows:

1. **Completeness** - Systems such as Virtual Data System have already argued the value of so called '*complete*' provenance. Whilst it is unknown what constitutes total completeness for these systems, it is known that by bringing context to a provenance enabled system, it will be possible to ask more complicated queries of the nature of that process. If provenance queries are unknown at the time documentation of a process is made, it follows that the complexity of those queries will also be unknown. If context exists for a system, but is left undocumented, those potential use cases may not be able to be satisfied.

2. **Structure** - Context for actions usually is documented by other software systems that are unable to answer provenance queries. This is because the collection of information which is captured is not recorded to detail relationships which occur in the data. Restructuring such information (that is readily available for use in provenance queries) would be beneficial, because it allows context information to be related to those actions that are documented in a provenance system (and hence the relationships have been asserted between them). Questions concerning the provenance of data items could then be answered through the structured information.

3. **Sharing** - Some contextual information may not be readily available for clients because, for instance, it is only available by the actor's particular network or deemed unimportant for clients. By recording it at a mutually agreed location, such as a provenance store, we allow any client who has access to the provenance store to be able to make use of it.

4. **Immutability** - Many of the monitoring systems reviewed in section 2.4 only keep the highest resolution data available for a restricted amount of time. Whilst this reduces resources necessary for extended periods of time, it also reduces the amount of information available over shorter periods, which may be more interesting to a user. The original data in such cases will typically be recorded at a lower granularity than possible in order to reduce the storage requirements necessary for it. If such data is however immediately captured in a provenance system, and cannot be altered after storage, queries over the specific periods a user has an interest in can be supported.

4.2 Automaton for Provenance

4.2.1 Context Model Alternatives

We first explore a variety of modeling techniques appropriate for context in order to support our choice of using a automaton model. Using Hidden Markov Models for instance, a model is constructed from observations which are able to be made of a system. These observations are assumed to be indicative of an underlying model made up of states and transitions. Using a Hidden Markov Model and a series of system observations, it is possible to determine the probability of a particular sequence of observations occurring or a likely state sequence for a observation sequence. In the black-box actors with which we are working, the relationships which hold between an observation and underlying state is difficult to determine, as there is no mechanism for determining the relationship or state. It is possible that an actor owner could divulge such information, but typically in an open system this will not be the case. A better model may be to consider only working with observations of a system in an attempt to predict future actor behavior. Another modeling technique, Petri nets, are well suited to modeling pre and post conditions for observations and concurrency in systems. We assume that no concurrent processes occur in our actors, due to the complexity such a model would create for a black-box actor. A black-box actor which was modelled with concurrency for example would experience difficulty in relating observations back to the particular process which caused them. Petri nets would therefore introduce a unnecessary complexity to our model and therefore are not a suitable choice. These assumptions do however have limitations, where multiple actors may be running simultaneously on a single host. Features observed in context could be incorrectly asserted as related to one actor, when they were in fact related to another.

4.2.2 The P-Automaton

We now introduce the p-automaton (or provenance automaton), our own mechanism of representing the evolution of an actor's current state given the distinct set of previous states an actor has been through over a period of time. We choose automaton over and above the other models considered in section 4.2.1 as it most appropriately represents how state may change over time for an actor. To minimise the complexity of such a model, we make the assumption we are attempting to model context for a single process upon a actor. We also assume an automaton which is event driven, i.e. it consumes events or messages as the characters/tokens which cause transition between states. As opposed to a traditional finite state automaton, which defines the set of tokens the machine can accept (and will always accept), our automaton is *observation based* and a reflection of the events and state transitions *which have already been observed in the past.* These automaton therefore do not model future events, although they may be used as a mechanism to predict them based on past behaviour. We refer to this representation of observations as a *provenance automaton* or *p-automata* as it is a description of the process which has led to the evolution of an actors current state. Such an automaton allows any observations of state to be represented in a formal manner to assist in interpretation of the observation data, instead of attempting to analyse large collections of statistical data manually from a wide variety of sources. Using this representation, we can quickly see if the actors' state has changed, by inspecting whether or not a set of transitions between states exist for an actor. We also show that it is possible to make comparisons between multiple executions of the same workflow using state observations, to determine if there is any similarity in the sequence of states which have been observed (over the same interval for each execution).

Formally a provenance automaton consists of the following:

- The input alphabet Σ, consisting of a number of *observed* events $\Sigma = \{e_0, e_1, e_2, ..., e_n\}$ where e_n is the last observed event.
- S, The non-empty set of finite *observed* states $S = \{s_0, s_1, s_2, ..., s_m\}$
- s_0, the initial state, an element within S.

- δ, the state transition function $\delta : S \times \Sigma \rightarrow S$
- F, the set of final states over the period observations are made, $F \subseteq S$

A provenance automaton is based on the set of all monitored events M, $M \subseteq \Sigma$, and all monitored states T, $T \subseteq S$

We can represent the complete automata with the 5-tuple description

$$A = \{S, \Sigma, \delta, s_0, F\}$$

As the automaton only represents those changes of state that have occurred in the past, the alphabet Σ and set of states S can vary if determined at a different point in time. That is, a greater variety of transitions between states are possible if observations are made at a later date for the same actor. The provenance automaton may be either deterministic (only one state returned by Σ for each event in Σ) or probabilistic (a set of states is returned by Σ, with a probability of occurrence, which we shall discuss further later in this chapter).

Temporally, the evolution of the state of an actor can be viewed as a single chain of states and transitions between each observation of state. This may be represented through a deterministic p-automata which only ever accepts a single event for a particular state. Each state may occur in this temporal chain more than once, that is, the conditions in order for a particular state to be caused may have occurred multiple times over a single observation period. Each of the transitions represented by δ between states are those functions which map from an initial state s_i in this chain to a subsequent state s_{i+1}, once the set of observed states have been temporally ordered. An alternative model (which occurs in practice) is that an actor may enter any number of states and return to those states also. This alternative model would be built through aggregation of the observations made for the same actors over a period of time, such as the same service being invoked multiple times.

From our fMRI example in Chapter 3, an automaton would represent a model of the states that each of the actors (*align warp*, *softmean*, *slicer* and *convert*) had previously been in and the events deemed to have caused transitions between them. Events could be messages exchanged between the actors, or those observed upon the actors through user created tools co-located with the actors. User-created tools monitor application specific metrics compared to generic monitoring systems (for example, quality of a given result as defined by the application). Each p-automaton could represent the complete evolution of a state space for a single actor (from when the actor was first deployed), e.g. the states observed in a single execution of the atlas workflow. By building an automaton over an extended period of time, it is possible to calculate the alternative final states each actor has previously ended in for an observation period when a particular sequence of events occurred. This information also would be used, to for instance, approximate the likely states of the align, slicer and convert services would be left in following communication if the client were ever to communicate with them in the future. Such a tool is a powerful mechanism for determining if a desired state will ever be reached by an actor, or taking preventative measures to avoid those states which are deemed undesirable by a user (for example a never-ending loop).

4.2.3 The Extended Transition Function

We borrow the concept of the extended transition function $\hat{\delta}$ of non-deterministic finite automata [42], to define a function for event driven p-automaton that takes a state q and an ordered set of events μ, and returns the set of states that the automaton could be in, if it starts in state q and processes the set of events μ. It is called 'extended' as it operates upon an ordered set of events rather than a single event as the transition function δ does.

BASIS: $\hat{\delta}(q, \epsilon) = \{q\}$. That is, if we start in state q without reading any events, the automata remains in state q.

INDUCTION: Suppose w is an event set of the form $\{x, e\}$, where e is the final observed event of w and x is the remaining set of events until e. Also suppose that $\hat{\delta}(q, x) = \{p_1, p_2, ..., p_k\}$. Let:

$$\bigcup_{i=1}^{k} \delta(p_i, e) = \{r_1, r_2, ..., r_m\}$$

i.e., we take the union of each of the set of events returned by the transition function δ starting in the state p_i and processing e (the final event of w), for every event in w. Then $\hat{\delta}(q, w) = \{r_1, r_2, ..., r_m\}$. We compute $\hat{\delta}(q, w)$ by first computing $\hat{\delta}(q, x)$, and then by following a transition from any of the states that is labeled e. An example non-deterministic finite state automaton is shown in figure 4.1.

Example: Let us use $\hat{\delta}$ to describe the processing of the set of ordered events $\{e_1, e_1, e_2, e_1, e_2\}$ by the automaton in figure 4.1.

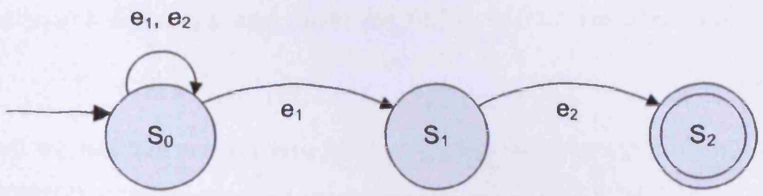


Figure 4.1: An example non-deterministic automaton

	e_1	e_2
s_0	s_0, s_1	s_0
s_1	$\emptyset$	s_2
s_2	$\emptyset$	$\emptyset$

Table 4.1: Transition table for the NFA shown in figure 4.1

1. $\hat{\delta}\{s_0, \epsilon\} = \{s_0\}$
2. $\hat{\delta}\{s_0, e_1\} = \delta\{s_0, e_1\} = \{s_0, s_1\}$
3. $\hat{\delta}\{s_0, \{e_1, e_1\}\} = \delta\{s_0, e_1\} \cup \delta\{s_1, e_1\} = \{s_0, s_1\} \cup \emptyset = \{s_0, s_1\}$
4. $\hat{\delta}\{s_0, \{e_1, e_1, e_2\}\} = \delta\{s_0, e_2\} \cup \delta\{s_1, e_2\} = \{s_0\} \cup \{s_2\} = \{s_0, s_2\}$
5. $\hat{\delta}\{s_0, \{e_1, e_1, e_2, e_1\}\} = \delta\{s_0, e_1\} \cup \delta\{s_2, e_1\} = \{s_0, s_1\} \cup \emptyset = \{s_0, s_1\}$
6. $\hat{\delta}\{s_0, \{e_1, e_1, e_2, e_1, e_2\}\} = \delta\{s_0, e_2\} \cup \delta\{s_1, e_2\} = \{s_0\} \cup \{s_2\} = \{s_0, s_2\}$

This set of ordered steps can be read as follows:

- In step 1 we determine the transition the p-automaton progresses to if starting in the state s_0 and the event ϵ occurs. In this case, the automaton remains in the state it was originally in.
- In step 2, we apply the transition function to s_0 (the resultant state from step 1) and e_1 (the final event in w). This returns the set of two possible states which the automaton may progress to, $\{s_0, s_1\}$.
- In step 3 we determine the states the p-automaton may progress to given the ordered sequence of events $\{e_1, e_1\}$. As we are already aware what the transition function returns for $\delta\{s_0, e_1\}$ (from step 2), we apply the transition function to each of the

states in the set $\delta\{s_0, e_1\}$ and take the union of the results. This returns the set $\{s_0, s_1\}$.

- In steps 4-6 we use the same method of applying the transition function δ as in step 3 to each of the elements returned in the previous step and returning the union of each of the result sets.

Had this been a deterministic automaton, we would not have had to take the union of δ applied to each state (and event observation) returned by $\hat{\delta}$ for all events observed prior to the current event. If we relate this to our original brain atlas workflow, each of the states the automaton progresses through would be associated with the configuration of variables which were observed upon the single actor, slicer. The events $\{e_1, e_2\}$ would correspond to those monitored at the slicer service, such as service request and response messages being sent and received. The extended transition function $\hat{\delta}$ would tell us the state that had previously been observed on slicer when the set of ordered events μ had occurred and slicer had started in a state q.

4.3 Characteristics of Provenance Automaton

The p-automaton representing an actor also has an associated event driven finite state machine for that same actor. Each p-automata has characteristics which make it different from its finite state machine, as explored in section 4.3.2.

4.3.1 Finite vs Provenance Automata

The core difference between a p-automaton and its finite state machine (FSM) is that the FSM for a system is representative of state changes for all possible (past and future) events. A p-automaton however is representative of observations made in the past. As a FSM is a way of modeling a system that may be in one of many finite states, it will always fully describe a system's behaviour. Describing a FSM for a system relies on us having all knowledge about a systems behaviour prior building a model for it. For actors in a system whose future behaviour is unknown, it cannot be modeled with a FSM.

A p-automaton may be constructed to describe past state transitions, when a set of observations for a system have been made to a particular point in time. However, depending on the observation interval which is chosen and how each state is described, the number of states (and transitions between them) will change. Further, for any given actor the number of states is unknown for the p-automaton representing it over the period of its entire (past and future) existence. It is however possible that the total number of all states is known, due to the choices which have been made for converting variable observations into states (this will be further explored later in this chapter).

4.3.2 Previously Observed Acceptance Criteria

Acceptance criteria for transition between states for a p-automaton is created from those observations which have actually occurred. The finite state machine however, describes those events that will always cause a change in state, which events will be accepted whilst in a particular state and the complete language (set of events which result in some final state) which it is capable of accepting. This means the FSM not only contains all the transitions, events and states from the p-automaton for the same actor, but also includes all future transitions, events and states as well. In the event-driven model we present for state, attempting to determine the FSM for an actor is extremely difficult due to the actor's future behavior being unknown. The events which cause state change in a given actor's finite state machine in the future may not necessarily be the same as those which have been monitored in the past and hence any transitions observed for an actor's p-automaton may not match those of its FSM.

Example 4.1. Consider the finite state machine with the following properties.

- The input alphabet Σ, consisting of a finite number of events $\Sigma = \{e_1, e_2, e_3, e_4\}$
- S, the non-empty set of finite states $S = \{s_0, s_1, s_2, s_3\}$
- s_0, the initial state, an element within S.
- δ, the state transition function $\delta : S \times \Sigma \rightarrow S$
- s_3, the final state for a given observation period

The FSM can be described with the 5-tuple $A = \{S, \Sigma, \delta, s_0, s_3\}$. This automaton can be seen in figure 4.2. A p-automaton represents the observations made upon an actor until a particular time. These observations capture the changes in the state of that actor during a time interval. We assume temporally ordered set of event observations which gives the following when applied to the extended transition function $\hat{\delta}$ (defined in section 4.2.3): $\hat{\delta}(s_0, \{e_1, e_3, e_4, e_2\}) = \{s_3\}$.

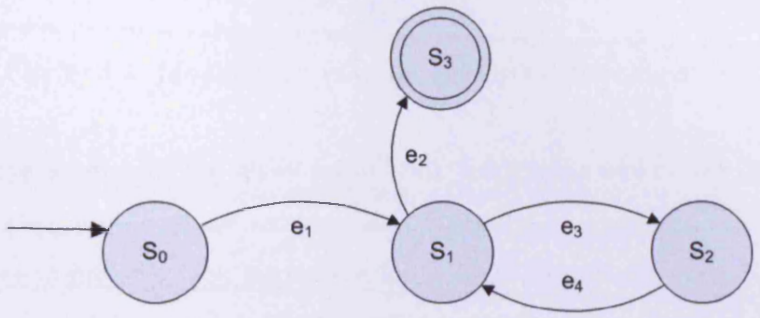


Figure 4.2: A event-driven finite state automaton

The structure of the p-automaton is visibly very different and has a completely different acceptance language than its FSM. We give an example of the differences using the FSM described by the following 5-tuple, shown in figure 4.2 and its associated p-automaton, shown in figure 4.3:

$$A = \{\alpha, \beta, \gamma, s_0, s_3\}$$

Where:

- γ is the state transition function $\gamma : \alpha \times \gamma \rightarrow \alpha$.
- $\alpha = \{s_0, s_1, s_3\}$
- $\beta = \{e_1, e_2\}$

As state observations and event observations only include a subset of those which occur in the FSM, we may miss some important elements of the states an actor has been in. We can see that by not monitoring the event e_3 or variables which indicate state s_2 has

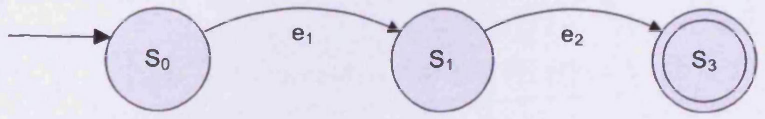


Figure 4.3: A provenance automaton

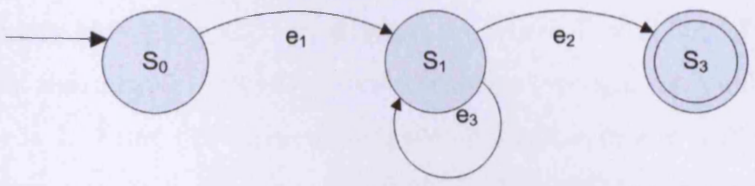


Figure 4.4: Modified provenance automaton from figure 4.3

occurred, we have no idea of the likely set of state transitions which may have occurred or that indeed whether another state occurred also. Even if these elements were included in our p-automaton, we would still lack important knowledge about how states have arisen. For instance, if we add e_3 to our set of monitored events (and subsequently) observed events β, we derive the provenance automaton in figure 4.4. This still is not completely representative of the entire FSM for the actor shown in figure 4.2.

Maximal Observations

From state s_1, given that we are now monitoring event e_3, we are able to see that it has indeed occurred, but as s_2 is still not in our set of observation states, it can only be assumed that $\hat{\delta}(s_1, e_3) = \{s_1\}$. We can see that our p-automaton can only give a true reflection of all states an actor has been in when the set of observed states are exactly the same as those in S, and observed events are exactly the same as those in Σ. In order to illustrate this we prove by extension that the maximum possible amount of observed states and transitions from a set of observations is achieved when $M = \Sigma$ for a p-automaton.

Example 4.2. Consider the automaton that has a transition function δ, where $\delta(s_i, e_i) = s_{i+1}$ holds (where every new event leads to a different state). Consider also that the observations of e_{i+1} always occur later than observations of e_i. We consider $|\delta|$ (the number of transitions) for a number of example scenarios, where the set of all monitored states $T = S$ but the set of all monitored events $M \subset \Sigma$.

1. $\Sigma = \{e_0\}, M = \{\emptyset\}, T = \{s_0, s_1\}, S = \{s_0, s_1\} \implies |\delta| = 0$
2. $\Sigma = \{e_0, e_1\}, M = \{e_0\}, T = \{s_0, s_1, s_2\}, S = \{s_0, s_1, s_2\} \implies |\delta| = 1$
3. $\Sigma = \{e_0, e_1, e_2\}, M = \{e_0, e_1\}, T = \{s_0, s_1, s_2, s_3\}, S = \{s_0, s_1, s_2, s_3\} \implies |\delta| = 2$

By extension we say that $|\delta| = i - 1$ in all scenarios where $T = S$ but $M \subset \Sigma$. Another description is that the number of events observed is always less than the number of transition events which are in Σ. From our earlier description of *latent events* in section 3.5.2, we can say that if latent events exist, they are within the set $\Sigma - M$. Next we consider examples where $T \subset S$ but $M = \Sigma$.

1. $\Sigma = \{e_0\}, M = \{e_0\}, T = \{s_0\}, S = \{s_0, s_1\} \implies |\delta| = 0$
2. $\Sigma = \{e_0, e_1\}, M = \{e_0, e_1\}, T = \{s_0, s_1\}, S = \{s_0, s_1, s_2\} \implies |\delta| = 1$
3. $\Sigma = \{e_0, e_1, e_2\}, M = \{e_0, e_1, e_2\}, T = \{s_0, s_1, s_2\}, S = \{s_0, s_1, s_2, s_3\} \implies |\delta| = 2$

By extension we say that $|\delta| = i - 1$ in all scenarios where $T \subset S$ but $M = \Sigma$, or the number of states observed is always less than the number of states which exist in S. From our earlier description of *latent states* in section 3.5.2, we now can say that latent states always occur in the set $S - T$.

4.3.3 Language Differences

It is possible to derive the language $L(A)$ which consists of the set of events seen by the p-automaton until a certain point in time. This describes the sets of events w over the set Σ, that on being given to the p-automaton has resulted in some final state from F. This is different to a standard FSM whose acceptance language describes the set of strings such that $\hat{\delta}$ contains at least one accepting state for each string [42]. Whilst an event-driven finite state automaton's language describes all possible sequences of events that are triggers between that automatons state, a p-automatons language only describes events that have been observed. Therefore a p-automaton represents information about an actor up to the time when observations have been made. It cannot be used to make any assumptions on the events which may be accepted in the future, but can only be used to explore the behavior which has been exhibited so far. Formally, if $A = \{S, \Sigma, \delta, s_0, F, M, T\}$ is a p-automaton then

$$L(A) = \{w | \hat{\delta}(s_0, w) \cap F \neq \emptyset\}$$

That is, $L(A)$ is the set of events w in Σ such that $\hat{\delta}(s_0, w)$ contains at least one accepting state.

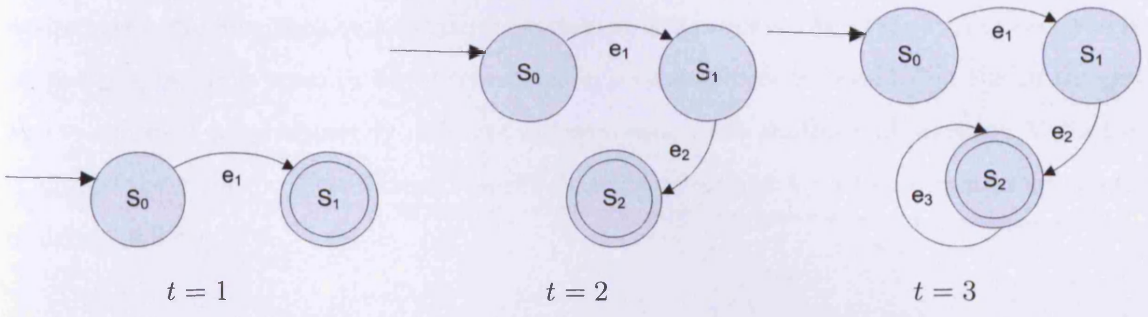


Figure 4.5: Evolution of an p-automaton over time; additional observations build on previous ones

4.3.4 Evolution and Variation of p-automaton

We have shown that by using a p-automaton, context is able to be captured for an actor by recording observations of a chain of states up unto the present point in time. What also may be of interest to a user is how the current p-automaton has evolved from a previous one or the differences in p-automaton recorded for the same process interval. Such evolution indicates those states and transition events which are new or missing over those of previous observations. These differences will occur depending on how both the variables which context is representing, as well as the transition events which indicate state change have been chosen to be modeled with p-automaton. A system variable such as memory may fluctuate through many different values for instance and if each one is modeled as a separate state, then it is possible the different states will be observed for the same process.

We class the use of additional observations with a previous p-automaton's observations which refer to the same actor as the *evolution* of an actors p-automaton. In this case, additional observed states and events may only ever be added to the final observation of state of a previous p-automaton. This is due to the contributions the additional observations make over the previous automaton. We therefore say that the automaton p_1 has an alphabet which is a subset or equal to the set of later observed events comprising an automaton for the same actor, p_2. Formally, if $p_1 = \{S, \sigma_1, \delta, s_0, F\}$ is a p-automaton observed ending at time t_1 and $p_2 = \{S, \sigma_2, \delta, s_0, F\}$ is a p-automaton which refers to the same actor ending at time t_2, and $t_2 > t_1$, and for both p_1 and p_2 observations started at the same time, then $\sigma_1 \subseteq \sigma_2$.

When comparing two p-automata which are calculated from observations upon the same actors within a process, we class this as the *variation* between the p-automata. A user may desire to observe how the actor's state is developing in a process, over a number of executions of that process. It is possible for observations in such scenarios to reveal that the states and events captured are completely different between one p-automaton and another. Variation is akin to the inclusion of additional events that are monitored for a p-automaton, as shown in figures 4.3 and 4.4.

4.4 Modeling State in Actor based Systems

So far in this chapter we've introduced theory to describe how the definitions we made in chapter 3 for context can be used to describe a model of state in actors. Central to the implementation of this model is the means by which actual observations are represented as state, which could be performed in a variety of ways. We focus on time series observations, which are a common means for collection of observations throughout monitoring systems we reviewed in section 2.4. Determining the states present within a set of time series observations can be a laborious and time consuming process. For scientists, the boundaries of each state observation require precise definition and an experimenter may not know sensible values which they may hold. In this section we introduce how the concept of state in p-automata may be derived from observations upon an actor and represented succinctly using a language for extraction of knowledge from multi-variate time series.

4.4.1 Motivation for an Interval Based Representation

In an actor based system, actor states which are outside the periods of time over which a process is invoked are not usually deemed "interesting", for defining context. Instead of mining the state of an actor when it is not executing any functionality, we turn our focus to the period over which an action is invoked. As actions are usually invoked in actor systems through message exchange, we claim that the most interesting states are those which occur within the interval when a request message was sent to an actor and the associated response message was sent by that actor. In this period, the actor's observed state may be documented as part of any documentation that is recorded for a process.

In figure 3.5 we showed a typical scenario in an actor based system where a client invokes a service. We can see that from the point of view of the client, this consists of two events - sending a request and receiving a response, and for the service actor two events also -

receiving a request and sending a response. One approach to documenting the action's invocation might be to document it from the point of view of each actor and the two events which they each have knowledge of. In our diagram, the response from the service is shown to be not sent immediately - as the action which is performed has a duration over which it executes. This is the case for all service based systems, although their response may be seen as almost instantaneous by a client, depending on the complexity of the action and method of deployment chosen. For documentation of the actor's properties during invocation, we could document every time something of interest occurs upon the actor. Such a model is appropriate where responses from each actor are assumed to be immediate and involve a few observations of state. However, it is also possible that observations of state are available consistently over the time the actor is invoked. In such scenarios, documentation of every internal observation made might be inappropriate due to the many similar measurements that may result. Our example in section 3.2 also motivates for a representation other than an event-based model due to the length of time over which consistently observed states are held.

An appropriate data format for the variables which comprise the states held during execution of actions is therefore that of *time intervals*, (see definition 3.5.3). Using intervals, a variable is able to be recorded along with the start and end time points over which it holds a particular value. Modeling state using intervals therefore allows representation of the concept of duration for a state, something which is not possible through current event-based representations often used in provenance systems to date.

4.4.2 Deriving Series of State from Variable Measurements

Where multiple variables are collected as actor state, each point in time may reference a collection of variables which were observed at that point in time and which may all be queried from a monitoring system repository. This type of data is known as a set of *multi-variate* measurements. We demonstrate how algorithms applicable to multi-variate data observed over time may be used to derive state observations.

Pattern observations within temporal measurements have long been represented using Allen's interval relations [9], where summaries of concurrently occurring intervals may be made using 13 different relations (overlaps, meets, during, finishes etc). Whilst these summaries attempt to represent the coincidence of simultaneously occurring intervals they have

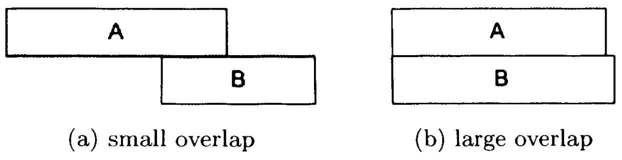
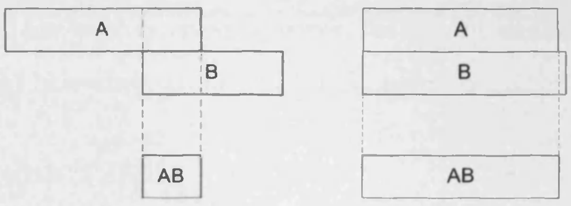


Figure 4.6: Variations of Allen's *overlaps* relation upon the same intervals

also been described as ambiguous, not easily comprehensible and non-robust [58]. The examples shown in figure 4.6 demonstrate how the same two coinciding intervals can have many degrees of how much they overlap, which Allen's *overlap* relation is incapable of expressing.

The Time Series Knowledge Representation (TSKR) is a language for representing interval relationships, described as a better tool than Allen's relations [57], able to summarise simply the relationships between many variables observed across the same time period. The TKSR is comprised of 3 different levels, with each describing concepts of duration, coincidence and partial order. These levels are described as would be with musical notation (tones, chords, phrases). The lowest level consists of *tones* which are built from a label, symbol and a description about the interval when that tone is observed. Essentially a time interval description, tones can be obtained through the division of a set of observations of a single variable into several smaller sets using a number of different thresholds for that value with an approach known as *segmentation* [45]. Segmentation provides the benefit of a reduced (and more manageable) number of states than if calculated using the discrete values originally recorded. For example, given a set of temperature values over the last year, providing thresholds to indicate when the temperature was high, normal or low would yield a series of intervals for when high, normal or low measurements had been made. These resultant intervals would not coincide with one another as it is impossible for measurement of a single variable to be within two of the ranges at the same time. It would not make sense to say for instance that our temperature measurements were both high and low at the same point in time. *Chords* are a summary of which tones coincided with one another and when. For example, two tones A and B which occur simultaneously may be summarised through the chord "AB" over the interval they coincide. Figure 4.7 shows how chords solve the problem of Allen's expression of the overlaps relation, with chords below the intervals which coincide. TSKR therefore enables the intersection of sets of time series measurements, which could be derived from another segmented set of measurements of another variable. If our temperature example included a second variable, such as humidity, measured over the last year also, which was also segmented into high, medium or low intervals, the derived chords would be able to express all the possible distinct combinations of both the humidity and temperature series. Once the two series had been intersected, this collection of chords would summarise the measurements which had been made for each variable. It would be possible for example, to identify the times at which temperature was high *and* humidity was low (or any combination of high, normal or low). Introducing further variables would increase the complexity of each chord, and also the subsequent queries which could be made of any data based on them.



(a) small overlap (b) large overlap

Figure 4.7: Chords in TSKR represent coincidence of intervals

Defining State Boundaries

As we present an observation based model, initially it is impossible to predict the behavior of an actors' host system as the complete set of transition events, (δ) and future states (S) are unknown. It is necessary to be able to define when a transition between states occurs and how many states are possible for each actor. We can progressively construct such a model from execution of a single process multiple times and making observations of δ and S. Such a model will hopefully improve any future prediction of state which may be made as more observations are collected. Observations which are to be made upon particular actors therefore need to be clearly outlined, along with the conditions which indicate that a change of state has occurred, referred to as state thresholds. Given that we have collected a set of variable measurements we would like to distinguish when an actor is in one state or another.

Definition 4.1. *The state thresholds are a set of finite thresholds for each observation* *variable which subdivide their total ranges into a number of non-overlapping subsets. These variables are then capable of changing across these subsets. These detail the boundaries at which state transition will occur.*

If both the complete set of events which have been observed and the state thresholds are known for an actor, given a time-ordered set of each, we can determine the set of transition functions which correspond to Σ. Referring back to our original example in section 3.2, this would indicate that we had managed to capture a number of time-stamped observations about the variables represented within β. By ordering observations of both events and state by time-stamp, and grouping states using thresholds (to reduce the total number of observed states), we can see where transitions between state occurred, possibly attributing them to events. When Σ has been derived however, we cannot assume that the p-automaton provides an indication of how state transitions will always occur in the future.

Representing State with TSKR

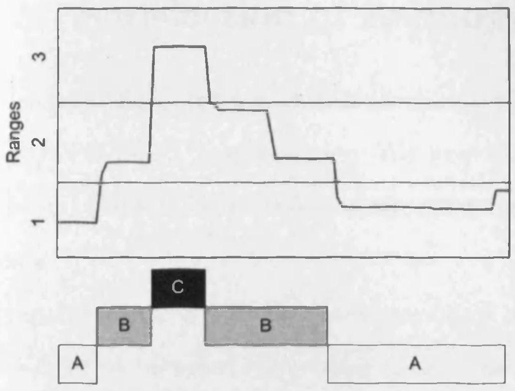
The TSKR is a very natural representation for the measurement of variables which are collected over the same time period. As observations of actor state are comprised of many time series variables observed for the interval over which that actor is in use, it is possible to use TSKR to represent and derive knowledge about the unique states an actor has been in. For instance, each tone in the TSKR is equivalent to the interval in which a single variable observed upon an actor holds a value between two thresholds. The coincidence of these intervals represents a particular observation of state, which is equivalent to TSKR's chord description. Each chord is able to be represented simply through a 'pattern' which holds numerical references for the tones which coincide for that interval. As thresholds are set for tones prior to finding the periods over which they occur, each chord pattern element references the limits for each of the conditions of that chord. The most common sequences which appear in the order of chords can be found quite simply through a depth first search of ordered chord data. Using this method to describe monitored variables as state significantly reduces the amount of information which is captured, as similar variable values are grouped together and recorded as part of the same tone. However, important information about when changes across thresholds occur are documented concisely. The benefit of using TSKR to derive states lies in this simplicity of representation of the derived state observations. Once a state has been found and labeled, the original monitoring data is no longer necessary for the purposes of reasoning with it, as the knowledge desired is represented as the single pattern referring to the conditions which were true as it was observed. This knowledge will be represented at a level specified through the thresholds set by a user. As the TSKR already has a number of algorithms defined for it for derivation of chords [58] (states) and phrases originating from tones (partially ordered sets of $k > 1$ chords), we are able to adopt them for deriving and reasoning over observations of state.

Using TSKR

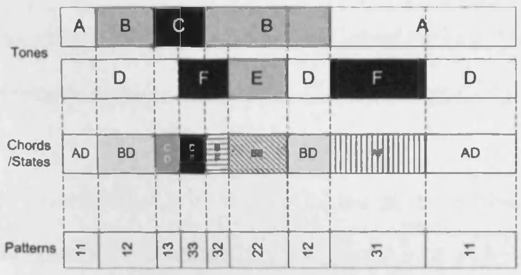
We now proceed by applying the TSKR to our example introduced in section 3.2.

Example 4.3. We assume a collection of 3 variables as β from our example. Each of these variables is measured over the same time interval yielding a set of observation values for a particular time in the interval. For each variable, 2 thresholds are given to segment its measurements into 3 time intervals indicating a high, normal or low value, corresponding to TSKR's basic primitives - using a tone based representation. The decision of how these thresholds are chosen may be undertaken by a user using prior knowledge, or using an equal frequency or equal width histogram derived from the range of each sets of values associated with the recorded variables. The application of such thresholds therefore for any single variable will give a set of tones which do not overlap one another (as a variable can only ever be within the limits of two thresholds). Collections of these intervals can be represented as *interval series* which describe the same property as being true over a number of non-overlapping intervals.

For example, consider the series shown in figure 4.8(a), which is calculated from measurement of a variable's value over a period of time. Using thresholds to segment [45] the measurements, we can determine when this variable was within a number of pre-defined ranges (in our example these are described as high, medium or low). The resultant series is shown as a set of non-overlapping tones with symbols A, B and C. Each of these has an associated range into which the variable falls for the period over which the tone was observed. The tones may be mined for coincidence across different variables, where a number of different tones coincide for an interval exceeding a given duration to yield a description of a chord. Each time series has 2 thresholds applied to yield 3 tones, giving 2 sets of tones with tone symbols A-F. In our example, chords derived from these tones are labeled with the symbols which coincide from the tones they represent (AD, BD etc). These chords



(a) Variable segmentation using thresholds to find series



(b) Mining states and their patterns from coinciding intervals

Figure 4.8: Mining unique context series from numeric time series

are representative of states, as they describe when an actor has a particular set of values for variables. In figure 4.8(b), we coincide our derived variable series with another series calculated from another variable measured over the same time period. By coinciding the two series, we are able to determine the unique states which an actor was in over that period by using the two variables as the state components. Our resultant (coincidence) series therefore describes the periods over which each variable is described as high, medium or low. The pattern reference for each state is calculated by holding references to the ranges into which each variable fell for the duration of a state. The final context series therefore describes both the conditions true for each state and the intervals in which those conditions were true.

The final series which represents states is said to be *contiguous*, i.e it will have no gaps between intervals which follow one another. Analysis in this manner yields a set of states where each state may occur multiple times over an observation interval. From the p-automaton which may be constructed, we have no knowledge of the set of events which cause transitions, merely the states that have occurred and that transition from one state to the next occurs. See [59] for a more complete description of the concepts described here.

4.5 Satisfaction of Requirements Outstanding

In section 3.5.1 we noted that a number of requirements weren't able to be met with use of the PReServ model alone. We now show how these are able to be satisfied using the representations for context which we have introduced in this chapter. These are outlined below:

Requirement 3.2 - The provenance of an actor should include values of an actor's state variables at instants other than when the actor triggered recording.

Requirement 3.3 - The provenance of an actor should include the trend of change of an actor's state variables over the period the actor performs an action, not just a set of static values of measured variables.

The use of the TSKR model to represent changes of state means they are able to be recorded as time intervals. It is a much more succinct summary of observations of metrics as a collection of thresholds and start and end times than collecting many values which may not often change. These records of when states begin and end mean that assertions of state can contain detail of features without having to assert a large number of different actor state assertions.

Requirement 3.4 - The provenance of an actor should be documented in such a way as to enable subsequent collections of state observed for the same action to be compared against one another.

The pattern representation which we use for each state makes it possible for comparisons of observations made for the same actor. This is done through comparing the patterns of states which occur at the same time in each observation. We demonstrate an example of this later on in this chapter, in section 4.6.3.

Requirement 3.5 - The provenance of an actor should provide information relevant to the history of an item even when causal relationships are unknown.

Through use of the interval representation which we propose, we are able to concisely summarise contextual information collected for an actor. It is possible for this to either be asserted as part of documentation for a process, or left upon each of the actors upon which it was collected. Therefore it is possible to collect information about an actor, even if causal relationships may not be known.

4.6 Use Cases Revisited

In this section we revisit the use cases we presented in section 3.4. We demonstrate an approach to how these could be satisfied using the p-automaton and interval based model we have presented so far.

4.6.1 Context Analysis Using Action

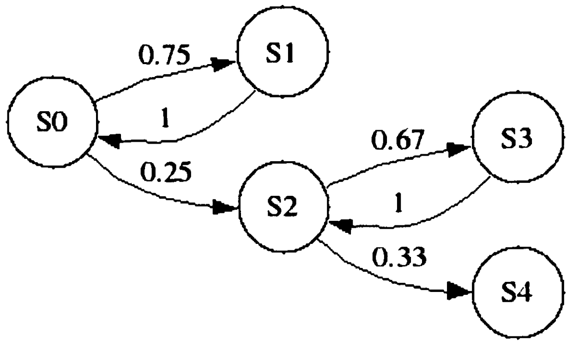
We can determine why certain states are observed in a system where both context is now able to be recorded as state and every action is documented, by inspecting how previous actions have affected the state of an actor. If for example, the same state has always been observed following a particular action execution, we can infer that an observation of that state is caused by that action. Hence if the same state is later observed, we can suggest that it was caused by that action being executed. In situations where the same host has multiple actors located upon it, we can use this method to find the particular action which gave rise to a state.

4.6.2 Future Prediction of Context

Given that the states and transitions observed for previous processes are represented using the model presented in this chapter - it is now possible to be able to approximate future states for a process executed multiple times. For all states related to the same event, a transition table listing the probability of state transition between two past states (given an observed event) may be calculated. This matrix represents a probabilistic automaton which may be used as the basis for a prediction of state - the most likely next state for an actor being that transition with the largest probability value corresponding to the current state. As an example, consider the transition table shown in figure 4.9(a) for a set of states. The table is generated based on a set of observations of state and the transitions between them. It is read from row to column, so if an actor's current state is S2, there is a probability of 0.33 that the next transition will be to S4. It is however more likely the next transition will be to S3 (a probability of 0.67) based on the observations recorded in the table. Given that further observations would alter this transition probability, it may be the case that in a future calculation of the table, the transition when in S2 may be to a different state.

	S0	S1	S2	S3	S4
S0	0	0.75	0.25	0	0
S1	1	0	0	0	0
S2	0	0	0	0.67	0.33
S3	0	0	1	0	0
S4	0	0	0	0	0

(a) A state transition table



(b) The finite state automaton for a)

Figure 4.9

4.6.3 Comparison of Past Actions

In sections 3.4.2 and 3.4.3 we described use cases for how comparison of actions and processes using context could enable swift navigation of process documentation. Here we define a measure of similarity to support such a comparison. This is based upon how similar two states are given that their associated chord representations do not match one another. As we have adopted an approach of segmentation, it is simple to make comparisons when the two measurements of state to be compared have been made using the same variables. The two states represented as chords will therefore have pattern references which also correspond to conditions which were observed for the same variables. For instance if the first pattern element for a state is a measurement of load on the actor, the same pattern element also corresponds to load during a second observation (if the boundaries for thresholds remain the same). A difference in any one of the pattern elements indicates that the measurement for the associated variable fell within different threshold boundaries. In our model, pattern elements which are higher correspond to higher thresholds for tones and a distance measure is therefore appropriate to determine how similar two states are. We define the similarity between two states as a simple distance measure between each of their corresponding pattern values as shown in equation 4.1, where q and r are the two patterns being compared and p and t are the number of items in the patterns and the number of possible values for each of those items. Here, each element in each state pattern is compared with one in another state, for the same original variable. The total number of states possible for an actor is p^t, though for any given process run not all states may be observed.

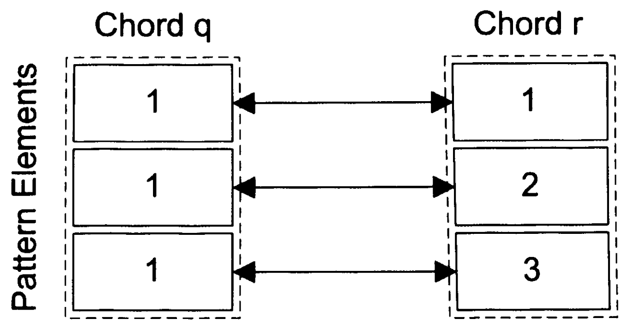


Figure 4.10: Similarity of chords is based on the distance of their constituent pattern elements

$$s = 1 - \frac{\sum_{n=0}^{p} |q_n - r_n|}{p \times (t-1)} \tag{4.1}$$

The distance between two states therefore is the total measured error between each of the states pattern elements, divided by the maximum total distance possible for error on each of those elements. Our measure differs slightly from a distance measure such as the Levenshtein distance (or edit distance) [48]. With such a measure the distance is computed as the number of changes necessary to convert one string to another. If we consider our computed chords pattern as those to compare the Levenshtein distance is unable to distinguish between the degree of change in any one of our pattern elements, just that elements differ. Any pattern element therefore that requires being altered would be given the same score value. As pattern elements correspond to numerically ordered ranges in our model, taking account of the difference in these elements is important. This similarity value gives us a single measure of how similar the conditions under which each of the actors involved were operating were. The two chords shown in figure 4.10 demonstrate how this approach works; given that we have 3 variables which were measured, resulting in 3 pattern elements. The maximum possible distance between each of these pattern elements will always be 2 $(t-1)$, resulting in a maximum chord distance of 6 $(p \times (t-1))$. The total distance in our example is 3, giving a similarity of 0.5.

4.6.4 Comparison of Past Processes

Given that we can compute the similarity of two states observed during a process, we can also compare the similarity of the larger process the actor took part in against observations made in other processes. This is dependant upon the same process being invoked multiple times, with the same actors used within the process. Our measure for the process is computed by determining the total of all similarity values, the greater the value indicating the states observed in the the process were more similar. This value is divided by the maximum possible similarity for that process, which is the total of the maximum similarity values for each actor involved. The resultant measure falls between 0 and 1, and is described in figure 4.2 where a is the number of actions in the process.

$$s = \frac{\sum_{n=0}^{a} |s_n|}{\sum_{n=0}^{a} |\max s_n|} \tag{4.2}$$

For example, we consider a process comprised of 4 actions which is executed twice and state pattern values are compared against one another. The first and second actions both have a similarity value of 0.65, the third 0.5 and the fourth 0.7 between each process run. Therefore the total of all similarity values for the process is 2.5, with a maximum possible similarity of 4 (1 for each action in the process). The total similarity s of the process is therefore 0.625 (2.5/4).

4.7 Mining State Transition Events

Transition events between states in our model are represented either as a change in β between two states, or through temporal ordering of the observations which have been made. If we choose when a variation in state occurs, we are able to capture transitions as the tuple $\{s_1, s_2\}$ and only need calculate where differences in state occur - as we have shown in section 4.4.2. Each transition would be associated with a given instance in time at which that transition occurred. There are a number of alternative ways in which transition events may be considered in our system.

4.7.1 Temporal Order of Observations

The most simple approach to determining transitions between states is that all event and state observations are ordered temporally and the primary assumption is that any event which occurred in the instance a state changed is the given transition event for that transition. We assume that the convert service from our example progresses through the set of observed states S, $S = \{s_0, s_1\}$ with the set of observed events E, $E = \{e_0, e_1, e_2\}$ up until

	e_0	e_1
s_0	s_0	s_1
s_1	$\emptyset$	$\emptyset$

(a) Automaton derived from real-time data

	e_0	e_1	e_3
s_0	s_0	s_1	$\emptyset$
s_1	$\emptyset$	$\emptyset$	s_1

(b) Inclusive of additional latent events

Table 4.2: Transition tables

the present time. We introduce the functions $t_e()$ and $t_l()$, which return the earliest and last observation time of the event/state passed to them respectively. Given that $t_l(s_0) > t_e(e_0)$, that is, the last known observation of state s_0 occurs before the earliest observation of event e_0 and: $t_l(e_0) > t_e(s_0)$; $t_l(s_0) > t_e(e_1)$; $t_l(e_1) > t_e(s_1)$; $t_l(s_1) > t_e(e_2)$.

We are able to infer the transitions which correspond to the first state transition table in 4.2. Notice that no transitions relating to e_2 have been described in the table. This is because although e_2 has been observed, no subsequent states have and therefore it cannot be assumed whether that actor will move to a new state or return to a previous one. If however it is known that $t_l(s_1) < t_e(e_2)$ we can derive the second state transition table shown in table 4.2.

While assuming transition events always occur between the times at which states are observed, it must also be assumed that the transition effect resulting from such event is always instantaneous and there is no associated time delay for a change in state to occur.

4.8 Summary and Discussion

We have introduced the p-automaton and shown how it allows us to see and investigate the causes of a set of observed states for an actor. These causes are based on the set of observed views of state on the actor and the set events indicated as transitions by the user. A p-automaton allows an alternate view of monitoring data compared to approaches such as state diagrams and monitoring tool graphs. Through definition of particular state boundaries upon a collection of observations, it is possible to ascertain a description of a provenance automaton. A p-automaton gives us a mechanism of being able to tell when a change of state has occurred upon an actor. We can also observe the effects of changing the set of monitored states on the provenance automaton produced. i.e It is possible to show a variety of assumed cause events for observations of state if any exist. Through choosing sets of monitored events which overlap one another, we can obtain a list of those events which are frequently assumed to be the most common causes of state transition. By removing the monitored events which only give transitions from a state to itself, we highlight those events which are deemed to be important by a user but give no transitions to other states.

We also demonstrated how modeling observations of context using an interval representation is a novel approach to gathering metadata about processes within the provenance community. Particular focus has previously been paid to documenting the set of events which comprise a process, without discussion of whether any one of these events have effects which give rise to properties and conditions holding true for longer than a single point in time. By adopting a model capable of representing intervals we hope to capture such knowledge.

As time series data may be recorded within a particular community exclusively for that community's use, they may desire that particular metrics are not revealed to any other parties. The conversion of time series data into state interval data described here also serves a purpose of anonomysing the metrics which are observed. In just exposing the states which occurred, rather than describing the configuration of that data which comprises each state, we anonymise that information which may need to remain confidential within that group. In this manner, users of such data are able to see that changes in state observation have occurred without necessarily knowing how the metrics from each of those observations are built.

There is however a number of limitations to the use of the p-automaton we present. Firstly, any p-automaton is unique to the particular actor original measurements have been taken from. This means that it is not able to be used for analysis of state observations upon other actors. Instead a p-automaton needs to be built for each actor involved in a particular process and that p-automaton used for any analysis of state observations taken upon that host. However, it may be possible to use automaton to locate differences in actors providing the same functionality which are located within similar environments. For example, the same web service deployed on two different hosts, both of which have the same hardware specification and configuration.

In this Chapter we have introduced a conceptual model for actor state using finite state automaton using the definitions for state presented in Chapter 3. Our terminology however is equally usable within other modeling techniques which are suited to representing the definitions introduced. We demonstrated an approach to mining observations of context made upon contributing actors in a process to the causal process with which they were related. We now proceed by demonstrating application of this approach to representation of context within an existing process documentation environment.

Chapter 5

Enabling Context Capture in Process Documentation Systems

We begin this chapter by reviewing the work this thesis has presented so far. Firstly, we described the importance of context to action and processes and motivate that its introduction in provenance systems would enable a variety of use cases to be satisfied. Later, in Chapter 3 we defined terminology appropriate for use in describing context in general, followed by a model in Chapter 4 which demonstrated how such terminology could be applied to represent context for an actor based system.

We now describe how our model is capable of capturing observation of the states of actors within a pre-existing process documentation system. By extending a pre-existing system we are able to focus on the core of our context concerns, rather than that of the more general provenance problem, largely solved by many existing systems. These include providing a means to capture context along with automating and making capture transparent to a user. We attempt to tackle the many challenges listed in Chapter 2 for context capture by providing a system which is able to be customised to meet the needs of a particular application scientist, with as little intrusion to the application as is possible and instead making observations in a passive manner. This system, known as the State Assertion Registry (StAR) is proven to be flexible enough to represent the evolution of state upon an actor and collects observations in a manner consistent with our definitions and model already presented. Our choice of architecture features both capture of assertions of interaction and a capability for capture of actor state. We demonstrate that through assertion capture which includes assertions of state conforming to our definitions given in Chapters 3 and 4 we are able to capture a model for the past and potential behaviour of actors that a user of a process would otherwise be unaware of. Through the context capture possible with the resultant joint architecture, users will be able to determine differences during the execution of repeated actions and processes.

5.1 Introduction

In documenting the history of context along with the actions of each of the contributing actors, we seek to understand the circumstances of why an actor may have exhibited a particular behavior and what can be done to avoid or continue those events which may have caused it. The information collected within this history, as has been seen in Chapter 2, is extremely subjective according to the purpose for which it is intended. Attempting to define context as a core set of variables will not satisfy all those use cases which may have requirements for elements of it, with previous attempts at schemas to do so failing to come into fruition due to application dependance [33,38]. If specification of variables which are to be recorded are instead left to a domain scientist we enable capture of those metrics which are deemed most relevant by that user. There is also the possibility to modify the data items according to what is required at a later date. Here we revisit the features necessary for an architecture to successfully record context within a pre-existing provenance environment. Our aim is not to prescribe how process documentation is recorded, but how assertions about the context of actions within such documentation are generated and recorded. We therefore have built our architecture around the use of a pre-existing provenance protocol and enabling libraries. These provenance tools do not prescribe what may be collected as the contents of an assertion of state, which allows us to provide a mechanism so that a user/administrator may do so instead. These have been tested in a wide variety of scenarios including those within a number of different application domains [36].

5.2 Tradeoffs in Context Capture

We have previously outlined a number of qualities which would be desirable when documenting the context of actions, along with outlining those tradeoffs which need to be evaluated in building a process documentation system [87]. We summarise a number of considerations which are derived from these below:

- Level of Actor Co-operation - In [87] we describe actors which may have full, reduced or no actor co-operation with a provenance environment. The actor will make decisions on the level of detail at which it exposes information about itself. In a system within which we have full actor co-operation for instance, direct availability to implementation may be possible with no restrictions to available resources. Such a situation might occur if process documentation and context capture are all administrated by the same user. In a system where multiple users exist, such as an administrator for an actor and a client who makes use of it a reduced level of access to information or even no access at all may be provided. Given a variety of scenarios are possible, it may be increasingly difficult to extract contextual information which is of use to an end user, even though it may exist.

- Availability of Context Resources - If an actor does not have any resources available upon it, it is going to be unable to provide any contextual information about itself, regardless of whether or not it wanted to co-operate with a client. It may be the case that resources do exist but are not yet integrated with the process documentation system, meaning that the amount of work necessary to use them could be relatively minor.

- Level of Actor Intrusion - The level of intrusion which we introduce into a system through recording documentation of process and the amount of information such intrusion enables a system to capture has previously been investigated within job execution environments [73]. The most desirable system is described as one with no intrusion to system or user, but providing all information about the two. In a service based system, instrumentation of actors may not be possible, due to their loosely-coupled interaction with a querying actor. Where minimal intrusion is only possible, the level of useful information able to be captured is also minimal. It has also previously been suggested that the level of granularity at which provenance is collected bears a relation to its usefulness in its application domain [78]. In order to capture semantically meaningful, relevant, information in service-oriented-architectures (SOA's), intrusion would be required into every one of the services which are being used as components within a workflow. Dependant on the co-operation of each of the process actors, such intrusion into an application may be impossible. We consider the situation that as a finer granularity of information is collected, it incurs a greater overhead in the time to record provenance documentation and a larger level of intrusion to the actors.

5.3 The State Assertion Registry

We constructed our solution to the outstanding requirements of context for a specific provenance system. The State Assertion Registry (StAR) is a tool for capturing the state of actors involved in a process within a service oriented architecture. It works by relying on the observations made by trusted tools co-located with actors, rather than enforcing a core set of variables to be captured as observations in all application scenarios. The interval based model we adopt for capture (presented in section 4.4.2) can be applied to other architectures, and is not restricted to working within a service oriented environment.

Our registry based approach is an attempt to allow users to specify when and what is to be recorded as observations of actor state. It is up to the administrator of the architecture to decide which variables and resources are exposed publicly and may be queried on execution of a service. An experimenter would finally reason against the observations of variables which are captured, using such an analysis technique as the Time Series Knowledge Representation (TSKR) to derive observations of state. As policies may differ between actors, so too will the observations of state made across them. This is because the context of each actor is unique to that actor. All observations therefore will only make sense on comparison to other observations made about that actor.

5.4 The Round Robin Database Tool

The Round Robin Database Tool (RRDTool) is a tool for time series monitoring used extensively in cluster and Grid environments [1,3,5,51]. Within a cluster/Grid, RRDTool is used to capture metrics of interest quickly with minimal interruption to its host system. This method of capture may be applied to actors within an SOA also. It can then subsequently be queried in order to determine values of variables over a period of interest, to later be mined for observations of state. As described in Section 2.4.2, RRDTools' strength lies in how data is captured to a database of fixed size, drastically reducing the storage requirements for variable observations made over long periods of time. By interfacing StAR with the

RRDTool storage medium rather than the monitoring tool, we are able to allow StAR to be capable of data capture from those other monitoring system tools which use RRDTool. This includes a wide variety of possible measurements with many of the tools and RRDTool itself providing the capability of capturing user-defined observations.

5.5 Provenance Recording for Services

Our choice of provenance capture mechanism is an implementation of the Provenance Recording Protocol(PReP), called Provenance Recording for Services [34, 35](PReServ) made by the University of Southampton. The choice of adopting PreServ was made for a number of reasons; Firstly, its actor based specification makes it ideal for use within open environments such as SOA's. The use of a third party repository known as a "provenance store" to store evidence means that it is possible to document a view of any process to storage from each of those actors involved in a process. This is then able to be queried at a later date easily from any other actor. Secondly, as this model is applicable in many scenarios, it has been extensively tested within a variety of application domains. The libraries built for this purpose were created as a result of the PASOA project * and are freely available to provenance enable other applications. Finally, PreServ leaves specification of internal information to the application domain in which it is used. Representation of provenance is broken down using a number of types of p-assertion or "provenance assertion". Through the p-assertion, PreServ is capable of asserting descriptions of interaction, actor state and any relationships that exist between assertions. As no structure is specified for the body of actor state, it offers a suitable capture mechanism for our own observations of context. Our specification for context differs slightly than that which was designed for actor state however, as it may include assertions made over the entire time it takes an action to execute and not just before or after messages are exchanged.

The lack of structure for actor state p-assertions has been implemented in such a way as to enable applications which do have specific requirements to implement a schema as they see fit. Doing so however also means that any knowledge which is represented is only able to be interpreted by the particular project that created it. If this knowledge were to be shared, it may require explanation of the structure adopted. Our intention is that a

*http://www.pasoa.org

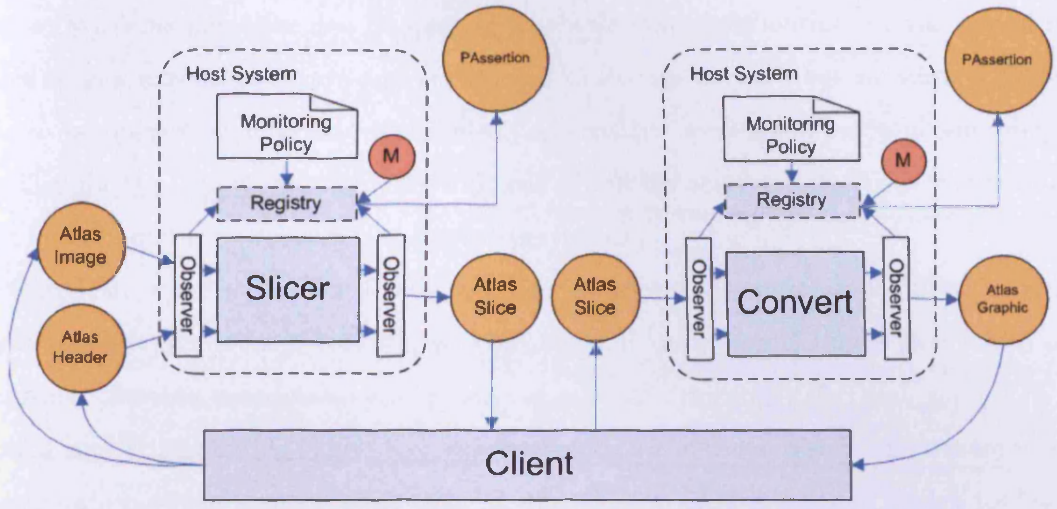


Figure 5.1: Provenance capture using a registry

representation of context which has some formal structure to it, but which is still able to be customised, will enable each of PreServ's assertions to be created in a generic manner and be interpretable between applications.

5.6 Application Example

To demonstrate the use of our architecture we introduce a simple scenario (taken from the Atlas workflow previously presented in Section 3.2), consisting of a workflow comprised of the two services (slicer and convert), which is shown in figure 5.1. The slicer service in this workflow takes as input a single averaged brain image (atlas image) and metadata about that image (atlas header), returning a representation of a slice of the brain along a particular dimension (x, y, z). Each service interacts with actors which capture observations of state over the period which the service is invoked, and this data is later combined to form an assertion of provenance (or *p-assertion*). Multiple assertions of state may be asserted in this manner, across the period over which observations are made, or gathered together and represented as a single assertion.

We refer to our own Java implementation of the architecture, known as the State Assertion Registry (StAR) to demonstrate its main features. This implementation extensively uses Apache Axis Handlers as a mechanism for triggering the capture of provenance assertions, with a RRDTool storage database [51] co-located with the actor as the monitoring source (M). This database can be queried to obtain state information for the period over which service execution occurs and is recorded to storage in real time, meaning it is available to be queried at alternate times also. Alternative application environments may be specified for the registry through development of specific observers (for trigger events) and plug-ins to monitoring sources (for state observations).

Our client in the scenario is Triana, a visual workflow composition tool [50] which allows a user to easily construct a workflow without internal knowledge of the services which they are using. Service descriptions are given via a WSDL document for each service in the process, and Triana is able to give a visual representation of them, which can be manipulated to pass data between them on execution, as with the workflow presented. Such a tool is not strictly necessary for the simple workflow described, but allows us to demonstrate how actor state capture may be achieved as provenance in a manner that is completely independent of the use of a service, providing the transparency available in the existing approaches we reviewed in section 2.3. All assertions in the given scenario are therefore captured on the service side using the wrappers described previously.

5.7 StAR Actors

Here we outline those actors within our system which are necessary in order to fulfill the recording of actor state and the role they play in capturing actor state assertions. We refer to figure 5.1 as a visual aid for their interaction.

Monitor Source - These are sources from which actor provenance is available. Each source would have associated with it a "plug-in" which would be located at the coordinator to collect and expose variables which are desirable to the user. Each of these form part of a state observation. The monitoring source therefore initiates no communication and only responds to requests via a plug-in called through StAR. In our implementation, monitoring information is collected directly from the monitoring sources database through a plug-in class before being passed to the registry after the service has been invoked.

Event Observer - The event observer is located upon the same system as the service actor in question and monitors it for events of interest. If an event occurs, such as a

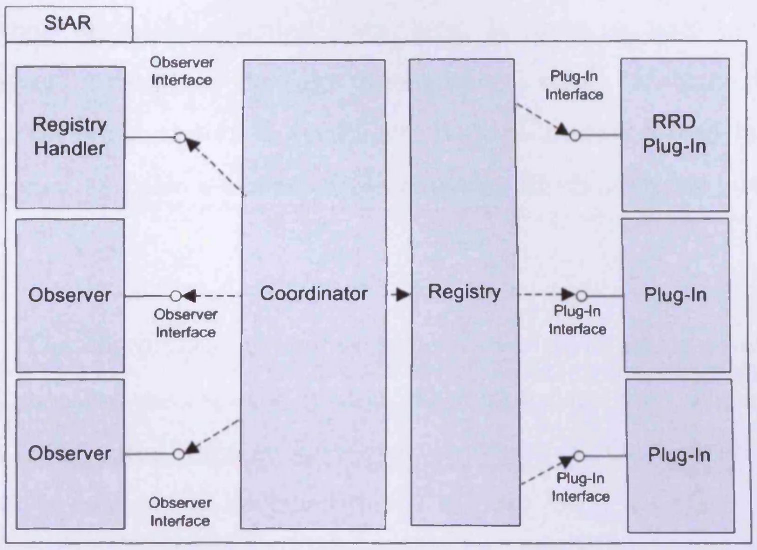


Figure 5.2: The nesting between different StAR components

invocation request or a log file update, the event observer communicates this change back to a coordinator for inspection. This actor can be most likened to a daemon process, which would run as a background process upon the service, making passive observations of the host. The event observer is given details of the events which it is to monitor by the registry upon request.

In the example given, our observer is a Java wrapper to the Apache Axis service which has access to both request and response messages which are received and sent by the service. Essentially, observation of these messages triggers the capture of state information. The use of such a wrapper does not require modification to the logic of the service and requires only a configuration parameter to be set upon the server to indicate it is to be used. From the clients perspective, use of a wrapped service is no different from invoking the service as usual. It may be possible that a user is completely unaware that state observations occur as a workflow is invoked.

Service requests are not the only type of events which might be monitored with our implementation in a SOA. It is possible to capture other event types. An unauthorised access event, deadlock, log file updates could all be exposed by event observers as events which are important to state capture.

Registry - The registry is an actor which holds details of all registered monitoring sources and the rules which are to be executed upon them. It issues requests to event observers and to coordinators for details of the rules to execute as a result of triggered events. In our example a policy file which describes events which are of interest is read by an implementation of the registry, and also describes those observers which are to be initiated when the system starts.

Coordinator - The coordinator in our system receives descriptions of monitored event occurrence and contacts the registry to find the appropriate rules which correspond to functionality to be executed when such an event is triggered. Once these rules have been found, behaviour is executed as specified in the registry, using interfaces appropriate for the monitor source in question. In our system, the coordinator is an event listener which captures events described in each observer plug-in class. If events are triggered which are not described within an observer, exceptions are thrown to indicate this.

Provenance Store - Our system makes use of a provenance store outlined by the Provenance Recording Protocol (PReP) [34] to achieve persistent storage of the results of all rule executions performed by coordinators. Its only purpose is persistent storage of assertions captured concerning workflows and does not initiate any communication itself. It only receives requests and sends acknowledgement messages. Our service based implementation communicates across a SOAP interface to achieve persistent storage of actor state assertions. It is possible through an implementation of PReP called Provenance Recording for Services (PReServ) to capture assertions of interaction also for the period over which services are invoked.

5.8 System Interaction

Here we outline the communication that occurs between system components for recording actor state assertions, which makes use of the system actors listed in section 5.7. Within our system, each component forms part of the same set of Java libraries which are co-located with an actor upon its host system. We describe a complete interaction between a client and service actors as illustrated in figure 5.1, with the subcomponent interaction shown in figure 5.3.

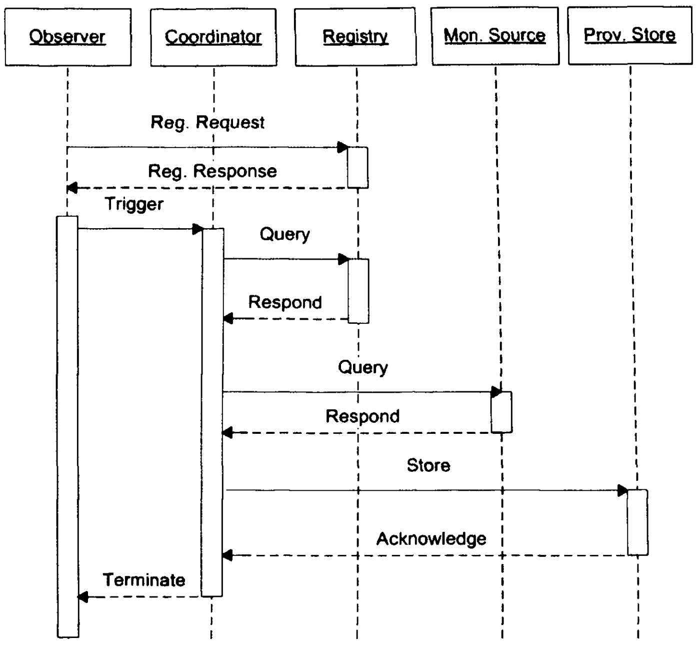


Figure 5.3: Interaction between architecture components

Specification of a registry policy will occur prior to the invocation of the actor(s) co-located with that registry, either by the administrator of that service or a remote configuration mechanism. Our policy allows a user to specify the actor state elements to be read/queried from the monitor database post invocation. As the system is initiated, each observer registers itself. As the trigger of events is specified within observer logic, StAR can determine where invalid event specification occurs within a registry policy file. This is done by indicating events which have been specified in the policy which have not been declared within observers. Once an observer is triggered by a monitored event, a coordinator is created to deal with the events which have been configured for its associated observer. It determines its events through querying the registry and then finds the event monitoring sources' associated plug-in. This plug-in description is loaded into memory for later execution. As the coordinator is now able to query the monitoring source to communicate observations to our system, it is queried for those state variables specified in the policy. Once this information is at the coordinator, an actor state assertion is generated for the time over which the service was executed. The mechanism for describing what has occurred may include the extra step of data mining to determine the unique states which have held over the period over which that actor was invoked. The information is recorded within a provenance store which is specified by the state asserting actor which may or may not be available for public query. It is therefore possible that assertions of actor state do not completely document a full execution of a workflow and not every actor in a workflow agrees to release its evolution of state as assertions publicly. As observations are made passively also, an experimenter may never be aware they occur. Hypotheses however can still be formed, using only the body of information available from assertions for the actor.

5.9 Observation Contents

StAR makes use of a number of unique descriptions for specification of the contents of observations and when they are captured. In essence, this serves as a publish/subscribe type system, where the publisher is the monitoring source and the subscriber is the observer of events. Such descriptions are necessary to allow users to specify when, where and what is captured within the content of an assertion of state. *Rules* are described as user defined instructions to be executed on occurrence of a particular event. Each rule description consists of the plug-in to use to retrieve monitored output and the event which triggers the rule. *Plug-Ins* are used to retrieve information from monitoring sources. A plug-in description consists of the monitored source upon which it is to be executed and the process to execute on being triggered. The process to execute may specify a number of inputs to the plug-in for the associated monitoring source depending on those which have been configured. Observers, as described in sections 5.7 and 5.8, observe events of interest upon an actor. They are used as trigger events by specifying when events of interest are to occur within an actor or external to it. For instance, an observer plug-in may be a class hard-coded within a users' system, or a wrapper to a service each with trigger events to be thrown at particular points of interest. Our system uses a Java based implementation of plug-ins (which we describe later in this chapter), with the primary one being a wrapper to services (though as described earlier is not limited to monitoring only services).

5.9.1 Observation Policies

Observers make use of configuration policies to describe their behavior at run time. These policies define those rules which are to be used to collect information, based on particular trigger events occurring. We represent such a policy through a description written in XML,

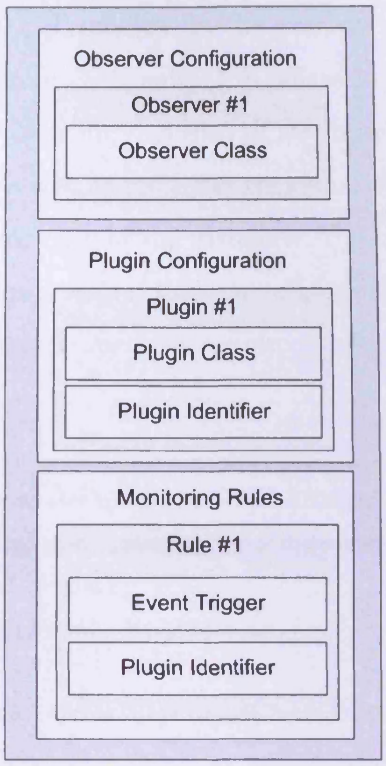


Figure 5.4: The structure of StAR's policy files

whose general structure is shown in figure 5.4. The policy begins with information detailing how the observers which are to be used for event observation are configured, followed by how the particular plug-ins are configured for capturing this information and finally the monitoring rules which describe which plug-ins to execute on observation of an event. We demonstrate how observations are made from a policy in StAR using an observer which collects the current clock time of an actors host system. The policy which defines the functionality to invoke when this observer fires trigger events is given in listing 5.1. Through specifying string descriptions of events to occur at particular points of observer functionality and describing appropriate responses through policy rules, we are able to associate a particular rule with an observer.

We show in figure 5.5 the sequence of interactions which occur for the policy in listing 5.1. In our example, if the event "`Axis Handler Response`" is fired, the rule "`TSKR RRDs`" would be executed. The rule is configured with a plug-in id of 1 which is a reference corresponding to the class to be used for processing the trigger event. As can be seen, the associated plug-in is named "`TSKRPlugin`" and is called with three alternate *variable* arguments, specifying particular thresholds. In the example given, when the TSKRPlugin is executed, it extracts data from each rrdtool database for the variable(s) which is/are passed as an argument to it. Following this each of the unique states which were observed are calculated by the plug-in, as well as the times they were observed through segmentation using the specified thresholds for each of the variables. The plug-in stores the state data to the resource pool used for output. When the system is terminated, the temporary storage which the registry has written to is read for resources, which are retrieved and built into actor state assertions.

```
<monitors>
        <plugin classname="org.pasoa.star.plugin.types.TSKRPlugin" id="1" />
        <observer classname="org.pasoa.star.observer.types.RegistryHandler" />
        <rule name="TSKR RRDs" pluginid="1">
            <event description="Axis Handler Response" />
            <input>
                <rrdpath>/home/scmimw/rrds/ProvenanceCluster/hgrid01.grid.cf.ac.uk</rrdpath>
                    <rrdfiles>
                        <rrdfile name="bytes_in">
                            <thresh>35200</thresh>
                            <thresh>69900</thresh>
                            <thresh>110000</thresh>
                        </rrdfile>
                        <rrdfile name="load_one">
                            <thresh>0.056</thresh>
                            <thresh>0.101</thresh>
                            <thresh>0.2</thresh>
                        </rrdfile>
                        <rrdfile name="mem_buffers">
                            <thresh>228000</thresh>
                            <thresh>232000</thresh>
                            <thresh>300000</thresh>
                        </rrdfile>
                    </rrdfiles>
            </input>
        </rule>
</monitors>
```

Listing 5.1: Example Policy Specification

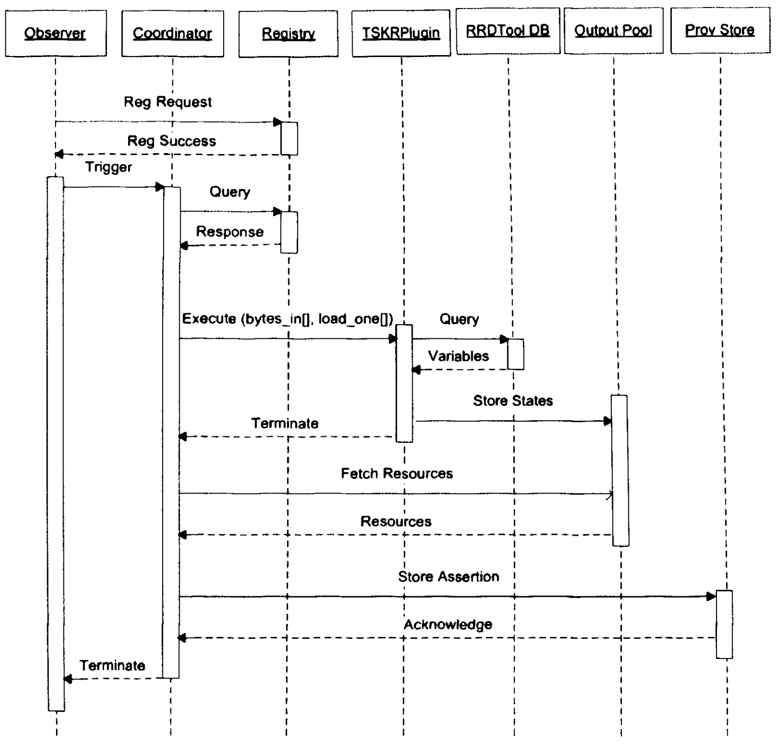


Figure 5.5: Interaction Example of StAR Components

We believe that the way in which policies are specified could be made easier through a point and click web interface for each actor. Currently specification of triggers of plug-in execution are made possible by reference to appropriate descriptions made within observers - such as "Axis Handler Request" in listing 5.1. As the interface for which plug-ins and observers are defined is standard throughout all extensions, it would be possible to define a web interface which enables rules to be defined in a simpler way. For example - in the case of listing 5.1, an observer and plug-in could be selected from all those available, followed by the event which triggers plug-in execution. If any specific inputs needed to be passed to the plug-in, their content could also be specified.

In summary, a user may configure rules to operate upon their own plug-in implementations and observers to monitor the variables they desire in an environment of their choice. We believe that such customization makes the system well suited to any number of actor based scenarios.

5.9.2 Recording Observations

The content of observations which are asserted to storage is decided by a plug-in author. During actor invocation plug-ins are able to capture assertion content as "resources" to a temporary storage location held in memory within StAR, known as a resource pool. Each resource stored in the pool comprises of its variable name and its associated value expressed as string content. The contents of the pool are copied into an actor state assertion when an actor finishes executing and are serialised to XML. Following copy of the content of the pool, it is emptied by an observer. In this manner content of assertions may be built up over the time the actor is invoked and then finally written to a provenance store and resources are cleared from memory.

5.9.3 Plug-In Development

During the development of StAR we created a number of plug-ins to easily achieve the capture of common actor state observations. The first of these is a SystemTimePlugin, which captures the clock time of an actors host system when executed. The open nature of many of the actor based systems we evaluated in chapter 2 meant that system clocks may not be synchronized between multiple machines. As PReServ itself does not document the time at which assertions are captured (as temporal ordering does not indicate any causal relationship), the SystemTimePlugin is useful. The second we developed was an RRDPlugin, which extracts the content of a number of round robin databases to their numeric values. This plug-in was used at stages during experimentation prior to selection of the TSKR interpretation method. Whilst it documents the highest detail for an observation, it also captures many data points which are at the same level. The final and most complex plug-in, TSKRPlugin, uses the principles described in section 4.4.2 to segment round robin measurements for a number of variables and then intersects them to determine the unique states which were observed over a period of interest. For the actors we make use of, this is typically a service whose execution period is examined to analyse how it performed when invoked.

Developing plug-ins for the registry requires extension of the abstract Java class *Abstract-Plugin* and using the interface *PluginInterface*. We show two such plug-ins, SystemTime-Plugin and ExecutionTimePlugin (which captures the execution time of an actor) in listings

A.1 and A.2 in Appendix A. The abstract class enables a class which extends it to store important observations to disk using a resource pool. As the same resource pool is available within all plug-ins, resources may later be used again within the same or different plug-in (as in listing A.1). Resources can also be added to an output pool (as in listing A.2, for capture as part of a state assertion. The abstract class also enables a plug-in to read policy settings, which may include custom settings for it at execution time. The plug-in interface ensures that each class which implements it has a method `executePlugin(ObserverEvent event)` which describes the action which should be taken when a event of interest occurs.

In the listings given, the system time plug-in stores the time at which events are triggered by using their string descriptions as a key. These are added to both the temporary resource pool and the output pool. The execution time plug-in is then able to read the values which have been captured as resources, in order to operate on them as desired. In this case it is possible for the ExecutionTimePlugin to calculate the total execution time of the actor from the resources given and assert that as part of output.

All of our own plug-ins operate on the two events which are triggered at the beginning and end of a service call. If however a custom set of events need to be captured, possibly in an entirely different (non-service based) environment, then a developer can create their own implementation of an observer. This firstly requires implementing the interface *ObserverInterface*, which ensures that specification of observer events is correctly made. Also, the custom implementation needs to use an *Observer* object to indicate when each event of interest occurs. Listing A.3 demonstrates how we setup descriptions of custom observer events and subsequently trigger them in our own observer, RegistryHandler.

5.10 Visualising State Assertions

The contents of assertions made by StAR to a provenance store contain a description of state observations expressed in XML. Listing A.4 in Appendix A shows an example of the content of such an assertion, which has been built from the TSKRPlugin described in section 5.9.3. In the case of this observation, it is made by a single actor about its execution between 9.29AM and 9.31AM on Tuesday 30 Sep 2008 (unixtime 1222766990 and 1222767080). Here detail of two states which have been observed are documented which each last for approximately half the overall execution time. Each of the state patterns which are listed refer to a segmented region into which each monitored variable falls, which have been specified in a policy (such as the one in figure 5.1). These particular observations indicate that 3 variables are monitored and the state has changed due to a difference in the amount of buffered memory (the third variable indicating a reference to the variable mem_buffers in the policy), whilst the other two variables have remained at the same value. The symbol, label and rule elements describe TSKR attributes which describe the condition represented by the pattern and when it occurs. The support attribute describes the ratio between the length of time that the state occurred to the overall length of time of execution of that actor. Depending on the policy configuration for the actor and the duration taken for its execution, the number of states observed and recorded within an assertion could be far more than we list here.

Over the course of the StAR library's development we have experimented with different ways in which to visualise the information which is recorded as state assertions. It is difficult to interpret assertions by just inspection of their content, especially in our own case where references are made to particular policy components (such as thresholds). However, visualisation is dependant on the content of observations and therefore also dependant on the application which records it. One method we use for visualisation of TSKR involves stacking segmented variable measurements above one another to determine which of the variables has given rise to a particular state observation. The final state observations are put underneath the variable observations and each unique state and variable condition is represented by a different colour. In the example in figure 5.6, red segments indicate when a variable has a high value, orange segments indicate when it was normal and yellow indicates when it was low. A different or higher number of colour choices could be used to visualise a larger number of thresholds or those segments which indicate different properties. This allows TSKR based results to be understood very quickly and how a state is built up from variable values to be determined. A visualisation such as this is built from observations of many executions of the same actor, merged together in time order. Our visualisation library currently produces a single image from a TSKR based assertion, but could be built upon to create a more dynamic environment for exploration of observations. Such an environment could for example allow comparisons of context to be performed for portions of a process visually, rather than using the similarity measure we have defined in section 4.6.3.

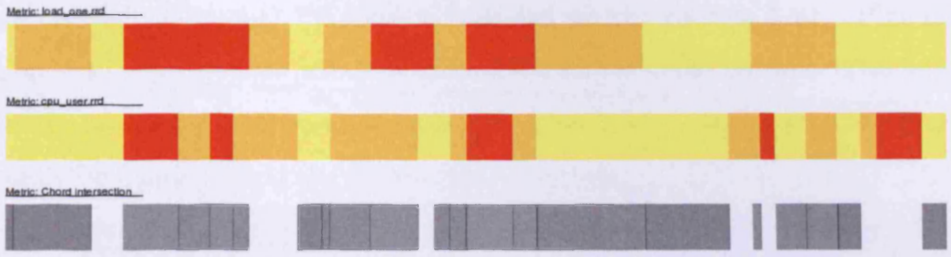


Figure 5.6: Dissemination of state through series visualisation

5.11 State Aggregation Heuristics

Within the proposed registry system, it is difficult to know how information collected from plug-ins should be aggregated. For a simple service which executes quickly, it may be better to return information from plug-ins at a high resolution. For long running services (where execution is over many days), a summary of what has been observed over that time would perhaps be more appropriate. It becomes apparent, that some specification for the heuristics employed within a service is necessary on configuring those observations made upon a service. This should be undertaken prior to the invocation of that Web service. We therefore propose a threshold value to indicate when metrics are to begin being aggregated. For a service where time of execution is below a given threshold, the state information should be collected at the same frequency as that of the plug-ins monitoring the actor. For a time of execution which is above the threshold value, the information collected should be aggregated into a summarised format.

5.12 Further Application Examples

Our approach throughout this thesis has been based on a single motivating example in a service based environment, but the libraries we have created are generic and are able to be used within a variety of scenarios. Here we outline how StAR can be used in a variety of applications.

Non Distributed Processes - If an entire process is executed upon a single machine, StAR can be used to document context too. In this instance, observers would be created to trigger as each individual function is executed on the process host. Plug-ins would be developed to capture data which is of interest to the stake holders. The information would still be asserted to a provenance store as an actor state assertion by StAR and could subsequently be queried using the stores query interface.

Bioclimatic Modeling - Previously we have used our approach to demonstrate the capture of context within a bioclimatic modeling experiment [89]. Bioclimatic modeling attempts to predict the possibility that a species may become endangered or hazardous pests as a result of climate change. Data sources and analytic tools in the environment consist mainly of "legacy" resources which cannot be easily recreated and may not have been originally intended for use together or exposure within a service oriented architecture. In order to standardise communication, they are wrapped by a Web Service which is tailored to that particular resources' inputs and exposed by a standard set of methods. These methods are invoked through use of the Triana workflow enactment and composition tool. The Web Service handler we have created as a wrapper for StAR is able to be used as an observer in this experiment, enabling the transparent capture of actor state for each service involved. Actor states are able to be recorded as we have presented in this chapter, through monitoring tools upon an actors host. In this manner, subtleties in the experiment's conditions can be revealed to better enable experimenters to reason over evidence of past actions. For the work presented in [89], these were use cases to try and improve and understand the performance of the application.

Aerospace Engineering - TENT is an engineering integration system which provides an environment for the graphical composition of workflows for simulations held at the German Aerospace Center [46]. Within the TENT application, context of processes is captured during the simulation of complex flight manoeuvres to answer a number of questions, such as: i) Given some data item, what was the simulation configuration?; ii) Given some parameter, in which simulation(s) has it been used?; iii) What data has been recorded in a simulation with a specific parameter?

Experiments are represented as a workflow which consists of some pre-processing of data to initialise the simulation, a variation component iterating a single parameter to study which is passed to a simulation component and finally a visualisation component.

Following a single simulation several options may occur, the parameter variation component may be notified that calculations are complete, results may be passed to a visualisation component for some direct feedback on results or they may be transferred to a data server for storage (for a more complete description, see [46]). Computation completion states (finished, crashed, interrupted etc), file transfer statistics (time, protocol, number of bytes), name and simulation configuration parameters (bounds, increment to parameter for study) are all captured as context. Capturing the simulation's configuration for example allows experimenters to understand which changes in the system led to a change in results. Some computations in the simulation can take up to two weeks to return results - so repeated execution of an experiment to determine environment settings is not feasible. StAR can capture these context elements, with observers and plug-ins built for the TENT environment - such as a plug-in built to return the completion state for a computation, or transfer statistics for a file. It would also be possible to use a p-automaton to represent the completion states for actors observed over the time a process executed. The p-automaton for instance could be used to attempt to understand how past events (such as interaction from a experimenter) led to a simulation crash. State mining using TSKR would be unnecessary, due to states having been predefined by the application. However, Instrumentation of application actors is able to occur due to direct availability of their implementation to TENT developers.

5.13 Points of Discussion

In our implementation of the architecture proposed, we use a provenance store with the available PreServ libraries to achieve persistent storage for the assertions of state which are collected from our actors. Depending on the definition used for provenance, this could be seen as an inconsistent use of the provenance store due to the type of data which is actually stored. PASOA describes the assertion of state of an actor as the documentation provided by an actor about its internal state in the context of a specific interaction. More specifically, (and recently) the assertion type we use is described as contributing information applicable either immediately before or immediately after an interaction message is sent. If assertions were made which used more than just this information (a long running service or a hypothesis built across several invocations for example), then what is captured is not a true reflection of the knowledge ascertained immediately before or after an interaction, but collected over a period of time. The information asserted as a hypothesis would be information built as a result of several invocations. When using StAR's TSKR plug-in, we make observations concerning the period between two interactions (a service request and response). Our implementation makes use of the store for ease of use (as the observations made can either reflect the content retrieved from tools or a hypothesis) and extends this existing work. We believe state observations from TSKR are appropriate for assertion to a provenance store, as they summarise a single actor execution and hence provide context which is relevant for a particular process. It is possible for a more appropriate storage (according to the PASOA definition) of the data to be achieved through keeping state information in a separate store. A appropriate reference would link to it within an actor state assertion. This is possible when using StAR by asserting references to a remote provenance store which link to the local round robin databases only containing state. Our own experimentation does not however currently employ this method.

Although it is not necessary for the experimenter to know the exact queries which may be later made of state in order to record it, it may serve as an advantage. Through knowledge of the types of queries which may be asked of documentation, it is possible to configure states to be based on particular variables when configuring their policies. This approach may yield states which are more useful for the types of questions being asked during reasoning, rather than attempting to configure policies with variables without knowing their later application.

5.14 Summary

Determining the state an actor was in is extremely important in understanding the interactions which were observed as a workflow is invoked. Although it is possible to determine the sequence of interactions which gave rise to a particular set/piece of data, without knowledge of the context under which those assertions were made it is impossible to fully understand the process described. Interactions between actors occur and are captured, but if the experimenter is unable to answer their queries from the information provided by just interactions, they are left with solutions which manually request monitoring history from each of the actors involved. We have shown a novel way in which an actors state is able to be represented and asserted through extending the pre-existing libraries and model from the PASOA project. Through capture of state data over the time the actor was executing, we ascertain a context for a particular action and assert it as a p-assertion to storage. The assertions captured represent changes in state which occur as the workflow is executed and are later able to be queried from the provenance store. As the state observations are based upon changes that occur in observed variables on the actors local system, it is possible for an experimenter to be made aware of changes in behaviour of that actor without necessarily knowing the events which caused them. Without the integration of a provenance system, the only alternative to this would be to manually determine relevant data from monitoring tools. This is a key difference in our approach compared to alternatives of context collection such as log file analysis or Web service monitoring and is also suited to a variety of open environments.

Chapter 6

Evaluation

We now evaluate how well StAR fulfills the original use cases for which it was designed. We conduct some performance tests when using StAR in a documentation system and demonstrate its use in documenting actor state for a process for the five use cases presented in Chapter 3. These include attempting to predict future actor properties for processes based on previously documented ones and facilitating comparison of actions based on the states observed in a monitored process. We refer back to our earlier workflow originally presented in Chapter 3 when evaluating the use of state.

6.1 Performance Evaluation

We begin by measuring how the State Assertion registry (StAR) introduced in chapter 5 performs when recording actor state p-assertions against a non-provenance enabled and a provenance enabled scenario where only the causal process is recorded. For this performance evaluation, StAR is used to vary the number of and type of assertions recorded during invocation of a data modeling Web Service [8]. The service which we use creates statistical models using Quantitative Structure-Activity Relationship (QSAR) [75], which attempts to correlate biological activity to chemical compound structure described in the data set which is sent to it. The modeler has a number of data processing techniques and neural network and statistical models which take incoming data sets from a client and generate models based upon them. There are a number of modeling algorithms exposed which vary the accuracy of the model produced.

6.1.1 Test Environment

Our performance tests are conducted with a Ubuntu Linux System (1.83GHz processor, 2GB of RAM) operating as the client and an IBM JS20 blade centre machine (2 x 2.1GHz processors, 1.5GB RAM) hosting the service. This provenance enabled scenario has been achieved using StAR and PReServ libraries available from the PASOA project website, using the most recent library version at the time of writing (v0.3). Thorough scalability tests for the PreServ libraries have been performed previously [32, 36].

6.1.2 Experimentation

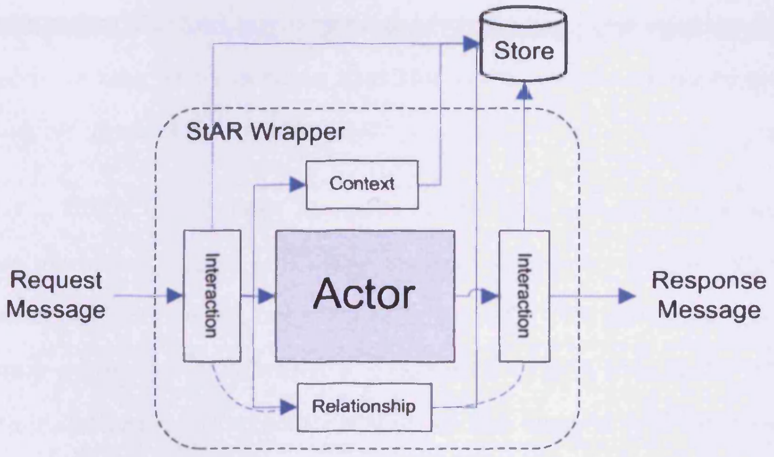


Figure 6.1: Assertion Types and their capture using StAR

Each of our performance experiments are performed under 4 different scenarios, which are each shown in figure 6.1.

1. Scenario 1 (0 assertions): Firstly, to ascertain a usual invocation time of the service no process documentation at all is recorded and therefore no assertions of any type are captured.

2. Scenario 2 (2 Interaction Assertions, 1 Relationship Assertion) : During the second experiment only detail describing interactions with an actor are recorded. Their content describes the messages which where exchanged whilst the service was invoked along with a relationship assertion documenting how each interaction assertion is related to

one another. In our single actor representation, this would be a causal relationship between the interaction assertions representing request and response messages.

3. Scenario 3 (1 Actor State Assertion) : In the third execution, only actor state assertions are recorded for the period when the service receives a request message and returns its response message (the period in which it is assumed the service is executing).

4. Scenario 4 (2 Interaction Assertions, 1 Actor State Assertion, 2 Relationship Assertions): In the fourth execution all assertions described in the 2nd and 3rd experiments are recorded along with a relationship assertion detailing the relationship between the process documentation and any observed actor states. The relationship we document in this case is causal, as we assume that the states which are observed upon an actor were caused by interaction with it.

We carry out a single experiment for each of the four scenarios above, which involves invocation of the modeling service 100 times for each scenario. In the experiment, the size of the data sent to the service for correlation is varied. The datasets are each filled with the same randomly generated data which is structured so it is able to be interpreted by the service. For larger datasets, the execution time of the service will increase as the service attempts to correlate information in this larger dataset.

6.1.3 Results

Figure 6.1.2(a) shows invocation times of the service on variation of its payload size, with the raw data results in Appendix B. We observe that execution times increase as would be expected, with a larger dataset evaluation incurring a longer response time. The majority of the datasets show that full provenance recording is the most expensive, incurring overheads between approximately 4-8% respectively compared with no assertions being recorded. However the 6.7 and 8MB datasets indicate a capture time for all documentation that was less than observed whilst capturing documentation of interactions. This difference is fairly minimal compared to the time to invoke the service at just under 3 seconds (or 1.3% of time of invocation), so could be a result of machines clearing local memory, or the amount of work our TSKR plug-in had to do to interpret observations of state. As each different actor in our experiment uses a different registry configuration policy with details unique to

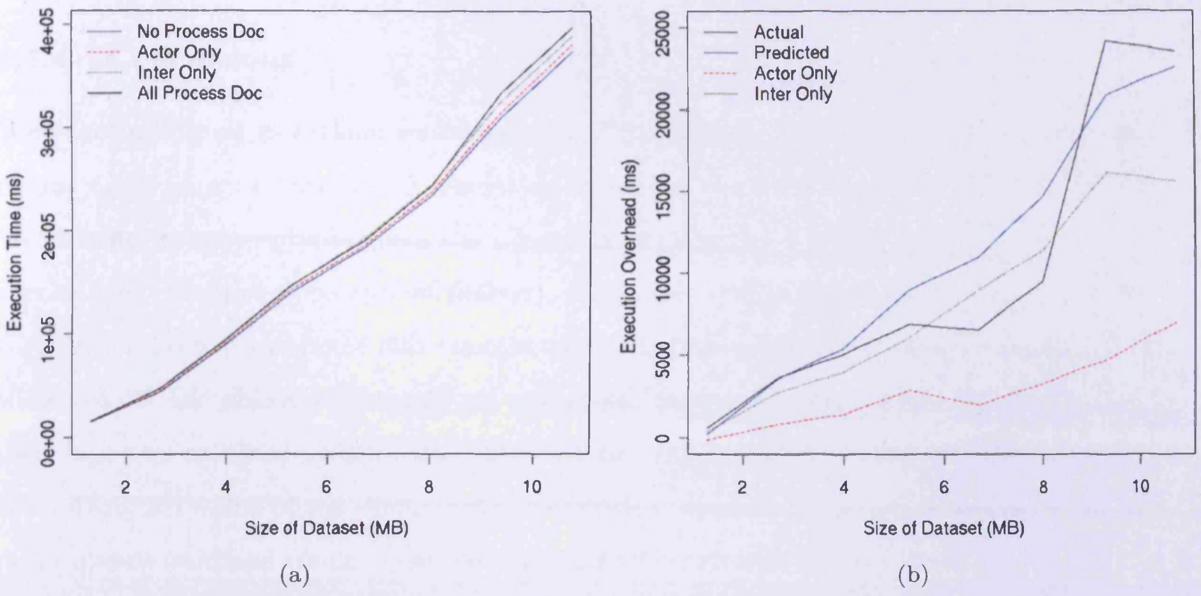


Figure 6.2: a) Execution time and b) execution overhead on use of different payload sizes with StAR

the host, this also could be the cause of the differences we observe. In our policies, these differences are in the thresholds set for variables and are based on previous minimum and maximum values observed upon that actor, an example of which was presented in listing 5.1. Figure 6.1.2(b) shows the actual overheads incurred through capturing assertions in each scenario and a predicted invocation time for scenario 4. This predicted overhead is the combined measured overheads in scenarios 2 and 3 and is used as a comparison for the actual overhead observed when recording all process documentation. The predicted overhead is shown to not be accurate in several cases, indicating that the overheads of recording all assertions is not a direct product of recording evidence of actor state and the causal processes individually in our system. The small differences from the predicted trend for 6.7 and 8MB datasets we have noted above, cause a sharp rise to the measured overhead for 9.3MB dataset. As all process documentation includes an additional relationship assertion (relating the actor state assertion to the causal process) than interaction and actor state assertions recorded alone, a longer invocation time than that predicted can be explained. However, lower than expected times could be a result of the random nature of the datasets which we created for use with the data modeling service and could lead to lower values for the base (no process documentation) which we calculate all overheads from.

6.1.4 Conclusions

When recording all assertions we observed a 612-24193ms overhead (or 4-8% of the invocation time) above recording no assertions at all for our data modeling (QSAR) service. We consider this acceptable given the overall invocation times which were observed for the service used (between 1500 and 440000ms). Given the level of detail about the process captured when both provenance and context enabled, this overhead is relatively small. Both these results are obtained through an automatic instrumentation with our system, so if used in a service based system, the time cost to context enable would be minimal. Given the subsequent value of the documentation which is captured it is likely that such a small performance overhead would be acceptable to administrators of such services.

6.2 Use Case Evaluation

We now show how our capture model is capable of satisfying the 5 original use cases which motivated its creation in section 3.4. We use the State Assertion Registry (StAR) to automatically record assertions of interaction and state to a provenance store for each of the services in a workflow. The workflow converts an averaged brain image (determined from the average of intensities of a number of MRI scans) gathered from a collection of high resolution anatomical data into graphics files showing slices of the brain.

6.2.1 Scenario Review

The process we use to evaluate StAR against the original use cases to create population-based brain atlases from high resolution anatomical data, previously introduced in section 3.2 and shown in figure 3.2. The data we use is available from the Functional Magnetic Resonance Imaging (fMRI) Data Center*. The workflow takes some input image data and produces a slice image subject to the axis specified as an input. Each of the processing steps in our workflow is a Web Service hosted upon a one of 8 machines, with images and data slices being exchanged between the client and services following successful invocation of each step. Each of the services in the workflow is hosted on a IBM JS20 blade machine (2 x 2.1GHz processors, 1.5GB RAM) and the provenance store used for them all is a Sun x2100 machine (1 x 2.2GHz processor, 4GB of RAM). We show our experimental setup in

*http://www.fmridc.org/

figure 6.3. Each of the host machines also has a ganglia monitoring daemon (M) to collect metrics and our State Assertion library used to determine the state of the machine at a particular point in time.

When the workflow executes, a client makes calls to each of the remote services to perform the entire process. A single action is performed by each of the services used and a set of assertions are recorded for it by StAR. The client adds an additional tracer string to each call to services, which acts as an identifier for each process. This tracer is also propagated to service response messages by the handler operating on the service. Using a monitoring policy, it is possible to alter the content of the observations which are captured. We make use of both the system time and TSKR plug-ins introduced in section 5.9.3 to document the execution time and any observed states for all services. We consider the features observed in the variables we monitor as being able to continue for longer than just the duration of a single action. Therefore, as the process is performed 1200 times, each subsequent invocation is delayed to allow the systems used in the workflow to recover. This is undertaken in order to ensure that the effects of one invocation of the process are not observed as part of the process documentation which immediately follows in our experiments. We use this particular workflow because of its usage within the provenance community as a comparative tool for the features of different provenance systems. The total number of actions which are executed on each invocation of the process is 12, comprised of 4 align_warp actions, 4 reslice actions, 1 softmean action and 3 slicer actions. Due to the order in which requests are made, there is never a case where two services are executed concurrently upon a single host. We chose not to perform stage 5 of the original challenge workflow due to the extremely small execution time of the last function (convert) and the likelihood that capturing context for such a small interval would not be possible.

Context Recording Policy

The policy we use for recording of context makes use of the plug-ins introduced in Chapter 5. We use a system time plug-in to record the request and response time of services and our TSKR plug-in to mine actor states. The resultant actor state assertions identify the interval over which the actor is invoked (the time between request and response messages) based upon TSKR mined series from the segmented values using three variables: bytes in per second, one minute load average and the amount of buffered memory. We chose these

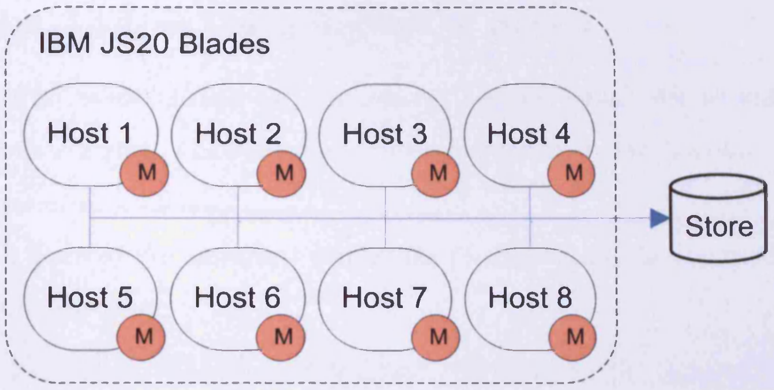


Figure 6.3: The workflow environment used for experimentation

variables as they provide a variety of interesting features which are to be monitored (such as network activity and the current amount of available memory), along with sufficient variation in their value to allow segmentation. There are however many more variables available from other monitoring systems which can use the same underling technologies.

Querying Collections of Context

Our analysis of the context for a particular process requires that we first obtain documentation which describe each causal process related to a particular data item. We use a provenance query [54] to collect the set of relationships which describes how each atlas slice which is created from the workflow execution. The provenance query functionality exists within the PReServ library we make use of. Given that we have documentation in which each process is described, we are able to query detail from the causal relationship set to find the actors that were involved in each process and their context over the intervals they were invoked. Our particular workflow environment only makes use of 12 actors, but where an open environment is used - such as a grid, it may be possible that many more actors are made available to offer the same functionality. We retrieve a local copy of context for each actor in order that we might easily interpret our results and use it throughout our use case evaluation. Within the entire collection of process documentation ($PROCESSES$), we refer to a function $PROCESS(n)$ as returning the entire causal process for process number n and $CONTEXT(x, n)$ as returning the context of an actor x during process n.

6.2.2 Context Analysis Using Records of Action

Use Case 1. *The administrator of the convert service would like to understand why a particular documented state has been made more frequently in the last day.*

Use Case 2. *A user of the workflow would like to understand how a particular actor arrived at a documented state.*

Solutions to our first two use cases require interpretation of the entire collection of context records available for an actor. These are retrieved from recorded process documentation using information about the actions which have been invoked. We can determine when an actor has changed states by querying records of actor state for a process and then querying detail of that same actors state for the process which was executed immediately after it. If the state which follows matches the first, we assume that state has not changed for the actor. However, if there is a difference in state, we assume a single change of state occurred for that actor caused by executing the functionality provided by that actor. A description of our method of building transitions is given in listing 6.2.2. We give an example of what this looks like in figure 6.4 for information collected for the reslice action during our experiments. The coloured reslice boxes show the first and final actions performed and the observed states for the action over this time are shown below them. Each box represents an action which plays a part in a single process, so we show the state transitions in total for 4 processes. As it is executed multiple times over the period it is made available, observed states are recorded as actor state assertions. Over the course of the 1200 invocations of the brain atlas workflow, we obtain a history of state transitions for each of the actors used. In our own experiments, only one action is executed at a time on each host machine and a causal relationship is captured between a resultant state and the interaction event which caused it. We will always therefore be able to determine the event which gave rise to a particular state. Our experiments therefore allow us to determine which instance of the process invocation gave rise to a sequence of observed states, through querying the causal process related to a state. However, it is possible in other experimental setups that multiple actors could be co-located upon a single host and that multiple actions could be performed simultaneously. In these cases inspection of records of interaction will reveal the invocation of which actor caused a state (or state sequence) to occur.

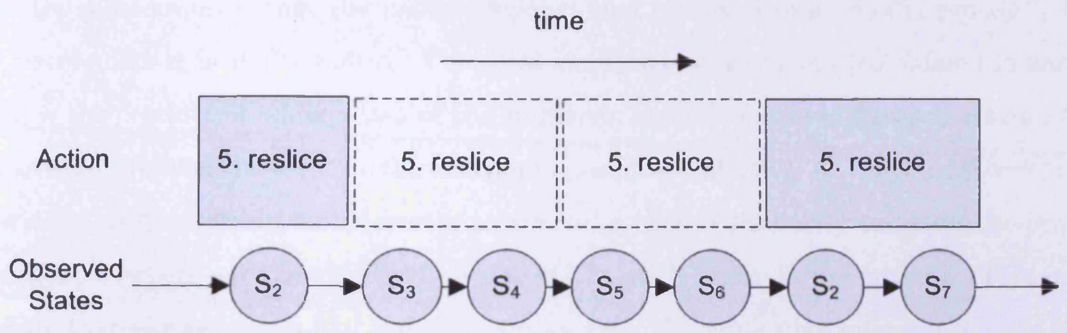


Figure 6.4: Observed state transitions for reslice action 5

Given that all state transitions for an actor have been found, the frequency at which they occur over a particular period is able to be calculated. Our state descriptions include the time over which a state occurs, meaning that experimenters are able to build transition lists for states over particular intervals.

```
transitions[] = $\emptyset$
find $actors \in PROCESSES$
    foreach $actor \in actors$
        tl = $\emptyset$
        for i = 1 to $COUNT(PROCESSES)$
            to = find state in $CONTEXT(actor, i)$
            from = find state in $CONTEXT(actor, i - 1)$

            if(to != from)
                t = new transition(from,to)
                tl = tl $\cup$ t
    transitions[actor] = tl
return transitions
```

Listing 6.1: Determining past actor state transitions for each actor

6.2.3 Comparison of Past Processes

Use Case 3. *A user of the fMRI workflow would like to perform multiple invocations of the workflow and draw comparisons in the observations of state made for actors involved.*

In order to compare past processes in our scenario we determine the similarity of a single state for each action within the process against that observed in a "model process", with our results shown in figure 6.6(a). The total similarity of a process (as defined in section 4.6.3) is the product of adding each of the similarity values calculated for each action in the process and dividing the total by the maximum possible similarity. As our workflow features 4 services, with a total of 12 actions in a process, a perfect similarity value for the process would be 12 (where a perfect similarity value is 1 for each action in the process). This would indicate that two processes had the same states, over the same time intervals, for the same actions. A scientist may then use these distances as a filter to locate interesting processes from a large collection of process documentation. We show an example of this procedure for the first 5 steps of the workflow in figure 6.5, where the 1st and 4th process are compared. The distance between each of the states' (which are compared) associated pattern elements are inspected to calculate their similarity. This method of pattern inspection is applied to every single invocation of an actor in each process to calculate every similarity score against the model process.

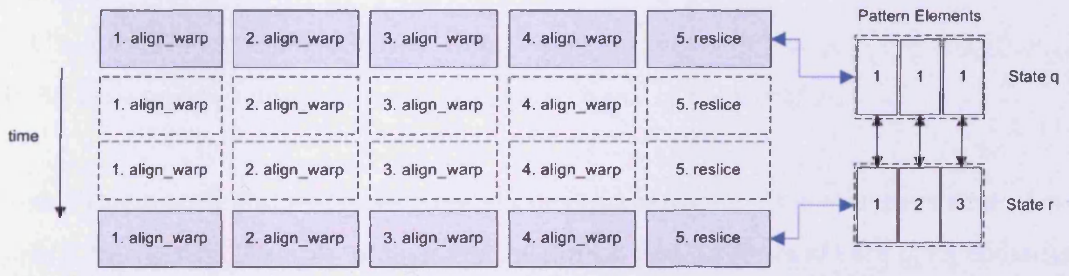
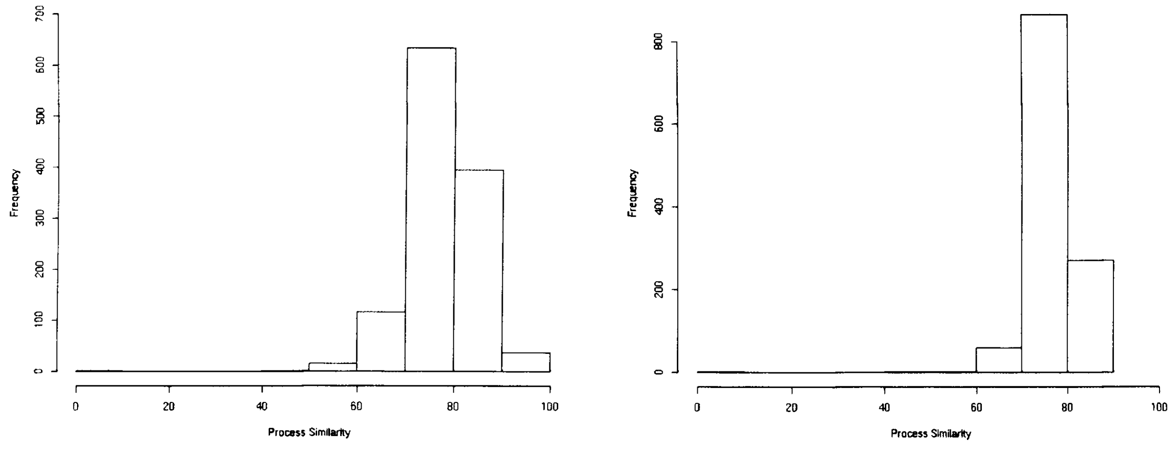


Figure 6.5: Measuring Similarity of Past Processes

Our results in figure 6.6 show the distribution of process similarity values. For our scenario, we are able to see that the total documentation can be reduced dramatically when searching for either those processes with a high or low similarity ($\geq$ 0.9 or $\leq$ 0.6). This corresponds to 3% and 1.3% of all of the documentation recorded in the experiment. If we look at the lowest similarity processes ($\leq$0.5), we can reduce this figure even further to 0.3% of all documentation. Figure 6.6(b) shows the same processes being compared, but with an average similarity value corresponding to the observation of multiple states within each invocation. We reduce the number of interesting processes with a large and low similarity value ($\geq$ 0.9 or $\leq$ 0.6) by doing this, but increase the number of processes



(a) (b)

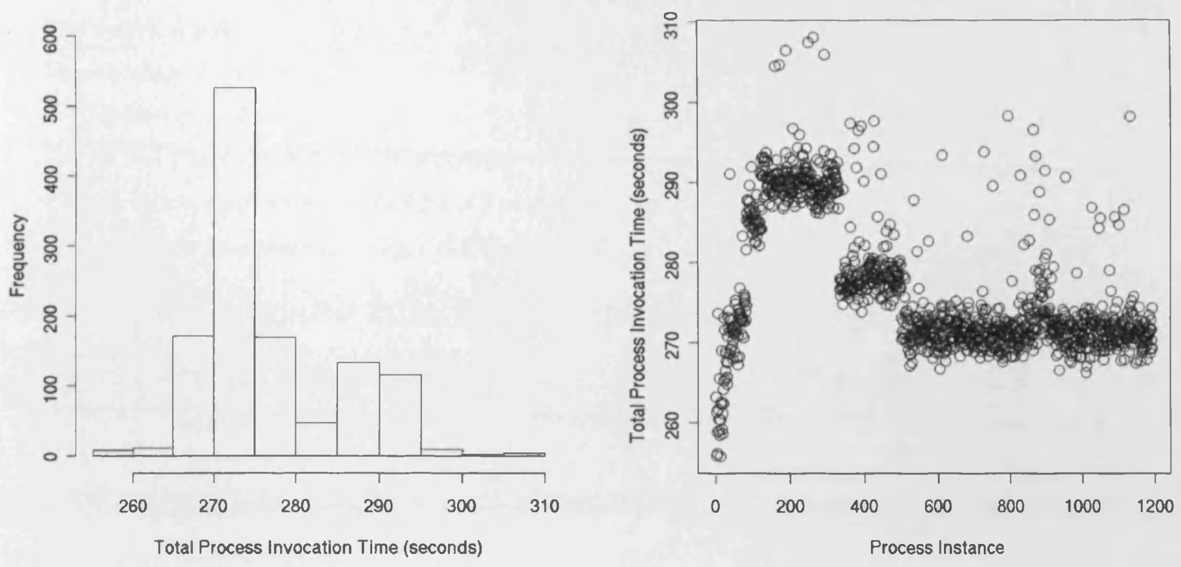
Figure 6.6: Distribution of process similarity values when compared to model process

falling into the 0.7-0.8 range. Without the documentation of actor states, it is perfectly feasible that the navigation of records of all 1200 causal processes (totalling 56MB's worth of XML documentation to be queried in our own experiments) would have to be navigated manually.

Use Case 4. *The experimenter would like to ensure that differences in the execution time of each process are minimal over multiple invocations of the workflow.*

Using the request and response times of our process's actors, the execution time of every process is calculated. For each process the sum of invocation times of each of its constituent actions is calculated. We show the distribution of our results for the 1200 processes we invoked in figure 6.7a. The majority of these times spread across a range of 15 seconds (265-280 seconds), which can be expected with the large amount of service invocations which are being undertaken. A smaller number of outliers increase the overall range to 60 seconds, which is still acceptable given the volume of processes which we execute. Figure 6.7((b)) shows when these invocation times were observed over the process instances. The majority of longer running processes occurred in the initial set, with later times settling to an average of about 270 seconds. An experimenter therefore would be able to conclude that over the course of experimentation differences in execution time of the process are minimal.

```
times = $\emptyset$
for i = 1 to $COUNT(PROCESSES)$
    find $actors \in PROCESS(i)$
```



(a) (b)

Figure 6.7: Invocation times for the process and their distribution

```
        foreach $actor \in \{actors\}$
            time = find invocation time in $CONTEXT(actor, i)$
            times = times $\cup$ time
    return times
```

Listing 6.2: Querying Process Invocation Times

6.2.4 Prediction of Future Actor Properties

Use Case 5. *An experimenter would like to determine the most likely future conditions under which an actor will be operating.*

A likely set of future properties can be determined for an actor by using a transition table (as described in section 4.6.2) for each actor in our process. A state prediction is made for each actor after a single process is executed, based upon the last known state of that actor and the most common previous transition observed for that state. The actual state which was observed as part of the next process invocation is added to the table of transitions, gradually building a state transition history. For this experiment, results are based on the entire set of processes being executed twice. We show a listing for the experiment below.

```
  tmatrix = $\emptyset$
  find $actors \in PROCESSES$
  foreach $actor \in \{actors\}$
    prediction = $\emptyset$
    for i = 1 to $COUNT(PROCESSES)$
      this = find state in $CONTEXT(actor, i)$
      last = find state in $CONTEXT(actor, i-1)$

      if(this != last)
        t = new transition(last,this)
        tmatrix = tmatrix $\cup$ t

      if(prediction == this)
        m = m + 1

      prediction = lookup highest prob from tmatrix

    matches[actor] = m
return matches
```

Listing 6.3: Predicting Future Actor States

The resultant automaton we derive for the slicer actor (action 10) is shown in figure 6.8, which is found following all invocations of the process. The direction of each arc shows the initial and subsequent states and the values on each refers to the probability that a particular transition will occur (over all transitions). We also give the associated transition table for this p-automaton in table 6.1. As our experiments use a policy which segments variables into 3 distinct series and we use 3 such variables, the total number of possible states which may be observed is 27 (or 3^3). Our transition table indicates that many of these possible states were not observed however, as with many of the actions we make observations for. We observed less than half of all possible states during this experiment, indicating the vast majority of transitions occur between a few states. Most transitions occurred between states s_0 and s_1 whereas transitions between states s_{10} and s_{12} occurred very few times. This could be attributed to the host system remaining relatively stable across the time experimentation was performed, resulting in few changes in the variables observed. We can see from the state automata in figure 6.8 that the majority of transitions were initially made between states falling between s_0 and s_7. The bottom part of the

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}
s_0	0	0.661	0.276	0.005	0.016	0.026	0	0.005	0.005	0.005	0	0	0
s_1	0.632	0	0.281	0.018	0.041	0	0	0	0.029	0	0	0	0
s_2	0.714	0.238	0	0	0	0.038	0.01	0	0	0	0	0	0
s_3	0.25	0.75	0	0	0	0	0	0	0	0	0	0	0
s_4	0	0.9	0.1	0	0	0	0	0	0	0	0	0	0
s_5	0.667	0.222	0.111	0	0	0	0	0	0	0	0	0	0
s_6	1	0	0	0	0	0	0	0	0	0	0	0	0
s_7	0	0	1	0	0	0	0	0	0	0	0	0	0
s_8	0	0.833	0.167	0	0	0	0	0	0	0	0	0	0
s_9	0	0	0	0	0	0	0	0	0	0	0.182	0.727	0.091
s_{10}	0	0	0	0	0	0	0	0	0	0.6	0	0.4	0
s_{11}	0	0	0	0	0	0	0	0	0	0.778	0.222	0	0
s_{12}	0	0	0	0	0	0	0	0	0	0	1	0	0

Table 6.1: Transition table for the probabilistic p-automaton shown in figure 6.8

figure indicates a collection of states which were never left (due to the arc pointing in one direction to s_9), indicating they were the final set of states observed from all those made. From inspecting the patterns held for each state, we can tell this group show those states where the network transmission (bytes_in) has risen to a maximum value, where all other states are those observed with a minimum value.

Figure 6.9 shows the percentage of matches for predicted states to actual ones for our approach along with a history based and simple monte-carlo prediction. For the history based prediction, we randomly chose a state based on those which have already been observed for that actor. Each point represents the match rate for a single actor in the process. All machines were able to consistently predict states with variable success rates (35-85%), which was always above that of the monte-carlo based predictions. The monte-carlo approach was even susceptible to suggesting a state which was never observed, ultimately producing extremely poor results. The average trend indicates that as more transitions are observed, state becomes more difficult to predict in the future. This is due to the increased complexity of the model which is built when more transitions are found. However, this trend is not conclusive due to the amount of available data. It is likely that a more sophisticated analysis of the transition patterns leading to a state could further increase this success rate. This would require looking at transitions over a large number of invocations of the process and seeing if there were any patterns leading up to a particular state. If so, then a prediction could be made based on a pattern analysis, rather than the frequency of a transition. The two outliers at 1500 and 2200 transitions represent the actor softmean. They suggest that the highest amount of processing is undertaken by it, leading to a large

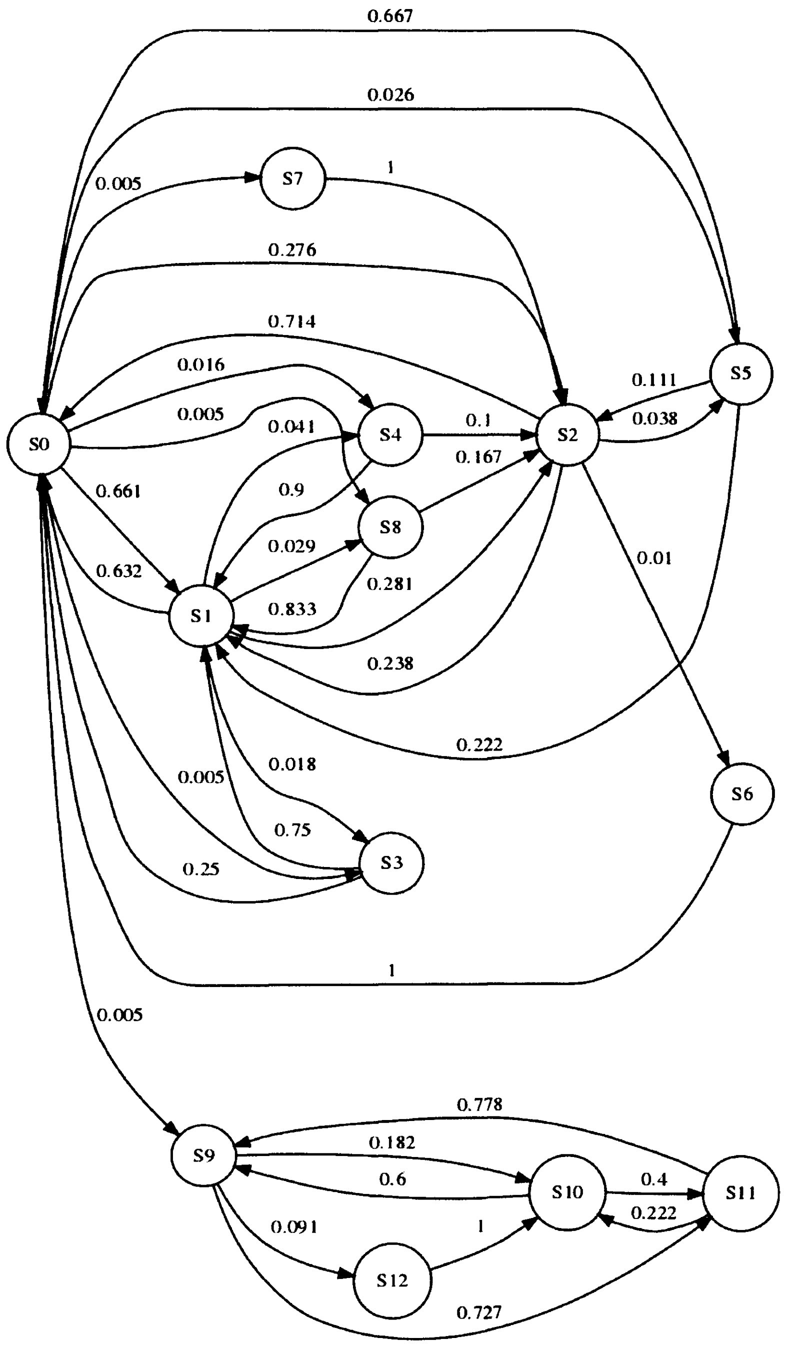


Figure 6.8: P-automaton based on observed state transitions for slicer action

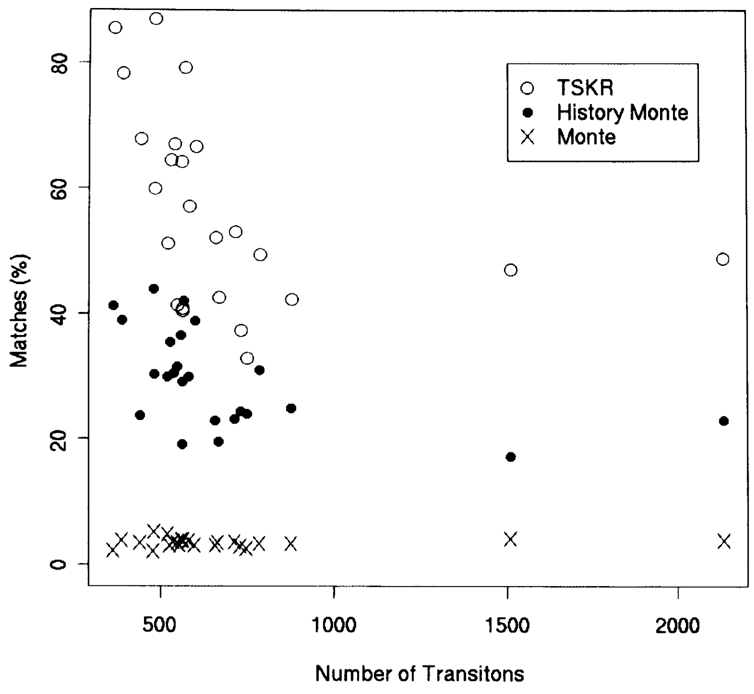


Figure 6.9: Match rate of predicted states to those observed

amount of state transitions. Using our approach, the scientist executing the process is able to form a hypothesis detailing the most likely states to occur for each actor.

6.3 Answering Additional Queries using Context

Several additional questions can be answered through the use of both static and dynamic metrics captured using context. We provide a selection of examples of these below which can be satisfied from the context captured during our experimentation:

1. Static Data - That data which does not change throughout the lifetime of an actor. As a result, static data need only be recorded once per process during its execution. Such data items have been previously investigated, and include: (i) Per-Node: node identity, operating system, etc.; (ii) Per-Actor: actor identity, name, owner, version, capability, etc. Such information is similar to that published by an actor to a registry service in a SOA.

   - **Which user created image X?**

Through use of the plug-in architecture for the State Assertion Registry, a user identification plug-in details which scientist invoked a particular workflow using a client. In the case of our experimentation, each experiment was performed by the same user "scmimw", which is an additional element (meaning it plays no part in TSKR state derivation) of each actor state assertion recorded.

- **Which operating system was used on actor Y?**

  Through use of the plug-in architecture for the State Assertion Registry, a operating system plug-in details the operating system used when a service in a workflow was invoked. This would also be captured as an additional element recorded as part of an observation. Querying this value from an assertion made by any actor shows the OS to have been Linux version 2.6.18-1.2747.el5 (Red Hat 4.1.1-30).

2. Dynamic Data - That data which may change during the lifetime of an actor. It is therefore necessary to record this data at periodic intervals over the lifetime of that actor. Such data items may include: (i) Per-Node: memory usage, network traffic, etc. (ii) Per-Actor: service execution time, uptime, availability, etc. Such dynamic data is usually derived from other, less complex recorded metrics.

   - **Which process/procedure took longest to execute?** On comparison of all retrieved invocation times of actions which were executed, we are able to see that the softmean action has the highest average invocation time. This is understandable due to the complexity of the operation it is performing which averages all the results of previous actions in the fMRI process into a single image. The align_warp action has the next longest invocation time which also performs many comparisons to determine how each new image is to be adjusted to match the reference image.

     We calculate the maximum execution time by determining the sum of invocation times of each action which were part of a process. Based on this we are able to determine the maximum of these values, which is the 269th process which was executed and has a total invocation time of 308.1s. From inspection of the invocation times of actions/processes we can see that this was the net result of prolonged invocation times across all actions, which seemed to stabilize in later processes.

- **Which process/procedure uses the most memory?**

  We calculate which process uses the largest amount of memory by determining which action sustained the longest amount of time in a state with the memory recorded at a maximum value. This maximum equates to observations which have been made over the highest threshold set as part of the actors provenance recording policy. The process with the largest sum of this periods therefore is taken as the process which uses the most memory. In our observations, this was the softmean actor which remained in states with the highest memory value for a total of 16264s over all 1200 invocations of the process. The remaining actors had very few observations with states in a high memory state, which indicates either that; 1) the thresholds which were used were inappropriate; 2) the states of maximum memory usage occur for less than the minimum time at which an observation of state is made (10s) upon the other actors.

  The maximum time a single process was observed to be in a state of high memory was 111s in the 119th process, all of these 111s were observed as part of the softmean actor's execution.

- **Which process/procedure has the highest load?**

  We calculate which process has the highest load (using the same technique as with memory) by determining which action sustained the longest amount of time in a state with the load recorded at a maximum measurement. In our observations it was again the softmean actor whose load was recorded to be in a maximum value for the longest period of time (10239s over all 1200 invocations). The align_warp actors also had periods of high load, but all remained under a 400s period for all processes.

3. State Queries

- **Which host has the largest number of transitions of state?**

  We are able to calculate the host with the largest number of transitions of state through application of our principles of time series representation (shown in Section 4.4.2) and mining the results to determine which host most frequently moved between states. From our results we see this is the softmean actor (on host hgrid08) once again with a total of 2137 transitions for all processes. This allows us to assert another important fact about this actor. Not only does it remain for the longest period in a high memory state, but it is also the most frequent to change between its states. Again, we presume that threshold adjustment could alter this number of transitions, but our results are based on average thresholds for each actor.

- **Which process/procedure has the largest amount of state "oscillation"?**

  We describe state 'oscillation' as an actor holding the same state as it held immediately prior to the state preceding the actors current state. In other words, oscillation is caused if an actor moves from one state into another and then back into the original state. Although not an indication of unwanted actor behavior in itself, it does again show that policy thresholds for variables may not be set at values which are suitable. We measure oscillation by stepping through each observed state and determining if the last but one state is the same as the one currently being inspected. In our tests, the softmean actor is shown to have the largest frequency of oscillation at a level of 353 transitions - which is over 7 times any of the other actors. The 80th process is shown to have the largest state oscillation across the entire process (at 4 transitions) and unusually the softmean actor plays no part in any of these observations.

- **When did the largest amount of state "oscillation" occur?**

  As we've been able to previously find the process in which the largest amount of oscillation occurred, we are subsequently able to query the context which is captured for each of the actions in the process to determine the time over which it was carried out. By querying the first and final actions context recorded for the 80th process, we find the start and end time were at unixtime 1222785080 and 1222785290, giving a total execution time of 210 seconds.

6.4 Scalability

Exhaustive performance testing of the State Assertion Registry on a multi-node system such as a cluster was felt to be unnecessary, as our system is limited by the speed of its central provenance store. Thorough scalability testing for the store has already been carried out [32]. The results of these tests detail the impact of the store size for a variety of controlled recording scenarios, with the results showing that the store's implementation was scalable. As the only additional overhead which is incurred is through the use of our StAR wrapper, we restricted our own performance tests to a single machine. Our interest did not lie in measuring how well the provenance store coped with being stress tested during the recording of large numbers of assertions to single repositories as this has been extensively tested elsewhere [36], but with the resultant overhead incurred by a single actor when making use of our system.

6.5 Query Interface Limitations

Our tests revealed significant performance problems when querying actor state from a provenance store. The PReServ software (v0.3) we adopt currently uses an XQuery interface to query recorded process documentation, which we found to be extremely slow when dealing with large quantities of provenance records. In situations where only a few specific queries are to be made of recorded documentation this is not so much of a problem, but our experiments required all documented actor state to be returned from a provenance store. This was necessary so that the context for each process which was invoked could be compared against one another. For the 1200 processes, total experimentation took 222220 seconds (2.5 days) to execute in our setup and 8632805 seconds (10 days) to query the information back from the store. Such a query overhead might not make this comparison entirely practical for projects wishing to adopt our model of context using our current implementation.

Comparisons against a local and remotely hosted store were made to determine if the large response time was down to the remote nature of our queries, revealing it to be much the same. By manually navigating through a local copy of the captured process documentation and thus bypassing the query interface, we were able to query the same documentation in 98 seconds or 0.001% of the original query time measured using XQuery. This involved extracting actor state assertion elements from the XML assertions we had stored using knowledge of its Document Object Model (DOM) structure. This huge reduction in query time suggests the interface to the provenance store could have its performance massively improved upon in a future version. Situations such as our experiments where large amounts of process documentation need to be inspected would be able to achieve significant speed improvements if another, more efficient interface alternative were offered. An alternative to the approach adopted during our experiments is to record to multiple provenance stores and query records from each. PReServ offers the ability to do this through placing links to other stores within process documentation so that relevant records may be found on query. However, even though some improvement could be achieved by recording to multiple stores, our investigations suggest an underlying problem would remain through the use of the XQuery interface.

6.6 Summary

In this chapter we showed the State Assertion registry (StAR) to both be efficient and capable of solving the original fMRI use cases. Our performance experiments showed a 4-8% overhead when using StAR to document the process, which we believe to be acceptable given the additional use cases able to be satisfied once provenance enabled. Our use cases for the context model included analysing records of context using documented actions, facilitating comparison of past processes and predicting future actor properties. We also demonstrated StAR's versatility through answering several additional queries using the data which was collected during experimentation. Our experimentation has been based on a single motivating example, but we note that our approach is general. StAR could be used to document context in a variety of application scenarios including our own method of mining for states.

We noted that with the query interface currently adopted, answer of each use case is limited by the time it takes to retrieve large amounts of documentation from a provenance store. However, we also observed a much reduced query time when bypassing the interface, indicating that a much faster response time is possible once the current interface has been improved. This also indicates that the approach we adopted can also be used where more relaxed access is possible to monitoring installations. As we have already said when introducing the original experiment in section 3.2, we have assumed a worse case scenario in terms of availability of information. However, the techniques we have adopted for interpretation of context are equally applicable in a variety of cases.

Chapter 7

Conclusion

A scientist performs an experiment by running a set of activities multiple times within an open system. Minute differences in the execution environment manifest themselves in an extended time it takes that workflow to execute. The scientist is able to observe differences in the execution time, but is ultimately unable to understand how the differences arose due to the open nature of the system and a lack of documentation which is recorded about the experiments host systems.

This scenario is just one example of how a scientist may choose to use functionality in a remote environment - often to enable access to facilities which are far superior to their own. It also demonstrates a problem given the use of black box systems to carry out functionality, where little is known about the system on which a process is executed. Scientists are left unable to answer many of their specific questions following experimentation.

The grand vision which *provenance* has introduced, that evidence recorded which describes a process should be thorough enough to answer all possible future provenance queries, would be the ultimate goal in solving these scientists' problems and fulfilling their confidence in their results. However, without understanding the circumstances such processes have been subject to - the *context* in which they have executed, not all provenance queries can be answered.

This dissertation has sought to address this problem, that existing solutions built to understand processes post execution are not thorough enough to satisfy many provenance problems. We have shown that through structured representation and collection of context this problem can, to a large extent, be solved.

The rest of this chapter now summarises the key contributions presented in this dissertation and possible future work which we believe would be valuable.

7.1 Contributions

7.1.1 Context, Process and Provenance

Our first contribution was to make clear the distinction between *context*, *process* and both of their relevance to *provenance*. We showed from current literature that provenance is a type of query which is used to instill confidence about how a particular process has been conducted. Context was introduced as the situation or setting of a process which had some bearing on the process outcome and therefore is seen as a constituent part of answering a provenance query.

Through our literature review, we observed that provenance work to date focuses on describing how a process is constructed from sequences of steps to perform some desired functionality, resultant from their combination. Several approaches to this problem have been achieved through structured representation of observations of system interaction, often built to solve specific problems for a particular domain. Through the use of process documentation, in the first instance records of each of the steps used to conduct processes are captured as evidence to support provenance queries. However, context for these processes is not captured in a structured manner and given little attention as a credible method of contributing in the answer of provenance queries.

Making this distinction also enables us to determine the best method to collect records of both action and context to sufficiently answer provenance queries. The problem of documenting processes has largely been solved through research to date. However, we observe that much of the necessary content for records of context is available through use of existing systems and easily adaptable for use in current provenance solutions. We assert that through separate, structured representation of both process and context, we can answer many more provenance queries than through use of either type of information alone.

7.1.2 Actor State and Provenance Automaton

Functionality which is chosen to be executed in an open environment over other, local alternatives is usually complex. Due to this complexity, executed functions usually take some time to yield results. We noted that the values of recorded context components in these systems can change over this time, meaning the associated process needs to be documented in such a way as to record the changes that have occurred. The concept of actor state was introduced and a model was created to represent and understand such changes, based on outstanding requirements that remain after solely documenting interactions and their relationships in a system.

Actor state allows for the same collection of values observed as context to be distinguished and compared against one another over a period of time. This is important for systems which document processes where documentation of context is usually collected at a single point in time, rather than across intervals for each participant. This also means that changes in these collections are able to be identified even if the experimenter does not understand the relation which these changes hold with a process, which is important given that future queries may not be known at the time a process executes.

The provenance automaton (or p-automaton) was developed to represent the chain of transitions which the state of an actor in a system might undergo over a period of time. Its one key difference to that of a finite state automaton is that it represents a sequence of transitions that have occurred until a particular point in time. Therefore it may not represent behavior for an actor for all time, but could be used to determine what a future state of an actor may be. A p-automaton can be built using a collection of observations of actor state, to model an actors most current state and the sequence of past transitions that led to it. By using a number of these provenance automaton for all the actors in a system in which a process is executed, we represent the history of context of a complete process.

The intervals over which actors execute functionality can also be represented using time series. We demonstrated how use of the Time Series Knowledge Representation (TSKR) could be used to determine unique states from a collection of observations. This helps in mining unique states from several observations of the same set of variables, collected as actor state. Due to the pattern representation TSKR adopts for the series it represents, it also allows for comparison of specific states. We demonstrated how this could be applied through specification of a similarity measure for states observed upon the same actor.

7.1.3 Documenting Actor State

Our final contribution was to show how documentation of actor state could easily be achieved through existing provenance recording strategies. Recording evidence of causal process execution has largely been achieved in a variety of forms to date, so we chose to enable one of these approaches to model the situation of these processes also. The P-Structure was chosen as a method of representing all process documentation. By demonstrating how context can be applied using the P-Structure we indicate how the extensive number of applications which have previously adopted it can record context in their own particular scenarios.

We developed the State Assertion Registry (StAR) to capture assertions of actor state in an automated manner. By using the P-Structure to represent assertions of state, we can capture detail of the causal processes through existing process documentation software. Construction of the software had to be pursued with a consideration of a number of trade-offs, such as the amount of actor state data available or if it is exposed by its owner. StAR was designed to be able to be customised, due to the variety of applications with which it might be used. Through using observers to define events of interest, plug-ins to describe information sources and policies to describe how StAR behaves at runtime, it is capable of representing a wide variety of complex scenarios. Our own observer and plug-in implementations allow capture of actor states in a transparent manner for service based systems. Once stored, we are able to answer context queries from the documentation captured due to our use of PreServ and the query interface it has.

7.2 Experimental Findings

We used StAR to conduct performance experiments against a data mining service to determine the impact of its use. We found that an overhead of between 4-8% was incurred on the invocation time when using StAR to document the process. This involved both collection of records of action and context and therefore we considered it acceptable given the value of the documentation captured for answering provenance queries. We also observed in these experiments that documentation of actor state is usually collected in less time than documenting the interaction that occurred, most likely due to the number of assertions which are necessary to document each of these scenarios (1 vs 3).

The main contribution from our experimentation however was in answering our original 5 use cases which served as the motivation for the definitions of actor state and provenance automaton. Using a service based implementation of the Functional Magnetic Resonance Imaging (fMRI) workflow as an exemplar for the provenance automaton, we recorded both records of action and context to a central storage repository. From the recorded information that was collected over 1200 invocations of this workflow, we demonstrated how context can be understood using records of action, how evidence from past processes could be compared against one another and how states may be predicted for actors in the future. None of these use cases could be satisfied in a non-context recording environment and even if elements of context were able to be captured, a significant amount of interpretation of this evidence would be necessary to answer these questions. The actor state model serves as a natural method of understanding how actors are affected over time and through use of provenance automaton we can model and interpret their behaviour. Further satisfaction of a number of additional provenance queries was also possible due to the extensible nature of our system, when evaluating a number of examples.

7.3 Future Work

In this section we describe future work which we have not yet conducted, but feel would be valuable contribution to this research.

7.3.1 Alternative Environments

Understanding how context affects system behaviour is a desired quality not unique to service based systems. Our example of documenting context in a service based system has been undertaken to explore context's relationship with answering provenance queries - which are often implemented using such environments. However, an interesting avenue for future work would be to demonstrate how the actor state model and provenance automaton could be used to document context in a system which was not service based. The State Assertion Registry could also be evaluated as a means of capturing context as assertions in such systems, by implementing observers appropriate for the scenario. As with our own observers, trigger events would be thrown to indicate specific functionality being executed and the process's context could be recorded as it occurs. Use of the provenance automaton to evaluate what occurred during execution from documentation would also still be possible.

7.3.2 State Mining Approach

In this dissertation, we used the Time Series Knowledge Representation (TSKR) as the means to interpret variable values and distinguish between actor states. This approach yielded good results during evaluation (When making predictions about future states, we were able to match between 30-85% of all states which were observed), but is just one method by which a collection of variables may be interpreted on an actor over a period of time. A more thorough investigation into how actor states may be found from sequences of variables may reveal that an alternative representation is able to give a higher or more consistent success rate.

TSKR also has the ability to perform analysis of several of collections of time series and find common patterns in the data, which we did not investigate in this work. If this were to be integrated with our work, it would mean it would be possible to find those sequences of actor state transitions which occurred most regularly and organise them so as to group together sequences which were sub-sequences of each another. This could potentially improve the recommendations made for future prediction of state.

7.3.3 Interface

Documentation which is captured detailing actor states is relatively complex and difficult to understand through manual inspection alone. During our research we prototyped some simple mechanisms to visualise how an actors state changed over time (shown in section 5.10). This involved displaying the entire time series using a different colour for each unique segment observed for the variables collected, with each series of observed actor states shown beneath the variables. From our usage the approach seemed a far more intuitive way of interacting and navigating records of actor state than attempting to inspect records manually. Both records of causal process and context have little specific investigation into the mechanism which is used to disseminate this evidence. In [24] a portlet based visualisation tool has been demonstrated to visualise both relationships and interactions which are observed in a healthcare process. However, this is only shown in use against a single scenario and the tool does not provide the ability to visualise assertions of state. From the large number of applications which have been built to satisfy provenance questions, it would be interesting to study if there was any discernable differences in each of the particular approaches which have been adopted in visualising documentation of processes. In particular, the provenance community is currently working toward a common description for representing causal processes across applications in the form of the Open Provenance Model (OPM). It would therefore be useful to investigate if creation of a means of visualisation of OPM based representations were possible - which would also be uniform across applications.

7.4 Concluding Remarks

This dissertation has given a solution to the problem of documenting and interpreting the context of processes by building upon existing process documentation strategies. Awareness of the circumstances of events often serves as a strong motivation for change of confidence about how well those events have been performed. We could liken the task of ensuring particular conditions have been met to that performed by a quality controller or safety inspector. However, with the increased use of remote, loosely coupled organisations to conduct processes or steps within them, desirable conditions for them cannot always be assumed to have been met. Fulfilment of total confidence in results can only be achieved when all evidence desired by an individual is made available to answer their own specific provenance questions. With the variation of primary concerns that exist for users who wish to understand what has happened in their systems, our contributions are a part of the significant overarching problem of solving user confidence in data created in their own environments.

Appendix A

Supporting Code Listings

This appendix contains supporting code listings used in construction of the State Assertion Registry, which are described in detail in Chapter 5, sections 5.9.3-5.10.

```java
/** A plugin which returns the current time of the host system */
public class SystemTimePlugin extends AbstractPlugin implements PluginInterface{
        static Logger logger = Logger.getLogger("org.pasoa.star");

        public SystemTimePlugin(){
                pluginName = "System Time Writer";
        }

        /** Stores the current system time against the descriptor of the event */
        public void executePlugin(ObserverEvent event){
                this.addToResourcePool(event.EVENT_DESCRIPTOR + " Time",
                        System.currentTimeMillis());

                ResourceValue resourceToAdd = new ResourceValue(System.currentTimeMillis(),
                        event.EVENT_DESCRIPTOR);

                this.addToOutputPool(event.EVENT_DESCRIPTOR, resourceToAdd);
        }
}
```

Listing A.1: System Time Capture Plug-In

```
/**
 * A plugin that takes start and end times from file to determine the execution time of a service
 * @author Ian Wootten
 */
public class ExecutionTimePlugin extends AbstractPlugin implements PluginInterface {

    private long starttime;
    private long endtime;

    public ExecutionTimePlugin(){
        pluginName = "Execution Time Writer";
    }

    public void executePlugin(ObserverEvent event){
        try{
            starttime = (Long) this.getResource("Axis Handler Request Time");
            endtime = (Long) this.getResource("Axis Handler Response Time");
        }
        catch (PluginProblem e){
            e.printStackTrace();
        }

        this.addToOutputPool("exectime", new ResourceValue(Long.toString(endtime - starttime)));
    }
}
```

Listing A.2: Execution Time Capture Plug-In

```
    public class RegistryHandler extends BasicHandler implements ObserverInterface{
        private ObserverEvent requestEvent = new ObserverEvent("Axis Handler Request");
        private ObserverEvent responseEvent = new ObserverEvent("Axis Handler Response");
        ....
        ....
        public void doServer(MessageContext context) throws AxisFault{

            if( ! context.getPastPivot() ){
                Observer observer = new Observer();

                observer.setRegistry(coordinator.getRegistry());
                observer.eventTriggered(requestEvent);

                setCoordinator(coordinator);
                setObserver(observer);
            }
            else{
                Observer observer = getObserver();

                observer.eventTriggered(responseEvent);
            }
        }
    }
```

Listing A.3: Execution Time Capture Plug-In

```
<?xml version=“1.0” encoding=“UTF−8”?>
<state pattern=“1 : 1 : 2”>
    <symbol>0</symbol>
    <label>S0</label>
    <rule>when bytes_in is low and load_one is low and mem_buffers is med</rule>
    <support>0.45054945054945056</support>
    <interval start=“1222766990” end=“1222767030” />
</state>
<state pattern=“1 : 1 : 3”>
    <symbol>1</symbol>
    <label>S1</label>
    <rule>when bytes_in is low and load_one is low and mem_buffers is high</rule>
    <support>0.5604395604395604</support>
    <interval start=“1222767030” end=“1222767080” />
</state>
```

Listing A.4: Example State Assertion Content

Appendix B

Raw Experimental Data

	Execution Times (ms)					
Dataset Size (MB)	No Prov	Actor	Interaction	All		
1.3	15422.37	15257.45	15776.94	16034.42		
2.7	44258.09	45088.53	47018.85	47889.01		
4	88980.34	90415.02	92972.07	94058.18		
5.3	139589.62	142498.03	145645.13	146466.15		
6.7	182528.38	184521.54	191498.96	189048.69		
8	233985.3	237279.41	245379.05	243527.18		
9.3	309021.22	313737.85	325203.96	333214.27		
10.7	373723.34	380704.1	389342.24	397258.7		
		Execution Overheads (ms)				Overhead (%)
Dataset Size (MB)	Exec Time (ms)	Actor	Interaction	Predicted	Actual	
1.3	15422.37	-164.92	354.57	189.65	612.05	4
2.7	44258.09	830.44	2760.76	3591.2	3630.92	8
4	88980.34	1434.68	3991.73	5426.41	5077.84	6
5.3	139589.62	2908.41	6055.51	8963.92	6876.53	5
6.7	182528.38	1993.16	8970.58	10963.74	6520.31	4
8	233985.3	3294.11	11393.75	14687.86	9541.88	4
9.3	309021.22	4716.63	16182.74	20899.37	24193.05	8
10.7	373723.34	6980.76	15618.9	22599.66	23535.36	6

Bibliography

[1] Cacti: The Complete RRDTool-based Graphing Solution. [Online: http://www.cacti.net/, Accessed Jun 2009].

[2] Data Collector for IBM Web Services Navigator. [Online: http://www.alphaworks.ibm.com/tech/wsdatacollector, Accessed Jun 2009].

[3] Munin. [Online: http://munin.projects.linpro.no/, Accessed Jun 2009].

[4] The Globus Alliance. [Online: http://www.globus.org, Accessed Jun 2009].

[5] Zenoss: Open Source Enterprise Monitoring. [Online: http://zenoss.com/, Accessed Jun 2009].

[6] Marcos Kawazoe Aguilera, Jeffrey C. Mogul, Janet L. Wiener, Patrick Reynolds, and Athicha Muthitacharoen. Performance debugging for distributed systems of black boxes. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 74–89, New York, NY, USA, 2003. ACM.

[7] Varol Akman and Mehmet Surav. Steps toward Formalizing Context. *AI Magazine*, 17(3):55–72, 1996.

[8] Ali Shaikh Ali, Omer F. Rana, Ian C. Parmee, Johnson Abraham, and Mark Shackelford. Web-Services Based Modelling/Optimisation for Engineering Design. In *OTM Workshops: On the Move to Meaningful Internet Systems*, pages 244–253, Agia Napa, Cyprus, 2005. Springer Berlin / Heidelberg.

[9] James F. Allen and George Ferguson. Actions and Events in Interval Temporal Logic. *Journal of Logic and Computation*, 4:531–579, 1994.

[10] Ilkay Altintas, Oscar Barney, and Efrat Jaeger-Frank. Provenance Collection Support in the Kepler Scientific Workflow System. In Luc Moreau and Ian T. Foster, editors, *Proceedings of IEEE International Provenance and Annotation Workshop (IPAW06)*, volume 4145 of *Lecture Notes in Computer Science*, pages 118–132. Springer, 2006.

[11] Mark Baker and Garry Smith. GridRM: An Extensible Resource Monitoring System. In *Fifth IEEE International Conference on Cluster Computing (CLUSTER'03)*, page 207. IEEE Computer Society, 2003.

[12] Gavin Bell. What is your Provenance? May 2007. XTech 2007, Paris, France.

[13] Rajendra Bose and James Frew. Composing Lineage Metadata with XML for Custom Satellite-Derived Data Products. In *16th International Conference on Scientific and Statistical Database Management (SSDBM'04)*, volume 0, page 275, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[14] Uri Braun, Simson L. Garfinkel, David A. Holland, Kiran-Kumar Muniswamy-Reddy, and Margo I. Seltzer. Issues in Automatic Provenance Collection. In *Proceedings of IEEE International Provenance and Annotation Workshop (IPAW06)*, pages 171–183, 2006.

[15] Peter Buneman, Sanjeev Khanna, Keishi Tajima, and Wang Chiew Tan. Archiving Scientific Data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD '02)*, pages 1–12, Madison, Wisconsin, 2002.

[16] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and Where: A Characterization of Data Provenance. In *International Conference on Database Theory (ICDT)*, volume 1973 of *Lecture Notes in Computer Science*, pages 316–330. Springer, 2001.

[17] Rob Byrom, Brian Coghlan, Andrew W Cooke, Roney Cordenonsi, Linda Cornwall, Abdeslem Djaoui, Laurence Field, Steve Fisher, Steve Hicks, Stuart Kenny, Jason Leake, James Magowan, Werner Nutt, David O'Callaghan, Norbert Podhorszki, John Ryan, Manish Soni, Paul Taylor, and Antony J Wilson. Relational Grid Monitoring Architecture (R-GMA). Talk, August 15 2003. UK e-Science All-Hands meeting, Nottingham, UK, September 2-4, 2003. 7 pages of LaTeX and 5 PNG figures.

[18] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos Eduardo Scheidegger, Cláudio T. Silva, and Huy T. Vo. Using Provenance to Streamline Data Exploration through Visualization. Technical Report UUSCI-2006-016, SCI Institute - University of Utah, 2006.

[19] L. Chen, N. Shadbolt, F. Tao, C. Goble, C. Puleston, and S. Cox. Managing Semantic Metadata for the Semantic Grid. In *Proceedings of Knowledge Grid and Grid Intelligence (KGGI) workshop, Beijing, China.*, 2004.

[20] Liming Chen, Zhuoan Jiao, and Simon J. Cox. On the use of semantic annotations for supporting provenance in grids. In Proceedings of Euro-Par 2006 Parallel Processing, editor, *Proceedings of Euro-Par 2006 Parallel Processing*, volume 4128/2006 of *Lecture Notes in Computer Science*, pages 371–380. Springer, 2006.

[21] He Chuan, Zhihui Du, and Sanli Li. GMA+ - A GMA-Based Monitoring and Management Infrastructure for Grid. In Minglu Li, Xian-He Sun, Qianni Deng, and Jun Ni, editors, *Grid and Cooperative Computing*, volume 3033 of *Lecture Notes in Computer Science*, pages 10–17. Springer, 2003.

[22] Herbert Clark and Thomas B. Carlson. Context for Comprehension. In *Attention and Performance IX*, pages 313–330. Lawrence Erlbaum Associates, Mahwah, New Jersey, 1982.

[23] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems*, 25(2):179–227, 2000.

[24] Vikas Deora, Arnaud Contes, Omer F. Rana, Shrija Rajbhandari, Ian Wootten, Kifor Tamas, and Laszlo Z. Varga. Navigating Provenance Information for Distributed Healthcare Management. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI '06)*, pages 859–865, Washington, DC, USA, 2006. IEEE Computer Society.

[25] Luc Moreau (Editor), Beth Plale, Simon Miles, Carole Goble, Paolo Missier, Roger Barga, Yogesh Simmhan, Joe Futrelle, Robert McGrath, Jim Myers, Patrick Paulson, Shawn Bowers, Bertram Ludaescher, Natalia Kwasnikowska, Jan Van den Bussche, Tommy Ellkvist, Juliana Freire, and Paul Groth. The Open Provenance Model (v1.01). [Online: http://openprovenance.org, Accessed June 2009], July 2008.

[26] Ian T. Foster. The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration. In *In Proceedings of the 14th Conference on Scientific and Statistical Database Management*, page 11. IEEE Computer Society, 2003.

[27] Ian T. Foster, Jens-S. Vöckler, Michael Wilde, and Yong Zhao. Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In *In Proceedings of the 14th Conference on Scientific and Statistical Database Management*, pages 37–46. IEEE Computer Society, 2002.

[28] Juliana Freire, Cláudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos Eduardo Scheidegger, and Huy T. Vo. Managing Rapidly-Evolving Scientific Workflows. In Luc Moreau and Ian T. Foster, editors, *Proceedings of IEEE International Provenance and Annotation Workshop (IPAW06)*, volume 4145 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 2006.

[29] James Frew and Rajendra Bose. Earth System Science Workbench: A Data Management Infrastructure for Earth Science Products. In *Proceedings of the 13th International Conference on Scientific and Statistical Database Management*, pages 180–189. IEEE Computer Society, 2001.

[30] James Frew, Dominic Metzger, and Peter Slaughter. Automatic Capture and Reconstruction of Computational Provenance. *Concurrency and Computation: Practice and Experience*, 20(5):485–496, 2008.

[31] Jennifer Golbeck. Combining Provenance with Trust in Social Networks for Semantic Web Content Filtering. In *Proceedings of IEEE International Provenance and Annotation Workshop (IPAW06)*, pages 101–108, 2006.

[32] Paul Groth. *The Origin of Data: Enabling the Determination of Provenance in Multi-institutional Scientific Systems through the Documentation of Processes.* PhD thesis, School of Electronics and Computer Science, University of Southampton, October 01 2007.

[33] Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau. An Architecture for Provenance Systems. Technical Report (v0.6), University of Southampton, 2006. [Online]. Available: http://eprints.ecs.soton.ac.uk/12023/.

[34] Paul Groth, Michael Luck, and Luc Moreau. A protocol for recording provenance in service-oriented Grids. In *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04)*, Grenoble, France, December 2004. [Online]. Available: http://eprints.ecs.soton.ac.uk/11914/.

[35] Paul Groth, Michael Luck, and Luc Moreau. Formalising a protocol for recording provenance in Grids. In *Proceedings of the UK OST e-Science second All Hands Meeting 2004 (AHM'04)*, Nottingham, UK, September 2004. [Online]. Available: http://eprints.ecs.soton.ac.uk/10216/.

[36] Paul Groth, Simon Miles, Weijan Fang, Sylvia C. Wong, Klaus-Peter Zauner, and Luc Moreau. Recording and Using Provenance in a Protein Compressibility Experiment. In *The 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, 2005. [Online]. Available: http://eprints.ecs.soton.ac.uk/10910/.

[37] Paul Groth, Simon Miles, and Luc Moreau. PReServ: Provenance Recording for Services. In *Proceedings of the UK OST e-Science second All Hands Meeting 2005 (AHM'05)*, 2005. [Online]. Available: http://eprints.ecs.soton.ac.uk/12570/.

[38] Paul Groth, Simon Miles, Victor Tan, and Luc Moreau. Architecture for Provenance Systems. Technical Report (v0.4), University of Southampton, 2005. [Online]. Available: http://eprints.ecs.soton.ac.uk/11310/.

[39] Ramanathan Guha. *Contexts: a formalization and some applications.* PhD thesis, Stanford University, Stanford, CA, USA, 1992.

[40] Ceki Gülcü. *The Complete Log4J Manual.* QOS.ch, 2004.

[41] Denise Head, Abraham Z. Snyder, Laura E. Girton, John C. Morris, and Randy L. Buckner. Frontal-Hippocampal Double Dissociation Between Normal Aging and Alzheimer's Disease. *Cerebral Cortex*, 15(6):732–739, 2005.

[42] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation, Second Edition.* Addison-Wesley, 2001.

[43] Kamran Karimi. *Discovery of Causality and Acausality from Temporal Sequential Data.* PhD thesis, University of Regina, 2005.

[44] Kamran Karimi and Howard J. Hamilton. TimeSleuth: A Tool for Discovering Causal and Temporal Rules. In *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI02)*, pages 375–380. IEEE Computer Society, 2002.

[45] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. Segmenting Time Series: A Survey and Novel Approach. In *an Edited Volume, Data mining in Time Series Databases*, pages 1–22. World Scientific, 1993.

[46] Guy K. Kloss and Andreas Schreiber. Provenance Implementation in a Scientific Simulation Environment. In *Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, Chicago, USA, May 2006. Springer-Verlag.

[47] Geoffrey N. Leech. *Semantics: The Study Of Meaning.* Penguin UK, 1981.

[48] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[49] (Editors) Mark Little, Eric Newcomer, and Greg Pavlik. Web Services Context Specification (WS-Context). Technical Report Version 0.8, The Organisation for the Advancement of Structured Information Standards, 2004.

[50] Shalil Majithia, Matthew S. Shields, Ian J. Taylor, and Ian Wang. Triana: A Graphical Web Service Composition and Execution Toolkit. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 514–524, Washington, DC, USA, 2004. IEEE Computer Society.

[51] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, July 2004.

[52] John McCarthy and Saša Buvač. Formalizing Context (Expanded Notes). In A. Aliseda, R.J. van Glabbeek, and D. Westerståhl, editors, *Computing Natural Language*, volume 81 of *CSLI Lecture Notes*, pages 13–50. Center for the Study of Language and Information, Stanford University, 1998.

[53] Simon Miles. Second Provenance Challenge. [Online: http://twiki.ipaw.info/bin/view/Challenge/SecondProvenanceChallenge, Accessed Jun 2009].

[54] Simon Miles. Electronically Querying for the Provenance of Entities. In Luc Moreau and Ian Foster, editors, *Third International Provenance and Annotation Workshop*, volume 4145, pages 184–192, Chicago, Illinois, US, 2006. Springer. [Online]. Available: http://eprints.ecs.soton.ac.uk/12567/.

[55] Simon Miles. Practical Definition of Causation for Distributed Applications. Internal Document, Nov 2006. Southampton University.

[56] Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The Requirements of Using Provenance in e-Science Experiments. *Journal of Grid Computing*, 5:1–25, 2006. [Online]. Available: http://eprints.ecs.soton.ac.uk/13242/.

[57] Fabian Mörchen. A better tool than Allen's relations for expressing temporal knowledge in interval data. The Twelveth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, 2006.

[58] Fabian Mörchen. Algorithms for time series knowledge mining. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '06)*, pages 668-673, New York, NY, USA, 2006. ACM.

[59] Fabian Mörchen. Unsupervised pattern mining from symbolic temporal data. *ACM SIGKDD Explorations Newsletter*, 9(1):41–55, June 2007.

[60] Luc Moreau. Usage of 'provenance': A Tower of Babel Towards a concept map. Position paper for the Life Cycle Seminar, Mountain View, July 2006.

[61] Luc Moreau, Juliana Freire, Joe Futrelle, Robert McGrath, Jim Myers, and Patrick Paulson. The Open Provenance Model, December 2007.

[62] Luc Moreau, Bertram Ludäscher, Ilkay Altintas, Roger S. Barga, Shawn Bowers, Steven Callahan, George Chin Jr., Ben Clifford, Shirley Cohen, Sarah Cohen-Boulakia, Susan Davidson, Ewa Deelman, Luciano Digiampietri, Ian Foster, Juliana Freire, James Frew, Joe Futrelle, Tara Gibson, Yolanda Gil, Carole Goble, Jennifer Golbeck, Paul Groth, David A. Holland, Sheng Jiang, Jihie Kim, David Koop, Ales Krenek, Timothy McPhillips, Gaurang Mehta, Simon Miles, Dominic Metzger, Steve Munroe, Jim Myers, Beth Plale, Norbert Podhorszki, Varun Ratnakar, Emanuele Santos, Carlos Scheidegger, Karen Schuchardt, Margo Seltzer, Yogesh L. Simmhan, Claudio Silva, Peter Slaughter, Eric Stephan, Robert Stevens, Daniele Turi, Huy Vo, Mike Wilde, Jun Zhao, and Yong Zhao. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience*, 20(5):409–418, 2007.

[63] James D. Myers, Thomas C. Allison, Sandra Bittner, Brett Didier, Michael Frenklach, Jr. William H. Green, Yen-Ling Ho, John Hewson, Wendy Koegler, Carina Lansing, David Leahy, Michael Lee, Renata Mccoy, Michael Minkoff, Sandeep Nijsure, Gregor Von Laszewski, David Montoya, Luwi Oluwole, Carmen Pancerella, Reinhardt Pinzon, William Pitz, Larry A. Rahn, Branko Ruscic, Karen Schuchardt, Eric Stephan, A. Wagner, Theresa Windus, and Christine Yang. A Collaborative Informatics Infrastructure for Multi-Scale Science. *Cluster Computing*, 8(4):243–253, 2005.

[64] James D. Myers, Carmen M. Pancerella, Carina S. Lansing, Karen L. Schuchardt, Brett T. Didier, and C N. Ashish, Goble. Multi-scale Science: Supporting Emerging Practice with Semantically Derived Provenance. In *ISWC workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*, March 06 2006.

[65] Tobias Oetiker. RRDTool: Logging & Graphing. [Online: http://oss.oetiker.ch/rrdtool, Accessed Jun 2009].

[66] David Ogle, Heather Kreger, Abdi Salahshour, Jason Cornpropst, Eric Labadie, Mandy Chessell, Bill Horn, and John Gerken. Canonical Situation Data Format: The Common Base Event. Technical Report v1.01, International Business Machines Corporation, August 2003.

[67] Thomas M. Oinn, R. Mark Greenwood, Matthew Addis, M. Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole A. Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter Li, Phillip W. Lord, Matthew R. Pocock, Martin Senger, Robert Stevens, Anil Wipat, and Chris Wroe. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, August 2006.

[68] Carmen Pancerella, James D. Myers, Thomas C. Allison, Kaizar Amin, Sandra Bittner, Brett Didier, Michael Frenklach, William H. Jr. Green, Yen-Ling Ho, John Hewson, Wendy Koegler, Carina Lansing, David Leahy, Michael Lee, Renata McCoy, Michael Minkoff, Sandeep Nijsure, Gregor von Laszewski, David Montoya, Reinhardt Pinzon, William Pitz, Larry Rahn, Branko Ruscic, Karen Schuchardt, Eric Stephan, Al Wagner, Baoshan Wang, Theresa Windus, Lili Xu, and Christine Yang. Metadata in the Collaboratory for Multi-Scale Science. 2003 Dublin Core Conference: Supporting Communities of Discourse and Practice - Metadata Research and Applications in Seattle, WA, 28 September - 2 October 2003, 2003.

[69] Carmen Pancerella, Jim Myers, and Larry Rahn. Data Provenance in the CMCS, 2002. Chicago, Illinois, 17-18 October.

[70] Wim De Pauw, Michelle Lei, Edward Pring, Lionel Villard, Matthew Arnold, and John F. Morar. Web Services Navigator: Visualizing the execution of Web Services. *IBM Systems Journal*, 44(4):821–846, 2005.

[71] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Number ISBN-10: 1558602380. Morgan Kaufmann, 1992.

[72] Shrija Rajbhandari, Arnaud Contes, Omer F. Rana, Vikas Deora, and Ian Wootten. Trust Assessment Using Provenance in Service Oriented Applications. In *Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops (EDOCW '06)*, page 65, Washington, DC, USA, 2006. IEEE Computer Society.

[73] Christine F. Reilly and Jeffrey F. Naughton. Exploring Provenance in a Distributed Job Execution System. In *Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, Chicago, USA, May 2006. Springer-Verlag.

[74] John P. Rouillard. Real-time Log File Analysis Using the Simple Event Correlator (SEC). In *18th USENIX System Administration Conference (LISA '04)*, pages 133–149, November 2004.

[75] Ana Gallegos Saliner. *Molecular Quantum Similarity in QSAR: Applications in Computer-Aided Design.* PhD thesis, IQC Institut de Qumica Computacional, 2004.

[76] Carlos Scheidegger, David Koop, Emanuele Santos, Huy Vo, Steven Callahan, Juliana Freire, and Claudio Silva. Tackling the Provenance Challenge One Layer at a Time. *Concurrency and Computation: Practice and Experience*, 20(5):473–483, 2008.

[77] Reinhard Schwarz. Causality in distributed systems. In *EW 5: Proceedings of the 5th workshop on ACM SIGOPS European workshop*, pages 1–5, New York, NY, USA, 1992. ACM Press.

[78] B. & Gannon D.; Simmhan, Y.L.; Plale. A Survey of Data Provenance Techniques. Technical Report TR-618, Computer Science Department, Indiana University, 2005.

[79] Feng Tao, Liming Chen, Nigel Shadbolt, Fenglian Xu, Simon J. Cox, Colin Puleston, and Carole A. Goble. *Semantic Web Based Content Enrichment and Knowledge Reuse in E-science*, volume 3290 of *Lecture Notes in Computer Science*, pages 654–669. Springer, 2004.

[80] Valerie E. Taylor, Xingfu Wu, and Rick L. Stevens. Prophesy: an infrastructure for performance analysis and modeling of parallel and grid applications. *SIGMETRICS Performance Evaluation Review*, 30(4):13–18, 2003.

[81] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A Grid Monitoring Architecture. Memo, Global Grid Forum / Grid Monitoring Architecture Working Group, 2002.

[82] R. Vaarandi. SEC - a Lightweight Event Correlation Tool. In *Proceedings of the 2002 IEEE Workshop on IP Operations and Management (IPOM 2002)*, pages 111–115, 2002.

[83] R. Vaarandi. A Data Clustering Algorithm for Mining Patterns From Event Logs. *Proceedings of the 2003 IEEE Workshop on IP Operations and Management*, pages 119–126, 2003.

[84] R. Vaarandi. A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs. *Proceedings of the 2004 IFIP International Conference on Intelligence in Communication Systems*, Vol. 3283:293–308, 2004.

[85] Jasmin Wason, Marc Molinari, Zhuoan Jiao, and Simon J. Cox. *Delivering data management for engineers on the grid*, volume 2790/2004, pages 412–416. Springer, 2003.

[86] A. Woodruff and M. Stonebreaker. Supporting Fine-Grained Data Lineage in a Database Visualization Environment. In *Proceedings of the 13th International Conference on Data Engineering (ICDE'97)*, pages 91–103, Washington - Brussels - Tokyo, April 1997.

[87] Ian Wootten, Shrija Rajbhandari, and Omer Rana. Automatic Assertion of Actor State in Service Oriented Architectures. In *Proceedings of the IEEE International Conference on Web Services (ICWS07)*, 2007.

[88] Ian Wootten and Omer Rana. Recording the Context of Action for Process Documentation. In *Proceedings of IEEE International Provenance and Annotation Workshop (IPAW08)*, pages 45–53, Berlin, Heidelberg, 2008. Springer-Verlag.

[89] Ian Wootten, Omer Rana, Shrija Rajbhandari, and J.S.Pahwa. Actor Provenance Capture with Ganglia. In *Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGRID06)*, 2006.

[90] Ian Wootten, Omer F. Rana, and Shrija Rajbhandari. Recording Actor State in Scientific Workflows. In Luc Moreau and Ian T. Foster, editors, *Proceedings of IEEE International Provenance and Annotation Workshop (IPAW06)*, volume 4145 of *Lecture Notes in Computer Science*, pages 109–117. Springer, 2006.

[91] Xuehai Zhang and Jennifer M. Schopf. Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2. *CoRR*, cs.DC/0407062, 2004.

[92] Jun Zhao. *A Conceptual Model for E-Science Provenance.* PhD thesis, School of Computer Science, Manchester, 2007.

[93] Jun Zhao, Carole Goble, Mark Greenwood, Chris Wroe, and Robert Stevens. Annotating, linking and browsing provenance logs for e-Science. In *Proceedings of the 2nd International Semantic Web Conference (ISWC2003) Workshop on Retrieval of Scientific Data*, Florida, October 07 2003.

[94] Jun Zhao, Carole A. Goble, and Robert Stevens. An Identity Crisis in the Life Sciences. In Luc Moreau and Ian T. Foster, editors, *Proceedings of IEEE International Provenance and Annotation Workshop (IPAW06)*, volume 4145 of *Lecture Notes in Computer Science*, pages 254-269. Springer, 2006.

[95] Jun Zhao, Chris Wroe, Carole A. Goble, Robert Stevens, Dennis Quan, and R. Mark Greenwood. Using Semantic Web Technologies for Representing E-science Provenance. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2004.

[96] Yong Zhao, Michael Wilde, and Ian T. Foster. Applying the Virtual Data Provenance Model. In Luc Moreau and Ian T. Foster, editors, *Proceedings of IEEE International Provenance and Annotation Workshop (IPAW06)*, volume 4145 of *Lecture Notes in Computer Science*, pages 148–161. Springer, 2006.