UNIVERSITY OF SOUTHAMPTON

The Origin of Data

Enabling the Determination of Provenance in Multi-institutional Scientific Systems through the Documentation of Processes

by

Paul T. Groth

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

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

September 2007

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY of ENGINEERING, SCIENCE and MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

The Origin of Data

Enabling the Determination of Provenance in Multi-institutional Scientific Systems through the Documentation of Processes

by Paul T. Groth

The Oxford English Dictionary defines provenance as (i) the fact of coming from some particular source or quarter; origin, derivation. (ii) the history or pedigree of a work of art, manuscript, rare book, etc.; concr., a record of the ultimate derivation and passage of an item through its various owners. In art, knowing the provenance of an artwork lends weight and authority to it while providing a context for curators and the public to understand and appreciate the work's value. Without such a documented history, the work may be misunderstood, unappreciated, or undervalued.

In computer systems, knowing the provenance of digital objects would provide them with greater weight, authority, and context just as it does for works of art. Specifically, if the provenance of digital objects could be determined, then users could understand how documents were produced, how simulation results were generated, and why decisions were made. Provenance is of particular importance in science, where experimental results are reused, reproduced, and verified. However, science is increasingly being done through large-scale collaborations that span multiple institutions, which makes the problem of determining the provenance of scientific results significantly harder.

Current approaches to this problem are not designed specifically for multi-institutional scientific systems and their evolution towards greater dynamic and peer-to-peer topologies. Therefore, this thesis advocates a new approach, namely, that through the autonomous creation, scalable recording, and principled organisation of documentation of systems' processes, the determination of the provenance of results produced by complex multi-institutional scientific systems is enabled. The dissertation makes four contributions to the state of the art.

First is the idea that provenance is a query performed over documentation of a system's past process. Thus, the problem is one of how to collect and collate documentation from multiple distributed sources and organise it in a manner that enables the provenance of a digital object to be determined.

Second is an open, generic, shared, principled data model for documentation of processes, which enables its collation so that it provides high-quality evidence that a system's processes occurred. Once documentation has been created, it is recorded into specialised repositories called provenance stores using a formally specified protocol, which ensures documentation has high-quality characteristics. Furthermore, patterns and techniques are given to permit the distributed deployment of provenance stores. The protocol and patterns are the third contribution.

The fourth contribution is a characterisation of the use of documentation of process to answer questions related to the provenance of digital objects and the impact recording has on application performance. Specifically, in the context of a bioinformatics case study, it is shown that six different provenance use cases are answered given an overhead of 13% on experiment runtime. Beyond the case study, the solution has been applied to other applications including fault tolerance in service-oriented systems, aerospace engineering, and organ transplant management.

Contents

List of Figures

Declaration of Authorship

I, Paul Groth, declare that the thesis entitled The Origin of Data:Enabling the Determination of Provenance in Multi-institutional Scientific Systems through the Documentation of Processes and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published as listed in Section 1.6.

Signed:

Date:

Acknowledgements

When I first started this process, I had images of a lone scholar in the midst of a library somewhere surrounded by books writing away in isolation. For both my sanity and my tremendous benefit, this was not the case. Without the support of the following people, this dissertation would have never been done. More importantly, this process would not have been so fun.

First, a big thank you goes to my parents, without your confidence in me this would not have been possible. Your passion and belief in education are an inspiration. Thomas, your advice on writing, creativity, and my serve is second to none. LaRue, your unconditional encouragement, concern, and love have been a rock to stand on during this process.

One piece of advice my parents gave me before starting this PhD is that picking a good supervisor is absolutely critical in order to achieve success. Luckily for me, I picked a great one, Professor Luc Moreau. Luc, thank you for teaching me the importance of being systematic, how to actually do a proof, how to write like a scientist, and the importance of fluid intake. The time you have spent having intense discussions with me, reading my work, and putting up with my realistic schedules is very much appreciated. In summary, good job.

Thanks to Michael Luck for giving me an another perspective on academia and being a nice guy. Klaus: Ich danke dir vielmals. Ohne dein Experiment und Rat würde diese Doktorarbeit nie gegeben. To the Iridis guys, Ivan Woolten and David Baker, without your help (and responding to my strange requests), I would have never pulled this off.

A special shout out goes to Paul Townend for being the first and best user of my software. Thanks to the people on the PASOA and EU Provenance projects for using the software and your insightful comments on my ideas. Andrej and Tibo, your summer projects were cool, thanks for coming on board to help.

Thanks to everyone in the IAM Lab, you make it a great place to work. Thanks to Victor Tan and Weijian Fang for always taking the time to discuss technical stuff and putting up with my Americanisms. Maria, it was great having another North American around. The Lab and my time in Southampton would not have been as interesting or fun without Roxana and Shiv. I hope your current endeavours take you where you want to go. The last bit of crunch time before submission was made bearable by my chats with Maira. Don't worry you'll be finished soon. I would be amiss not to mention Danius, who went from scary lab management guy, to a good friend. Thanks for the time at Ceno looking at barmaids and giving me advice on my thesis.

For over two years in the lab, the desk next to mine was occupied by one Chris Bailey who calmly put up with my commentary both verbal and via MSN as well as the occasional

Nerf gun attack. Dude, thanks for the coffee in Starbucks and listening to my grandiose ideas.

Beyond sitting in Starbucks, playing tennis has kept me in good spirits. Thanks to Kat & the Tennis Team for showing me what British student culture is really like, oh, and the tennis was fun as well. Thanks to Eyre for letting Moss take hours on the weekend to do battle with me on court. Moss, hopefully, we can have a hit again one day soon.

From Pensacola, I have to thank Joe for his extended emails that remind me what the "real" world is like. Shamma, the once a year lessons on how to be both hip and a computer scientist are good but they still haven't rubbed off. Thanks to Niranjan, whose recommendation got me here in the first place and whose New Year's Eve parties are always the place to give an update on life's progress. Also thanks to David Eccles, who introduced me to the cheeky pint before I even got to England and who showed me that TGIs is a great place to do research.

My time in Southampton would not have been nearly as fun without the people of 25 High Road. Thanks to all of you for your epic support. Steve, women!?, what more can I say. Thanks for your wisdom and for proofing thesis chapters. Seb, the french food, the french wine, the bbqs, the server, and the unbelievable guitar have made it an awesome trip. Mischa, thanks for the talks about life, academia, and Web 2.0 as well as proofing this entire dissertation. It is very much appreciated. I'll always fondly remember sprawling out on your sofa. Martin, thanks for showing the way and letting me move into High Road in the first place. Respect. Laura, talking to you about the wider world has made life more colourful. In the end though, who is right? The dreads: Tony stay metal. Simon thanks for being proof that scientist actually use multi-institutional scientific systems. Ben, thanks for showing me what life is really about. Sofie, thanks for the living room chats.

Finally, the quality of this dissertation would be much less without the help of Geraldine and Simon. Geraldine, thanks for making Simon and I talk about things other than provenance. Without your intervention, dinner and drinks would not have been as exciting. Simon, working with you has been a stupendous experience. I will miss our lunches together. Your ideas, your arguments, your questioning have taught me a lot about how to do research, write code, and approach life. Thank you.

Chapter 1

Introduction

Science is changing. Increasingly, it can no longer be done in the confines of a single scientist's lab or for that matter in a single department, research institute, or university. Major scientific questions, such as how proteins fold, existence of the Higgs boson particle, and the human impact on the climate, require the application of skills, knowledge, and facilities from multiple institutions. For example, to analyze the data produced by the largest scientific instrument in the world, the Large Hadron Collider, the cooperation and computing facilities of over 100 institutions across the world is needed [79]. In proteomics, the advanced simulations needed to understand how proteins fold are computed using personal computer cycles donated by thousands of volunteers [167]. It is not just computing power that needs to be shared. Biologists rely on the cooperative construction and availability of open curated sequence databases to develop new drugs and understand the intricate workings of complex biological systems [189]. To study the dynamics of matter and the structure of blackholes, scientists and engineers from over five institutions have cooperatively built the Laser Interferometer Gravitational-Wave Observatory that spans two sites located 3000 kilometres apart [2]. In astronomy, the collobarative use of telescope and processing resources is allowing astronomers to access a complete digital catalogue of the sky [176]. By sharing large-scale experimental apparatus over the Internet, earth quake engineers have democratised their field [146].

Therefore, it is no longer sufficient to just share results through the standard scientific practice of producing publications. These problems can only be solved through the sharing, not only of published results, but also of other resources including sensor data, equipment, experimental processes, and human know-how. A system that requires the sharing of resources from multiple institutions to achieve its scientific aims is what we term a *multi-institutional scientific system* or application.

While this new approach to science has already been successful, it is still in its childhood and a variety of issues must be addressed for it to become as mature as other scientific techniques and tools. In this dissertation, we begin to address one of those issues,

namely, confidence in the results produced by these systems.

1.1 A Problem of Confidence

Consider, a scientist, Alice, studying an image of a supernova acquired from an online database. This image is composed from from thousands of distinct images provided by telescopes from around the world. Alice then runs the supernova image through various analysis routines made available by different institutions. The outputs of these analyses are then used by Alice's own bespoke algorithm to generate a final set of results, which she publishes in a journal. The journal paper contains the highlights of her results, a description of the algorithm used and the references to the services relied upon.

Another scientist, Victor, takes a keen interest in Alice's paper and following standard scientific practice attempts to reproduce the experiment. He recreates Alice's bespoke algorithm from the paper's description, he then finds that some of the services Alice relied on are no longer available or have changed versions. Adapting to the problem, Victor uses the newer versions of the services and tries to recreate the unavailable services from code he finds on the Internet. After running the experiment, Victor finds that the results are different than those in Alice's paper. Victor contacts Alice who then tries to recreate her own experiment. Unfortunately, Alice no longer has the original outputs from the services she used and using the outputs generated by the current services Alice's algorithm produces a different result. The ability to *reproduce* results is one of the cornerstones of the scientific method and the inability to do so, in this case, undermines the *confidence* Alice and Victor have in the original results.

This simple story illustrates the difficulty in reproducing results when they are produced using decentralised systems that dynamically evolve. In such systems, no one authority has control over the services or data within the system. Hence, the system's components can evolve independently without the knowledge of other components and thus it is hard to maintain knowledge about the past state of the system. This lack of knowledge makes reproduction difficult. Decentralisation and dynamicity are both hallmarks of multi-insitutional scientific systems.

While reproduction is an important factor in the confidence scientists have in a result, there are other factors that also contribute. These factors include:

- The ability to interpret and understand a result.
- The ability to understand the experiment and chain of reasoning that was used in the production of a result.
- The ability to verify that the experiment responsible for a result was performed according to acceptable procedures.

- The ability to identify what the inputs to an experiment were and where they came from.
- The ability to know who performed an experiment and who is responsible for its results.

The standard scientific practices of peer-review and publication take into account these factors and provide the bedrock on which the confidence in scientific results is based. However, in the context of multi-institutional scientific systems, that may involve hundreds of individuals, institutions, and components, it becomes difficult for scientists, reviewers, and the public to obtain all the information they need to be confident in the results these systems generate. Fundamentally, users need to understand how these results were produced, their history, their origins, their *provenance*.

1.2 The Assurance of Provenance

The Oxford English Dictionary defines provenance as (i) the fact of coming from some particular source or quarter; origin, derivation. (ii) the history or pedigree of a work of art, manuscript, rare book, etc.; concretely, a record of the ultimate derivation and passage of an item through its various owners.

In the field of art, knowing the provenance of an artwork provides collectors, curators, and the public a context, which provides the means to understand, verify, and evaluate that artwork. Provenance gives assurance that the artwork has value; that it is, for example, truly painted by Johannes Vermeer and is not actually a forgery by Han van Meegeren. Similarly, when Starbucks Coffee produces a brochure like the one in Figure 1.1, they are using a guarantee about the provenance of their coffee to both reassure customers and indicate the quality of it.

Just as knowing the provenance of a work of art provides it with greater weight, authority and context, knowing the provenance of a digital object or data item offers similar benefits. In particular, when detailed enough, the provenance of a digital object contains all the information necessary to provide confidence to its users. Each of the various factors used by scientists in their confidence judgements are addressed by having a comprehensive record of a digital object's derivation.

In different domains and environments, what constitutes a comprehensive record of derivation may vary radically. For example, in art, the provenance of a painting usually only details its chain of ownership. However, in some cases, it is not only necessary to know the chain of ownership but also the various restorations the painting went through. In food science, the provenance of food purchased at a grocery store would include where the food was grown, how it was transported, packaged, and processed. An example of

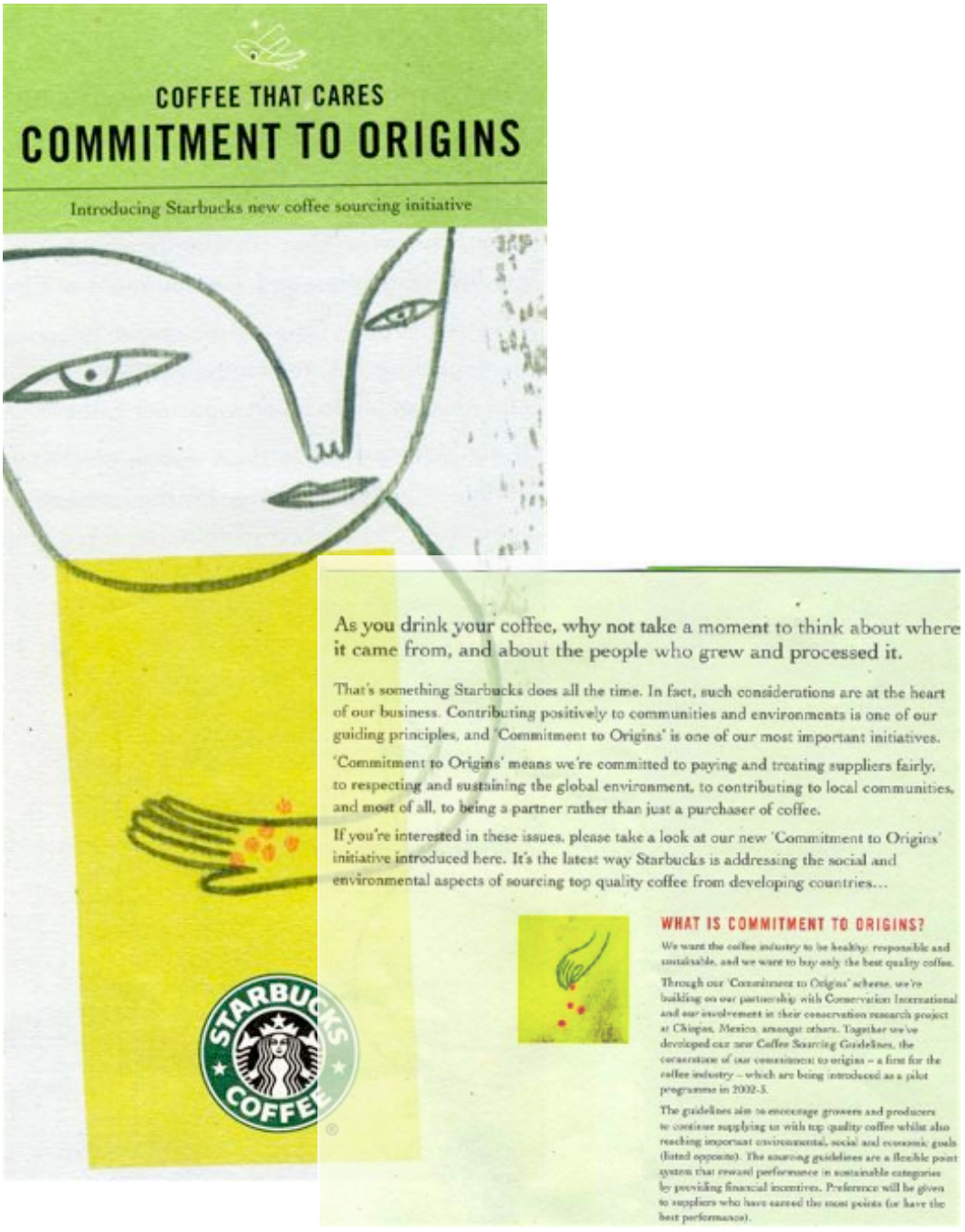


FIGURE 1.1: Brochure from Starbucks discussing the origins of their coffee

FIGURE 1.2: The labels on these eggs show their provenance

the provenance of food is the label put on all eggs sold in Germany as shown in Figure 1.2. This label indicates how the hen that laid the egg was raised, what country the egg is from, the farm where the egg was produced, and the cage where the egg was laid. For digital objects, the provenance could include everything from the algorithms used in processing to the user who started a computational simulation.

Thus, depending on what information gives a user confidence, the kind of information returned from a query about an item's provenance varies. The unifying theme between the above records of derivation is that they document part of the *process* that led to an item in a particular state. For example, the restorations of a painting are part of the larger process that led to the painting in its current state. Thus, knowing the entirety of the process that led to the painting as it is would also include the various restorations it has undergone. Therefore, conceptually, we define the provenance of a result produced by a system as follows:

Definition 1.1. The provenance of a result is the process that led to that result.

In computational systems, results are usually data items, and thus throughout this work we focus primarily on the provenance of data, which would be the process that led to the data item in question. By understanding the process that led to the result produced by a multi-institutional scientific system, a scientist can have confidence in it.

1.3 The Role of Documentation

Processes, however, are ephemeral, they occur and then are gone. Thus, to show that a process did in fact happen some evidence is necessary. Revisiting the Oxford English Dictionary's definition of provenance, we note it concretely defines provenance as "a record of the ultimate derivation and passage of an item...". We view such a record

of derivation as consisting of documentation that taken together is the evidence that a process that led to a particular item occurred. This view is natural and reflects itself in many different settings. In law, prosecutors and defenders find various forms of documentation to show that a crime (i.e. a process) occurred or did not occur in a particular manner. Similarly, historians use documentation to make the case that a historical event happened in a particular manner.

Documentation, however, comes in many forms including letters, contracts, email, stamps, memos, input data, output data, algorithms, executable programs, code, meeting minutes, and lab notebooks. Therefore, to understand the provenance of a result one must sift through all this various documentation and find the set of documentation that best represents the process that led to the result in question.

Thus, provenance is a *query* answered by searching over documentation. The problem that exists in multi-institutional scientific systems is that documentation is in multiple locations, represented in different formats, and is not easily queried. Furthermore, documentation is often lost, deleted, or modified so that it can no longer be found or used as accurate evidence of provenance. Hence, determining the provenance of results produced by these systems is difficult. We term this problem the *provenance problem*.

To solve it, we propose that documentation of all of a system's processes should be created according to a shared model. By specifically documenting a system's processes, all the various forms of documentation are connected so that together they provide comprehensive evidence of those processes. We term documentation that describes a system's processes, *process documentation*.

It is important to ensure that process documentation enables the accurate determination of provenance in multi-institutional scientific systems. We have identified six characteristics that documentation should possess to achieve this goal. These characteristics are enumerated below and discussed and justified in more detail in Section 3.2.

1. Factual - Process documentation should provide an accurate representation of the process that occurred and thus should be factual.

2. Attributable - Knowing who is responsible for process documentation is critical to accuracy because it provides both the ability to judge the quality of a source and the ability to make creators of documentation accountable for it.

3. Autonomously Creatable - In a multi-institutional or distributed system, no one central entity can accurately create documentation about the entirety of a process, thus, process documentation must be able to be created by multiple independent sources. Furthermore, it should be created in manner so that it can be collated together such that the provenance of data can be determined.

4. Process Oriented - From Definition 1.1, provenance is defined in terms of process. Therefore, accurate process documentation should reflect a system's processes and allow an individual process to be distinguished and found.

5. Immutable - Once process documentation is created, it should not be deleted or modified. Without this guarantee, it is difficult to establish that process documentation accurately and comprehensively reflects the processes that occurred.

6. Finalizable - When multiple components create process documentation, it is important to be able to determine when a component is finished creating documentation for part of the process. Once a component is done creating documentation, it can then be held responsible for the completeness and thus the accuracy of its account.

Process documentation with these characteristics is what we term *high quality*. In this dissertation, we introduce and specify a shared model that helps components or entities within a multi-institutional system to create high quality process documentation.

Once documentation has been created according to our model, it can be collected into specialised repositories, called *provenance stores*, responsible for maintaining and preserving it beyond the lifetime of the components that created it. We term an application that creates and records process documentation, a *provenance-aware application*. After process documentation is collected into these repositories, it can be queried to find documentation that represents the provenance of a particular digital object. Thus, the provenance of a digital object can be determined by performing a query over process documentation. By filtering and searching the representation of provenance returned by such a query, various questions can be answered. For example, who authorised an action, what computers were used in the creation of a result, and why did a process take so long to complete. We term these questions, *provenance questions*. This explicit separation of concerns between creation and querying allows components to be designed specifically for their purpose.

1.4 Thesis Statement and Contributions

Our solution to the provenance problem can be summarised in the following thesis statement.

The autonomous creation, scalable recording, and principled organisation of documentation of multi-institutional scientific systems' processes enables the problem of determining the provenance of results produced by these systems to be solved.

This dissertation makes the following contributions to the state of the art:

1. The provenance of a digital object can be answered by a query over documentation of a system's processes. Making this distinction explicit brings benefits in terms of *system design*. It enables a *separation of concerns* between creators and queriers, which allows queries to be performed by independent parties. Furthermore, it caters for the specialisation in the design of creation, recording, and querying components.

2. A data model for process documentation, which allows for the provenance of results to be obtained and that is based on two key principles:

   (a) Process documentation must represent *causal* relations between entities for the provenance of results to be determined.

   (b) To enable provenance queries to be answered accurately, documentation should be *high quality*. It should have the characteristics of being *factual*, *attributable*, *autonomously creatable*, *process oriented*, *immutable* and *finalizable*. These characteristics are supported both by the data model and the recording of process documentation into provenance stores.

   These principles are necessary to enable provenance questions to be answered accurately in distributed multi-institutional settings. This contribution is discussed primarily in Chapter 3

3. A protocol and patterns that enable the scalable recording of documentation into provenance stores (Chapter 4). The protocol enforces the recording of process documentation with high-quality characteristics. This is shown through a series of proofs (Section 4.4.5). Scalability is shown through controlled experiments conducted on an implementation of the repository that follows the protocol specification (Section 6.3).

4. A characterisation of the use of documentation of process to answer questions related to the provenance of digital objects and the impact recording has on application performance. Specifically, the solution is evaluated in the context of a real world application from bioinformatics (Chapter 5). It is shown that six different provenance use cases are answered given an overhead of 13% on experiment runtime (Section 6.4). While these use cases are specific to the case study, they reflect a range of provenance questions that scientists might pose.

1.5 Presentation Overview

This dissertation is organised as follows.

Chapter 2 discusses in greater detail the nature of provenance and its relationship to processes. It also analyses the state of the art for determining provenance in computa-

tional systems with respect to multi-institutional scientific systems. From this analysis, conclusions are drawn about the key attributes a provenance system should have.

Chapter 3 defines a data model, the p-structure, designed to support both the requirements outlined in Chapter 2 as well as high-quality characteristics. The data model is defined conceptually using concept maps and thus is not bound to any particular technology or implementation.

Chapter 4 introduces the concept of a provenance store, a specialised repository for storing process documentation. It defines three design patterns that can be used to determine how to best deploy a set of provenance stores. Additionally, the chapter specifies how distributed documentation can be linked. Finally, a protocol for recording process documentation is formally specified. Using this formal specification, the protocol is shown to support the recording of process documentation with high-quality characteristics.

Chapter 5 introduces the bioinformatics case study, the Amino Acid Compressibility Experiment, and its associated provenance use cases.

Chapter 6 presents an evaluation of an implementation of our approach. It considers three different aspects of the implementation. One, the scalability of the implementation in a controlled environment. Two, the impact recording process documentation has on the case study and three, whether the use cases from the case study can be effectively answered with the approach. Recommendations on the use of the implementation in applications are also given.

Chapter 7 outlines various avenues for future work and concludes the dissertation.

1.6 Publications

The work in this dissertation is derived from a number of peer-reviewed publications, which are listed below:

- P. Groth, M. Luck, and L. 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
- P. Groth, M. Luck, and L. Moreau. A Protocol for Recording Provenance in Service-Oriented Grids. In T. Higashino, editor, *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04)*, volume 3544 of *Lecture Notes in Computer Science*, pages 124–139, Grenoble, France, December 2004. Springer-Verlag

- P. Groth, S. Miles, W. Fang, S. C. Wong, K.-P. Zauner, and L. Moreau. Recording and Using Provenance in a Protein Compressibility Experiment. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, pages 201–208, July 2005
- P. Groth, S. Miles, and L. Moreau. PReServ: Provenance Recording for Services. In *Proceedings of the UK OST e-Science Fourth All Hands Meeting (AHM05)*, September 2005
- P. Groth, S. Miles, and S. Munroe. Principles of High Quality Documentation for Provenance: A Philosophical Discussion. In Moreau and Foster [135], pages 278–286
- P. Groth, S. Miles, and L. Moreau. A Shared Model for Documentation of Processes Enabling the Determination of Provenance. *ACM Transactions on Internet Technology*, 2007. Under Review

In addition, results of this dissertation were used as the basis of other applications, and published as follows:

- P. Townend, P. Groth, and J. Xu. A Provenance-Aware Weighted Fault Tolerance Scheme for Service-Based Applications. In *Proceedings of the 8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2005)*, pages 258–266. IEEE Computer Society, May 2005
- V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau. Security Issues in a SOA-Based Provenance System. In Moreau and Foster [135], pages 203–211
- S. Miles, S. C. Wong, W. Feng, P. Groth, K.-P. Zauner, and L. Moreau. Provenance-based Validation of e-Science Experiments. *Journal of Web Semantics*, 5(1):28–38, 2007
- S. Miles, P. Groth, S. Munroe, S. Jiang, T. Assandri, and L. Moreau. Extracting Causal Graphs from an Open Provenance Data Model. *Concurrency and Computation: Practice and Experience*, 2007
- S. Miles, P. Groth, S. Munroe, M. Luck, and L. Moreau. AgentPrIMe: Adapting MAS designs to build confidence. In *Proceedings of 8th Internation Workshop on Agent Oriented Software Engineering*, 2007

Chapter 2

A Critical Analysis of Provenance Systems

In this chapter, we provide a *critical analysis* of the state of the art for determining the provenance of data produced by applications that span multiple institutions. The analysis results in six conclusions:

1. The Service Oriented Architecture style is the primary software engineering approach to designing multi-institutional applications.
2. Provenance systems should take into account the important distinction between *past processes* and *prospective processes*.
3. A data model for provenance should be well-defined and independent from any one domain or technology to cater for multiple platforms and programs.
4. Multiple levels of abstraction must be supported to satisfy a range of queries.
5. The storage of provenance-related information should be separated from its collection point to ease management and query processing.
6. Causal dependency tracking is critical for understanding the provenance of data.

The rest of the chapter is organised as follows. We begin with a description of multi-institutional scientific systems and a review of the technologies used to implement them. This review identifies provenance as a critical concern for such systems, which leads to a discussion of provenance as a concept. From its use in art, several characteristics of provenance are derived including the relationship between provenance and process. After this discussion, the distinction between past and prospective processes is presented. With this context, a review of various systems for determining provenance is given with respect to their effectiveness for multi-institutional scientific systems. After this review,

we identify three cross-cutting concerns in the development of a provenance system for multi-institutional systems. Following the discussion of the cross-cutting concerns of abstraction, querying and causality, we present the conclusions of our analysis. Finally, we summarise the chapter.

2.1 Multi-institutional Scientific Systems

As the complexity of scientific problems has increased, it has become difficult to assemble the necessary resources (manpower, data, processing power, apparatus, etc.) to solve these problems within one institution or at one site. For example, in earthquake engineering, large physical resources such as shake tables and buildings are often not in the same place. Furthermore, many simulations require large scale computational resources often hosted at specialised supercomputing facilities such as the National Center for Supercomputing Applications (NCSA) [146]. Because it is difficult to perform earthquake engineering at a single site, the problem arises as to how these various distributed resources can be brought together effectively.

Foster, Kesselman and Tuecke stated this problem precisely as coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organisations, where a virtual organisation (VO) is defined by the rules that govern the sharing of resources between a set of individual and/or institutions [70]. VOs provide a useful model for coordinating the interactions of diverse resources controlled by multiple participants. It allows diverse resources to be assembled and qualities of service to be negotiated without the possibility of adversely effecting resource providers. A VO typically follows the following four stage lifecyle [172].

1. Resources are discovered.
2. The terms and purpose of the VO are negotiated.
3. The resources are used by the VO to generate a result.
4. The VO is disbanded.

During this lifecyle resources can be dynamically added or subtracted and the rules of the VO may be renegotiated. Systems that require the sharing and coordination of resources across multiple institutions for a particular scientific domain or question are what we term a *multi-institutional scientific system*. Such systems can be identified by three properties [69].

- Sharing - A multi-institutional scientific system requires the sharing of resources in order to accomplish its task.

- Heterogeneous - The kinds of resources available in a multi-institutional scientific system often vary widely. Furthermore, the rules and policies that govern the various resources vary dramatically across institutions.

- Dynamic - The resources available to a multi-institutional scientific system vary over time: members may join or leave, facilities may be introduced, removed or upgraded and the problem set may evolve. Furthermore, there may be a dynamic range of participation in the system. For example, some institutions could be core members of the system whereas others are on the periphery providing only a small contribution. Likewise, there may be a core set of problems that the system addresses and a series of smaller related problems.

Multi-institutional scientific systems exist in a variety of domains including earthquake engineering [146], high energy physics [85], chemistry [75], climatology [20], weather forecasting [148], astronomy [177], and medicine [166]. These systems rely on software and hardware infrastructure collectively known as the Grid [64]. This infrastructure enables systems to share and cope with dynamic heterogeneous resources provided by multiple parties. The software portion of the Grid is provided through common middleware (i.e. software libraries and services used by applications). There are various Grid middleware packages available including the Globus Toolkit [66], the UNiform Interface to Computing Resources (UNICORE) [173], gLite [79], and the Open Middleware Infrastructure Institute stack [14]. These middleware packages provide services ranging from security to resource discovery and thus ease the creation of multi-institutional scientific systems by allowing them to take advantage of existing software that deals with the complications of aggregating distributed resources.

2.1.1 Web Services and Service Oriented Architectures

One of the goals of these middleware packages is to facilitate interoperability between disparate systems. To encourage and strengthen interoperability, the Grid community, as represented by organisations like the Open Grid Forum and the Organization for the Advancement of Structured Information Standards (OASIS), have embraced World Wide Web technologies, particularly, Web Services [14]. By using standard technologies that are widely deployed such as eXtended Markup Language (XML) [33], Uniform Resource Locators (URLs) [19] and the Hypertext Transfer Protocol (HTTP) [63], Web Services provide cross-platform communication and data interoperability between applications. The adoption of Web Services means that the plethora of Web-based resources can also become part of the Grid infrastructure. For example, iSpecies.org combines data from the Google Scholar and Yahoo Image search websites with processing power from the National Center for Biotechnology Information (NCBI) to generate species specific web pages.

The adoption of Web Services also reflects the move towards the use of the Service Oriented Architecture (SOA) style of designing multi-institutional systems [69, 65]. SOA is an architectural style that views applications as a set of loosely coupled services communicating via a common transport. A service, in turn, is defined as a well-defined, self-contained, entity that takes input and produces output in accordance with a well-defined interface[1]. In the context of Web Services, a service's interface can be expressed in the Web Services Definition Language (WSDL) [42] and it can communicate using the common transport protocol SOAP [130].

The SOA style provides three benefits when building multi-institutional scientific systems. First, it *hides implementation* behind an interface allowing a service's implementation to change without impacting the user. For example, in the case of iSpecies.org, NCBI could change the underlying hardware or programming language it uses to process requests while not impacting the website. This is important in a multi-institutional context because an institution may wish to change the implementation of its services without having to involve collaborators. Loose coupling achieves the second benefit of the SOA style, *service reuse*. The ability to reuse services is particularly important in dynamic systems, where institutions are transient members and the landscape of resources changes. In such an environment, it is critical for institutions to be able to reuse and reallocate the services they provide to different collaborations. Finally, the SOA style encourages *platform independence*. By only requiring a common transport, the SOA style allows the use of the programming language, operating system, or implementation technique that is best suited for a particular service. For example, in the case of earthquake engineering, a shake table will use bespoke hardware and software whereas computational simulators would use a parallel programming platform like MPI (Message Passing Interface) [182].

Because of the benefits of SOA and Web Services, Grid middleware (such as Globus Toolkit 4 [66], gLite [79] and UNICORE [124]) has transitioned to these technologies. Furthermore, many multi-institutional systems from a variety of domains, including climate modelling [20], weather forecasting [148], and astronomy [177], have adopted these techniques and technologies. Bioinformatics is a particularly good example of a domain that has used and benefited from service orientation [189, 169, 81]. Databases containing a variety of genetic and biomedical data along with services to analyse that data have been made available by a variety of institutions including the NCBI and the European Bioinformatics Institute [170, 187]. These services have been then integrated in order to investigate biological problems such as Williams-Beuren syndrome [171]. Because the bioinformatics community has widely adopted the service oriented approach to building multi-institutional systems, we have chosen a case study from the field to evaluate our provenance solution.

[1] This definition was derived from [186], [121] and [69].

2.1.2 The Use of Workflows

Once institutions make their resources available as services using Grid middleware, they can then be assembled either dynamically or through an off-line process to create new multi-institutional systems. In order to tie services together, one technique that is often used is workflow and workflow enactment [197]. Workflows are scripts that describe the dependencies between different tasks. These workflows are then executed using a workflow enactment engine, which invokes various services to accomplish the tasks specified. In essence, workflows allow for a scripted form of VO where the script specifies the resources used by the VO to generate a result [70].

Workflow can be represented as graphs [76] and divided into two types, abstract and concrete [55]. Abstract workflows are those in which the task dependencies are defined but are not bound directly to a particular service. In contrast, concrete workflows are those where the tasks are bound to services. Software such as Pegasus [55] and FreeFluo [194] take abstract workflows and generate concrete workflows dynamically taking advantage of available resources. Users can create workflows directly either specifying them by hand or by using workflow editing software such as Taverna [98]. These workflows are then executed with a workflow enactment engine like Condor [76]. The trend in the development of workflow systems is to enable workflows to be specified at an evermore abstract level, which enables scientist to focus on science rather than the underlying technology [82].

As workflows become more generic and reusable, they are beginning to be exposed as services themselves [196]. Furthermore, services often rely on other services to achieve their functionality. Essentially, the use of SOA is encouraging the movement away from coordinating services centrally to multi-level hierarchies of workflows and peer-to-peer interactions between services [49].

The workflow community has identified a significant problem in current support for multi-institutional scientific systems, namely, the inability to reproduce scientific analyses or processes [56]. This inability is caused by inadequate information describing the *provenance* of results produced by such systems. Without provenance related information, it is difficult for scientists to repeat, analyse or understand results. Thus, a core component of the scientific method cannot be practised effectively by scientists working in a multi-institutional environment [56]. Beyond its necessity for the practice of science, provenance is also regarded as a key component in determining data quality [163].

While there are still some difficulties in determining provenance in systems that run only within one institution, the problem has already been extensively addressed. As long as a computational experiment or program is run within a single execution environment, whether it is an operating system, workflow enactment engine, or database, it is possible to capture everything that has occurred. For example, the Provenance-Aware

Storage System [137] captures the execution of any program executing within the Linux Operating System. In a single institution, it is possible to mandate the adoption of a particular execution environment. However, in systems spanning multiple institutions, it is difficult for institutions to impose such a mandate on each other. Furthermore, in multi-institutional scientific systems, it is highly unlikely that all provenance-related information pertaining to a particular experiment can be stored or aggregated into one centralised location. Because these systems are dynamic and no one institution would either want to be responsible for maintaining all the information after the experiment's end or give up its own information to some other institution, a unique challenge arises that is not present in a single institution scenario. Thus, the inability to determine the provenance of scientific results in multi-institutional scientific systems is a significant problem and the one we aim to address.

Before detailing our solution to the provenance problem, we analyse the available systems to see their deficiencies and strengths with respect to a multi-institutional environment. To help in this analysis, we first discuss the concept of provenance and its relation to process in more detail.

2.2 Provenance and Process

Provenance has a long history of usage in art. By understanding its usage in that domain, some general characteristics of the concept can be derived, which are useful in understanding the concept within computer systems, particularly the relation between provenance and process. We now discuss provenance in art from which we explicate two perspectives on process. When we discuss process throughout this dissertation, the word should be understood by its common sense definition: a continuous and regular action or succession of actions, taking place or carried on in a definite manner, and leading to the accomplishment of some result (Oxford English Dictionary).

2.2.1 Provenance in Art

In art, the term provenance is used to describe the history of ownership of a work of art (Oxford English Dictionary). For example, Figure 2.1[2] shows the provenance for a painting by Johannes Vermeer. Notice that several statements in the provenance are of the form "possibly...", this exemplifies the uncertainty of the provenance of the painting. This uncertainty about the provenance of an artwork is not unusual; on the contrary, it is the norm especially with works created before the 17th century [101]. Thus, much of the work in determining the provenance of an artwork is finding, analysing and judging the

[2]The image and provenance information in Figure 2.1 are used with permission of the National Gallery of Art, Washington.

Woman Holding a Balance c. 1664
Johannes Vermeer (Dutch, 1632 - 1675)

Possibly Pieter Claesz van Ruijven [1624-1674], Delft; possibly by inheritance to his wife, Maria de Knuijt [d. 1681], Delft; possibly by inheritance to her daughter, Magdalena van Ruijven [1655-1682], Delft; possibly by inheritance to her husband, Jacobus Abrahamsz. Dissius [1653-1695], Delft;(sale, Amsterdam, 16 May 1696, no. 1); Isaac Rooleeuw, Amsterdam; (sale, Amsterdam, 20 April 1701, no. 6); Paolo van Uchelen [d. 1703], Amsterdam. (sale, B. Tideman, Amsterdam, 18 March 1767, no. 6); Kok. Nicolaas Nieuhoff, Amsterdam; (sale, Ph. van der Schley, Amsterdam, 14 April 1777, no. 116); Van den Bogaard.[3] (sale, Maximilian I Joseph (1756-1825), Munich, 5 December 1826, no. 101, as by Gabriel Metsu). Louis CharlesVictor de Riquet de Caraman [1762-1839], Paris; (sale, Lacoste, Paris, 10 May 1830, no. 68). Casimir Péreir; (sale, Christie & Manson, London, 5 May 1848, no. 7); Péreir's son; by inheritance to Comtesse de Ségur-Péreir; (P. & D. Colnaghi & Co., London, and M. Knoedler & Co., New York); sold 11 January 1911 to Peter A. B. Widener, Lynnewood Hall, Elkins Park, Pennsylvania; inheritance from Estate of Peter A. B. Widener by gift through power of appointment of Joseph E. Widener, Elkins Park, Pennsylvania; gift 1942 to National Gallery of Art, USA.

FIGURE 2.1: Provenance of the painting Woman Holding a Balance by Johannes Vermeer [141]

authenticity of various pieces of documentation. This is a difficult task. It involves the physical examination of the work itself as well as research in catalogues, photo archives, correspondence and registrar records. Additionally, the task is made more difficult by false claims and actual suppression of provenance information, for example, when a piece has been stolen, smuggled or illegally excavated. However, if the complete provenance of an artwork (i.e. accurate documentation of its movement from its origin to its current location) is known, it can be an indicator of superior quality [101].

Three general characteristics of provenance can be taken from its use in art. First, provenance is demonstrated by accurate documentation. The role of a curator, collector, or historian is to find, verify and then present documentation so that it gives a convincing account of the artwork's history. Second, provenance is fundamentally about the past: the provenance of the work tells one where it *was*, not where it will be. Third, provenance is about tracking processes. In art, the process of importance is the movement of an artwork through a chain of custody or its restorations. In terms of multi-institutional scientific systems, the processes considered are broader. However, just as in art, knowing the process that led to a digital object lends the object greater authority and allows it to be better understood.

2.2.2 Two Perspectives on Process

Given that process is intertwined with provenance, we now aim to distinguish between two different but complementary perspectives that can be taken on the notion of process.

On one hand, when considering a future, *prospective process*, tools may be used to model such a process so that it can be analysed and simulated, in order to understand

its properties, and decide whether they meet requirements [23]. Being satisfied with a process model, notations can be used that describe the model and that are processable by computers [115, 12]. Such notations can also be optimised or compiled to ensure that computers can execute them efficiently [5]. In other cases, one may not be interested in identifying all the different steps that must be followed, but instead broad goals can be specified, from which a computer system is expected to infer the necessary actions to take, so that goals become satisfied [54]. What these process-related activities all have in common is that they are undertaken *without ever needing the process to actually take place*.

On the other hand, there is a whole set of activities that pertain to *past processes*, such as the calculation of an operation's actual duration, the determination of the actual input data used in an end result's production, or finding evidence that an experiment occurred as planned. These activities are about analysing processes *after they have taken place*. Broadly, activities related to past processes include judging reliability and quality, auditing, reuse and reproduction, managing process change and evolution, and ascertaining credit and ownership [83].

In the context of multi-institutional scientific systems, prospective processes are denoted by workflows, programs for services, and policy statements. Fundamentally, these define what *could happen* in the system. In contrast, a past process is the execution of a workflow, program, or policy. This execution is what *has happened* in the system. Past processes have information that a workflow or plan cannot contain. Examples of this information include the following:

- runtime decisions such as choice of branches or selected input values,
- errors or problems that arise during execution,
- performance characteristics such as the length of time for a program to execute in a particular environment.

Therefore, when answering questions related to past processes, workflows provide inadequate information and may result in possibly inaccurate answers. For example, a workflow may show that a particular service was called when, in fact, an error occurred and the service was never contacted.

From our brief discussion of provenance in art, we asserted that provenance is about the past. Understanding where, why, and how a result was produced is dependent upon knowing the past process that to led it, not how the process was intended to occur. Thus, computational systems for determining provenance should consider past processes.

Having described the environment we consider and refined the concept of provenance, we now analyse related work with respect to systems that support provenance in computational systems.

2.3 Provenance Systems

The subject of provenance has not gone without notice in the literature. Under the heading of lineage, a comprehensive overview of provenance related systems is given by Bose and Frew [27]. Similarly, a focused survey of provenance in the domain of e-Science is presented by Simmham et al. [161]. Two compilations of the current state of the art are given in [136] and [134]. The former provides detailed descriptions of the work done by various teams for the Provenance Challenge[3], a community effort to understand and compare systems addressing provenance. The literature refers to provenance using several other terms including audit trail, lineage [113], and dataset dependence [9]. We use these terms interchangeably to refer to provenance.

Our goal in this section is to review the various systems for the determination of provenance in computational settings (provenance systems) and judge whether they adequately address the needs of multi-institutional scientific systems. We divide these systems into the following categories:

- Version Control Systems
- Application Specific Systems
- Operating System Level Provenance Systems
- Provenance in Database Systems
- Distributed Debugging, Monitoring and Recovery
- Workflow-centric Systems
- Models

Each category of system has different inadequacies and benefits with respect to multi-institutional scientific systems. Therefore, we intersperse the review of each category with our own conclusions to provide a context for the analysis.

2.3.1 Version Control Systems

We begin our review with version control systems, which are systems that maintain multiple versions of a document or file to track changes in them. These systems allow the changes in files to be seen overtime, previous versions of files to be retrieved, changes in files to be associated with particular users, and comments to be made about those systems. Version control systems such as Concurrent Versions System [97] and Subversion

[3] http://twiki.pasoa.ecs.soton.ac.uk/bin/view/Challenge/WebHome

[147] support multiple users modifying files by maintaining a master file, including all previous versions of the file, on a centralised server. Users then must synchronize their local copy of a file with the server after editing the file or to retrieve the edits of another user. Thus, the user is responsible for indicating when a file is a new version. Version control systems also support the merging of changes between between the local copy of a file and the master file. Version control systems are widely used in software development, an overview of various version control systems and techniques in that context is given in [45].

Versioning file systems are similar to version control systems except that they are integrated with the file system. Every time a file is saved or modified on disk a new version is created. Thus, the user is not responsible for indicating the creation of a new version. Versioning file systems allow for the immediate undo of accidental operations and the long term history of a file to be viewed by the user [153]. Examples of versioning file systems include Wayback [47] and Elephant [153]. Like a versioning file system, the Mac OS X's integrated backup management system, Time Machine[4], keeps track of all changes to files on disk and provides an easy to use interface for retrieving earlier versions.

While version control systems are used by scientists everyday, for example when writing notes on Wikis[5], they cannot provide a comprehensive view of the provenance of data because they do not describe the process by which a file or document was generated. For example, if a JPEG image was produced by converting an image output from an image registration algorithm applied to two other files, a version control system would be unable to inform a scientist that JPEG conversion and image registration were involved in the creation of the JPEG image. Thus, the provenance for that end result is incomplete. The systems we describe next not only capture the changes to data (i.e. files) but also how those changes occurred.

2.3.2 Application-Specific Systems

Many provenance systems are designed for the needs of one application or domain. We review these systems here.

Some of the first research in provenance was in the area of Geographic Information Systems (GIS)[113]. Knowing the provenance of map products is critical in GIS applications because it allows one to determine the quality of those derived map products [113]. Lanter developed two systems for recording retrieving the provenance of map products in a GIS. The first system was a meta-database for recording data about GIS processes.

[4] http://www.apple.com/macosx/leopard/features/timemachine.html

[5] The Oxford English Dictionary defines Wiki as a type of web page designed so that its content can be edited by anyone who accesses it, using a simplified markup language. Implementations of Wikis allow for the revision history of a page to be seen and older versions retrieved [11].

The second system was for tracking operations in the Arc/Info GIS system's graphical user interface and command line [112, 114].

Provenance is also needed in the area of statistical analysis. S is an interactive system for statistical analysis where the results of user commands are automatically recorded in an audit file [18]. These results include the modification or creation of data objects as well as the commands themselves. S's AUDIT utility can then be used to analyse the audit file to retrieve the provenance of a statistical analysis. This utility can also create a script to reexecute a series of commands from the audit file.

Manipulating arrays is an important procedure in many computational models. In the context of the Array Manipulation Language (AML), the Sub-pushdown algorithm tracks all the array operations performed when executing AML programs [120]. Using an implementation of the algorithm called ArrayDB, the provenance of output arrays produced by AML programs can be retrieved.

The above systems are inadequate for multi-institutional environments because they are domain specific, program specific, and cannot work in distributed environments. The next five systems are designed for distributed settings but are again application specific.

Another system in the GIS domain is GOOSE [8], a tool for the creation of large geographical models by multiple participants. GOOSE was designed to work in a GIS environment that resembles a multi-institutional system where multiple tools, data sources, and researchers work together to create scientific results. GOOSE deals with this environment by providing a user interface to a GIS modelling engine and a shared repository accessible by multiple users. As the user goes about creating an object for a geographical model an operations log is kept. When the user stores the object the operations log is kept along with it. Then, when another user makes use of the object, they can retrieve the operations log for that object and thus its lineage. GOOSE mandates a common user interface and storage layer and is specific to one domain and is therefore not sufficient for the kind of environment we consider.

Another domain where provenance is of interest is satellite image processing. The Earth System Science Workbench (ESSW) is designed for processing satellite imagery locally [73]. It provides a lab notebook service for tracking processing steps and a No-Duplicate Write Once Read Many storage service for storing files. Essentially, the system provides wrappers for each program a user calls, which transparently gather the input and output of the executing programs. After the fact, ESSW can recreate the lineage of science objects from the data stored in the notebook service. Newer versions of the software, named ES3 [74], are closer to the operating system level provenance systems discussed below in that the software captures operating system calls. To aggregate the data produced by separate workbenches, a lineage server is proposed that merges data to produce a repository of lineage data that can be searched [26]. ESSW identifies important parts of a generic provenance solution, however, it is still tied to the particular

domain of satellite image processing.

The Collaborative Analysis Versioning Environment System (CAVES) and Collaborative Development Shell (CODESH) are designed to provide a virtual logbook for distributed collaborative groups [28]. Both CAVES and CODESH are interactive shells that users log into to perform various data analysis tasks. Similar to S, the systems track the user interaction with the shell and stores them as session logs. These logs are then published to a server allowing other members of the collaborative group to investigate and replay other users' sessions. The system is designed specifically for sharing users' interactive analysis sessions in multi-institutional collaboratories, however, it is not a complete solution because it does not capture what goes on outside the interactive shell.

In the context of distributed job execution on the Grid, work has concentrated on gathering statistical information and re-running jobs. Both Quill++ [151] and gLite Job Provenance [60] support these tasks. These systems are designed to be scalable and to mimimise the impact of provenance on job execution. Capturing data for provenance in job execution environments is an important part of an overall solution for multi-institutional scientific systems, however, both Quill++ and gLite are tied completely to their execution engines (Condor and gLite respectively) and thus are not adequate as a total solution for heterogeneous systems.

2.3.3 Operating System Level Provenance Systems

By capturing the execution of programs and their dependencies between each other through the operating system, operating system level provenance systems are more generic than the systems discussed above because they are application and domain independent.

One example of such a system is the Transparent Result Caching (TREC) prototype [184]. TREC uses the Solaris UNIX proc system to intercept various UNIX system calls in order to build a dependency map between those calls. Using this map, a trace of a program's execution can be transparently captured, which can be used for example to automatically build a makefile from the users interaction with the operating system.

Similar to TREC and ES3 [74], the Provenance-aware Storage System (PASS) integrates with the Linux kernel to capture all operating system calls, storing arguments to those calls as well as the dependencies between them. PASS stores this data in the Berkeley DB database, which allows a variety of queries and processing to be performed on the database [137]. Although PASS successfully completed the Provenance Challenge and was able to run the scientific workflow described, that workflow had to be run on a single computer [156]. PASS differs from TREC by integrating with the kernel directly. By not executing in user-space as TREC does, PASS is able to gather more information than TREC. For example, PASS captures read and write system calls whereas TREC does

not, which means that TREC must infer whether a specific file is input or output to an executing program. Furthermore, PASS supports queries that TREC does not [32].

Both PASS and TREC provide for the transparent documentation of program execution on *individual* computers. They are not designed for gathering and integrating data across multiple distributed computers. Additionally, because the data is captured with respect to the operating system, these systems have a problem in providing meaningful results to a scientist [32]. Thus, PASS and TREC do not provide the facilities necessary to determine provenance in multi-institutional scientific systems. However, they can provide valuable input data to a provenance system that is designed for the multi-institutional environment [137].

2.3.4 Provenance in Database Systems

Much of the data used by scientists for their experiments resides in databases. Furthermore, for some time, the database community has actively addressed the problem of the provenance of data stored within databases [190]. Therefore, we now briefly look at the work from this community.

Provenance in database systems has focused on the data lineage problem [53]. This problem can be summarised as given a data item, determine the source data used to produce that item. Woodruff and Stonebraker looks at solving this problem through the use of the technique of weak inversion [191]. Given some output data, a weak inversion function attempts to lazily recreate the input data used to generate the output. Unfortunately, this requires that a user who creates a new database view must also define a weak inversion function for that view. This technique has been used to improve database visualisation [192].

Cui et al. formalises the data lineage problem and presents algorithms to generate lineage data in relational databases [53]. The generation algorithms are similar to automatically creating weak inversion functions for every new view in a database, which allows users to "drill through" the lineage of a data item seeing the source data (tuples) that contributed to the given data item [51]. This work was also extended to deal with general transformations of data sets inside a data warehouse [52]. While data warehouses collect and cleanse a variety of data from many different sources, they are designed to work within one organisation [106, 3]. Furthermore, they are designed specifically to work with data stored and available using standard database technologies [106, 7]. Building on this and other work, Trio not only captures lineage information but also provides mechanisms to manage and query it [188]. Trio provides an extension to SQL for queries both over lineage and accuracy information [3].

Weak inversion approaches have benefits in terms of the size of data needed to be maintained as well as, in many cases, the ability to generate the inversion function

automatically. However, these systems are not as comprehensive as other approaches. They do not capture, for example, the parameters used by functions, the version of a workfow executed, or the published source of the data [161, 2]. Furthermore, in many cases it may not be possible to generate inversion functions because particular operations are not invertible.

Another system that looks at the data lineage problem in a data warehouse context is AutoMed [62]. AutoMed is designed to track the changes between data formats or schemas. As data evolves between differing schemas the lineage of those schema transformations can be tracked. AutoMed, then, is not focused on capturing the process by which data is generated but instead on how the formatting of data changes over time.

Buneman et al. [37] redefine the data lineage problem as "why-provenance" and define a new type of provenance for databases, namely, "where-provenance". "Why-provenance" is why a piece of data is in the database, i.e. what data sets (tuples) contributed to a data item, whereas, "where-provenance" is the location of a data element in the source data [37]. Based on this terminology a formal model of provenance was developed applying to both relational and XML databases. This emphasis on the kinds of queries that need to be supported was an important step. However, these are only two kinds of provenance queries that need to be supported by a provenance system.

In other work, Buneman argues for a time-stamped based archiving mechanism for change tracking in contrast to the diff-based mechanisms used by version control systems. It is argued that these mechanisms may not capture the complete process of database modification because there may be multiple changes between each archive of the database. Therefore, a diff-based mechanism is not a reliable approach for the development of a general provenance system [36].

To address the problem of scientists constructing databases "by hand" through manual methods such as entering data directly in the system or copying data from other sources, a copy-paste model of database update can be used [35]. This model attaches information to database fields about the kind of updates that occurred to it during a transaction (i.e. whether the field stayed the same, data was inserted or copied into it).

While work in the database community is significant, it fails to completely address the requirements of multi-institutional environments. First, the systems presented do not address how to capture the execution of applications that span multiple heterogeneous platforms. Second, they do provide mechanisms to track the services used to achieve a particular result. Third, database systems are often not designed to cope with the unstructured data and complex operations that scientific systems use. Finally, they do not address the need to determine the provenance of data when the provenance includes documentation provided by distributed sites.

2.3.5 Distributed Debugging, Monitoring and Recovery

Thus far, the systems we have described are either specific to a given application or are not designed to deal with distributed environments. In distributed systems research, there has been much work on debugging and monitoring distributed applications as well as recovering when those applications fail. A common theme in the research is generating a trace of execution, which can then can be used to either determine what went wrong in an application or, if the trace is detailed enough, restart the application after failure. Such a trace of execution could be used to determine the provenance of a digital object. Because these systems support both a trace of program execution and are designed for distributed environments, we review them now.

Hollingsworth and Tierney provides a survey of current monitoring and debugging frameworks and tools for Grids [96], which follows on from earlier work in distributed systems [103, 17]. One can divide the components of an end-to-end monitoring and debugging framework into three levels (cf. 20.1, p.322 [96]). At the first level is *instrumentation*, which is the integration of probes or sensors into software or hardware to measure their state. Sensors produce what is known as event data. This is data that a particular event, such as reading from the network, has occurred at a particular time. At the second level is *presentation*. Components, at the presentation level, gather event data produced by instrumentation, store it, and make it available for use. They are also responsible for archiving event data and managing the underlying sensors. Thus, the storage of event data is separated from the capturing and analysis of the data. The third level is *analysis*. Components at this level analyse monitoring data to, for example, find bugs, spot performance problems, and detect security breaches.

An example of such a framework is the NetLogger toolkit [95]. It provides libraries for instrumentation, tools for collecting and archiving event data, and a visualisation analysis tool. To work in heterogeneous systems, Netlogger specifies a common format for all event data. Netlogger was built to capture very detailed information in a high performance setting, however, one of the drawbacks to the approach they take is their dependence on accurate synchronised clocks on all computers using Netlogger. While this is a reasonable assumption in an environment where all computers are controlled by one administrator, in a multi-institutional environment synchronized clocks are the exception not the norm.

Frameworks such as NetLogger and Ganglia [122] provide detailed logs of events that occur in large scale distributed systems. Using this data, the connections between events can be inferred. For example, the NetLogger analysis tool has the ability to visualise sets of events on a "lifeline" by associating events that operate on the same data. Other systems infer the causal connections between events (i.e. that event A was caused by event B). For example, the DeWiz system uses event data from Grid-based logging systems to infer the causal connections between events building an event graph model

[108]. DeWiz analyses, such as determining if a message was lost in transmission, can be performed on the event graph model using the Grid. Instead of relying on an event based log, Aguilera et al. views distributed systems as a set of connected "black boxes" and develops algorithms for inferring the causal paths between events from message based logs. By visualising these causal paths, performance bottlenecks can be identified by developers [4]. The approach of tracking messages passed between components modelled as black boxes can also be used for mobile agent security [180]. These systems are interesting because they provide a trace of the execution of distributed applications. However, they are not data-focused and thus do not help in determining the provenance of data produced by applications.

If the logs created by an application are detailed enough, then, the application can be successfully restarted using what is known as rollback-recovery protocols. Elnozahy et al. is a survey of the research in this area [61]. Rollback-recovery protocols rely on snapshots of a distributed systems' state called checkpoints. Essentially, each program executing in the distributed system captures its state periodically, then, when an error occurs, these states can be meshed together to form a total picture of the system's state [39]. The system can then be "rollled back" to a previous stable state and restarted. One difficulty these protocols have is how to mesh together states created at different times. To address this problem, checkpoints are combined with message logging data to allow the relationship between checkpoints to be ascertained. One way of identifying the relationship between checkpoints is to determine the *causal connection* between distributed checkpoints [174]. This approach is superior to related techniques because it isolates executing programs from the failure of other programs, avoids synchronized checkpointing, and reduces the overhead on storage because only the most recent checkpoint is stored [61]. Rollback-recovery protocols have been widely researched, however, they have not been widely deployed in practice, probably due to the complex modifications to the operating system or runtime that they require [61]. For example, support for checkpointing in Java required rewriting the Java Virtual Machine [175]. In science, checkpointing has been used in the context of long running programs on multi-processor computers [152].

Both rollback-recovery protocols and monitoring systems provide mechanisms to trace the execution of distributed applications, however, this is not adequate for determining provenance. First, they capture events and not data and thus do not allow the provenance of data to be determined. Second, none of the systems described provide a mechanism for identifying a particular digital object and retrieving its provenance. Third, even if the systems were modified to have this functionality, the data they provide is at too low of a level to be of use to scientists. The systems do not represent the data at multiple levels of abstraction: a scientist cannot view past processes at a scientific level and then "drill down" to obtain more technical information. In summary, distributed debugging, monitoring and recovery systems provide valuable insight

into capturing processes in distributed environments but do not provide the requisite functionality to determine provenance.

2.3.6 Workflow-centric Systems

As discussed in Section 2.1, workflows currently play an important role in allowing scientists to design multi-institutional experiments. Thus, there are several provenance systems that are workflow centric. In the Provenance Challenge, eight of the seventeen systems were based on a particular workflow environment.

Barga and Digiampietri proposes that the workflow enactment engine should be responsible for the automatic collection of provenance information at runtime [15]. The system described, called REDUX, is a modification of Windows Workflow Foundation [31] and stores provenance information according to a multi-layered model. The four level model starts with abstract service descriptions. The second level represents the actual services used at runtime. Providing even greater detail is the third level which contains the data and parameters used during workflow execution. The final level contains runtime specific information such as timing information and information about the machines used during execution. Thus, a user can start from an abstract representation of the provenance of an item and drill down to the actual instance level information. Leveraging the model, the size of the information stored can be significantly decreased [16].

In the context of the myGrid project, the Taverna workflow enactment engine has been modified to generate provenance according to a Resource Description Framework (RDF) based data model [200]. Data generated according to Taverna's provenance model is stored in a triple store. Once stored, a query application programming interface (API), ProQA, enables a wide variety of provenance queries to be executed over the provenance RDF graph[199]. Because provenance information is stored as RDF, it enables both workflow annotations and provenance information to be queried over simultaneously [198]. Moreover, using the semantic web browser, Haystack, provenance can visualised and browsed [200]. One of the drawbacks to the Taverna provenance model is that it has remnants of its origins in bioinformatics, namely, the use of Life Science Identifiers (LSIDs) to label all provenance information. The use of LSIDs requires that the infrastructure to issue them must be in place and accessible in order for Taverna to create provenance information.

Like Taverna and Windows Workflow Foundation, VisTrails provides a graphical user interface for building workflows. However, instead of just capturing the execution of a workflow, VisTrails also captures how the workflow was created by the user [154]. This system is the first to be specifically designed to capture the workflow evolution process [72]. While VisTrails motivating domain is visualization [159], the provenance-based querying and workflow construction techniques such as "Query by Example" and

"Creation by Analogy" are domain independent [155]. Using the system, users can return to previous versions of workflows and workflow runs to compare their results. Furthermore, portions of workflows can be combined together to create new workflows and the origin of this new workflow is also tracked. Essentially, VisTrails allows for the user to serendipitously explore a space of workflows while not losing any of their previous work. The data describing the provenance of a workflow is captured using an open, self-describing data model to enable sharing and publication [72].

Another workflow system is Kepler, which aims at supporting multiple kinds of workflows from those designed for high-level conceptual bioinformatics experiments to those designed for job control and data movement in Grids [117]. To allow for a wide range of workflows, Kepler adopts a formal model that supports different computational styles. It contains components called actors that are very similar to services, producing output and taking input. Furthermore, it adds a notion of a Director that governs how a pair of actors interact. For example, a Director may state that one actor cannot execute until receiving data from another actor [117]. Based on this formal model, a provenance model centered on recording read, write and state-reset events in an event log is outlined by Bowers et al. [30]. Using the log, write events (i.e. the output of data) can be paired with several read events (i.e. the input of data) and a dependency graph can be built. State-reset events in the event log identify when previous read events can be discarded as precursors to subsequent write events. Using this approach, several provenance queries can be answered [30]. However, these queries are dependent on the workflow to bound the query by having an event log per workflow. Furthermore, to support queries about the functional relationships between inputs and outputs, the model depends on actors implementing only one kind of functionality (i.e. actors cannot support more than one function). With modifications to support explicit dependencies and metadata, the execution of workflows in Kepler can be captured with the Kepler Provenance Recorder [10]. The implementation allows for the smart-rerun of workflows based on the algorithms from VisTrails [10].

The above systems capture provenance-related information only from the workflow enactment engine, the Karma Provenance Framework [165], on the other hand, supports the capture of this information both from the workflow enactment engine and from the services used. This capture is facilitated by a notification model. The workflow enactment engine and services publish information about their execution to a notification services as XML. The Provenance Repository then listens for those notifications and stores them. This has the benefit that services can asynchronously submit their provenance information and thus not delay execution. The data model Karma used is centered on the notion of workflow activities (i.e. the invocation of a service). To tie the various XML submissions about each activity, a workflow identifier is attached to each. Furthermore, activities are ordered by attaching a logical time stamp generated by the workflow enactment engine. The model also assumes that each service instance within a

workflow run is identified uniquely [162]. A variety of provenance related queries can be answered by Karma using a combination of SQL queries and a Web Services API [164].

The Virtual Data System (VDS) is a workflow system, which focuses on data intensive scientific applications [202]. The system takes a functional approach: Executable applications are described as transformations (i.e. functions) and the input to those applications are described by derivations that bind particular data to a transformation (i.e. function calls). The syntax to describe derivations and transformations is called the Virtual Data Language (VDL) [71]. Workflows described in VDL can then be submitted to workflow planners such as Pegasus [55] or converted to run in workflow enactment engines such as Condor DAGMan [76]. When the concrete workflow is executed the parameters along with information about the runtime environment are stored in the VDS. Like Karma this parameter and runtime information is submitted back to the VDS by the invoked services. Once stored in the VDS, parameter and runtime information can then be combined with lineage information inferred from the VDL to answer a variety of provenance queries [201]. This reliance on the existence of a workflow to infer provenance is one of the major disadvantages to the VDS approach because it does not allow provenance to be determined in cases where the workflow no longer exists.

Szomszor and Moreau argues for infrastructure support for provenance in Grid and Web Service applications [178]. An architecture and implementation was developed around a workflow enactment engine recording data into a separate repository. To cater for reproducibility all the inputs and outputs to Web Services are recorded along with the workflow script and the interface definitions of services. The recording interface provided by the implementation supports both the asynchronous and synchronous submission of data. The implementation also has a validation capability that determines if a particular result is current by re-executing the workflow and comparing the execution to the one documented in the repository.

The workflow systems described here can all answer a variety of queries about the provenance of data. However, as multi-institutional scientific systems become increasingly decentralised, centralised workflow enactment engines do not have all the information necessary to provide the complete provenance of various results. For example, if a workflow enactment engine, A, called a service, B, which also executes a workflow, the provenance-related information stored by A would not contain the information about how B produced its results and thus the complete provenance of the output of A could not be determined. Furthermore, with the exception of VDT and Karma, the systems described only capture the workflow enactment engines view of a service invocation, this leads to the possibility of manipulation of provenance-related information produced by the workflow enactment engine. In a multi-institutional environment, both the workflow enactment engine and the service in an interaction need to record documentation of their involvement with each other. Finally, while all these systems have accessible well specified data models, they are all tied to the particular notion of workflow implemented

by each system. In heterogenous environments, a model is needed that is not tied to any particular workflow environment.

2.3.7 Data Models for Provenance

From the above discussion, it is apparent that data models are particularly important for representing data so that the provenance of results can be retrieved. The necessity of a common data model is underlined by Bose and Frew [25], who identify the lack of a common data model as preventing researchers from determining the provenance of their data.

Much of the work in data models is focused on employing Semantic Web technologies, and particularly RDF. One goal of using RDF is to facilitate interoperability [78]. Frutelle and Myers describes [78] the integration of RDF encoded provenance-related information produced by two independent execution environment and the queries performed over the combined data. Because a specific model is not defined, queries are performed under an interpretation of the commonality between the data produced by the two execution environments. While this is possible for two environments, it becomes increasingly difficult as the number of environments and data models increase. Scientific Application Middleware takes a similar tack by allowing the querier to decide what types of RDF metadata define provenance [140]. Allowing the querier to define provenance, makes it difficult to provide generic tools for reasoning about and analysing the provenance of digital objects.

As the availability of RDF data becomes greater [59], there have been several proposals to track when and by whom various pieces of RDF are created [77, 38]. These proposals extend the RDF triple format to contain extra elements containing attribution, security and timing information. Knowing the creator of an RDF statement is important but that information alone does not describe other important parts of its provenance including how and why the statement was made.

Golbeck and Hendler define a provenance data model in the Ontology Web Language (OWL) to take advantage of the language's greater reasoning capabilities [84]. The OWL language makes this possible by adding to RDF a larger vocabulary of terms and a richer formal semantics [123]. The OWL provenance model assumes that services in an application create metadata about their execution and that all inputs and ouput files are placed on the Web and can be addressed by URIs. The assumption that data being processed by services can be made available on the Web does not take into account the large amount of data stored in private databases or that cannot be easily addressed by URIs.

In contrast to RDF based models, ZOOM is built around a relational model with transitive closures [44]. The model consists of a set of Step-classes that describe a type of

service (i.e a function, actor, black box). When a workflow is executed a partial order of Steps are created. Steps are instances of Step-classes with the input and output of the Step attached. Thus, there is a direct link between specification and execution representation. Furthermore, Step-classes can inherit from other Step-classes allowing multiple levels of abstraction to be expressed. However, the system relies on the presence of a workflow definition (step-classes), which may not always be available, to understand the functionality of a given service. Furthermore, when using the model, queries must infer connections between steps by using time stamps and input/output matching. In a distributed system, timestamps may not be correctly ordered thus causing false connections between steps to be present.

2.4 Cross-Cutting Concerns

Having reviewed a variety of provenance systems with respect to multi-institutional scientific systems, we now analyse three cross-cutting concerns, namely, the level of abstraction systems address, the nature of queries, and the fundamental role causality plays in provenance.

2.4.1 Levels of Abstraction

The literature often discusses the "granularity" at which provenance-related information is captured [161]. We find this term confusing, it is not clear what makes a system fine grain or course grain. Instead of discussing granularity, we focus on the notion of various levels of abstraction within applications. There are three axes in our notion of levels of abstraction for provenance systems. They are the nesting of components, the nesting of data, and the vocabulary used to describe processes. We address each in turn.

From a software engineering perspective, applications are often built using a hierarchy of components. In object-oriented systems, objects contain other objects which contain objects themselves. Likewise, in a functional systems, functions call other functions which in turn call other functions. This nesting of components is critical to the reusability of software. It allows applications to be built by reusing and hooking together components to create new functionality. With respect to provenance systems, it enables provenance to be queried at different levels of component nesting. For example, a Web Service that plots a graph contains components that perform various mathematical and drawing routines. A provenance system can capture the fact that a Web Service was called and a graph was returned but it can also capture the use of the various drawing or mathematical components within the Web Service. This allows a user to view the provenance of a result in terms of high-level components and then progressively break these components down to view how the nested components contributed to result's generation.

Data, like components, can also be nested. For example, an array consists of elements, which may be arrays themselves. A user could be interested in the provenance of the array, its elements, or all of them together. Thus, a provenance-system needs to be able to determine the provenance of data as well as its nested elements.

Finally, a provenance system can use different vocabularies to describe the same process. For example, in bioinformatics many services take textual input and process them using utilities like grep or gzip. While these services are manipulating text, they are also performing scientific operations through those manipulations. Thus, we could, for example, describe a service that estimates the information content of a protein sequence as a text compression service. Both these descriptions are valid and useful. They represent the exact same operation at different levels of abstraction. Thus, it is important for a provenance-system to allow for different vocabularies to be used in the description of the same operation or process.

Hence, a provenance-system can cater for multiple levels of abstraction in three different and complimentary ways. It can allow provenance-related information to be captured about the nesting of components, the nesting of data, and using various vocabularies. As previously mentioned, a scientist should be able to start at a high level of abstraction, perhaps describing scientific operations, and then "drill down" to lower levels, for example, to see problems with program instructions. Therefore, it is critical to be able to capture and handle information about these various levels of abstraction. However, most of the systems, we have described do not cater for this need. They are essentially fixed at their systems designated abstraction level. For example, the database systems discussed only capture the transformation of tuples, whereas versioning systems only describe the changes between files.

Braun et al. [32] recognised the need to provide provenance-related information at different levels of component nesting and proposed to integrate the PASS system with workflow-based systems to achieve better coverage. This integration between systems is an important endeavour and will help in the uptake of provenance systems by scientists. Likewise, the Kepler system provides inspiration for describing workflows using various vocabularies as there system is designed to cope with various kinds of workflows ranging from high-level scientific workflows to those for Grid execution. To the best of our knowledge, Kepler does not allow for multiple abstraction levels within one workflow.

From this analysis, we believe that a provenance system should cater for multiple levels of abstraction in terms of nesting of components, nesting of data, and multiple vocabularies. One way a provenance system can cater to this need is to support queries at different abstraction levels. We now analyse how provenance systems support queries and provide a framework for thinking about queries with respect to provenance.

2.4.2 Answering Queries Related to Provenance

The fundamental goal of provenance systems is to enable users to answer questions about the results produced by their systems. Expanding on Buneman's categorisation of why-provenance and where-provenance, the W7 model [150] identifies the broad range of queries that fall under the heading of provenance. This conceptual model categorises provenance into "what", "when", "where", "how", "who", "which" and "why" questions and provides Entity-Relationship diagrams defining data elements useful when answering each question. To give an intuition as to the questions that would be asked by scientists using a multi-institutional system, we now list example queries mapping to each category.

- What were the inputs to this experiment?
- When did the experiment run?
- Where did the experimental data come from?
- How fast did the experiment execute?
- Which data sources were accessed while running the experiment?
- Why did this part of the experiment fail?

To answer these questions, the reviewed provenance systems, query the data they have collected using a range of implementations from extensions to SQL [3] to their own query APIs [199]. A common thread to all these systems is that, to answer questions related to the provenance of data, they rely on the equivalent of a dependency graph between data or events, which is then traversed to obtain an answer. In the case of Workflow-based systems, the dependencies expressed by the workflow are used to generate the graph. In systems such as PASS and ES3, the dependencies between operating system calls are explicitly captured and used to create a graph. Likewise, in database systems, weak inversion functions express dependency information. These graphs may not contain all the information necessary to answer a specific question, however, they provide a method to connect the information required.

Thus, such a dependency graph is a representation of provenance and share two properties. First, the edges of the graph represent connections or relationships between data or events. These relationships denote functions or operations applied to data. In general, they represent the causal connection between data or events. For example, the output of a function is caused by its inputs or the execution of a program is caused by the user double clicking an icon. The notion of causality as a general way to represent these connections stems from work in distributed systems [4, 108, 119]. Furthermore, in rollback-recovering protocols, causal connections provide benefits over synchronisation and transaction based approaches [174]. For example, a synchronisation-based approach

involves synchronising application execution with checkpointing, which, unlike a causal approach, delays application execution [174]. Likewise, using transactions requires that systems be serializable and thus is not a generally applicable rollback-recovery approach [174]. Second, dependency graphs are also often directed and acyclic. This is a consequence of causation flowing from the past towards the future. In the next section, we will discuss causality in a more detail.

These dependency graphs or *causality graphs* are built from data (or documentation) captured by the provenance system. The documentation captured by these provenance systems are partial or complete representations of the process of a particular application or environment. For example, in versioning systems, documentation of the file modification process is captured. Similarly, Workflow-based systems create documentation of the workflow execution process. Therefore, provenance queries can be generalised to extracting the appropriate causality graph from the documentation of process. No matter what form the provenance question takes whether "why", "how" or "when", the mechanism for queries is the same.

Hence, the framework through which we view provenance queries is three steps.

1. Identify the data to find the provenance of.
2. Extract the causality graph representing the provenance from documentation of process.
3. Traverse the causality graph to answer a specific question.

This view is simpler than the W7 conceptual model of provenance because it focuses on one concept, process, from which a range of provenance questions can be answered. The framework motivates the development of our solution to the provenance problem, which takes into account the need to make extracting causal graphs easy. Given that causality plays a central role in our framework for queries and thus how we represent documentation, we now discuss the definition we adopt.

2.4.3 Causality

From our analysis, it is apparent that knowing the connections and relationships between data is critical for understanding the provenance of digital objects. Specifically, causal relationships allow us to create the causality graphs described in Section 2.4.2. Thus, we discuss the notion of causation in more detail and provide a specific definition suitable for provenance in multi-institutional environments. Intuitively, most people have a general idea that causality is the relationship between cause and effect. However, philosophers have argued about its exact meaning for thousands of years[6]. We best understand it

[6]See Aristotle's Metaphysics and Posterior Analytics for a 2300 year old discussion of causation.

using a counterfactual definition [116], that is: if A had not occurred, then B would not have occurred, all else being equal.

Much of the work on causation in computer science has focused on inferring causal relationships from data sets [168]. Inferring causal relationships helps solve problems in a variety of areas including artificial intelligence [144] and data mining [160]. Pearl gives a systematic and mathematical treatment of causality [145]. He defines causality in terms of probabilistic functions and directed acyclic graphs. Specifically, the following definition for causal structure is given.

> A causal structure of a set of variables V (defined as probability distributions) is a directed acyclic graph (DAG) in which each node corresponds to a distinct element of V, and each link represents a direct functional relationship among the corresponding variables.

From this definition, Pearl goes on to present tools for mathematically reasoning about and inferring causality. A wide variety of techniques based on similar models are available for inference of causal relationship relationships, particularly using Bayesian methods [168]. Unlike this work, we are not trying to infer causality from some data set but instead rely on observations to document causality within systems, specifically within distributed systems. We use the notion of "observation by participation" that is a component within a distributed system can observe data or events when it processes such data or generates such events.

In distributed systems research, causality is discussed primarily with respect to asynchronous distributed systems. Such systems are modelled by sets of automata that perform three kinds of actions: sending a message (send event), receiving a message (receive event), and internal events [119, 458]. For modelling reliable first in, first out communication channels between automata, Lynch defines a *cause* function that maps a receive event to a prior send event in the same channel, β [119, 460]. This function is defined as follows:

1. For every receive event E1, E1 and *cause*(E1) contain the same message argument.
2. *cause* is surjective (onto)
3. *cause* is injective (one-to-one)
4. *cause* preserves order, that is, there do not exist receive events E1 and E2 with E1 preceding E2 in β and *cause*(E2) preceding *cause*(E1) in β.

Intuitively, the definition is saying that receipt of a message is caused by the sending of that same message. Lynch expands the notion of causality to include the idea that an

event occurring in an automata is caused by all the preceding events in the automata. This is termed the *depends on* relationship and is defined as follows [119, 465]:

An event E2 *depends on* E1 if one of the following holds:

1. E1 and E2 are events of the same automata where E1 precedes E2.
2. E1 is a send event and E2 is the corresponding receive event (as defined by the *cause* relationship above).
3. E1 and E2 are related by a chain of relationships of types 1 and 2.

The definition given in Lynch for *depends on* is the same as the *happened before* relationship described by Lamport for the ordering of events in a distributed system [111]. Thus, as stated in Lamport, this definition encompasses all possible causal relationships within an automata (i.e. because all causal relationships are temporal capturing, all *happened before* relationships will capture all possible causal relationships). This approach is conservative and appropriate in systems where causality must only be understood in an abstract manner, for example, where understanding the causal relationships existing within an automata is unnecessary.

However, we are particularly interested in understanding causality between data and thus we need to know in more detail how data was exactly transformed and the explicit causal connections between data items within automata. The definition provided by Lynch does not support the kind of detailed specific expression of the causal connection between data items that is required for the provenance of data to be adequately expressed. To provide greater detail, we, therefore, define causality in terms of a combination of the ideas presented above. Because multi-institutional scientific systems are distributed, we adopt the notion that the receiving of a message is caused by its sending. Secondly within automata (or services), we use the definition provided by Pearl, namely, that causality is expressed as functional relationships between variables (i.e. data). We expect such functional relationships to be expressed by the automata that executes the function, i.e. the observer of the execution. Using this functional notion allows causality between data items to be clearly indicated. Furthermore, it allows for the type of causality to be identified. For example, we can say not only that data item D2 was caused by data item D1 but we can also say that D2 was caused by a a Fast Fourier transformation on D2.

To bring these two definitions together, we amalgamate data and events together under the notion of an occurrence. An occurrence is either an event or a data item at event. Therefore, sending a message is an occurrence in which sending is the event and the message is the data at that event. This association provides for the location of a data item at given point in time as defined by an event. Again, we reiterate the point that

an occurrence can only be known by the executor of it, any other automata or service would only be able to infer the existence of the occurrence.

Using the notion of occurrence, we define our own notion of causality as follows:

Definition 2.1 (Causation). An occurrence O2 is caused by an occurrence O1 if one of the following hold:

1. O2 is functionally related to O1 (i.e. O2 has a direct functional relationship to O1 from Pearl).
2. O1 is the sending of a message and O2 is the corresponding receiving of the message (as defined by the *cause* relationship from Lynch).
3. O1 and O2 are related by a chain of relationships of types 1 and 2.

In summary, Definition 2.1 differs from the definitions provided by Lamport and Lynch in two important aspects. First, the definition deals with both events and data. Second, it provides a specific traceable causal connection between the reception of a message and subsequent sending of another message. These two aspects make Definition 2.1 more suitable for determining the provenance of results.

In this section, we have briefly discussed two views of causality from distributed systems and causal inference research. Based on these views, we defined a particular notion of causality suited to provenance and multi-institutional scientific systems.

2.5 Analysis Conclusions

We have presented a wide range of systems and models that address the problem of provenance in computational systems. We also have investigated three cross-cutting concerns: multiple levels of abstraction, queries and causality. From our analysis, we have come to the following six key conclusions.

1. *The Service Oriented Architecture style is the primary software engineering approach to designing multi-institutional applications.*

   In Section 2.1.1, we noted that the SOA style is suited to multi-insitutional scientific systems because it hides implementation, enables service reuse, and encourages platform independence. Because of these benefits, the SOA style has been used in a variety of scientific systems that consider a range of domains from bioinformatics to weather forecasting. Furthermore, the SOA style is being adopted by a variety of Grid-middleware platforms, including the Globus Toolkit. These platforms provide the basis for a large number of multi-institutional scientific systems [67].

Therefore, as reenforced by Foster [69], the SOA style is the primary approach to engineering these systems.

2. *Provenance systems should take into account the important distinction between past processes and prospective processes.*

   From Section 2.2.2, we have shown the distinction between prospective processes and past processes. There is a fundamental difference between *what is supposed to happen* and *what has happened*. Provenance is by definition about what has happened and is thus about past processes. Moreover, relying on prospective processes can result in a possible incorrect representation of the provenance of a result, for example, when an error or problem occurs during execution. Thus, it is critical to maintain the distinction between past and prospective processes in provenance systems.

3. *A data model for provenance should be well-defined and independent from any one execution environment to cater for multiple platforms, programs and domains.*

   Because multi-institutional systems span a wide variety of platforms, a common data model is necessary to allow provenance to be determined in these systems [25]. Therefore, specifying a data model that is independent from a particular execution environment is critical. An example of such an independent data model is the OWL provenance model [84]. However, while the model needs to be independent it must also be well-defined enough so that it can be clearly understood by multiple institutions. Thus, adopting purely RDF without specific schemas as proposed by Frutelle and Myers [78] is too lax. Furthermore, many of these systems we discussed were specific to a particular domain and thus cannot generically address the provenance problem arising from a variety of scientific systems.

4. *Multiple levels of abstraction must be supported by a provenance system to satisfy a range of queries.*

   Several authors have recognised the need for provenance systems to provide different levels of detail and work at different levels within the scientific application software stack [161, 32, 74, 15]. In Section 2.4.1, we recast this as the notion that provenance systems must allow for multiple levels of abstraction in terms of the nesting of components, the nesting of data, and multiple vocabularies for process description. We introduced the concept that scientists need to be able to drill down from a high level of abstraction to more levels in order to best satisfy their needs. Such "drill-down" functionality was inspired by Cui [50] and is similar to the ideas of ZOOM [44].

5. *The storage of provenance information should be separated from its collection point to ease management and query processing.*

   In Section 2.3.5, we presented a framework from Hoollingsworth et. al. for debugging and logging in Grid environments that separates the collection (the instru-

mentation level) and storage of data (the presentation level) [96]. This framework has been successfully used on the Grid via the NetLogger toolkit [95]. We adopt this approach, given that the data needed to be stored by provenance systems is similar to logging data and Grids are a core part of many multi-institutional scientific systems.

6. *Causal dependency tracking is critical for understanding the provenance of data.*

   Section 2.4.2 identified causality graphs as a common representation of provenance used by different provenance systems. Furthermore, the ability to extract causality graphs is at the core of our framework for viewing provenance queries. Causal dependency tracking has also had favorable results in checkpointing systems [61]. Thus, tracking and being able to extract causal relationships is a core attribute of any provenance system.

Many of the systems discussed support some of these attributes; however, none supports them all and thus provides a comprehensive solution to determining provenance in multi-institutional scientific systems.

2.6 Summary

In this chapter, a detailed description of multi-institutional scientific systems was given. This description identified the evolution of these systems towards greater decentralisation through the use of the SOA architectural style. After describing the environment considered, three characteristics of provenance were given by analysing its use in the field of art. Provenance was identified as being about the past and process was identified as playing a key role. Thus, the important distinction between past and prospective processes was explained. In this context, we reviewed a range of provenance systems from those designed for specific domains to those integrated with workflow construction and enactment environments. We identified three cross cutting concerns: support for multiple levels of abstraction, a framework for understanding provenance queries, and the role of causality in understanding the provenance of a result. To understand the nature of causality, we briefly reviewed work in the area and arrived at a definition of causality suited to provenance in multi-institutional systems.

From our analysis, we arrived at six conclusions, which can be summarised as follows. First, current systems do not cater fully for heterogeneous SOA-based multi-institutional systems; they are often domain or technology specific. Second, they do not not take into the difference between a prospective process (such as a workflow) and past process (such as documentation of the workflows execution). Third, they do not specifically define the causal nature of the relationships they express between data nor do they support multiple levels of abstraction and nesting. Fourth, they do not distinguish between

the data they capture and store (process documentation) and the representation of provenance that they retrieve from that data. Finally, open, generic, data models are important in allowing provenance describing multi-site processes to be retrieved.

In the next Chapter, we describe an open data model that supports high-quality characteristics and takes into account our analysis conclusions.

Chapter 3

A Model of Process Documentation

At the beginning of this dissertation, we outlined the need for the *provenance* of results in order to establish confidence in those results especially when they are produced by dynamic multi-institutional scientific systems. Furthermore, we introduced the notion that provenance is a *question* answered by querying documentation of an application's process. This novel distinction between provenance and process documentation allows provenance questions, unknown at the time of application execution, to be successfully answered provided enough documentation has been produced. Additionally, this separation of concerns allows components to be specialised for their specific role, either the creation of process documentation or its querying. To enable the creation and querying of process documentation by distributed software components, there must be some *shared* understanding between all these components. In this chapter, we present a data model for process documentation that provides this shared understanding.

But what should this data model look like? In Chapter 2, we arrived at six conclusions about the state of the art for determining provenance in multi-institutional systems. Taking these conclusions into account, this chapter specifies a model of process documentation that is compatible with SOAs, provides explicit veridical relationships between occurrences, and is both technology and domain independent. Furthermore, as discussed later, the data model is designed to support high quality characteristics derived from a use case analysis.

Therefore, the contributions of this Chapter are as follows:

- A more detailed description of the set of characteristics that define high quality process documentation.
- A precise conceptual definition of a generic data model for process documentation

that supports the creation of high quality process documentation and allows for the provenance of results to be determined.

The rest of the Chapter is organised as follows. First, a generic, shared model of process documentation is further motivated and a simple example application is introduced. Next, a set of characteristics are enumerated, which define high quality process documentation. After which the data model is conceptually specified. The specification begins by describing our definition of process, it then proceeds to describe how to create process documentation for applications once they are mapped to this perspective. After completely defining the model, we show how the provenance of a data item can be extracted from it. We also see how the characteristics defined earlier are supported. Finally, related work is briefly revisited and we conclude.

3.1 Motivation for a Generic, Shared Process Documentation Data Model

Just as the provenance of a work of art may include multiple owners, institutions, and handlers, the provenance of a particular digital object may include processes that occurred at different sites, at different institutions, and at different times. Because these processes may be different in terms of domain focus, underlying assumptions, and implementation technology, it is helpful to have a *generic* data model for their documentation so that the provenance of results can be traced back through these various interconnected processes.

Take the simple example application in Figure 3.1, at the request of a client initiator a mathematical calculation is performed on a collated sample, S, provided by a service that collates a sample from data provided by several sources. This application contains at least two sub-processes: Process A, the mathematical calculation, and Process B, the collation of the sample. The process documentation for Process A would document actions of addition, subtraction, and formula evaluation. On the other hand, the process documentation for Process B would document the collation of S, for example, by documenting who was responsible for the data items being collated, what institutions the data items are from, and whether the data items were produced experimentally or were synthesised from publications. The documentation for these two sub-processes differ in their level of detail and the kind of information included. However, using a generic model, we can still obtain the provenance of the numerical result that includes the whole of the process.

A model that is applicable to multiple processes could be generated on a case-by-case basis. However, a number of benefits arise from a data model that is not only generic but

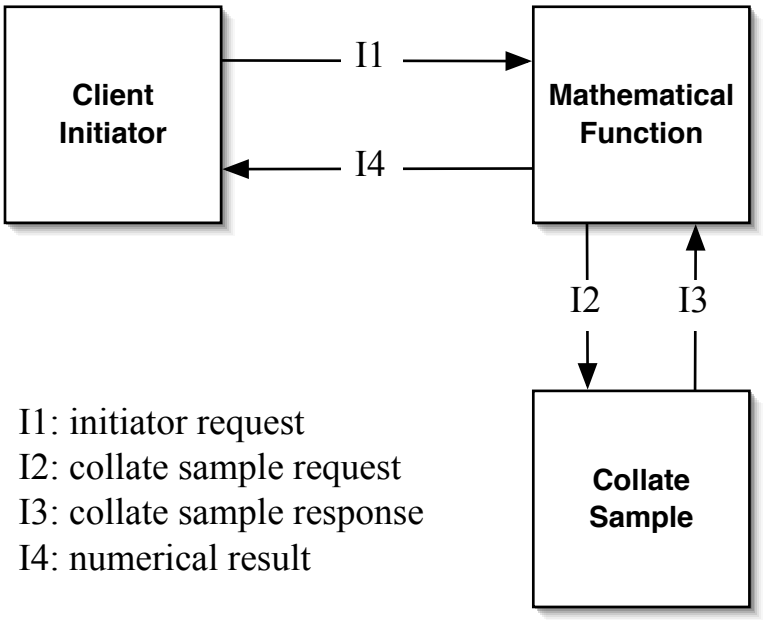


FIGURE 3.1: A simple example application

also shared across applications and application components. The benefits of a generic, shared model of process documentation are enumerated below.

1. **Future proofing**: It allows application developers to be sure that the process documentation their applications create today will be understandable by future applications and usable with process documentation generated by these future applications. This is vital since today's process documentation *will* be part of the provenance of tomorrow's results.

2. **Sharing**: It allows different institutions to share their process documentation without the need for conversion between models.

3. **Common tools**: Using it, tools can be designed that allow for the visualisation, reasoning, and filtering of process documentation irrespective of the domain.

4. **Independent creation**: With it, process documentation created independently by application components can be integrated together, which allows trailing analyses to be performed over unanticipated groupings of process documentation.

5. **Clear guidelines**: Application developers may want their applications to create process documentation, however, they may not know what data belongs in process documentation and what the structure of it should be. A generic, shared model provides a set of guidelines that help developers determine what data should be part of process documentation and how that data should be structured.

6. **Platform independence**: Internet applications are often developed using a variety of platforms (i.e. operating systems, programming languages, architectures). Such a model allows for provenance to be determined from process documentation generated by application components running on any platform.

3.2 Advantageous Characteristics for a Shared Data Model

While any generic, shared data model could provide these benefits and allow the provenance of results to be determined. The data model that we specify is designed to support the creation of *accurate* process documentation. In Section 1.3, we outlined a number of characteristics that help ensure accurate process documentation. We termed process documentation that adheres to these characteristics, high quality. These characteristics were derived from an analysis of use cases from several domains [126]. We looked at both the technical requirements enumerated in the analysis as well as the use cases themselves. We found that in a majority of the use cases, process documentation provides *evidence* that a process occurred. Thus, these characteristics are justified by philosophical arguments that equate process documentation to evidence. Beyond these philosophical arguments, a number of the characteristics also directly support a technical requirement enumerated in the use case analysis. We now revisit these characteristics and justify them in greater detail below.

Characteristic 1 (Factual). In the previous chapter, we noted that a number of provenance systems produce documentation that contains both factual and inferred information. With this combination, it is difficult to determine whether the process evidenced by the documentation actually occurred as described. Thus, we introduce the notion that process documentation should be factual: it should only be about what is known to have occurred in an application. To support this characteristic, we adopt the notion of observation by participation that is process documentation that evinces a particular application operation should only be created by the component that performed the operation.

Characteristic 2 (Attributable). In a court of law, evidence, particularly testimony, is judged by the person or institution who provides it. Furthermore, if it is found that the evidence given is false then remedial action can be taken against the provider. Similarly, if a user deems that process documentation is somehow erroneous, the user must know who is responsible for the creation of the documentation so that the party can be held accountable. By insuring users know the accountable party, they will have greater confidence in process documentation.

Characteristic 3 (Autonomously Creatable). In both criminal and scientific investigations, evidence is gathered at the most appropriate time and by the most appropriate person, device, or institution. By analogy, the distributed components of a

multi-institutional scientific system should be able to create process documentation at a convenient point in time without having to synchronise with any other component. Furthermore, process documentation created in an autonomous manner should be able to be collated together to present a complete representation of processes that occur across multiple components.

Characteristic 4 (Process Oriented). Evidence is only useful in a court if it can be put together to make a convincing case that a particular crime occurred. Likewise, process documentation is only useful if it can be put together to show that a process occurred. Thus, for a process documentation to be useful, it must be connected such that the process that led to a result can be determined from it. We term this notion of connectedness, process orientation. Therefore, for the data model defined here to meet the motivating use case, it must be process oriented.

Characteristic 5 (Immutable). In a legal setting, once evidence has been collected, it is criminal to tamper with it because it corrupts the view the court has on what occurred. Therefore, if we treat process documentation as evidence, once an application has created it for a particular execution, it should not be modified or deleted. Many times, it is not apparent that process documentation is useful until it is needed. Indeed, many users are caught off-guard when data they previously produced is deleted. Thus, it is important to maintain process documentation even when it is thought to be unimportant. Furthermore, in a multi-institutional setting, where the provenance of data might be used for scientific or legal verification, process documentation must not be tampered with. Hence, process documentation should be made immutable after creation.

Characteristic 6 (Finalizable). If process documentation is created in the context of a dynamic multi-institutional system, it is helpful to know when the components within the system have fully documented their processes so that the system can be disbanded or the component can be relinquished. Furthermore, it is helpful for users to know when full evidence (i.e. process documentation) has been provided for a particular process, otherwise, it is hard to know when a judgement can be made about the evidence. Thus, process documentation should be finalizable (i.e. markable as the final representation of a past process).

We term process documentation that possesses these six characteristics *high quality* process documentation. We now present our data model for process documentation called the p-structure. After presenting the p-structure in detail, we show how the design decisions for the p-structure support both these high quality characteristics and our analysis conclusions from Section 2.5.

3.3 The Data Model

We now present the various concepts that underpin the p-structure using our simple example shown in Figure 3.1. First, we define a specific notion of process represented by the p-structure. Next, we detail the data model and its constituent parts. After which, requirements on component behaviour are defined. Finally, we describe how provenance can be determined from process documentation organised using the model.

For each set of concepts, we provide a concept map [143] that gives an overview of the concepts and the relationships between them. Concept maps were chosen because they are designed for human consumption. Computer parsable representations are available in XML [138], OWL[1] and Java [89]. The use of concept maps was inspired in part by the Web Services Architecture specification [24], which also uses concept maps. The concept maps, shown in Figures 3.2, 3.3, 3.5 and 3.7, contain concepts represented by shaded rounded rectangles and relationships linked by lines between concepts. The words in the middle of a line denote the kind of relationship between the linked concepts. Maps are read downward, or if an arrow is present, in the direction which the arrow points. For example, the top portion of Figure 3.2 can be read as "Actors play a role", 'Actors have points of communication'. In the text, we italicise the first occurrence of concepts that appear in a concept map.

3.3.1 Process

The concepts discussed in this section are summarised by Figure 3.2. Applications are developed to address a variety of problems using different programming languages, design approaches, and execution environments. To represent this dynamic range of situations, we take a particular perspective on all applications, which embraces the principles of encapsulation and abstraction to enable process documentation to be created at varying levels of detail while still preserving coherence across applications and their components.

The perspective we take is to view applications as composed of entities, called *actors*, each of which represents a set of functionality within the application. Actors interact with other actors by the *sending* and *receiving* of messages through well-defined *points of communication*. Such a view naturally fits with service oriented architectures, one of the primary software engineering approaches for complex multi-institutional applications [69]. Our decomposition of applications is conceptual and is not restricted to applications already based on message passing. For example, as we view messages as information exchanged by actors, two threads communicating by a shared memory can also be viewed as actors.

To aid developers in the mapping of their applications to this conceptual perspective,

[1] http://www.pasoa.org/schemas/ontologies/pstruct025.owl

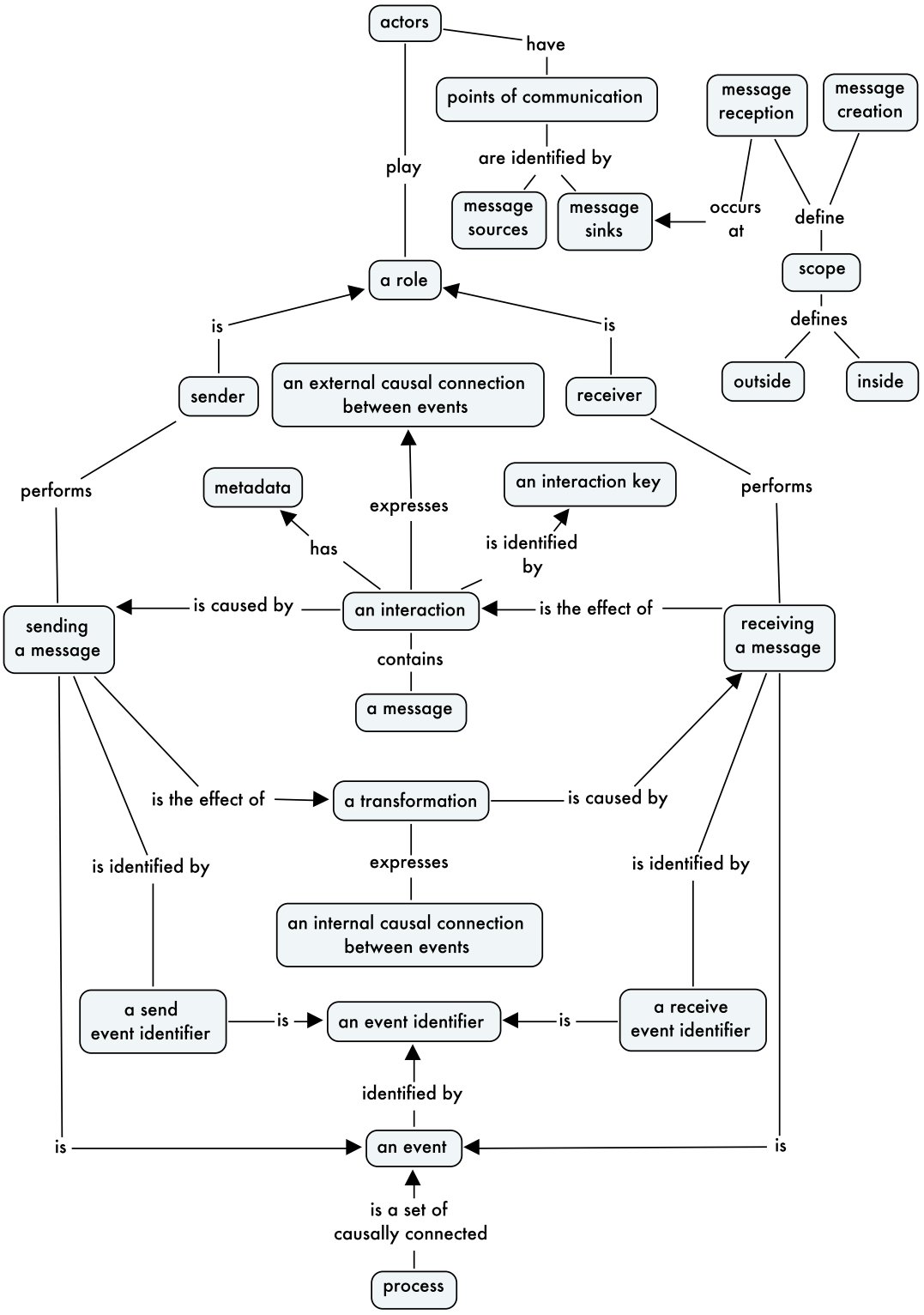


FIGURE 3.2: Concept map describing process

a software engineering methodology for the decomposition of applications into actors has been created [139]. For example, using this methodology, the example application was decomposed into actors that map to each step in the workflow i.e. Client Initiator, Mathematical Function, and Collate Sample. Each actor represents some functionality at a level of abstraction. Through the addition of actors, application functionality can be represented at a greater level detail. For instance, some of the functionality encapsulated by the Mathematical Function actor is represented in more detail by the Collate Sample actor.

When decomposing an application, specific points of communication are pinpointed and given an identifier called a *message source* or *message sink* that respectively denote where messages are sent and received. An actor may have any number of points of communication; the only restriction is that they are clearly identified. For example, in a Web Services context, a message sink would be the endpoint reference of an actor. An actor's functionality may only operate or work upon messages and by extension the data within messages that it has received or that have been created by the actor's functionality. Therefore, to define the boundary of an actor, we define the *scope* of an actor as the set of messages that have been received or been created by the actor (i.e. *message creation* and *message reception*). Thus, the scope defined here is not a static scope defined by software components but a dynamic scope that exists in the space of execution. Concretely, an actor that represents a procedure would contain within its scope not only all the inputs to the procedure and all the output data the procedure returns, but also all the data it may read or store in memory.

From scope, we define the notions of *inside* and *outside* an actor. Data is said to be *inside* an actor when it is part of a message that is an element of an actor's scope. Likewise, data is said to be *outside* an actor when it is not part of a message that is an element of an actor's scope. This definition also applies to the *events* within an application. If an event happens to data inside an actor, then it is also said to be inside the actor. Correspondingly, if an event happens to data outside an actor, then the event is also said to be outside the actor. Thus, both receive and send events are inside an actor when the message being sent or received is also inside that actor.

Scope provides a mechanism that allows data or events to be located within an application. However, a mechanism is also needed that expresses how events and data are connected together to form a process within an application execution. Given our message passing perspective, all data is contained within messages and the basic events that we consider are the sending and receiving of messages. Therefore, we define how these primitives are connected.

From Part 2 of Definition 2.1, we maintain that the receipt of a message is caused by the sending of that same message. The combination of a send and receive event along with the message exchanged is termed an *interaction*. Therefore, an interaction

expresses both a causal connection between a sending event and a receiving event as well as the contents of the message being exchanged. Specifically, an interaction describes an *external causal connection*, a logical connection formed where, for a given actor, the event inside the actor is caused by an event outside the actor. An interaction matches this definition because the internal receive event is caused by a send event outside the receiving actor. An example of an interaction is the sending of a response from the Collate Sample actor to the Mathematical Function actor.

With respect to interactions, actors can also play different *roles*. An actor may have the role of a message *sender* in one interaction and may play the role of a message *receiver* in another. An interaction can have *metadata* associated with it. This metadata is usually embedded within the message being exchanged. However, it may be associated in some other manner. Using this metadata, a sender can share information with a receiver that enables process documentation produced by these separate actors to be collated together. Specifically, a sender can generate a unique key for an interaction, termed an *interaction key*, and share this with the receiver. Thus, allowing the two actors to refer to a specific interaction using the same identifier. Based on the interaction key both the sending and receiving events can be identified by *event identifiers*. As we later show, these identifiers are crucial to organising process documentation.

Interactions express the causal connection between the sending and receiving of a message. However, they do not express the causal connection between the receiving of a message and the sending of another message within the scope of an actor. The sending of a message by an actor is caused by the execution of some functionality within the actor and this, in turn, is caused by the receipt of a set of messages. This causal connection follows from Part 1 of Definition 2.1. We term the causal connection between the receiving of messages and the sending of messages caused by an actor's functionality a *transformation*.[2] This causal connection is termed an *internal causal connection* because the events being connected are both inside the actor.

Consequently, in an application, the receiving of data by an actor may cause a transformation to occur. This transformation may cause the sending of data, which itself causes the receipt of data in another actor, which in turn may cause a transformation and so on. Thus, a process can be defined as follows:

Definition 3.1. (Interaction-based Process Definition) A process is a causally connected set of interactions and transformations.

Using this definition, precise organised process documentation can be created. We now describe a model, the p-structure, tied to this definition of process.

[2]Our model of process does not allow sequences of connected transformations inside an actor. This restriction ensures a simple decomposition rule: if more detail is needed about a transformation, the actor must be decomposed into more actors. Furthermore, the introduction of transformation sequences within an actor would necessitate the introduction of more primitives, thus, adding complexity to the model.

3.3.2 Process Documentation

The concepts discussed in this section are summarised by Figure 3.3. Thus far, we have discussed process documentation in the abstract. We now distinguish between the whole of process documentation and its individual parts. A *p-assertion* is an assertion that is made by an actor and pertains to a process. *Process documentation* then consists of a set of p-assertions. Actors that create p-assertions are termed *asserters*.

We place a restriction on all asserters that they only create p-assertions for events and data (what we termed occurrences in Section 2.4.3) that are inside their scope. Thus, asserters only create process documentation about what they know to be the case because they observe them through participation in either the creation or reception of messages. Scope defines participation and therefore defines observation. Asserters, then, are restricted to reporting on what they know to have occurred in a process, which supports the characteristic of factuality.

Every p-assertion also contains an *asserter identity*. This is the identity of the responsible party or parties for the p-assertion. Thus, the asserter identity will always contain the identity of the actor but may also contain, for example, the identity of the owner of the actor.

We now define two p-assertions that can be used to document the causal connections previously discussed.

3.3.2.1 Interaction P-assertions

Interactions are represented by *interaction p-assertions*, which contain four parts:

1. An asserter identity.
2. An event identifier.
3. A representation of the message exchanged in the interaction.
4. A *documentation style* describing how the representation was generated.

As with all p-assertions the interaction p-assertion contains an asserter identity. The event identifier uniquely identifies an event through a combination of a role identifier and an interaction key. The interaction key names the sender and receiver in the interaction by their message source and message sink respectively. It also contains a field that distinguishes this interaction from any other interaction happening between the message source and sink. The role identifier denotes whether the actor creating the p-assertion was the sender or receiver in the interaction and thus whether the event being documented is the sending or receiving of a message.

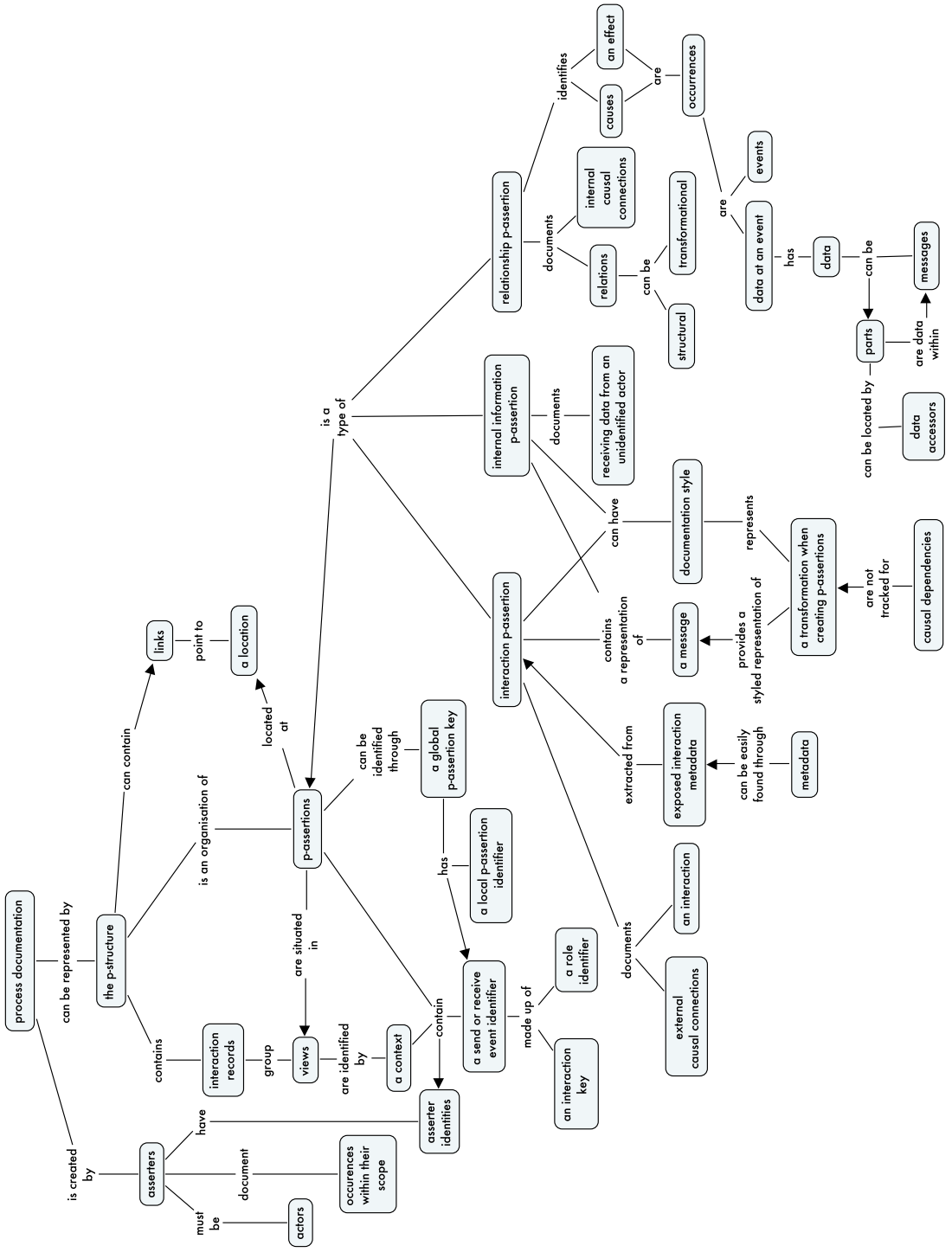


FIGURE 3.3: Concept map describing process documentation

The representation of the message contained in the interaction p-assertion often contains an exact duplicate of the message, but, in some instances it may not be feasible to have such a representation, for example, when the data being transferred needs to remain anonymous to users of process documentation or is of a large size. In the example application, this occurs when the sample generated by the Collate Sample actor is replaced with a reference to save storage space. To allow for these cases while still preserving an accurate representation, we allow a message to be transformed in a well-defined manner during the generation of a p-assertion, which is termed styling the p-assertion. The styling that is performed is defined explicitly by a *documentation style*. Causal dependencies are not tracked for these styling transformations because they pertain to the creation of process documentation as opposed to the production of application results. Likewise, the created p-assertions are not seen as application data and are not in the scope of an actor. For example, when the Collate Sample actor receives a large message, it may store that to a local database. To document the reception of the message, it generates a p-assertion with a reference to the data within it. While documentation produced after styling may not be as detailed as a copy of a message, it still provides critical evidence that a process occurred while allowing practical issues such as anonymization and scalability to be catered for. Therefore, like all other process documentation, it should should be immutable. We discuss the particular case of references in more detail in Section 6.6.2

Interaction p-assertions document both the data within applications as well as the external causal connections between the actors within those applications.

3.3.2.2 Relationship P-assertions

Unlike interaction p-assertions, *relationship p-assertions* represent internal causal connections between *occurrences*, which are defined as events or data items involved in events. For example, in our simple example application, an occurrence is the reception of an initiator request by the Mathematical Function actor. The data items in question can be entire messages or *parts* of messages. To locate a part of a message within process documentation *data accessors* are introduced, which are descriptions of how to find parts within p-assertions that document messages. Therefore, an occurrence within a relationship p-assertion is identified by locating the p-assertion where the event is documented and, if necessary, a data accessor.

A relationship p-assertion identifies one or more occurrences that are causes and one occurrence that is the effect of those causes. We limit a relationship p-assertion to one effect to make it easier to find the provenance of a particular occurrence: with this approach, there is no need to disambiguate which causes are associated with a particular effect. The specific relationship between these causes and the effect is described by a relation. The two types of causal relationships that are allowed between occurrences are

listed below:

1. *Structural.* Relationships of this type describe the composition of a data item from its constituent parts. They do not describe how a data item was composed from other data items, only that there exists a causal dependency between the data item and its parts. Structural relationships allow for the expression of the nesting of data. For example, a database is causally dependent on the entities it contains. If one of the entities in the database changed then the database itself would change. Relationships of this type can only be created between data items at the same event.

2. *Transformational.* This type of relationship describes, at some level of abstraction, a transformation and represents the internal causal connection from the receiving to the sending of messages. See Figure 3.2 for the conceptual description of a transformation. For example, a transformational relationship could represent the PPMZ compression algorithm applied to some input data within a received message (cause) to get compressed data within a sent message (effect).

Every relationship is described using a term from an appropriate vocabulary and the same transformation or structuring of data can be described by multiple relationship p-assertions using different vocabularies. However, in all cases, the relation described is causal as defined by Part 1 of Definition 2.1 (O2 is functionally related to O1).

Relationship p-assertions document both the data flow and control flow within an actor and thus are critical to understanding the process within an application.

3.3.2.3 Levels of Abstraction

The combination of interaction p-assertions and relationship p-assertions provide the information necessary to document processes. Furthermore, they allow process documentation to be created at different levels of abstraction. In Section 2.4.1, we described ways of describing parts of processes at multiple levels of abstraction: data nesting, multiple vocabularies, and component (or actor) nesting.

Relationship p-assertions cater for data nesting. Specifically, structural relations allow the nesting of data to be explicitly represented in process documentation. Thus, the causal connection between a container and its parts can be used in the determination of provenance. During the determination of a data item's provenance, a user can choose to include or exclude the provenance of the data item's constituents.

Relationship p-assertions also cater for multiple vocabularies. The same transformation in a process can be described using multiple relationship p-assertions each using its own

vocabulary. This enables the expression of what has occurred at both scientific and computational levels. Multiple vocabularies are useful for describing a transformation at different levels of abstraction within one actor. However, actors may implement the functionality they describe in relationship p-assertions by interactions with other actors and their transformations. This nesting of actors is common in software, therefore, we now discuss how the data model handles actor nesting.

Figure 3.4 shows the documentation for the collation process of the simple example application at two different levels of abstraction. Relationship p-assertions are shown by dotted arrow-capped lines with labels, where arrows point from effect to cause. Interaction p-assertions are shown with solid-arrow capped lines with labels, arrows point from the sender to the receiver of the message.

The left side of the figure shows documentation of the collation process at high level of abstraction. It states that the collate sample response was generated from the collate sample request. The relationship p-assertion provides an abstract description of the functionality the Collate Sample actor executed to achieve the response from the request. This level of abstraction may be useful for some users of process documentation who are interested in a "summary" of this activity. However, other users may need a more detailed picture of the collation process. To provide such a view, the actors used by the Collate Sample actor can be exposed. On the right hand side of the figure, process documentation is shown that includes the Collate Sample actor using a Database actor. This documentation states that on the receipt of a collate sample request, data items are retrieved from a Database, which are then collated together in to the collate sample response.

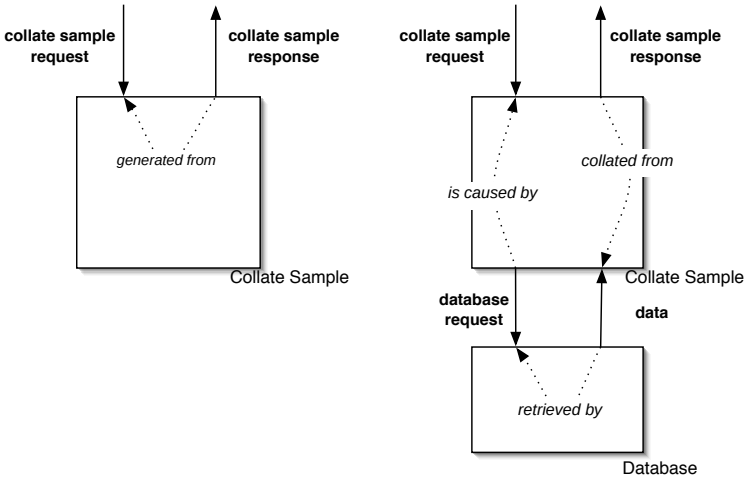


FIGURE 3.4: An example of documenting process at different levels of abstraction

If even further detail is required, more actors can be exposed that model progressively more detailed functional components within the application. No matter what level of

abstraction is required, process documentation for each level can coexist and complement one another to facilitate analyses.

3.3.2.4 Internal Information P-assertions

We now discuss one final type of p-assertion that facilitates abstraction and is introduced for convenience when using the model. It is often the case that a piece of data plays an important role in a process but the manner of its generation is not of interest. Examples of this include the time, the memory usage of an actor, and the configuration of an actor. All of these data items can be represented using relationship p-assertions and interaction p-assertions. However, using *internal information p-assertions*, the detail of how these data items were obtained can be abstracted away and one is left with just the data item and its basic causal connection to the process. We make the restriction that the data item is obtained by the actor either just before the sending of a message or just after the receipt of a message. This ties the data item explicitly to a particular occurrence in the process. Essentially, it allows an actor to assert, for example, that it sent a message at a particular time or that its memory usage was 20% after receiving a message. Thus, an internal information p-assertion represents the receipt of data by an actor from some other unidentified actor and is causally connected to the sending or receiving of a message by the former actor. The causal connections represented by internal information p-assertions are different for sending and receiving events and are as follows:

1. **Sending:** The sending of a message is caused by the receipt of the data within the internal information p-assertion, obtained just before the sending of the message.
2. **Receiving:** The receiving of a message causes the receipt of the data within the internal information p-assertion, obtained just after the receipt of the message.

We note that because internal information p-assertions represent the receipt of messages they can be used as causes within relationship p-assertions. Also, an actor can style the data during the creation of the internal information p-assertion. Thus, an internal information p-assertion consists of four parts: an asserter identity, the data, the documentation style of the data, and the event identifier of the event to which the data is causally connected. Internal information p-assertions allow data items to be made explicit without, the sometimes unnecessary, overhead of creating documentation for their generation.

3.3.2.5 The P-Structure

P-assertions contain the elements necessary to represent a process. However, without some organisation it would be difficult to discover distinct processes within process

documentation. Therefore, we introduce the *p-structure* which is an organisation of p-assertions that provides several elements to help isolate, find, and understand sets of p-assertions.

The p-structure situates each p-assertion in a container termed a *view*, which is identified by an event identifier. This means that p-asssertions are grouped together by the event that they are most closely associated with. The associations for the three types of p-assertions are as follows.

- Interaction p-assertions are associated with the event they document as given by event identifier within the interaction p-assertion.
- Relationship p-assertions are associated with the event identified in the effect occurrence of the relationship p-assertion. This allows users of process documentation to find the causes of a particular occurrence; once the occurrence is known the relationship p-assertion documenting its causes can be found in the same view.
- Internal information p-assertions are associated with the event that they are causally related to. Again, this provides a way to easily find data that is fundamentally about a given occurrence because the data is in the same view as the occurrence.

Each p-assertion within a view is given a local p-assertion id that, when combined with the event identifier for the view, allows the p-assertion to be uniquely identified within the p-structure. This combination is termed a *global p-assertion key*.

Because actors are only allowed to create p-assertions about events and data in their scope, all the p-assertions in a given view will have been asserted by the same actor. That is to say, the actor who participates in an event is the only actor that can make p-assertions about the event (observation by participation) and thus those p-assertions should be the only ones within a view. Therefore, to ease the identification of p-assertions produced by an actor, the common asserter identity shared between a view's p-assertions is also placed within the view.

Views contain p-assertions identified by local p-assertion id and an asserter identity. They also contain an element termed *exposed interaction metadata*. In Section 3.3.1, we introduced the notion that interactions can have metadata associated with them. This metadata is used to exchange interaction keys. It is also used to exchange information that helps demarcate and isolate processes. When the metadata is embedded in a message, it is documented using interaction p-assertions. Otherwise, it is documented using internal information p-assertions. Because communication systems have different message formats, metadata may be in different places within a message. To enable queriers to easily find this metadata without having to be aware of the underlying message format, an asserter can expose it within a view by placing a copy of it inside an exposed interaction metadata element. Essentially, instead of traversing the content

of p-assertions to find a piece of metadata, queriers can look in a well defined location that is independent of any particular message format.

We now look briefly at one mechanism, *tracers*, that is metadata used to demarcate processes. See Figure 3.5 for an overview.

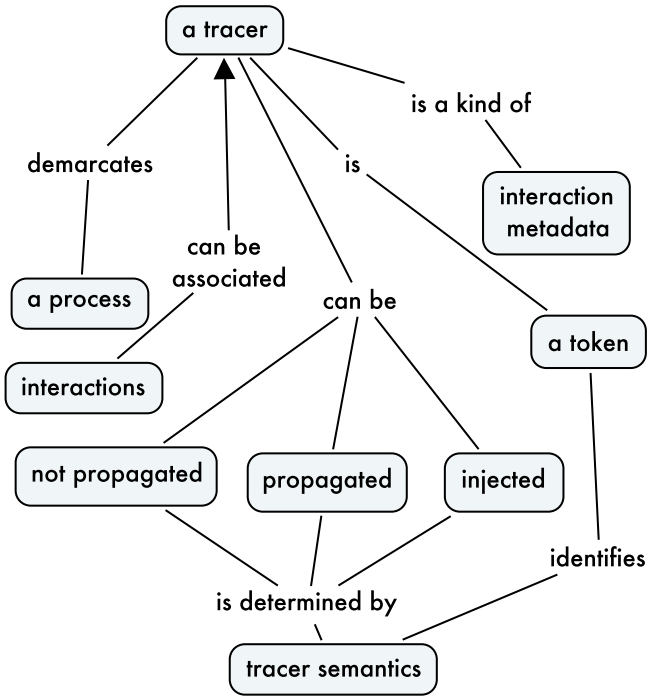


FIGURE 3.5: Concept map describing tracers

Tracers are tokens associated with interactions that identify the larger process that a particular interaction belongs to. Tracers are similar to transaction contexts within transaction processing systems. Just as a transaction context distinguishes a particular transaction, tracers distinguish or demarcate processes from one another in process documentation by identifying a set of interactions, typically involving several actors, that belong to a particular process. Actors can *inject*, i.e. add, tracers into an interaction's metadata. When an actor receives a tracer metadata, it can *propagate* or *not propagate* the tracer to subsequent messages that it sends. This is similar to the passing of a transaction context through the operations involved within a transaction. Injection and propagation are determined via *tracer semantics*, which are identified in the token. An actor has a choice as to whether it chooses to make use of tracers. When exposed interaction metadata contains a tracer, it is known that interaction documented was part of the process identified by the tracer. Thus, tracers assist in identifying particular processes within process documentation.

Given our interaction-centric perspective, the p-structure logically groups together the two views that document an interaction into what we term an *interaction record*. The

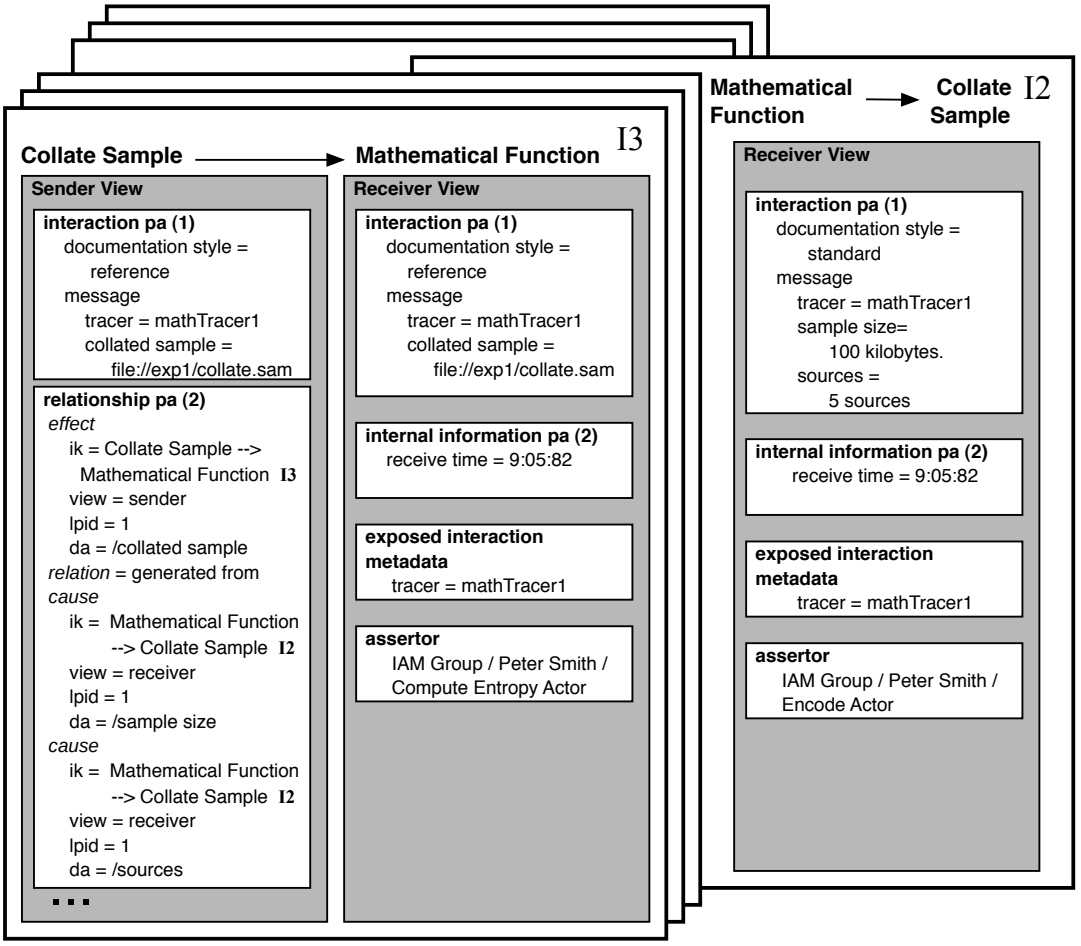


FIGURE 3.6: An example of the contents of a p-structure that documents the interactions I2 and I3 from Figure 3.1

grouping of views into interaction records has the added benefit of collating together process documentation created by independent actors. The p-structure then contains a set of these interaction records.

Figure 3.6 shows an example of a p-structure. The portions of the interaction records shown contain the documentation for the collation of a sample. Each large square is an interaction record labelled with the interaction key and label from Figure 3.1 of the interaction it represents. The label is in the upper-right hand corner of the square. The interaction key is shown by the name of the actor sending the message (acting as a message source) followed by an arrow pointing to the name of the actor receiving the message (acting as a message sink). Each p-assertion in Figure 3.6 is also followed by its local p-assertion identifier in parenthesis. The following abbreviations are used in the Figure: pa denotes p-assertion, ik denotes interaction key, lpid denotes local p-assertion identifier, da denotes data accessor.

Again, we note that the p-structure is implemented as a data structure in XML [138],

Java [89] and OWL.

3.3.3 Actor Behaviour Required by the Model

P-assertions are created autonomously by actors that document the occurrences within their scope. However, the p-structure does rely on actors following three rules that allow p-assertions to be correctly collated. These rules revolve around the correct creation and usage of interaction keys. They are as follows:

Behaviour Rule 1: Unique Interaction Key Rule *A sender asserting actor (i.e. a sender in the role of an asserting actor) must assign a globally unique interaction key to an interaction.*

There are many ways actors can obtain interaction keys. For example, an actor could generate an interaction key itself or obtain an interaction key from a naming service. The interaction key assigned to an interaction by a sender must be passed to the receiver in an interaction so that the receiver may also create p-assertions about the same interaction. For example, by passing the interaction key in a message header. To guarantee that this takes place, we introduce the interaction key transmission rule.

Behaviour Rule 2: Interaction Key Transmission Rule *A sender asserting actor must transmit the interaction key it assigns to an interaction to the receiver in that interaction.*

In order for the provenance of a piece of data to be retrieved, p-assertions must be associated with a particular interaction. The appropriate interaction rule governs how p-assertions should be associated with a particular interaction.

Behaviour Rule 3: Appropriate Interaction Rule *An asserting actor must use the interaction key associated with an interaction, I, when asserting p-assertions about I.*

These rules are simple and have been kept to a minimum so that actors can create process documentation as independently as possible. If these rules are not adhered to, the p-assertions created by various actors will not be able to be collated together to form a complete picture of a process. Specifically, the sender and receiver views of an interaction will not be placed within the same interaction record and will appear to be disconnected.

3.3.4 Extracting Provenance from the P-Structure

The concepts discussed in this section are summarised by Figure 3.7. The p-structure organises p-assertions so that the *provenance* of occurrences can be found. Specifically,

a *causality graph*, which represents the provenance of a particular occurrence can be extracted from the p-structure. Different forms of causality graphs (i.e. where the vertices and edges of the graph represent different entities) can be extracted from the p-structure depending upon usage. For example, one form of causality graph could have all edges representing a causal connection and all vertices being either a cause or effect, which could be useful in studying the purely causal relationship between occurrences whereas another form maybe useful when looking at the transformations applied. Here, we use a simple form for illustration purposes.

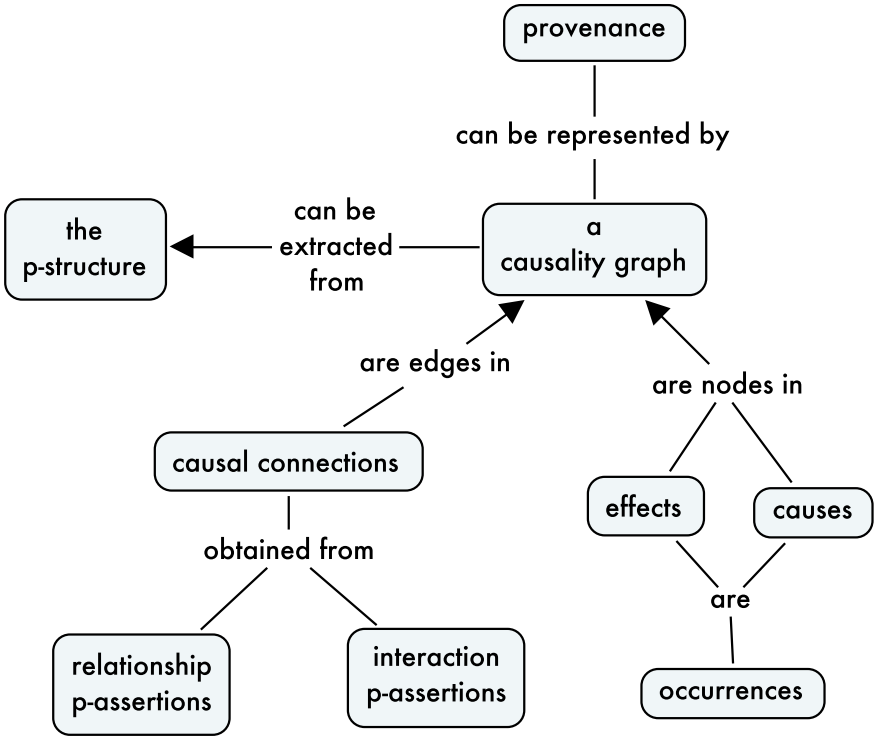


FIGURE 3.7: Concept map describing provenance

Figure 3.8 shows the provenance of a numerical result. The nodes in the graph are occurrences, in the role of causes, effects or both. The edges in the graph are hyperedges[3] and represent the causal connections extracted from relationship p-assertions. All arrows on the edges point from effect to cause. The external causal connections represented by interaction p-assertions are collapsed into the numbers shown to the bottom right of each node. These numbers map to the interactions shown in Figure 3.1. Internal information p-assertions are shown as annotations connected to the interactions by double-arrow headed lines. To save space, not all of these p-assertions are shown. The relationship p-assertions shown in Figure 3.6 are found in this figure.

Figure 3.8 evinces the claim that the provenance of an occurrence can be explained at different levels of abstraction. For example, the edge labelled as *is result of mathematical*

[3]A hyperedge is an edge that can have any number of vertices.

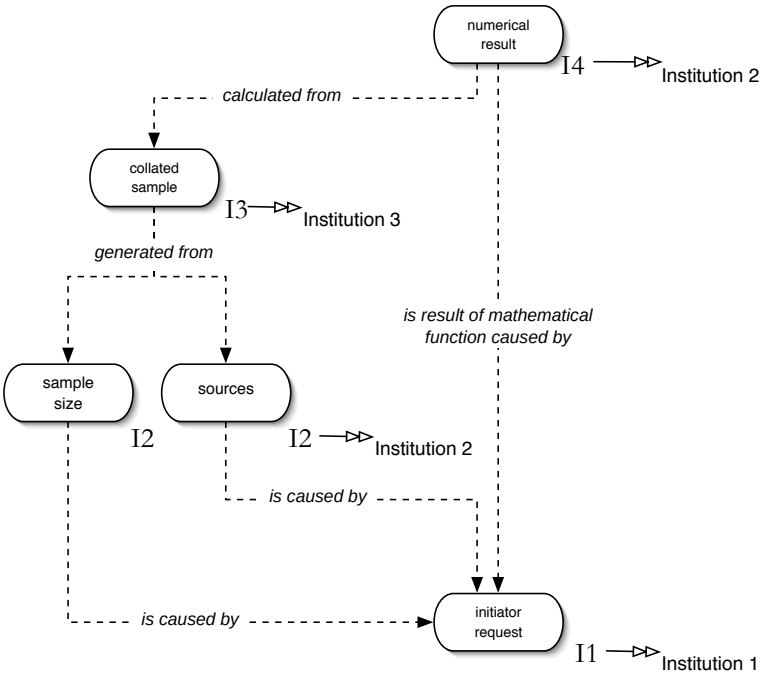


FIGURE 3.8: Causal graph describing the provenance of a numerical result

function caused by between *numerical result* and *initiator request* abstracts the three nodes and four hyper edges to its left.

3.4 High-Quality Characteristics Revisited

We now revisit each high-quality characteristic and discuss how each is addressed by the data model.

Characteristic 1: Factual

This characteristic is the basis of our notion of actors and thus is at the core of our data model. If application developers follow the data model specification, actors will only assert what is in their scope and thus process documentation will be factual. Essentially, the data model is a contract between the creator and the querier. In the case of factuality, it places an obligation on a creator that it only records data about what it knows to have occurred. Therefore, queriers can assume that process documentation will be factual and take action if they find that the contract is broken. The data model helps to enforce this contract by ensuring that effect of a relationship p-assertion is in the view with the relationship p-assertion, hence, the occurrence that is the effect must be documented as being part of the actor's scope. If the effect is not found within the same view, then it is a sign that factuality was not adhered to. Furthermore, a querier can check to see if the causes in a relationship p-assertion share the same assertor as the relationship

p-assertion, if they differ then it shows that factuality was not maintained. If routing information is available within messages, queriers can check to see whether a sender or receiver correctly identified themselves as such again helping to ensure factuality. Finally, we can detect when actors have conflicting views of the messages they exchange in interactions. This encourages actors to create factual documentation.

Characteristic 2: Attributable

The characteristic of attribution is supported through each p-assertion containing an asserter identity. Therefore, users of process documentation can identify who is responsible for a particular p-assertion and hold them to account for any information contained within the p-assertion. Cryptographic techniques such as digital signatures can be used to ensure that asserter identities are accurate and correctly associated with p-assertions. A more detailed discussion of the use of such security primitives is outside the scope of this dissertation, but can be found in Tan et al. [179].

Characteristic 3: Autonomously Creatable

Actors can create p-assertions in an autonomous manner through the use of sender generated interaction keys. Because the interaction key is a tuple based on information the sender has, the sender can guarantee its uniqueness without external dependencies. Thus, once an actor has either created or received an interaction key (via metadata which typically can be passed in message headers), it can then create p-assertions about that interaction without contacting an outside entity and at the time of its choosing. Thus, the decision to introduce the interaction key supports the collation of process documentation created by independent actors.

Characteristic 4: Process Oriented

The data model supports the creation of process oriented documentation. Two of its fundamental data structures, interaction and relationship p-assertions are about causally connecting occurrences. In Section 3.3.4, we illustrated how a graph describing the provenance of a particular data item could be extracted from process documentation. In the next chapter, we will further show that if applications correctly follow the data model, process documentation will be created such that the provenance of results can be determined. Finally, in Chapter 6, we will show that process documentation following the p-structure enables six practical use cases to be solved.

Characteristics 5 and 6: Immutable and Finalizable

These two characteristics can only be guaranteed by the correct storage and maintenance of process documentation. In this chapter, we have discussed the creation and organisation of process documentation and not how it is stored or maintained. In the next chapter, we will show how the recording of p-assertions in an appropriate repository can be used to guarantee these characteristics. We note that the organisation of

p-assertions into Views is helpful for supporting finalizable process documentation.

From this analysis, it is evident that documentation of process following the p-structure is high-quality. If actors create process documentation according to the p-structure, it will possess a majority of the aforementioned characteristics. This means that users of the process documentation can be confident that it is an *accurate* representation of a system's process. Hence, the provenance of results generated by queries over p-structure compatible process documentation will also be accurate. Being able to accurately determine the provenance of digital objects is vital for the practice of science using multi-institutional scientific systems.

3.5 Analysis Conclusions Revisited

As we have shown, many of our design decisions support high-quality characteristics. Likewise, other design decisions are in response to the analysis conclusions given in Chapter 2. We now revisit each analysis conclusion and show how each led to particular design decisions.

1. The Service Oriented Architecture style is the primary software engineering approach to designing multi-institutional applications.

The interaction-centric model we adopt reflects the SOA style. Just as the SOA style breaks down applications into services that communicate via message passing, the p-structure relies on applications being viewed as actors communicating via message passing where actors, like services, take inputs and produce outputs. Essentially, actors are services that can also call other services. Once adopting this view of applications, the concepts of capturing interactions and the internal functionality of actors follows, which leads to the interaction and relationship p-assertions concepts.

Ideas from Web Services, the primary implementation of the SOA style, have also influenced our design decisions. The concept of metadata associated with interactions is based on the header area within a SOAP message. This header area allows messages to be extended with additional information to cater for features such as security, routing, and message correlation [94]. Likewise, tracers, a form of metadata, allow for process demarcation by correlating messages. Web Services also led to the instantiation of the p-structure in XML.

2. Provenance systems should take into account the important distinction between past processes and prospective processes.

Process documentation is specifically about being evidence that a process has occurred. The data model assists asserters in creating process documentation about processes that have already occurred by providing a set of well defined concepts about the data that

needs to be provided when in process documentation. Furthermore, the p-structure organises process documentation so that questions pertaining to past processes, such as determining the provenance of data item, can be answered easily.

3. A data model for provenance should be well-defined and independent from any one domain or technology to cater for multiple platforms and programs.

The p-structure is defined at a precise conceptual level supported by Concept Maps designed for effective human consumption. We do not define the p-structure in terms of any one platform or technology and the conceptual model has been instantiated using three different technologies. Finally, the conceptual model is not tied to any particular scientific or business domain.

4. Multiple levels of abstraction must be supported to satisfy a range of queries.

The p-structure was specifically designed to cater for multiple levels of abstraction as discussed in Section 3.3.2.3. One of the purposes of introducing the mapping of applications into actors is to allow for the nesting of components to be expressed. When defining relationship p-assertions, structural relations were introduced specifically to cater for the nesting of data. Likewise, the decision to allow multiple relationship p-assertions to be created between the same occurrences was introduced so that multiple vocabularies could be used when describing a transformation that occurred within an actor. Finally, internal information p-assertions allow details of how internal information was produced to be abstracted away when it is convenient to do so.

5. The storage of provenance-related information should be separated from its collection point to ease management and query processing.

The next chapter focusses on the storage of process documentation. However, the definition of the p-structure is key to supporting the separation of the creation of process documentation and the querying of it. It provides a shared understanding between asserters of process documentation and queriers of it. Because of the p-structure, a querier can understand process documentation without having direct knowledge of the actors who created it. The p-structure's organisation of p-assertions into views, the pairing of views into interaction records, the introduction of exposed interaction metadata, the ability to identify each p-assertion uniquely, and the independent representation of transformations in relationship p-assertions were all designed to allow users of process documentation to traverse and understand it independently from its creators.

6. Causal dependency tracking is critical for understanding the provenance of data.

Causal dependencies or causal connections are at the heart of the p-structure. Two types of p-assertions, relationship p-assertions and interaction p-assertions are designed to capture causal connections. They follow directly from the definition of causality we defined in Chapter 2. Relationship p-assertions allow the functional relationships from

Part 1 of Definition 2.1 to be represented. Likewise, interaction p-assertions allow the causality of sending a message and another party receiving the message from Part 2 of Definition 2.1 to be represented. Thus, two of the core components of our data model are designed to express causality.

From this review, we have shown that the design decisions leading to the p-structure can be traced back to either the conclusions we obtained from our analysis of related work or the high-quality characteristics enumerated in Section 3.2.

3.6 Related Work

In Chapter 2, we reviewed a number of provenance systems. Here, we briefly revisit some of those systems' data models and distinguish them from the p-structure. A major difference between the p-structure and other models is that it is defined in terms of a conceptual model designed for human consumption instead of relying on a computer parsable syntax or formalism. This allows the model to be instantiated in a variety of languages and enables both syntactic and conceptual compatibility. For example, process documentation from two institutions could both be represented by the p-structure in XML and thus be conceptually and syntactically compatible. However, two institutions could adopt the p-structure but use different languages (i.e. one uses XML, the other uses serialized Java objects) and then their process documentation would only be conceptually compatible.

Unlike the workflow centric models from MyGrid [200], Kepler [29], REDUX[15], and Szomszor [178], the p-structure is specifically designed to handle both the workflow enactment engine's and service's view of an interaction. Furthermore, even workflow-based systems that support the modelling of both views of an interaction, such as Karma [165], still require the centralised coordination of the workflow enactment engine to properly demarcate process documentation from different workflow runs. In essence, the p-structure supports a peer-to-peer topology whereas these other models are designed for a centralised layout.

Additionally, these workflow-centric models often rely on the workflow definition to provide the causal connections between service inputs and outputs. Relying on the workflow definition is not always possible because services are implemented using a variety of implementation languages. The p-structure, on the other hand, provides a generic way to express causal connections independent of any particular workflow definition.

As discussed in Section 2.4.1, support for multiple levels of abstraction is key for scientist to easily use the process documentation. Many of the provenance systems discussed do not innately support multiple abstraction levels. The p-structure, however, is designed

from the ground up to support different levels of abstraction through an approach that enables both high level and low level descriptions of actor functionality to be expressed at the same time through the decomposition of actors and multiple vocabularies for relationship p-assertions. While ZOOM [44] supports abstraction of actor functionality, it does not support the documentation of causal connections in a robust manner because it relies on time stamps. Furthermore, it does support a direct mechanism like tracers to distinguish between processes nor does it support attribution inherently.

The p-structure strikes a balance between being open enough to support a variety of applications while being structured enough such that generic tools can be reliably built upon it. This is in contrast to Myers et al.'s approach [140], which advocates a completely open model with the only structure coming from the underlying syntax of RDF. Because of this totally open approach, different institutions could have completely different ways of organising process documentation. We believe that this approach prevents generic provenance specific tools from operating over a range of process documentation generated by different institutions. Likewise, process documentation tailored too specifically to one platform like PASS [156], S [18], CODESH [28] or Trio [188], prevents provenance from being determined across multiple institutions that use a variety of platforms.

Therefore, the p-structure is a novel data model for use in determining the provenance of results produced by systems that span multiple institutions.

3.7 Summary

In this chapter, we discussed how a generic, shared data model of process documentation facilitates the sharing of process documentation between institutions; it allows for the development of tools that work across domains and applications and for the creation of future-proof process documentation by independent application components running on a variety of platforms. We presented a detailed conceptual definition of just such a data model, the p-structure. The presentation was facilitated by the use of concept maps.

The specification of the p-structure began with a definition of process that embraces the Service-Oriented Architectural style. Using a simple example, we showed how the p-structure enables multiple levels of abstraction to be supported and how the provenance of a data item could be extracted from process documentation following our data model. After fully specifying the p-structure, we demonstrated that it supports the creation of high-quality documentation of process as well as the analysis conclusions drawn in Section 2.5.

The contributions of this chapter were two fold: a detailed description of high-quality characteristics and the precise conceptual definition of a data model that supports these characteristics. While other systems may allow the provenance of digital objects to

be determined, they do not directly consider the quality of the information on which they make that determination. Through an analysis of various use cases, we discovered that process documentation is used as evidence that a process occurred. Using this link between evidence and process documentation, we enumerated six characteristics of high-quality process documentation. By making these characteristics explicit, users are aware of the accuracy of the information they are relying on when trying to understand the provenance of their data. Furthermore, the p-structure is specifically designed to support these characteristics, which helps to ensure users' confidence in process documentation. This direct support for certain high-quality characteristics is novel.

The p-structure not only supports the creation of high-quality process documentation, it also provides a shared understanding between creators and queriers of process documentation. This separation of concerns allows queriers and creators to specialise on their particular activitives. The p-structure is specifically designed to allow for this clean separation. Finally, the adoption of a common data model such as the p-structure is critical for the origin of data to be tracked across applications that engage multiple institutions. Without such a technology and domain independent representation, it is difficult for the provenance of data produced by such applications to be determined.

The creation of process documentation compatible with the p-structure is the first part of our solution to the provenance problem. In the next chapter, we will see how correctly recording process documentation in specialised repositories creates a comprehensive solution.

Chapter 4

Recording Process Documentation

Chapter 3 presented a data model that enables process documentation to be created by separate, distributed application components and for that documentation to be organised in such a manner that the provenance of data items can be determined. Given the distributed nature of process documentation, this chapter presents a solution to *aggregate* and *store* it persistently over the long term while guaranteeing its immutability and conformance to the p-structure. We do not consider the storage layer itself. Instead, we consider the recording of data into the storage layer in a manner that preserves high quality characteristics.

The solution presented consists of four main contributions:

1. An architectural element for the storage of process documentation known as the provenance store.
2. Patterns for the appropriate deployment of provenance stores within applications.
3. A technique for connecting process documentation stored in a distributed manner.
4. A formally defined protocol that actors use in order to record process documentation into provenance stores. The protocol is asynchronous, stateless, and helps ensure that high quality process documentation is recorded.

The rest of the chapter is organised as follows. It begins by describing the notion of a provenance store and motivating its introduction. Next, the deployment of provenance stores within applications is described using patterns. Because documentation of process can be created by multiple actors, at multiple locations, in multiple institutions, we present a technique for the connection of process documentation stored in a distributed fashion. The chapter then proceeds to detail the protocol for recording process

documentation. A formal model of the protocol is given along with an analysis of its properties. Finally, the chapter is summarised.

4.1 The Provenance Store

Any architecture consists of sets of basic elements that are structured and connected to form a complete design for a system. In this section, we introduce one such basic architectural element that answers the question of *where* process documentation should be stored. The architectural element introduced is the *provenance store* which is defined as a specialised actor obligated to persistently store p-assertions. Furthermore, the actor must provide consistent and well-defined interfaces for the recording and querying of process documentation. The introduction of two separate interfaces for recording and querying reflects the important distinction between the documentation of process and determining the provenance of results from that documentation. The protocol defined later in this chapter specifies the recording interface, the query interface is beyond the scope of this dissertation and is defined elsewhere [125]. Thus, the provenance store functions as a server that actors contact to record their p-assertions and query for the provenance of digital objects.

Before further detailing the use of provenance stores within application architectures, we now justify their introduction by analysing several possibilities for the storage of process documentation.

One option is to store process documentation with the result itself. This solution has the benefit that once a result is found, its provenance is also immediately available. However, this solution suffers from several drawbacks. First, when a multi-institutional scientific system (or any system) executes, the final result may not yet be known, but process documentation needs to be simultaneously generated with the execution. Hence, process documentation cannot be stored with the result because it does not exist until the execution's end. Second, the result of a system may not be a digital object but may instead be a physical one (i.e. not contained in a computer system); thus, it becomes difficult to store process documentation with the result itself. Last, large amounts of process documentation can potentially be generated, perhaps related to the provenance of multiple results. In such cases, storing process documentation with each individual result is inefficient and, in many cases, may be impossible from a storage viewpoint. For example, in the bioinformatics application presented in the next chapter the quantity of process documentation for each result is larger than its size by a factor of 40.

Alternatively, asserters could keep process documentation locally. The benefit of this approach is that asserters have complete control of the p-assertions they create. However, this solution also has its own drawbacks. In a distributed system, it is often the case that actors are transient. Therefore, if an actor disappears, the process documentation

it has stored may also disappear. Furthermore, actors may not have persistent storage and thus may not be able to store p-assertions in a permanent manner. Even if an actor has access to some persistent storage, it may not have enough capacity to keep all the process documentation for every process that it contributed to.

Finally, if each actor maintains its own p-assertions, then enforcement of access control across an application's process documentation becomes challenging since it would require each actor to track the access control privileges of a myriad of queriers.

Given these difficulties, the approach we adopt utilises provenance stores, which allow process documentation to be stored during execution for multiple physical or digital objects; second, because they are specialised for the storage of p-assertions, provenance stores can be built to persistently store large amounts of process documentation and to deal appropriately with problems of security and access control. We note that a provenance store is a *role*. Any actor, as long as it supports the provenance store interface and designated qualities of service, can be a provenance store. Therefore, in actual deployments a provenance store can be integrated with other actors.

We now discuss how developers can deploy provenance stores within their application architectures such that process documentation can be recorded effectively.

4.2 Deployment Patterns

To be able to cope with documentation from a multi-institutional application, provenance stores may need to be distributed since there can be a large quantity of data, in a large number of p-assertions, recorded by a large number of actors, each with their own security domain, privacy requirements, etc. The requirement for creating process documentation in distributed applications, such that all documentation related to their execution can be retrieved again, presents a developer with several deployment problems. These include in what computer the provenance store should be located, how many provenance stores to deploy, and where in the network topology to deploy provenance stores. To address these problems, a set of *deployment patterns* are now introduced.

A pattern [7, 6, 80] describes a solution to a common design problem; the solution described must strike a balance between being concrete enough to be applicable and abstract enough so that it can be applied to a range of similar problem situations. The patterns presented here provide reusable solutions that developers can use to integrate p-assertion recording into their application architectures. The format of these patterns is as follows:

Name Each section title is a short name for the pattern that reflects the solution.

Diagram A diagram that shows the pattern visually. Diagrams have a common visual appearance. Provenance Stores are labelled and denoted by a 3D cylinder. Actors are denoted by boxes. A single message exchange is denoted by a line with an arrow head. The arrow denotes the direction of the message flow. Dotted lines follow the same convention but denote multiple message exchanges. A circle above a line denotes information inside the message.

Context The situation in which the pattern applies and why this pattern exists.

Problem Describes the problem that the pattern solves providing more detail as to when the pattern should be applied.

Solution A description of how to apply the pattern including the interactions between actors and any properties an actor is expected to have in order to function in the pattern.

We now present three patterns: SeparateStore, ContextPassing, and SharedStore for the deployment of provenance stores within applications.

4.2.1 SeparateStore Pattern

Diagram See Figure 4.1.

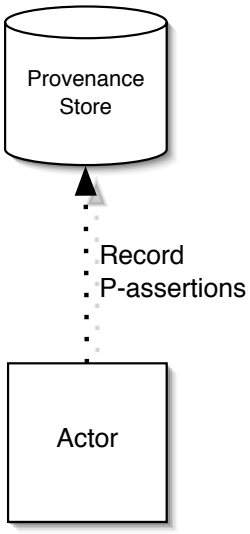


FIGURE 4.1: SeparateStore pattern diagram

Context Application actors want to make available information about their interactions and associated state. This pattern exists because querying actors want to know how application actors have interacted in the past in order to produce a result To know how application actors performed, these application actors must make information about their actions available.

Problem An application actor, A, may be involved in a large number of interactions over its lifetime and cannot retain all the process documentation itself. Likewise, querying actors would like to access information about A's previous message exchanges and states, even when A is not available. For example, A may have been shut down, moved or be under repair.

Solution A separately deployed provenance store is introduced to retain process documentation. An actor records p-assertions in a provenance store so that it does not have to retain this information itself. A provenance store should have the following properties:

1. It should be available in a long-term manner in comparison to the application actors that submit p-assertions to it. This property allows p-assertions recorded by an application actor to be accessed after the application actor has become unavailable.
2. It should provide a well-defined interface for the recording of p-assertions by an application actor.
3. It should provide a query capability to retrieve p-assertions, which makes the p-assertions available to querying actors.

4.2.2 ContextPassing Pattern

Diagram See Figure 4.2.

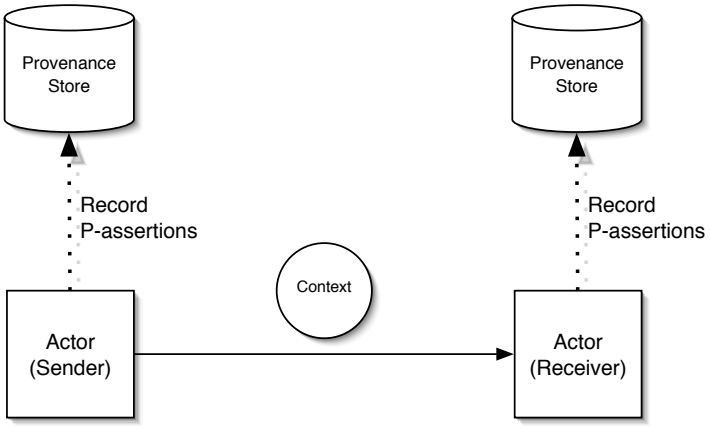


FIGURE 4.2: ContextPassing pattern diagram

Context Two application actors, A and B, exchange a message. A and B record p-assertions about this interaction in two provenance stores (see the pattern Separate-Store). Both actors record these interaction p-assertions because they want their view of that interaction to be documented. This allows other actors to determine if A's

and B's views of the interaction concur. Likewise, A and B may want to record internal information p-assertions and relationship p-assertions with respect to a particular interaction.

Problem The p-assertions that A and B record need to be identified as being the documentation for the same interaction. Otherwise, the actors' views of the interaction cannot be associated with one another; it then becomes difficult to determine if the recorded p-assertions are documenting the same interaction. Furthermore, it is not acceptable to exchange extra messages to establish a common identifier (i.e. interaction key) because the burden on actors should be limited to recording p-assertions.

Solution As discussed in the previous chapter, the sender in the interaction must obtain the appropriate interaction key to identify the interaction. It must then pass a *context* containing this interaction key to the receiving actor. Both actors use this interaction key to record their p-assertions in their respective provenance stores. The p-assertions for the interaction can then be matched using the interaction key. A method of passing this context is by attaching it to the application message exchanged by the sending and receiving actors. Beyond passing interaction keys, application actors may use a context to pass other information relevant to provenance including tracers.

Context information is metadata to the interaction (see page 49) and thus should be exposed to queriers in a more easily accessible fashion through the use of exposed interaction metadata (see page 56).

4.2.3 SharedStore Pattern

Diagram See Figure 4.3.

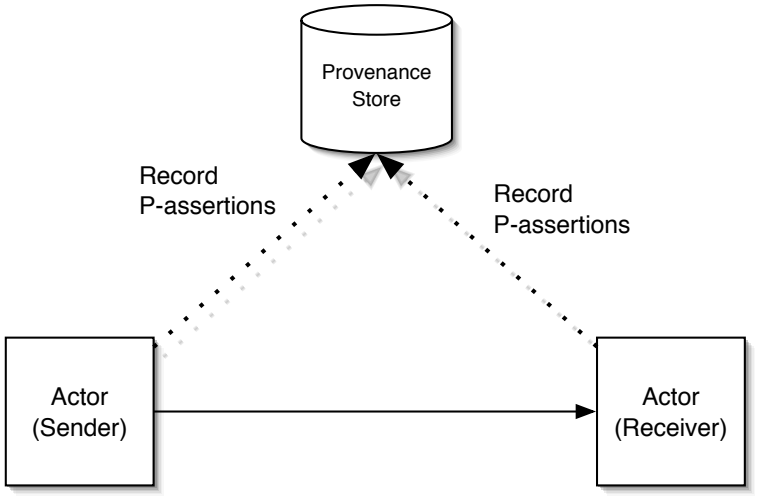


FIGURE 4.3: SharedStore pattern diagram

Context Actors record p-assertions in provenance stores following the SeparateStore and ContextPassing patterns.

Problem The SeparateStore and ContextPassing patterns may lead developers to believe that for every application actor, there is a corresponding provenance store. However, developers may not want to deploy a provenance store for every application actor, especially when the number of application actors is large. Also, in order to retrieve the provenance of a result, each provenance store must be contacted resulting in slower query performance.

Solution Application actors are allowed to record p-assertions into a shared provenance store. The SharedStore pattern clarifies the way in which SeparateStore and ContextPassing can be applied. Both SeparateStore and ContextPassing are agnostic as to what provenance store an actor may use to record its p-assertions. SharedStore emphasises that actors can record their p-assertions in any store they choose and provenance stores may hold p-assertions from multiple actors. It does not prescribe how many stores there should be and which provenance stores should be shared. It is left to the developer applying the pattern. SharedStore allows developers to determine the distribution of provenance stores that fits their application.

4.2.4 Pattern Application

The patterns that we have introduced show how p-assertions can be recorded in provenance stores by actors. The documentation of process can be recorded for an entire system by applying a selection of these patterns to every actor and every interaction in a system. We now show how these patterns can be applied using the simple example introduced on page 43 and shown again in Figure 4.4.

First, the SeparateStore pattern is applied so that each actor can record p-assertions into a provenance store. The application of this pattern is depicted in Figure 4.5.

Second, the SharedStore pattern is applied. Using this pattern, it is decided that the best deployment of provenance stores, in this case, is to have the Mathematical Function and Collate Sample actors share a common provenance store. Figure 4.6 shows the application of the SharedStore pattern to our simple example application.

Finally, to ensure that interaction keys are passed between actors and process documentation is connected, the ContextPassing pattern is applied as shown in Figure 4.7. Thus, all three patterns have been applied in order to appropriately record p-assertions.

These recording patterns allow for the flexible deployment of provenance stores. We also conjecture but do not show that these patterns aid scalability by allowing multiple provenance stores to be deployed for an application. The patterns can be applied to any number of interacting actors using any number of provenance stores to record

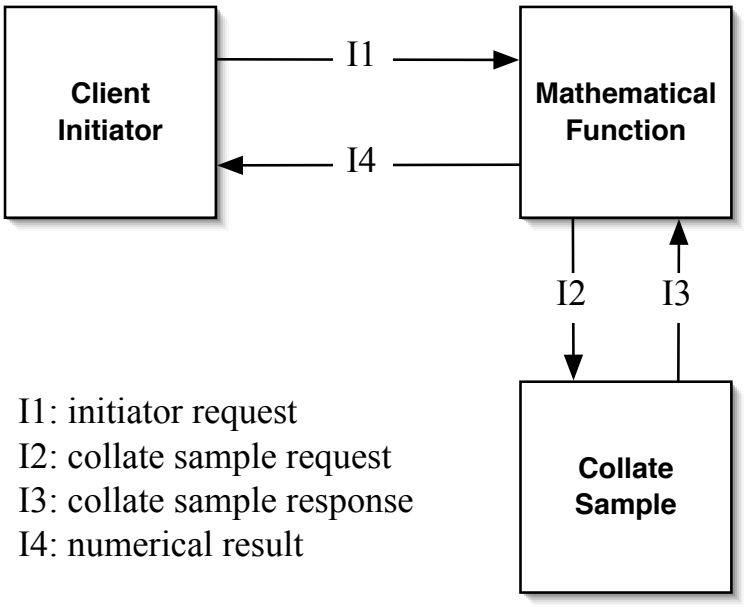


FIGURE 4.4: A simple example application

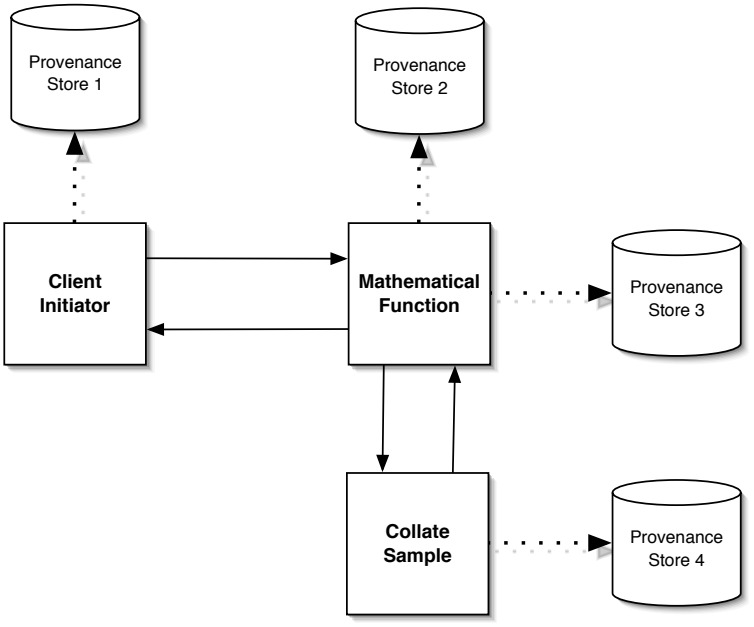


FIGURE 4.5: The SeparateStore pattern applied

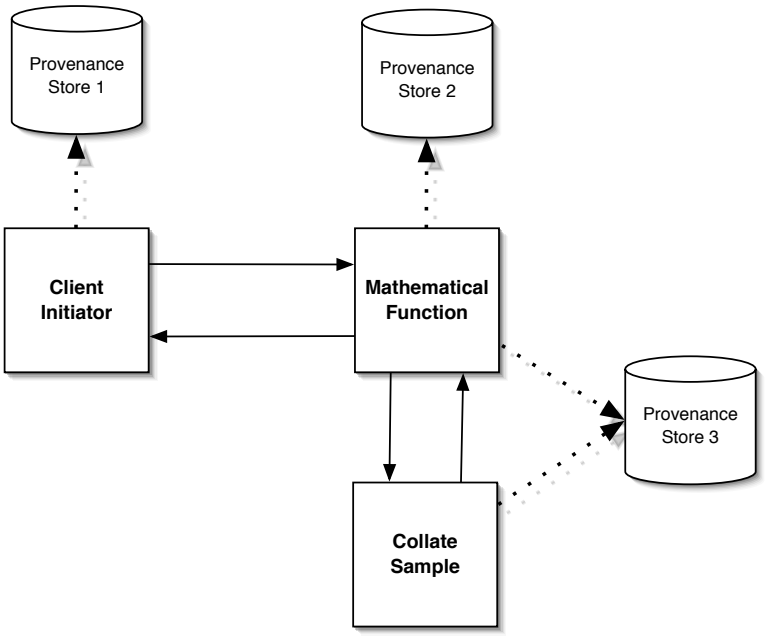


FIGURE 4.6: The SharedStore pattern applied

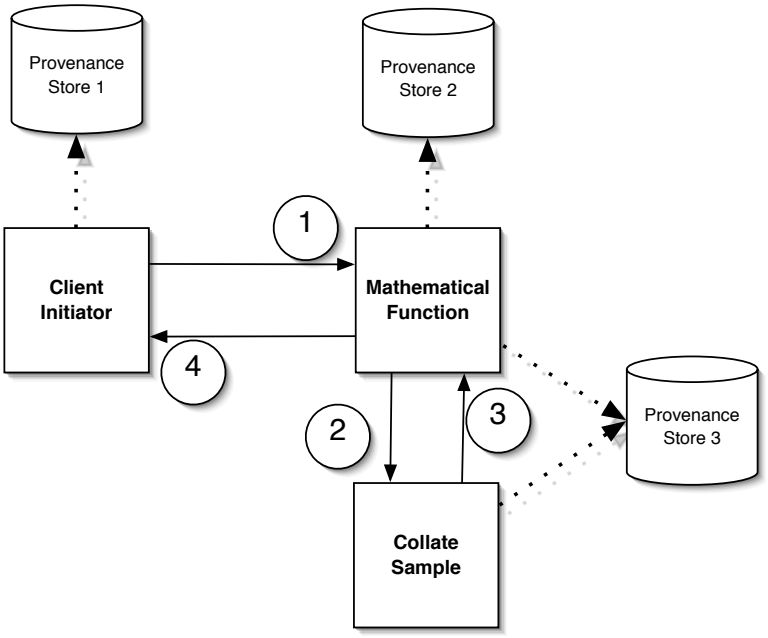


FIGURE 4.7: The ContextPassing pattern applied

p-assertions. These distribution patterns however do not mandate the number of provenance stores that must be used in a given application, nor the way they must be shared; it is left to the application developer to make those decisions.

4.3 Connecting Distributed Documentation

One of the consequences of applications that span multiple institutions and the use of the above deployment patterns is that process documentation need not be centrally located and can reside across various locations. There are several benefits to this: the elimination of a central point of failure, the spreading of demand across multiple services and the ability for provenance stores to exist in different network areas (for example, one provenance store may be behind a firewall whereas another is not). In general, allowing p-assertions to be recorded across multiple locations increases the flexibility and scalability of systems recording p-assertions.

However, to retrieve the provenance of a result, distributed process documentation must be connected so that the provenance of results can be found. The technique of *linking*, discussed below, enables this distribution.

Given that the p-assertions documenting a given execution may be spread across multiple stores, there must be some mechanism to retrieve these p-assertions in order to validate, visualise or replay the represented process. To facilitate such a retrieval mechanism, we introduce the notion of a link defined as follows.

Definition 4.1. A link is a pointer to a provenance store.

We note that links are necessarily *unidirectional*: a link always points to a remote provenance store location. Links are used in two instances, which we now describe.

4.3.1 View Links

The first use of a link deals with the situation where a sender's view of an interaction and a receiver's view of the same interaction as identified by a shared interaction key are stored in two different provenance stores. It is necessary for each actor to record a link, which we refer to as a *View Link*, that points to the provenance store where the opposite party recorded their p-assertions. Thus, the sender in an interaction records a link to the provenance store that the receiver used to record p-assertions for the given interaction, and vice-versa. This allows querying actors to navigate from one provenance store to the other in order to retrieve both views of an interaction. We note that View Links point to provenance stores only, not to particular pieces of data in a provenance store; the actual data of interest can be found by a local search of the provenance store.

If an actor, A, interacting with actor B has to assert, in provenance store P_A, that B is recording its view of the interaction in another provenance store, then actor A has to become aware that the store used by B is P_B. Either such knowledge is built into A, or it is communicated to A in the course of application execution. If it is built into A, then such knowledge is already part of A's scope, and can be asserted by A as an internal information p-assertion. Alternatively, if it is to be communicated to A, then such knowledge can be passed as part of a context, as shown by the ContextPassing pattern (for instance, when B returns a result to A). Hence, A can assert the link as part of an interaction p-assertion. When A and B do not have a request-response interaction, B can communicate to A the link to P_B by an out of band mechanism. One concern with View Links is the possible performance overhead of passing the provenance store location back to the sender. We have tried to minimize this impact by allowing the information to be sent in response messages. However, in a very dynamic system where an actor uses a different provenance store for every interaction there may be significant overhead. Developers should be aware of this limitation when using View Links.

View Links are stored in the provenance through p-assertions. If they are passed by in the context of a message, then they are stored in interaction p-assertions. If they are provided by some other mechanism, they can be stored as internal information p-assertions. To enable queriers to more readily discover and traverse provenance stores, View Links should be made accessible via exposed interaction metadata. Recall from Section 3.3.2.5, that exposed interaction metadata places metadata in a well defined location in the p-structure such that queriers know where to look for the information, without having to traverse application-specific p-assertion content.

4.3.2 Cause Links

Chapter 3 defined relationship p-assertions as internal causal connections between occurrences, where an occurrence is identified within process documentation by locating the p-assertion that represents it (See Section 3.3.2.2). Furthermore, the p-assertion containing the occurrence that is the effect of a relationship p-assertion must be located in the same View as the relationship p-assertion (See Section 3.3.2.5) itself. However, the p-assertions the represent the causes in the relationship p-assertion may be in various other Views and therefore located in different provenance stores.

To assist in finding these p-assertions, we introduce a second usage of links. For each relationship p-assertion it creates, an actor needs to identify which provenance stores the p-assertions representing causes are stored in; such a link is named a *Cause Link*. A Cause Link extends the relationship p-assertion data structure by adding the provenance store where each cause is stored. Like the View Link, a Cause Link only points to the provenance store and not to a particular piece of data in the store.

4.3.3 Linking Summary

Figure 4.8 shows an example of how both Cause and View Links are recorded. Actor A sends a message M2 to actor B as a consequence of message M1. The interaction exchanging message M2 is identified by interaction key 2. In the context (shown by the circle) of the message, A puts a link to the provenance store, P_A, that it uses for the interaction with B. Actor B then extracts the link from the context and records it as an interaction p-assertion in the provenance store P_B. As a result, a View Link from P_B to P_A is created (shown by the arc VL 1). We assume that A knows from its configuration that B always stores its p-assertions in P_B. Hence, A records a link to P_B as an internal information p-assertion in P_A, which creates a corresponding View Link shown by the arc VL 2. All View Links are made available by actors recording exposed interaction metadata. Finally, A makes a relationship p-assertion between its interaction with B and the previous interaction containing M1. As part of the relationship p-assertion, it adds a Cause Link to the provenance store, P_R, where the p-assertion related to the other interaction is stored. It then records the relationship p-assertion in P_A, thus connecting P_A to P_R shown by the arc CL. Figure 4.9 shows the contents of the provenance stores P_A and P_B after all p-assertions have been recorded.

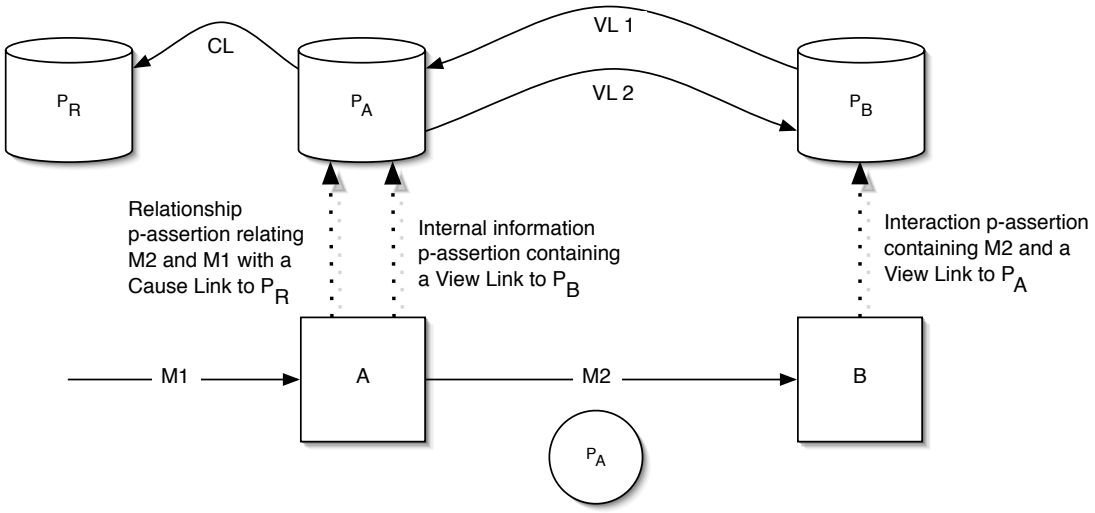


FIGURE 4.8: An example of linking

Both View Links and Cause Links allow data and p-assertions stored across provenance stores in multiple institutions to be retrieved by querying actors. View and Cause Links can be contrasted as follows.

- A View Link points to another store that contains a piece of data written by another actor (which is providing a different view on a same interaction).
- A Cause Link points to another store containing a piece of data asserted by the same actor (which is making assertions about another interaction).

Contents of P_A

interaction key	p-assertion type	p-assertion content
1	interaction	M1
2	interaction	M2
2	internal information	View Link to P_B
2	relationship	2 is related to 1, Cause Link to P_R

Contents of P_B

interaction key	p-assertion type	p-assertion content
2	interaction	M2, with View Link to P_A

FIGURE 4.9: Contents of provenance stores

Links provide a solution to the problem of connecting distributed process documentation. Similar to the Web, the unidirectional nature of links avoids the problem of having to synchronise between provenance stores when recording a link. Instead, each actor is responsible for recording a link just as each web page author is responsible for adding links to other pages as appropriate. Creating links is lightweight; the information needed to establish a link is minimal. Furthermore, the link structure provides a structured and simple mechanism for querying actors to traverse provenance stores hosted by multiple institutions.

So far, we have discussed the high level concept of recording process documentation into provenance stores and how this documentation can be connected so that it can be queried in a distributed environment. We now present a protocol by which actors can record their p-assertions into provenance stores.

4.4 PReP: The P-assertion Recording Protocol

The P-assertion Recording Protocol (PReP) defines the communication between actors and the expected behaviour of those actors when recording p-assertions. In this case, PReP has the following benefits:

1. It ensures that the data residing in the provenance store is compatible with the p-structure.
2. It provides a well-defined interface for actors to record p-assertions.
3. It guarantees certain beneficial properties in terms of both the data being recorded and its operation.

In the previous chapter, we enumerated the following characteristics that high quality process documentation should possess: factual, attributable, autonomously creatable, process oriented, immutable, finalizable. PReP supports these characteristics by enforcing several associated properties, which are discussed below.

4.4.1 Properties of PReP

Below are a six properties, which PReP enforces to help ensure that the process documentation within provenance stores is high quality. Essentially, each characteristic maps to a property, which can then be enforced.

1. As discussed in Chapter 3, a p-assertion's semantics dictates that it is always interpreted as being factual (i.e. about an occurrence within the scope of the actor creating the p-assertion). To ensure factual process documentation is contained within the provenance store, the *datatype safety* property is introduced, which guarantees that the protocol only allows p-assertions to be recorded. This property prevents untyped information from entering the provenance store, hence, factual process documentation can be more readily enforced.

2. Support for attribution is a fundamental part of the p-structure data model; an asserter identity is included in each p-assertion. It can also be used for access control on the provenance store. Thus, it is important to ensure that this identity is preserved during the recording of the p-assertion into the provenance store. To provide this assurance, PReP enforces the *identity preserving* property, which states that the asserter identity will not be changed during or after recording the p-assertion.

3. Within the p-structure, interaction keys are vital for supporting the autonomous creation of process documentation. To ensure that interaction keys are created and passed between actors correctly, Section 3.3.3 introduced three actor behaviour rules. Here, we show that the protocol has the property of being *actor behaviour compliant* i.e. that that the protocol enforces these three rules.

4. We introduce the property of *process reflection* that supports the characteristic of process orientation. It is defined as: eventually an application's execution will be described by process documentation recorded in provenance stores. This means that after the application has completely recorded the documentation of its execution in a set of provenance stores, a querier will be able to retrieve the provenance of any result produced by the distributed system.

5. In distributed systems, a safety property is one that states something will not happen [110]. In the context of PReP, we introduce the following *safety* property: no p-assertion is erased, overwritten or modified once recorded in a provenance store. This particular property is important because it supports the immutable characteristic, which as discussed earlier, is vital for queriers to have confidence in process documentation.

6. To support finalizable process documentation, we introduce the concept of *completeness*, which is that an object has all its constituents. In the case of PReP, we

define a View as the object that must have all its constituents. Thus, by inspecting a View, queriers can determine whether an actor has finished recording all the p-assertions for it.

Beyond these six properties, we introduce the scalability property of statelessness. A protocol is *stateless* when an actor can understand a message without relying on any previous or subsequent messages. This property is particularly useful in distributed systems where messages may be lost or received in any order.

We now present the protocol and corresponding definitions of actor behaviour that together satisfy these properties. We begin with a specification of the protocol.

4.4.2 Protocol Definition

PReP is an asynchronous protocol for an actor to record p-assertions into a provenance store. An asynchronous protocol allows actors to send their messages at any time, which means that actors can choose when to record p-assertions and thus not delay their execution. We define PReP in terms of both its messages as well as the expected behaviour of the actors exchanging the protocol messages.

With the previously listed properties in mind, we now proceed to define the protocol itself. As we defined earlier, an application can be described as actors communicating via message passing. In such a system, communication between actors gives a context to the individual execution of actors. Therefore, reflecting the p-structure, we base the protocol around the notion of an *interaction*. Actors record p-assertions in the context of a given interaction where they are either a sender or receiver. These roles were elaborated on earlier in Section 3.3.1.

With these general notions, we now present the protocol's messages and those assumed by it. After describing the messages, we then present the dependencies between them.

Let us consider the example of a distributed application that is not provenance-aware, i.e. one in which there are no provenance stores and p-assertions are not recorded. In such a case, application actors communicate via application messages that contain some application specific data. Figure 4.10 shows a *basic application message* that has one parameter that consists of an element from the set DATA. The parameter refers to the data typically transmitted by senders and receivers irrespective of p-assertion recording. Each message named in Figure 4.10 is given a notation that is used later in a formalisation. Likewise, the parameters of each message, which are defined by sets, are used in the same formalisation.

To transform such an application into a provenance-aware application, one or more provenance stores are introduced and actors record p-assertions into them. Additionally,

Name	Notation	Parameters
basic application message		DATA
application message	app	IK, DATA
record p-assertion	rec	IK, RI, A, LPID, P-ASSERTION
submission finished	sf	IK, RI, A, LPID, $\mathbb{N}^+$
acknowledgement	ack	IK, RI, LPID

FIGURE 4.10: The messages of PReP

basic application messages are extended with an identifier to be exchanged between actors, which is shown in Figure 4.10 as an *application message*. We label the identifier an interaction key.

As previously defined in Chapter 3, an interaction key identifies an interaction uniquely from all other interactions. The sender in an interaction is required to generate this key and send it to the interaction receiver. We have noted that it may not always be possible to extend application messages to add an interaction key. Instead, out of band mechanisms would be required to propagate similar information. During the following discussion, we assume that an interaction key can be added to basic application messages. The generation of interaction keys by senders supports a decentralised design where no centralised entity is necessary for senders or receivers to create or record p-assertions. The set of interaction keys is denoted by IK.

Application messages define the messages exchanged by application actors in a provenance-aware application. The rest of the messages defined in Figure 4.10 are exchanged between recorders and provenance stores.

The *record p-assertion* message is sent by a recording actor to a provenance store in order to record a p-assertion (the set of p-assertions is P-ASSERTION) about an interaction. The p-assertions we consider are those defined in Chapter 3, namely, interaction, relationship, and internal information p-assertions as well as exposed interaction metadata. The global p-assertion key is used by the recording protocol, hence we make its components explicit in the *record p-assertion* message. These include the interaction key the p-assertion is associated with, the role identifier (element of set RI), and the local p-assertion id (element of set LPID). The interaction key combined with the view kind and local p-assertion id ensures that every p-assertion can be referenced uniquely.

The *record p-assertion* message also includes an actor identity (element of set A) that is the asserter identity of the p-assertion (as defined in Section 3.3.2). Having a specific field for the asserter identity within the message simplifies the protocol definition as the structure of the p-assertions do not have to be modelled. Essentially, the messages define all the information necessary for the protocol to execute. We note that the protocol assumes that the creator of the p-assertion is also the recorder of the p-assertion. The asserter identity is essential for recording attributable process documentation. It connects the p-assertion in the message to the identity of the actor that creates, records,

and is responsible for it.

The *submission finished* message is similar to the *record p-assertion* message, except that the p-assertion parameter is replaced with an integer representing how many p-assertions a provenance store should receive in total from an actor for an interaction. By knowing how many p-assertions should be recorded, a provenance store can determine when an actor has finished recording p-assertions in the context of a particular View, which is used to determine when it is complete. Because of the asynchronous nature of the protocol, the *submission finished* message, like any other message, can be sent at any time. We note that in most cases an actor will send this message *after* it has recorded all its p-assertions. Thus, the *submission finished* does not imply that an actor guesses how many p-assertions it will create and record for a particular interaction. Instead, the ability to send the message at any time preserves the asynchronicity of the protocol. For example, consider an actor that has already created all its p-assertions, because of PReP's design, the actor can record all the p-assertions and declare that its submission is finished in parallel.

The last kind of message exchanged by recorders and provenance stores is the *acknowledgement* message. Each message received by a provenance store is acknowledged by an *acknowledgement* message, which contains the global p-assertion key contained in the message being acknowledged. There is some computation time in processing *record* messages and storing their contents. Therefore, *acknowledgement* messages allow actors to track whether their p-assertions have been stored within the provenance store. This is useful when the actor wishes to notify other actors that it has completed recording. Furthermore, the provenance store can use the acknowledgement message to return error messages or other implementation specific information. Thus, the acknowledgement message is not used for guaranteeing message delivery or flow control but, instead for the recorder to track the state of the provenance store.

We now describe the dependencies between the messages defined above. Due to the asynchronous nature of the protocol, the dependencies are minimal. They are as follows:

- For any *application message* in a given interaction, a *record p-assertion* or *submission finished* message about that interaction must contain the same interaction key as the *application message*.
- *Acknowledgement* messages must be sent after the receipt of the message that is being acknowledged.

4.4.3 PReP's Behavioural Constraints

The set of messages and their dependencies impose some behavioural constraints on the roles of sender, receiver, recorder, and provenance store. We now make such behaviour

explicit. Our intent here is to give an intuitive description of the required behaviour of these actors and then use the formalisation that follows to give a precise definition of that behaviour. We enumerate these **behaviour rules** below. The first three of these rules are the behaviour rules defined in Section 3.3.3.

1. (Unique Interaction Key Rule) A sender must generate a globally unique interaction key for every new interaction and assign it to that interaction.
2. (Interaction Key Transmission Rule) A sender must send the interaction key to the receiver by including it within the application message being sent.
3. (Appropriate Interaction Rule) Both receiver and senders must use the interaction key associated with an interaction, I, when asserting p-assertions about I.
4. A recorder must keep track of the messages it has sent to a provenance store for a particular interaction until the acknowledgements are received for them.
5. The provenance store must be waiting to receive messages and when it receives a message, it must process its content, store it and return the appropriate acknowledgement message. It is necessary that the provenance store determines how many p-assertions it has received from a particular actor for a given interaction and compare that to the number of p-assertions the recorder has declared so that it can determine if process documentation in a given View is complete. If documentation is marked as complete for an interaction, the provenance store must prevent any additional p-assertions from being recorded. Likewise, it has to detect attempts to overwrite previously stored p-assertions and respond with an acknowledgement message. If a p-assertion is received by a provenance store and its LPID has already been used, then the provenance store discards the p-assertion and the same acknowledgement message is returned. This means that once an actor has recorded a p-assertion, it cannot override that assertion.

We now present a formal model of PReP.

4.4.4 A Formal Model

To show that PReP satisfies the properties listed in Section 4.4.1, we now present a formalisation of PReP in terms of the behaviour of the actors involved in the protocol and the messages used. We have chosen to model PReP as an abstract state machine (ASM) because it provides a precise, implementation-independent means of describing the protocol. The ASM notation we adopt has been used previously to describe a distributed reference counting algorithm [133] and a fault-tolerant directory service for mobile agents [131]. The abstract machine characterizes the behaviour of actors with

respect to the messages they send and receive. This behaviour is specified by the permissible transitions that the ASM is allowed to perform. We begin by describing the state space of the ASM, we then proceed to discuss its transitions.

4.4.4.1 State Space

The state space of the ASM is shown in Figure 4.11. We model a distributed system as a set of actors, A, communicating via asynchronous message passing over a set of communication channels, $\mathcal{K}$. We identify specific subsets of actors in the system, namely, senders, receivers and provenance stores. An actor may be a member of all these subsets. These subsets map to roles defined previously.

Communication channels are assumed to be reliable and secure, and not to duplicate messages. No assumption is made about message order in the channel (i.e. sending message A before sending message B does not guarantee that A will arrive at its destination before B). Because of this assumption, channels are represented as bags of messages between pairs of actors. The messages listed in Figure 4.10 are sent over these communication channels and are formally defined as an inductive type producing set $\mathcal{M}$ in Figure 4.11.

Having defined the state space for communication between actors, we now model it for the internal functionality of each actor role.

Provenance Store State Space Informally, we can see a provenance store as an actor containing a table that maps interaction keys and a role identifier to a set of identified messages. This models the Views within a provenance store (V). An interaction key (κ) together with a role identifier (v) and set of identified messages is labelled a View. In Figure 4.11, the table ($store\_T$) is defined as a function that takes an interaction key and role identifier and returns a triple containing a submission finished message, several record messages and some local p-assertion ids. We use the power set notation ($\mathbb{P}$) to denote that there can be more than one of a given element. We define the set of provenance stores, PSS, as a mapping from an actor identity to a set of Views. Since each set of Views can be located at a different actor, our model allows for multiple provenance stores; practically, this allows process documentation to be located in multiple institutions.

Sender and Receiver State Space Now, we define the state space of sending and receiving actors in Figure 4.11. This state space describes the various tables that these actors use to keep track of the messages they need to send to the provenance store (TO_SEND), the messages they have sent to it (SENT) and the acknowledgements received from it (ACK). Furthermore, the state space describes the p-assertions that a

$$
\begin{array}{rcll}
\text{A} & = & \{a_1, a_2, \ldots, a_n\} & \text{(Set of Actor Identities)} \\
\textsc{Senders} & \subseteq & \text{A} & \text{(Set of Sender Identities)} \\
\textsc{Receivers} & \subseteq & \text{A} & \text{(Set of Receivers Identities)} \\
\text{PS} & \subseteq & \text{A} & \text{(Set of Provenance Store Identities)} \\
\text{REL} & = & \{r_1, r_2, \ldots, r_n\} & \text{(Set of Business Logic Descriptions)} \\
 & & & \\
\text{P-}\textsc{Assertion} & = & \{\alpha_1, \alpha_2, \ldots\} & \text{(Set of P-Assertions)} \\
 & & & \\
\mathcal{M} & = & \mathsf{app} : \text{IK} \times \textsc{Data} \to \mathcal{M} & \text{(Set of Messages)} \\
 & & \quad | \ \mathsf{rec} : \text{IK} \times \text{RI} \times \text{A} \times \text{LPID} \times \text{P-}\textsc{Assertion} \to \mathcal{M} & \\
 & & \quad | \ \mathsf{sf} : \text{IK} \times \text{RI} \times \text{A} \times \text{LPID} \times \mathbb{N}^+ \to \mathcal{M} & \\
 & & \quad | \ \mathsf{ack} : \text{IK} \times \text{RI} \times \text{LPID} \to \mathcal{M} & \\
 & & & \\
\text{SF} & = & \{m \in \mathcal{M} \mid m = \mathsf{sf}(\kappa, v, \iota, na)\} & \text{(Set of Submission Finished Messages)} \\
\text{R} & = & \{m \in \mathcal{M} \mid m = \mathsf{rec}(\kappa, v, \iota, lpid, \alpha)\} & \text{(Set of Record Messages)} \\
 & & & \\
\text{IK} & = & \textsc{Senders} \times \textsc{Receivers} \times \mathbb{N} & \text{(Set of Interaction Keys)} \\
\text{RI} & = & \{\mathsf{S}, \mathsf{R}\} & \text{(Set of Role Identifiers)} \\
 & & & \\
\text{V} & = & \text{IK} \times \text{RI} \to \text{SF}_\perp \times \mathbb{P}(\text{R}) \times \mathbb{P}(\text{LPID}) & \text{(Set of Views)} \\
\text{PSS} & = & \text{A} \to \text{V} & \text{(Set of Provenance Stores)} \\
 & & & \\
\text{TO\_SEND} & = & \text{A} \to \text{IK} \to \mathit{Bag}(\mathcal{M}) & \text{(Set of Messages To Send Tables)} \\
\text{SENT} & = & \text{A} \to \text{IK} \to \mathit{Bag}(\mathcal{M}) & \text{(Set of Sent Messages Tables)} \\
\text{ACK} & = & \text{A} \to \text{IK} \to \mathit{Bag}(\mathcal{M}) & \text{(Set of Acknowledged Messages Tables)} \\
\text{ASSERT} & = & \text{A} \to \text{IK} \times \text{RI} \to \mathit{Bag}(\text{P-}\textsc{Assertions}) & \text{(Set of p-assertions to be recorded)} \\
\text{LPID\_MAP} & = & A \to \text{IK} \times \text{RI} \to \mathbb{P}(\text{LPID}) & \text{(Map from actor to local p-assertion ids)} \\
\text{LC} & = & \textsc{Senders} \to \mathbb{N} & \text{(Set of Local Counters)} \\
 & & & \\
\mathcal{K} & = & \text{A} \times \text{A} \to \mathit{Bag}(\mathcal{M}) & \text{(Set of Channels)} \\
 & & & \\
\text{C} & = & \text{PSS} \times \mathcal{K} \times \text{TO\_SEND} \times \text{SENT} \times & \\
 & & \text{ACK} \times \text{ASSERT} \times \text{LPID\_MAP} \times \text{LC} & \text{(Set of Configurations)}
\end{array}
$$

Characteristic Variables:

$$
\begin{array}{rcl}
a & \in & A \\
a_s & \in & \textsc{Sender} \\
a_r & \in & \textsc{Receiver} \\
a_{ps} & \in & \text{PS} \\
r & \in & \text{REL} \\
m & \in & \mathcal{M} \\
d & \in & \textsc{Data} \\
\alpha & \in & \text{P-}\textsc{Assertion} \\
\kappa & \in & \text{IK} \\
v & \in & \text{RI} \\
na & \in & \mathbb{N}^+ \\
lpid & \in & \text{LPID}
\end{array}
\qquad
\begin{array}{rcl}
k & \in & \mathcal{K} \\
lpids & \in & \mathbb{P}(\text{LPID}) \\
recs & \in & \mathbb{P}(R) \\
store\_T & \in & \text{PSS} \\
to\_send\_T & \in & \text{TO\_SEND} \\
sent\_T & \in & \text{SENT} \\
ack\_T & \in & \text{ACK} \\
assert\_T & \in & \text{ASSERT} \\
lpid\_T & \in & \text{LPID\_MAP} \\
lc & \in & \text{LC} \\
c & \in & \text{C}
\end{array}
$$

Initial State / Configuration:

$c_i = \langle store\_T_i, k_i, to\_send\_T_i, sent\_T_i, ack\_T_i, assert\_T_i, lpid\_T_i, lc_i \rangle$

where:

$$
\begin{array}{rclcrcl}
store\_T_i & = & \lambda a \lambda \kappa v \cdot \langle \perp, \emptyset, \emptyset \rangle, & & k_i & = & \lambda a_i a_j \cdot \emptyset, \\
to\_send\_T_i & = & \lambda a_i \kappa_i \cdot \emptyset, & & sent\_T_i & = & \lambda a_i i k_i \cdot \emptyset, \\
ack\_T_i & = & \lambda a_i i k_i \cdot \emptyset, & & assert\_T_i & = & \lambda a_i \kappa_i v_i \cdot \emptyset, \\
lpid\_T_i & = & \lambda a_i \kappa_i v_i \cdot \emptyset, & & lc_i & = & \lambda a_i \cdot 0
\end{array}
$$

FIGURE 4.11: State Space

sender or receiver need to record in a provenance store (ASSERT) and how an actor keeps track of the local p-assertion ids it has already used (LPID_MAP). Finally, each sending actor has a local counter (LC) used to create interaction keys.

The state space that we have described may appear to be global in Figure 4.11. However, each table for a sender or receiver is indexed by an actor identity (A) and can be implemented with updates that are local to actors. Hence, the protocol does not require any global knowledge by actors of other actors' state.

For convenience, we define two accessor functions. The accessor function to access the state of the View is defined as follows:

If $store\_T(a)(\kappa, v) = \langle sf, recs, lpids \rangle$ then
$store\_T(a)(\kappa, v).sf = sf$,
$store\_T(a)(\kappa, v).recs = recs$,
$store\_T(a)(\kappa, v).lpids = lpids$

The function takes an actor identity, interaction key, and role identifier as input and returns a View. From the View, its contents such as record messages, local p-assertion ids and submission finished message can be retrieved. The inputs of the function act as a key to an index of Views within provenance stores.

We also define a function for accessing the state of a submission finished message. The function is defined as follows:

If $sf = \mathsf{sf}(\kappa, v, a, \ell, na)$ then
$sf.\kappa = \kappa$,
$sf.v = v$,
$sf.a = a$,
$sf.\ell = \ell$,
$sf.na = na$

This function just provides an easier notation for addressing the various contents of the submission finished message. This is helpful in the rules where the completeness of a View is checked.

Having described the state space of our ASM, a state (or configuration) of the machine is described in Figure 4.11. The machine's initial state can be summarised as

- empty interaction record stores,
- empty communication channels,
- all sending and receiving actors having empty p-assertion and message tables,
- and local counters being initialized to zero.

The machine proceeds from this initial state through its execution by going through *transitions* that lead to new states. These transitions are defined by the rules of the state machine discussed in the next section.

When describing the execution of a state machine, we use the following notation and definitions.

- A *transition* is the application of a rule to one configuration to achieve another configuration.
- A *reachable configuration* is a configuration of the ASM that can be reached by transitions from the initial configuration.
- $\longmapsto$ denotes a transition.
- $c \longmapsto^* c'$ denotes any number of transitions from a configuration c to another configuration c'.

We now discuss the specific rules of the ASM.

4.4.4.2 State Machine Rules

The permissible transitions in the ASM are described through rules, which are represented using the following notation.

$$
\begin{aligned}
&rule\_name(v_1, v_2, \cdots) : \\
&\quad condition_1(v_1, v_2, \cdots) \\
&\quad \wedge\ condition_2(v_1, v_2, \cdots) \wedge \cdots \\
\rightarrow\ &\{ \\
&\quad pseudo\_statement_1; \\
&\quad \cdots \\
&\quad pseudo\_statement_n; \\
&\}
\end{aligned}
$$

Rules are identified by their name and a number of parameters that the rule operates over. Any number of conditions must be met for a rule to fire. Once a rule's conditions are met, the rule can fire. The execution of a rule is a transition of the state machine and is atomic in order to maintain its consistency. A new state is achieved after applying all the rule's pseudo-statements to the state that met the rule's conditions.

We use *send*, *receive* and table update pseudo-statements. Informally, $send(a_1, a_2, m)$ inserts a message m into the channel from actor a_1 to actor a_2, and $receive(a_1, a_2, m)$ removes the message. Likewise, the table update operation puts a message into a table. The notation *table_T* is used to refer to any table in the state space. Formally, these pseudo-statements act as state transformers and are defined as follows.

- We use the operators $\oplus$ and $\ominus$ to denote union and difference on bags. If k is the set of message channels of a state $\langle \ldots, k, \ldots \rangle$, then the expression $send(a_1, a_2, m)$ and $receive(a_1, a_2, m)$ respectively denote the state $\langle \ldots, k', \ldots \rangle$, where $k'(a_1, a_2) = k(a_1, a_2) \oplus \{m\}$ and $k(a_1, a_2) \ominus \{m\}$, and $k'(a_i, a_j) = k(a_i, a_j)$, $\forall (a_i, a_j) \neq (a_1, a_2)$.
- If $table\_T$ is a component of state $\langle \ldots, table\_T, \ldots \rangle$, then the expression $table\_T(\ldots).y := V$ denotes the state $\langle \ldots, table\_T', \ldots \rangle$, where $table\_T(\ldots).x = table\_T'(\ldots).x$ if $x \neq y$, and $table\_T'(\ldots).y = V$.

To ease the readability of the rules, we also define the following pseudo functions.

The function below determines if a View is complete:

Definition
$complete$: A × IK × RI → $\{true, false\}$
$complete(a, \kappa, v)$:=
If $store\_T(a)(\kappa, v).sf \neq \perp$,
then return $store\_T(a)(\kappa, v).sf.na = |store\_T(a)(\kappa, v).recs|$
else return *false*.

The complete function takes as input an actor identity, an interaction key and a role identifier, which together identify a particular View within a provenance store. Once the View is found, the function tests to see if it contains a submission finished message. If the View does, the function tests whether the number of p-assertions within the View is the same as the number of expected p-assertions as specified in the submission finished message. If the numbers are equal, the function returns true. In all other cases, the function returns false.

The following pseudo function creates new interactions keys, updating an actor's local counter table.

Definition
$newIdentifier$: Senders × Receivers → IK
$newIdentifier(a_s, a_r)$:=
$lc(a_s) := lc(a_s) + 1$
return $\langle a_s, a_r, lc(a_s) \rangle$.

The function ensures that interaction keys are uniquely created. It takes two actor identities as inputs: one for the sender and one for the receiver. It then obtains the local counter of the sender and increments it by one. It then constructs a new interaction key using the two actor identities and the local counter. This interaction key is then returned by the function. This models the notion that senders are responsible for creating interaction keys.

$receive\_record\_passertion(a, a_{ps}, \kappa, v, \ell, \alpha)$:
$\mathsf{rec}(\kappa, v, a, \ell, \alpha) \in \mathcal{K}(a, a_{ps})$
$\rightarrow$ {

$receive(\mathsf{rec}(\kappa, v, a, \ell, \alpha), a, a_{ps})$;	-6
if $(\ell \notin store\_T(a_{ps})(\kappa, v).lpids \wedge \neg complete(a_{ps}, \kappa, v)), then$	
$\quad store\_T(a_{ps})(\kappa, v).lpids := store\_T(a_{ps})(\kappa, v).lpids \cup \{\ell\}$;	+1 or +0
$\quad store\_T(a_{ps})(\kappa, v).recs := store\_T(a_{ps})(\kappa, v).recs \cup \{\mathsf{rec}(\kappa, v, a, \ell, \alpha)\}$;	+1 or +0
$send(\mathsf{ack}(\kappa, v, \ell), a_{ps}, a)$;	+2
}	overall: -2 or -4

$receive\_submission\_finished(a, a_{ps}, \kappa, v, \ell, na)$:
$\mathsf{sf}(\kappa, v, a, \ell, na) \in \mathcal{K}(a, a_{ps})$
$\rightarrow$ {

$receive(\mathsf{sf}(\kappa, v, a, \ell, na), a, a_{ps})$;	-5
$if(\ell \notin store\_T(a_{ps})(\kappa, v).lpids \wedge store\_T(a_{ps})(\kappa, v).sf = \perp), then$	
$\quad store\_T(a_{ps})(\kappa, v).lpids := store\_T(a_{ps})(\kappa, v).lpids \cup \{\ell\}$;	+1 or +0
$\quad store\_T(a_{ps})(\kappa, v).sf := \mathsf{sf}(\kappa, v, a, \ell, na)$;	+1 or +0
$send(\mathsf{ack}(\kappa, v, \ell), a_{ps}, a)$;	+2
}	overall: -1 or -3

FIGURE 4.12: Provenance Store rules

Figures 4.12 and 4.13 show the ASM's transition rules, which formally define the behaviour described in Section 4.4.3. The annotations to the right of each rule are used in our termination proof and will be discussed later. We now discuss the two rules that govern the provenance store's behaviour shown in Figure 4.12.

Provenance Store Rules The *receive_record_passertion* rule takes an incoming record message from a particular actor and places it in the correct View in the provenance store as defined by *store_T*, and thereby also stores the p-assertion enclosed in the message. The View is looked up via the interaction key and role identifier located in the rec message. An acknowledgement message (ack) is then sent to the actor who sent the rec message. During its execution, the rule checks to see whether or not the local p-assertion id (ℓ) of the p-assertion contained within the message has already been used in the interaction record. If ℓ has not been used, the message is stored, otherwise it is not. Likewise, if the View is complete (i.e. it already contains the requisite number of p-assertions) the message is not stored. In all cases, the same acknowledgement is sent. This rule satisfies part of Behaviour Rule 5 (See Section 4.4.3.

The *receive_submission_finished* rule operates in a similar manner and satisfies the rest of behaviour rule 5. However, only one submission finished message (sf) is allowed to be recorded. If subsequent sf messages are received by the provenance store, the message is discarded and an ack message is sent to the actor.

We note that all local p-assertion ids made available in the provenance store correspond to a message stored in the provenance store. This correspondence is made in order to

simplify the rules. To show this, we establish the following lemma.

Lemma 4.2 (Local P-Assertion Id Correspondence). *For any configuration c, for all ℓ, κ, v, a, α, na, $\ell \in store\_T(\kappa, v).lpids$ if and only if there exists a $\mathsf{rec}(\kappa, v, a, \ell, \alpha) \in store\_T(\kappa, v).recs$ or a $\mathsf{sf}(\kappa, v, a, \ell, na) = store\_T(\kappa, v).sf$.*

Proof. We prove this lemma by induction on the length of the transition sequence from the intitial configuration c_i to a configuration c. In the base case, where the length is 0, all sets are empty and the lemma trivially holds.

In the inductive case, for transition lengths greater than zero, we assume that the lemma holds for $c_i \longmapsto^* c_n$ and we consider all possible transition sequences from $c_n \longmapsto c$. The only two rules that modify the provenance store are *receive_record_passertion* and *receive_submission_finished*.

In the case of the *receive_record_passertion*, if a local p-assertion id is added to the provenance store then a corresponding record message is also added. Likewise, if a record message is added then a corresponding local p-assertion id is added. This is shown in the third and fourth pseudo-statements of the rule. There is only one local p-assertion used in the rule as shown by the rules guard, thus, the local p-assertion id that is stored to the set $store\_T(\kappa, v).lpids$ is the same as the one inside the stored record message.

The same reasoning also holds true of *receive_submission_finished*, however, with submission finished messages not record messages.

In both rules nothing is deleted, therefore, the correspondence is always maintained. □

Sender and Receiver rules The rules shown in Figure 4.12 define the behaviour of a provenance store in the ASM. However, in order to prove the properties of safety, liveness and attributability, we need to prescribe some behaviour of the actors that use provenance stores. Our aim is to show that if an actor behaves in the manner defined then certain guarantees can be given. This behaviour is formalised in Figure 4.13.

The functionality of these rules can be summarised as follows. When involved in an interaction (i.e. either sending or receiving a message), an actor makes p-assertions related to that interaction. This behaviour is shown in the *send_app_msg* and *receive_app_msg* rules. We highlight the *makeAssertions* function which takes an actor identity (a), an interaction key (κ), some data (d), and a business logic description (r) and produces a set of p-assertions, $\{\alpha_1, \ldots, \alpha_n\}$. The business logic description is equivalent to the type of relationship p-assertion that should be produced by the *makeAssertions* function. An instantiation of *makeAssertions* is crucial in the proof of liveness.

Once the set has been created, a rec message for each p-assertion is then constructed and added to the table of messages to be sent to the provenance store (to_send_T).

$send\_app\_msg(a_s, a_r, d, r)$:
// triggered when d, produced by a function described by r, is to be sent by a_s to a_r
$\rightarrow$ {
  let $\kappa = newIdentifier(a_s, a_r)$
  in $\{\alpha_1, \ldots, \alpha_n\} = makeAssertions(a_s, \kappa, d, r)$;
  $send(\mathsf{app}(\kappa, d), a_s, a_r)$;
  $assert\_T(a_s, \kappa, \mathsf{S}) := assert\_T(a_s, \kappa, \mathsf{S}) \cup \{\alpha_1, \ldots, \alpha_n\}$;
} annotations not applicable

$receive\_app\_msg(a_s, a_r, \kappa, d)$:
  $\mathsf{app}(\kappa, d) \in \mathcal{K}(a_s, a_r)$
$\rightarrow$ {
  $receive(\mathsf{app}(\kappa, d), a_s, a_r)$;
  // receive business logic
  $\{\alpha_1, \ldots, \alpha_n\} = makeAssertions(a_r, \kappa, d, \bot)$;
  $assert\_T(a_r, \kappa, \mathsf{R}) := assert\_T(a_r, \kappa, \mathsf{R}) \cup \{\alpha_1, \ldots, \alpha_n\}$;
} annotations not applicable

$make\_passertion\_msg(a, \alpha, v, \kappa)$:
  $\alpha \in assert\_T(a, \kappa, v)$
$\rightarrow$ {
  let $\ell \notin lpid\_T(a, \kappa, v)$
  in $to\_send\_T(a, \kappa) := to\_send\_T(a, \kappa) \cup \{\mathsf{rec}(\kappa, v, a, \ell, \alpha)\}$; +9
  $assert\_T(a, \kappa, v) := assert\_T(a, \kappa, v) \ominus \alpha$; -30
  $lpid\_T(a, \kappa, v) := lpid\_T(a, \kappa, v) \oplus \ell$; + 1
  let $\ell_2 \notin lpid\_T(a, \kappa, v) \cup \{\ell\}$
  in *if* $(assert\_T(a, \kappa, v) = \emptyset)$, *then*
  $to\_send\_T(a, \kappa) := to\_send\_T(a, \kappa) \cup \{\mathsf{sf}(\kappa, v, a, \ell_2, |lpid\_T(a, \kappa, v)|)\}$; +0 or +9
  $lpid\_T(a, \kappa, v) := lpid\_T(a, \kappa, v) \oplus \ell_2$; +0 or +1
} overall: -10 or -20

$record\_message(a, a_{ps}, m)$:
  $m \in to\_send\_T(a, \kappa)$
$\rightarrow$ {
  $to\_send\_T(a, \kappa) := to\_send\_T(a, \kappa) \ominus m$; -9
  $sent\_T(a, \kappa) := sent\_T(a, \kappa) \oplus m$; +1
  $send(m, a, a_{ps})$; +5 or +6
} overall: -2 or -3

$receive\_ack(a, a_{ps}, \kappa, v, \ell)$:
  $\mathsf{ack}(\kappa, v, \ell) \in \mathcal{K}(a_{ps}, a)$
$\rightarrow$ {
  $receive(\mathsf{ack}(\kappa, v, \ell), a_{ps}, a)$; -2
  $ack\_T(a, \kappa) := ack\_T(a, \kappa) \oplus \mathsf{ack}(\kappa, v, \ell)$; +1
} overall: -1

FIGURE 4.13: The rules of the ASM used by sending and receiving actors

Furthermore, a sf message is constructed when all the rec messages have been added to *to_send_T*. The sf message could be constructed at any time, but we have chosen to do so at this stage to make the rules simpler. Creating sf messages in this manner is an eager strategy, they could also be generated using a lazy strategy. These steps are formalised in *make_passertion_msg*, which also obtains the asserter identity and keeps track of the local p-assertion ids that the actor has used.

Once a message has been sent, it is added to a table of messages that have been sent to the provenance store, *sent_T*, and removed from *to_send_T*. Finally, when an acknowledgement has been received, it is stored in the table *ack_T*. This models how an actor keeps track of its communication with a provenance store for a particular interaction identified by κ. The rules that correspond to this tracking functionality are *record_msg* and *receive_ack* and satisfy Behaviour Rule 4. Behaviour Rules 1 through 3 are discussed during the analysis of the protocol, which we now introduce.

4.4.5 Protocol Analysis

Based on the ASM above, we now analyse PReP according to the identified properties in Section 4.4.1. The analysis shows how PReP's properties support the recording of process documentation with high-quality characteristics. These protocol properties are one element that ensures high-quality characteristics are achieved in recorded process documentation. We first analyse the communication protocol property of statelessness; we then analyse the properties that support the characteristics of process documentation.

We establish these properties as *invariants* i.e. as the state machine proceeds these properties always stay the same. We rely on case analysis either on its own or in the context of a proof by induction to establish these invariants. Essentially, the rules of the ASM are analysed to show that after any number of transitions the particular property still holds for the resulting configuration of the state machine.

Induction is performed upon the length of the transition sequence from one configuration to another. In the base case, we establish that the invariant holds in some initial configuration. In the inductive case, the invariant is assumed to hold after a series of transitions from the initial configuration to another configuration. We then prove, using case analysis, that the property holds after an additional transition.

In places, we establish characteristics as theorems whereas in others we provide conclusions, which show how the various lemmas and invariants help support a specific characteristic.

We now begin the analysis.

4.4.5.1 Statelessness

A protocol is stateless when an actor can understand a message without relying on other messages. Stateless protocols tend to be simpler to implement and have greater scalability [158] because actors do not need logic to maintain knowledge from other messages in order to understand any other messages; the actor can accept a message, process it, and then discard its content. Alternatively, if an actor needs to maintain state in order to implement a protocol, that state can become corrupted or out-of-sync, which could cause messages produced by the actor to be incorrect. For these reasons, we have developed PReP as a stateless protocol.

A stateful alternative to PReP would be if each message did not have an interaction key. Instead, a *start submission* message could be introduced that contained the interaction key and all messages after that message would be assumed to have that interaction key until the submission finished message is received. Thus, the provenance store would need to keep state from the start submission message in order to process the other messages, which could lead to a situation where messages could not be processed if the state was somehow lost or the start submission message was never received. It would also make the parallel submission of p-assertions about different interactions difficult because additional mechanisms to distinguish between p-assertions about different interactions would be needed.

To show that PReP is indeed a stateless protocol, we define understanding a message in terms of the ability of the state machine to process the message:

Definition 4.3 (Message Processing). Processing a message is the storing of its content in the correct location independently of any other message. Practically, this is the storage of a message contents in the provenance store or the storage of an acknowledgements contents by a recording actor.

Because we are interested in the understanding of a particular message by a receiver, we only consider transitions that deal with the receipt of messages. We term such transitions message m consuming transitions, which we define as follows:

Definition 4.4 (Message m Consuming Transition). A message m consuming transition is defined as a transition that receives a message m and processes it. Concretely, there are 3 such transitions.

1. $receive\_ack(a, a_{ps}, \kappa, v, \ell)$ is an $\mathsf{ack}(\kappa, v, \ell)$-consuming transition.
2. $receive\_record\_passertion(a, a_{ps}, \kappa, v, \ell, \alpha)$ is a $\mathsf{rec}(\kappa, v, a, \ell, \alpha)$-consuming transition.
3. $receive\_submission\_finished(a, a_{ps}, \kappa, v, \ell, na)$ is a $\mathsf{sf}(\kappa, v, a, \ell, na)$-consuming transition.

We see that some of these messages are received by provenance stores and others are received by recording actors. Using this definition, we establish the following invariant.

Lemma 4.5 (Message Consumption). *For any configurations c, c', for any message m and for any message m-consuming transition, t, such that t leads from c to c' the following holds: If $m \in k(a_1, a_2)$ in c then $m \notin k(a_1, a_2)$ in c' and the arguments of m and the identities of the actors communicating that message are necessary and sufficient to cause t to process the message.*

Proof. We proceed by an analysis of the message based transitions that lead from c to c'. Each rule only uses the information provided in a message and the communication channel to fire as demonstrated by the rule's conditions. Furthermore, the information is necessary and sufficient to process the data (as defined in Definition 4.3). Analysing each transition individually:

- For $receive\_ack(a, a_{ps}, \kappa, v, \ell)$, the message $\mathsf{ack}(\kappa, v, \ell)$ contains three of the parameters and the remaining parameters are used to define the channel $\mathcal{K}(a_{ps}, a)$. The parameters a, κ are necessary and sufficient to reference the acknowledgement table, $ack\_T(a, \kappa)$, and thus store the message.
- For $receive\_record\_passertion(a, a_{ps}, \kappa, v, \ell, \alpha)$, the message $\mathsf{rec}(\kappa, v, a, \ell, \alpha)$ contains five of the parameters and the remaining parameters are used to define the channel $\mathcal{K}(a, a_{ps})$. The parameters a_{ps}, κ, v are necessary and sufficient to reference a View, $store\_T(a_{ps})(\kappa, v)$, and thus store the message, if appropriate.
- For $receive\_submission\_finished(a, a_{ps}, \kappa, v, \ell, na)$, the message $\mathsf{sf}(\kappa, v, a, \ell, na)$ contains five of the parameters and the remaining parameters are used to define the channel $\mathcal{K}(a, a_{ps})$. The parameters a_{ps}, κ, v are necessary and sufficient to reference a View, $store\_T(a_{ps})(\kappa, v)$, and thus store the message, if appropriate.

While the rules provide for consistency checks, all the information provided in the message is necessary and sufficient to properly store the messages in the correct location. Therefore, we have shown that, if a message is on a channel, it can cause the firing of a message based transition and the execution of the transition will store the message, if appropriate. To finish the proof, we note that when each rule fires, the *receive* pseudo-statement is executed removing the message from the communication channel and thus removing it from the channel in the next configuration, c'. Consequently, the invariant holds. □

Conclusion 4.6 (Statelessness). *All messages in PReP can be understood without relying on any other messages.*

From Definition 4.3, we have a precise definition of understanding a message. Invariant 4.5 shows that the parameters of messages along with the communication channel definition are necessary and sufficient for the ASM to proceed for all message m-consuming transitions. Thus, messages in PReP can be understood without relying on any other messages.

We now proceed to analyse, how PReP supports high-quality characteristics of process documentation. The first is the factual characteristic.

4.4.5.2 Factual

The key to factuality is to preserve the data types used within and for messages. This is particularly important with respect to p-assertions, which have a particular meaning based on their type. To show that the ASM does preserve this characteristic, we establish the following theorem based on the property of datatype safety.

Lemma 4.7 (Data Type Safety). *PReP preserves data types.*

Proof. Figure 4.11 defines types for all sets, messages, and tables. Thus, the state space of PReP is typed. Furthermore, the transition rules of the ASM as defined in Figures 4.12 and 4.13 preserve the typing structure defined in the state space. Therefore, PReP preserves typing information. □

Conclusion 4.8 (Factual). *Data type safety helps ensure factuality.*

Because PReP preserves typing information, queriers can assume that the process documentation within a provenance store has the semantics of the p-structure as defined in Section 3.3.2. Essentially, queriers are guaranteed to retrieve p-assertions that are typed according to the p-structure. Thus, both queriers and recorders share a common semantics of the data within a provenance store and therefore do not have to be aware of each other in order to function.

4.4.5.3 Autonomously Creatable

In dynamic multi-institutional systems, process documentation must be able to be created by actors in a flexible and autonomous fashion. To support this, we defined three behaviour rules pertaining for actors creating and using interaction keys. Interaction keys connect the two views of an interaction as well as index the p-structure. To demonstrate support for autonomous creation, we define the property of actor behaviour compliant.

Definition 4.9 (Actor Behaviour Compliant). Two actors passing a single message are actor behaviour compliant when they follow the Unique Interaction Key Rule, the Interaction Key Transmission Rule, and the Appropriate Interaction Rule.

Because the correct creation and use of interaction keys is of considerable importance to allowing process documentation to be autonomously creatable, we establish the following theorem in terms of Definition 4.9.

Theorem 4.10 (Follows Behaviour Rules). *Actors following PReP are actor behaviour compliant.*

Proof. To establish this theorem, we perform a case analysis. Rules *send_app_msg* and *receive_app_msg* govern the interactions between senders and receivers. *send_app_msg* sends the application message to a receiver. In the first pseudo-statement, the rule creates a new interaction key using the $newIdentifier(a_s, a_r)$ pseudo-function, which guarantees the creation of a unique interaction key (under the assumption that actor identities are unique). Furthermore, the third and fourth pseudo-statements of the rule assign the interaction key to the application message and send the application message to the receiver. Therefore, the ASM, through the *send_app_msg*, rule complies to the Unique Interaction Key Rule and the Interaction Key Transmission Rule.

In the second pseudo-statement of the *send_app_msg* rule, the $makeAssertions(a_s, \kappa, d, r)$ function uses the generated interaction key to create p-assertions about the interaction. Likewise, in the second pseudo-statement of the *receive_app_msg* rule, the $makeAssertions(a_r, \kappa, d, \perp)$ function uses the interaction key in the application message, $\mathsf{app}(\kappa, d)$, to create p-assertions about the interaction. Therefore, the ASM, through these rules, complies with the Appropriate Interaction Rule.

Because the ASM complies to the Unique Interaction Key Rule, the Interaction Key Transmission Rule, and the Appropriate Interaction Rule, actors following PReP are actor behaviour compliant as defined by Definition 4.9. □

Conclusion 4.11 (Autonomously Creatable). *Actor behaviour compliance caters for the autonomous creation of process documentation.*

Establishing Theorem 4.10 has shown that an important factor in catering for autonomously creatable process documentation is satisfied for actors following the PReP protocol. Allowing process documentation to be created when appropriate in an autonomous fashion is important for distributed systems that have various quality of service constraints.

4.4.5.4 Immutable

We now show that p-assertions that have been previously recorded will not be overwritten or modified. We begin by defining a function that retrieves the record messages from a View at a particular configuration.

Definition 4.12 (View Function). For all c, a_{ps}, κ, v, $View(c, a_{ps}, \kappa, v)$ is defined as $store\_T(a_{ps})(\kappa, v).recs$ where $store\_T$ is in the configuration c.

Using this function, we define the following invariant, stating that a view's contents are monotonically increasing.

Invariant 4.13 (Safety). *For any configurations c_1 and c_2, where $c_1 \longmapsto^* c_2$,*

$$View(c_1, a_{ps}, \kappa, v) \subseteq View(c_2, a_{ps}, \kappa, v)$$

for any a_{ps}, κ, v.

Proof. We show this invariant by induction on the length of the transition sequence $c_1 \longmapsto^* c_2$. In the base case, where the length is zero, c_1 equals c_2 and the invariant holds.

In the inductive case, we assume that for $c_1 \longmapsto^* c_n$ the invariant holds: therefore, $View(c_1, a_{ps}, \kappa, v) \subseteq View(c_n, a_{ps}, \kappa, v)$. We then consider the possible transitions from $c_n \longmapsto c_2$. There are only two rules of the state machine that govern the actions of the provenance store: *receive_record_passertion* and *receive_submission_finished*.

In the case of the *receive_record_passertion*, *store_T* is only added to as shown by the third and fourth pseudo-statements of the rule. Moreover, if the same local p-assertion id is used by a record message, the message is discarded and an acknowledgement is sent. This test is shown in the second pseudo-statement of the rule. Hence, $View(c_n, a_{ps}, \kappa, v) \subseteq View(c_2, a_{ps}, \kappa, v)$.

Likewise, in the case of the *receive_submission_finished* rule, if a sf message has already been recorded for a given View, then no other sf message is allowed to be recorded as seen in the second pseudo-statement. Furthermore, there are no rules that operate on *store_T* that delete or modify already recorded p-assertions. Hence,

$$View(c_n, a_{ps}, \kappa, v) \subseteq View(c_2, a_{ps}, \kappa, v).$$

By transitivity, we conclude that

$$View(c_1, a_{ps}, \kappa, v) \subseteq View(c_2, a_{ps}, \kappa, v).$$

For all other rules, the provenance store is not updated; hence, $c_n = c_2$, which completes the proof. □

Conclusion 4.14 (Immutable). *Process documentation recorded using PReP is immutable.*

Lemma 4.13 establish that recorded process documentation is immutable. Immutable process documentation is an important characteristic because it ensures that the evidence of a process will not be deleted or tampered with after its recorded, which gives confidence to both users and creators of process documentation. Users know that process documentation is the same as it was when it was originally recorded by the source. Likewise, creators know that process documentation they have entrusted to the provenance store will not disappear or be corrupted. Finally, immutable process documentation ensures that users will not be caught off-guard by its accidental or non-malicious deletion.

4.4.5.5 Attribution

We now show that for every p-assertion in a provenance store, there is an actor identity for that p-assertion that identifies p-assertion's asserter, which means that every p-assertion can be "tracked back" to its creator. Hence, actors can be held accountable for the p-assertions they create, by a means outside PReP. Again, we note that, within PReP, the creator and recorder of a p-assertion are the same actor. We begin by defining the following invariant.

Invariant 4.15 (Identity Preserving). *For all configurations,* c_1, c_2*, where* $c_1 \longmapsto^* c_2$*, for any message* m *in* c_1 *and in* c_2*, the actor identity in* m *is the same, for all* $m \in R \cup SF$*.*

Proof. We prove this invariant by induction on the length of the transition sequence $c_1 \longmapsto^* c_2$.

In the base case, where the transition length is zero, c_1 is equal to c_2 and thus the invariant holds trivially.

In the inductive case, for transition lengths greater than zero, we assume that the invariant holds for $c_1 \longmapsto^* c_n$ and we consider all the possible transitions from $c_n \longmapsto c_2$. There are four rules that deal with messages from the set $R \cup SF$: *receive_record_passertion*, *receive_submission_finished*, *record_message* and *make_passertion_msg*. None of the first three rules create record or submission finished messages neither do the rules modify them. Thus, the actor identity in c_2 remains the same as in c_n and the lemma holds. In the case where $c_n \longmapsto c_2$ using *make_passertion_msg*, the newly created message does not belong to c_n and the lemma holds. □

We now must show that a maps to the asserter of the p-assertion (α) within m.

Lemma 4.16. *An actor identity contained within a given* rec *message is always the identity of the actor that caused the generation of the* rec *message.*

Proof. In Figure 4.13, once a p-assertion is in the $assert\_T$ table, the *make_passertion_msg* rule can fire. In the first pseudo-statement in the rule, a is used to generate the rec message. Furthermore, the rule can only fire if α belongs to $assert\_T$ by a. Therefore, a in the rec message is the identity of the actor that caused the rec message to be generated. A similar argument also applies to the creation of sf messages. Finally, from Invariant 4.15, we know that these identities remain the same. □

Theorem 4.17 (Attribution). *For every p-assertion in a provenance store, there is an asserter identity for that p-assertion and that identity refers to the actor that generated the record message containing the p-assertion.*

Proof. Based on Lemma 4.15, Lemma 4.16 and Theorem 4.14, we conclude that indeed if a p-assertion is in a provenance store then the identity of the actor who created and recorded the p-assertion can be determined. Note, that p-assertions only appear within *rec* messages inside the provenance store. □

Ensuring that attribution information is correctly generated and maintained is crucial because it allows users of process documentation to hold asserters accountable for their process documentation, which helps to ensure that accurate process documentation is produced.

4.4.5.6 Finalizable

We now show that process documentation recorded by PReP is finalizable. We begin by defining an invariant.

Invariant 4.18 (Complete View). *Consider a configuration, c_1, where a View is complete. For any configuration c_2, such that $c_1 \longmapsto^* c_2$, then*

$$View(c_1, a_{ps}, \kappa, v) = View(c_2, a_{ps}, \kappa, v),$$

for all a_{ps}, κ, v.

Proof. We prove this lemma by induction on the length of transition sequence $c_1 \longmapsto^* c_2$. In the base case, where the length is 0, c_2 is the same as c_1 and the lemma trivially holds.

In the inductive case, a length greater than zero is considered. We assume that the invariant holds for $c_1 \longmapsto^* c_n$ and show that the property holds for all possible transitions from $c_n \longmapsto c_2$.

An inspection of the ASM rules shows that only two rules modify views. They are *receive_submission_finished* and *receive_record_passertion*. By hypothesis, the view is complete. Hence, by definition of complete, *sf* is not null. From Theorem 4.14, if *sf* is not $\perp$, *sf* cannot be modified. Likewise, the rules prevent messages from being recorded if *sf* is not $\perp$ (see the rules' second pseudo-statement and the complete function definition). Therefore, $View(c_2, a_{ps}, \kappa, v) = View(c_n, a_{ps}, \kappa, v)$ and the invariant holds. □

Conclusion 4.19 (Finalizable). *Documentation of a View recorded using PReP is finalizable.*

From Lemma 4.18, a View can be marked as complete and is then immutable. Thus, documentation of a particular View can be marked as complete and is finalizable.

Finalizable seals a View preventing future information from being added, which gives users a firm basis for making a judgement about the interaction the View documents; they know that no new information will suddenly arise. Furthermore, it allows a dynamic system to know when an actor is finished recording process documentation so that the actor can be removed from the system if necessary.

We now establish that documentation recorded using PReP describes a process. This characteristic of process documentation is shown in Theorem 4.38, which relies on the following termination proof.

4.4.5.7 Termination

In the case of PReP, we define the following termination property.

Definition 4.20 (Termination Property). Termination is defined as the execution of a finite number of ASM transitions excluding *send_app_msg* and *receive_app_msg* transitions.

First, only communication between actors and provenance stores (i.e. the recording of p-assertions) is considered. Thus, we show that there can only be a finite number of transitions that do not involve the *send_app_msg* and *receive_app_msg* rules. To prove termination, a system measure that indicates how far the ASM is from finishing its transitions related to the recording of p-assertions is introduced. The system measure is defined as follows:

Definition 4.21 (System Measure). Given any configuration, c, the *system_measure*(c) is the sum of the measures of each message and table in the system. Figure 4.14 lists these measures.

We now analyse how this system measure changes after an application message is either received or sent by an actor.

Table/Message	Measure
rec	6
sf	5
ack	2
$assert\_T(a, \kappa, v)$	$(1 + sizeOf(assert\_T(a, \kappa, v))) * 30$
$to\_send\_T(a, \kappa)$	$(1 + sizeOf(to\_send\_T(a, \kappa))) * 9$
$sent\_T(a, \kappa)$	$(1 + sizeOf(sent\_T(a, \kappa)))$
$ack\_T(a, \kappa)$	$sizeOf(ack\_T(a, \kappa))$
$lpid\_T(a, \kappa, v)$	$sizeOf(lpid\_T(a, \kappa, v))$
$store\_T(a_{ps})(\kappa, v).recs$	$sizeOf(store\_T(a_{ps})(\kappa, v).recs)$
$store\_T(a_{ps})(\kappa, v)).lpids$	$sizeOf(store\_T(a_{ps})(\kappa, v).lpids)$
$store\_T(a_{ps})(\kappa, v)).sf$	0 if $store\_T(a_{ps})(\kappa, v)).sf = \perp$, 1 otherwise.
$\mathcal{K}(a, a_1)$	sumOf (measures of messages in the channel)

FIGURE 4.14: Measures for tables and messages defined in the ASM

Invariant 4.22 (Decreasing Measure). *For any reachable configurations c, c' and for any transition t, such that t leads from c to c', and t is not send_app_msg or receive_app_msg, then the following inequality holds:*

$$0 \leq system\_measure(c') < system\_measure(c)$$

.

Proof. We proceed by an analysis of the transitions that lead from c to c', and we establish that the system measure decreases. The annotations in Figures 4.12 and 4.13 show the number deducted from the system measure for each transition excluding *send_app_msg* and *receive_app_msg*. The annotations also show how these measures were calculated. This demonstrates that the measure of the system is strictly decreasing. □

Lemma 4.23. *For any configuration, all transition paths that do not use send_app_msg or receive_app_msg transitions terminate.*

Proof. Because the system measure is strictly decreasing (Lemma 4.22) and always greater than or equal to 0 (Definition 4.21), from any configuration of the system a configuration without successor can be reached. □

Theorem 4.24 (Termination). *PReP satisfies the termination property.*

Proof. Lemma 4.23 establishes that PReP satisfies the termination property (Definition 4.20). This means that it takes a finite number of transitions to record a p-assertion in the provenance store and for the recording actor to receive acknowledgements of recording. □

We now show that all p-assertions created by an actor will be recorded and acknowledged.

4.4.5.8 Guaranteed Recording

Once p-assertions have been created, we show that they will be recorded in the provenance store and acknowledged. Before establishing this formally, we first define two invariants. The first invariant shows that every message sent to the provenance store will be recorded by it. The second invariant shows that once a message has been recorded the recorder will have a corresponding acknowledgement message. We make the following two assumptions:

1. Local p-assertion ids are not reused.
2. There is implicit conversion from sets to bags.

We also define a mechanism to select a group of messages from a table that contain a particular interaction key and view kind combination. This mechanism is defined as follows:

Definition 4.25 (Messages Selector). For all κ, v,

$$Bag(\mathcal{M}) \downarrow (\kappa, v) = \Big\{ \mathsf{rec}(\kappa, v, -, -, -) \in Bag(\mathcal{M}), \\ \mathsf{sf}(\kappa, v, -, -, -) \in Bag(\mathcal{M}) \Big\}$$

Invariant 4.26 (Always Recorded). *For any configuration c_1 reachable from c_i, for all κ, a, a_{ps},*

$$sent\_T(a, \kappa) \downarrow (\kappa, v) = k(a, a_{ps}) \downarrow (\kappa, v) \oplus store\_T(a_{ps})(\kappa, v).recs \oplus store\_T(a_{ps})(\kappa, v).sf$$

where v is the role identifier of a.

Proof. We prove this invariant by induction on the length of the transition sequence, $c_i \longmapsto^* c_1$, where c_i is the initial configuration of the state machine.

In the base case, where the length is zero, c_1 equals c_i. In c_i all tables and channels are empty and thus the invariant holds.

In the inductive case, we assume that for $c_i \longmapsto^* c_n$ the invariant holds. We then consider the possible transitions from $c_n \longmapsto c_1$. There are three rules that deal with the tables identified in the invariant. We address each individually.

- In the *record_message* rule, a message is added to the $sent\_T$ table and is also placed on the channel between a and a_{ps} through the *send* pseudo-statement. Hence, $sent\_T(a, \kappa)$ in $c_1 = sent\_T(a, \kappa) \oplus m$ in c_n and $k(a, a_{ps})$ in $c_1 = k(a, a_{ps}) \oplus m$ in c_n. Thus, the invariant holds for c_1 since it holds in c_n under the assumption that local p-assertions are not reused.

- In the *receive_record_passertion* rule, a message on the channel from a to a_{ps} is received and is therefore removed from the channel. The same message is then added to $store\_T$ in the fourth statement of the rule. Hence, $k(a, a_{ps})$ in $c_1 = k(a, a_{ps}) \ominus m$ in c_n and $store\_T(a_{ps})(\kappa, v).recs$ in $c_1 = store\_T(a_{ps})(\kappa, v).recs \cup \{m\}$ in c_n. Therefore, the invariant holds in c_1 because it holds in c_n under the assumption that local p-assertions are not reused.

- In the *receive_submission_finished* rule, a message on the channel from a to a_{ps} is received and is therefore removed from the channel. The same message is then added to $store\_T$ in the fourth statement of the rule. Hence, $k(a, a_{ps})$ in $c_1 = k(a, a_{ps}) \ominus m$ in c_n and $store\_T(a_{ps})(\kappa, v).sf$ in $c_1 = store\_T(a_{ps})(\kappa, v).sf \cup \{m\}$ in c_n. Therefore, the invariant holds in c_1 because it holds in c_n.

Therefore,

$$sent\_T(a, \kappa) \downarrow (\kappa, v) = k(a, a_{ps}) \downarrow (\kappa, v) \oplus store\_T(a_{ps})(\kappa, v).recs \oplus store\_T(a_{ps})(\kappa, v).sf$$

holds in the inductive case and the invariant is established.

□

We now show that the actor sending a message to a provenance store will receive a corresponding acknowledgement message. We begin by defining a function that converts the messages in a View to a set of acknowledgement messages.

Definition 4.27 (Message to Acknowledgement Conversion Function). For all a, κ, v, $viewToAck(store\_T(a)(\kappa, v))$ is defined as

$$Bag\Big(\mathsf{ack}(\kappa, v, \ell) \;\mid\; \mathsf{rec}(\kappa, v, a, \ell, \alpha) \in store\_T(a)(\kappa, v).recs \;\; or \;\; \mathsf{sf}(\kappa, v, a, \ell, na) \in store\_T(a)(\kappa, v).sf\Big)$$

By using this function, we show that the messages stored in the provenance store can be converted to acknowledgement messages and thus can be matched to the acknowledgement messages received by the recording actor. We establish this matching through the following invariant.

Invariant 4.28 (Always Acknowledged). *For all configurations c_1 reachable from c_i, for all κ, a, a_{ps}, v,*

$$viewToAck(store\_T(a)(\kappa, v)) = k(a_{ps}, a) \downarrow (\kappa, v) \oplus ack\_T(a, \kappa) \downarrow (\kappa, v).$$

Proof. We prove this invariant by induction on the length of the transition sequence $c_i \longmapsto^* c_i$, where c_i is the initial configuration of the state machine.

In the base case, where the length is zero, all tables and channels are empty and thus

$$viewToAck(store\_T(a)(\kappa, v)) = k(a_{ps}, a) = ack\_T(a, \kappa) = \emptyset$$

and the invariant holds.

In the inductive case, we assume that for $c_i \longmapsto^* c_n$ the invariant holds. We then consider the possible transitions from $c_n \longmapsto c_1$. There are three rules that deal with the tables identified in the invariant; these are m-consuming transitions 4.4. We address each individually.

- In the *receive_record_passertion* rule, a message is added to the $store\_T$ table. A corresponding acknowledgement message is also generated and added to the the channel $k(a_{ps}, a)$. Hence,

  $$viewToAck(store\_T(a)(\kappa, v)) \text{ in } c_1 = viewToAck(store\_T(a)(\kappa, v)) \oplus m \text{ in } c_n$$

  and $k(a_{ps}, a)$ in $c_1 = k(a_{ps}, a) \oplus m$ in c_n. Thus, the invariant holds in c_1 since it holds in c_n under the assumption that local p-assertions are not reused.

- In the *receive_submission_finished* rule, a message is added to the $store\_T$ table. A corresponding acknowledgement message is also generated and added to the the channel $k(a_{ps}, a)$. Hence,

  $$viewToAck(store\_T(a)(\kappa, v)) \text{ in } c_1 = viewToAck(store\_T(a)(\kappa, v)) \oplus \quad m \text{ in } c_n$$

  and $k(a_{ps}, a)$ in $c_1 = k(a_{ps}, a) \oplus m$ in c_n. Thus, the invariant holds in c_1 since it holds in c_n under the assumption that local p-assertions are not reused.

- In the *receive_ack* rule, an acknowledgement message is removed from the channel $k(a_{ps}, a)$ and then added to the $ack\_T(a, \kappa)$ table. Hence, $k(a_{ps}, a)$ in $c_1 = k(a_{ps}, a) \ominus m$ in c_n and $ack\_T(a, \kappa)$ in $c_2 = ack\_T(a, \kappa) \oplus m$ in c_n. Therefore, the invariant holds in c_1 because it holds in c_n.

Therefore, $viewToAck(store\_T(a)(\kappa, v)) = k(a_{ps}, a) \downarrow (\kappa, v) \oplus ack\_T(a, \kappa) \downarrow (\kappa, v)$ holds in the inductive case and the invariant is established. □

Using Invariants 4.26 and 4.28, we now establish that the messages and thus the p-assertions within those messages sent to the provenance store will be stored and an acknowledgement will be received by the sender. Thus, the acknowledgements received from the provenance store by the sender will be equal to what was originally sent by the sender (after a simple conversion step).

Lemma 4.29 (Messages Always Recorded and Acknowledged). *For all a, a_{ps}, in any reachable configuration, c_f where all channels between a and a_{ps} are empty, all messages from the set $R \cup SF$ that have been sent by a to a_{ps} have been stored in a_{ps} and a has received an acknowledgement.*

Proof. From Invariant 4.26, if $k(a, a_{ps}) = \emptyset$ then for all κ, v,

$$sent\_\mathcal{T}(a, \kappa) \downarrow (\kappa, v) = store\_\mathcal{T}(a_{ps})(\kappa, v).recs \cup store\_\mathcal{T}(a_{ps})(\kappa, v).sf.$$

By definition of $store\_\mathcal{T}(a)(\kappa, v)$, we can collapse this equation to be

$$sent\_\mathcal{T}(a, \kappa) \downarrow (\kappa, v) = store\_\mathcal{T}(a_{ps})(\kappa, v).$$

Furthermore, from Invariant 4.28, if $k(a_{ps}, a) = \emptyset$ then for all κ, v,

$$viewToAck(store\_\mathcal{T}(a)(\kappa, v)) = ack\_\mathcal{T}(a, \kappa) \downarrow (\kappa, v).$$

Hence,

$$viewToAck(sent\_\mathcal{T}(a, \kappa) \downarrow (\kappa, v)) = ack\_\mathcal{T}(a, \kappa) \downarrow (\kappa, v).$$

Essentially, what an actor sent to a provenance store has been acknowledged. Given that no rule removes messages from the acknowledgement table and from Theorem 4.14 the provenance store is immutable, all record and submission finished messages that have been sent to a provenance store are stored in the provenance store and acknowledgements have been received by the recorder. □

We now introduce two more invariants that state that after p-assertions are created, they end up in the $sent\_\mathcal{T}$. The proofs for these two invariants are similar to the above invariants and are thus omitted for brevity.

Invariant 4.30 (Always Sent). *For all configurations c_1, c_2, where $c_1 \longmapsto^* c_2$, for all a, κ, v,*

$$assert\_\mathcal{T}(a, \kappa, v) \oplus to\_send\_\mathcal{T}(\kappa, v) \oplus sent\_\mathcal{T}(\kappa, v)$$

is constant for all transitions excluding send_app_msg and receive_app_msg.

Invariant 4.31 (P-assertion Accumulation). *For all configurations c_1, c_2, where $c_1 \longmapsto^* c_2$, for all a, κ, v,*

$$\begin{aligned} &assert\_T(a,\kappa,v) \oplus to\_send\_T(\kappa,v) \oplus sent\_T(\kappa,v) \quad at \quad c_1 \\ \subseteq\ &assert\_T(a,\kappa,v) \oplus to\_send\_T(\kappa,v) \oplus sent\_T(\kappa,v) \quad at \quad c_2 \end{aligned}$$

for send_app_msg and receive_app_msg transitions.

The above lemma and invariants show that once p-assertions are created, they will end up in a provenance store and the creator of the p-assertions will have received acknowledgements that they have been stored. Thus, the key to having correct evidence within the provenance store is the creation of appropriate p-assertions. In the next section, we address the creation of appropriate p-assertions that reflect or mirror the process that has gone on within an application. Once created, PReP guarantees that p-assertion will be recorded.

4.4.5.9 Process Reflection

We now show that a process (i.e. the execution of an application) can be reflected by process documentation recorded in a provenance store.[1] To do this, we extend the ASM to consider the execution of actors. The actors execute in accordance with the following definition of process, repeated from Definition 3.1:

A process is a causally connected set of interactions and transformations.

Thus, the execution of actors is described by the exchange of messages between actors and the transformations they perform on received messages. We now describe this execution formally. We begin by defining the state space for the execution of the set of actors as follows:

Definition 4.32 (Extended ASM State Space).

$$\begin{aligned} \text{AS} &= \mathbb{P}(\textsc{Data} \times \text{IK}) && \text{(Actor States)} \\ \textsc{AppS} &= \text{A} \rightarrow \text{AS} && \text{(Application State)} \\ \text{APC} &= \text{C} \times \textsc{AppS} && \text{(Provenance-aware Application State)} \end{aligned}$$

Characteristic Variables:
$\langle d, \kappa \rangle \in \text{AC}$,
$as \in \textsc{AppS}$,
$apc \in \text{APC}$,
$\langle c, as \rangle = apc$

[1] By reflection, we mean common sense reflection. We do *not* mean introspection as offered by some programming language runtime environments.

As all the data that an actor works upon is located in received messages, we model the state of actor (AS) as data within received messages, where messages are identified by interaction keys. We assume that a garbage collector will collect any unused data, but, for simplicity, we do not model garbage collection here. We also define a table, AppS, that maps from actor identities (A) to actor states. The same accessor notation that we have used for other tables applies to AppS as well. So that $as(a)$ will access the state of the actor identified by a. The configuration of the application is defined by the combination of the state of all the actors in the system combined with the configuration of PReP. This is modelled by APC.

The execution of actors following Definition 3.1 can be modelled by rules that express the transition of states when sending and receiving messages. We define these as follows:

Definition 4.33 (Application Rules).

$consume\_msg(a_s, a_r, \kappa, d)$:
$\mathsf{app}(\kappa, d) \in \mathcal{K}(a_s, a_r)$
$\rightarrow$ {
$as(a_r) := as(a_r) \cup \{\langle d, \kappa \rangle\}$;
// receive business logic
}

$produce\_msg(a_s, a_r, d, f)$:
// triggered when d, produced by function f that is described by r,
// is to be sent by a_s to a_r
$\rightarrow$ {
}

Thus, after the receipt of a message, the state of the application is updated, whereas, after sending a message, the state stays the same. Sending a message is triggered by the execution of some business logic or a transformation on the actor state, which we define as follows.

Definition 4.34 (Transformation). A transformation is the execution of a function f on an actor state, $as(a)$, to achieve a tuple $\langle d, a_r \rangle$. This is represented as $\langle d, a_r \rangle = f(as(a))$ where a_r specifies the actor to which the data item should be sent.

The functions applied to actor states can be described by business logic definitions, which are specified by the set REL defined in Figure 4.11.

From these definitions, the execution of some application is the transitions between actor states denoted by $as_1 \longmapsto^*_{app} as_2$, where the transitions are defined by Definition 4.33. Following this definition, an application can execute independently of recording process

documentation. We now show that when integrated with PReP, process documentation representing the application's process will eventually be in the provenance store. To do this, we couple the application to PReP. First, we denote three different categories of transitions and provide a notation for each:

1. Provenance Aware Application Transitions are all the transitions that occur in both PReP and an application. These are denoted by $\longmapsto_{paa}$.
2. Application Transitions are the transitions defined by Definition 4.33. These are denoted by $\longmapsto_{app}$.
3. PReP Transitions are the transitions defined in Figures 4.12 and 4.13. These are denoted by $\longmapsto_{prep}$.

We term an application that records process documentation, a provenance aware application. The execution of such an application is denoted by $apc_1 \longmapsto^* apc_2$. In such an execution, PReP Transitions and Application Transitions are coupled together following Definition 4.35.

Definition 4.35 (System Coupling). For any application configurations, $\langle c_1, as_1 \rangle$, $\langle c_2, as_2 \rangle$, the following transition is allowed:

$$\langle c_1, as_1 \rangle \longmapsto_{paa} \langle c_2, as_2 \rangle$$

if one of the following conditions hold:

1. if $as_1 \longmapsto_{app} as_2$ with *produce_msg*, then
   $c_1 \longmapsto_{prep} c_2$ using the transition *send_app_msg*.
2. if $as_1 \longmapsto_{app} as_2$ with *consume_msg*, then
   $c_1 \longmapsto_{prep} c_2$ using the transition *receive_app_msg*.
3. if $c_1 \longmapsto_{prep} c_2$ then
   $a_1 = a_2$ when transitions other than *send_app_msg* or *receive_app_msg* are performed.

What is occurring is that when the *produce_msg* or *consume_msg* rules are fired in the application, the corresponding *send_app_msg* or *receive_app_msg* rule is fired as well. Essentially, the application and PReP rules are merged together. Therefore, we note that the pseudo-statements shared by the rules only execute once.

Therefore, the execution of the application is coupled with the execution of PReP via corresponding rules for sending and receiving messages. Using this system coupling definition, we will show that process documentation reflecting the application's execution

will end up in the provenance store. First, we define the function that creates documentation when an application rule fires. The function is called with $r = \perp$ when *receive_app_msg* fires and with $r \neq \perp$ when *send_app_msg* fires.

Definition 4.36.

makeAssertions(a, κ, d, r)*:*
  $PA = \emptyset$*; // set of p-assertions*
  if $r \neq \perp$
    $PA = \{$ *relationship p-assertion with the effect* d*, relation* r*, and causes* $as[a]$*,*
      interaction p-assertion with content d *and interaction key* $\kappa\}$
  else
    $PA = \{$*interaction p-assertion with content* d *and interaction key* $\kappa\}$
  return PA*;*

This *makeAssertions* function creates interaction and relationship p-assertions. When given a business logic description, r, the algorithm creates an interaction p-assertion to document, d in the interaction identified by κ as well as a relationship p-assertion using r to document the causal relationship between $as[a]$ and d. When not provided r, the algorithm creates an interaction p-assertion documenting the incoming data, d, within the interaction identified by κ. Therefore, this algorithm uses the concepts defined by the p-structure data model.

Using the above definitions, we now outline the proof of Lemma 4.37. To better illustrate the lemma and the proof outline, we use three state transition diagrams shown in Figures 4.16, 4.17, and 4.18. These figures follow the legend shown in Figure 4.15.

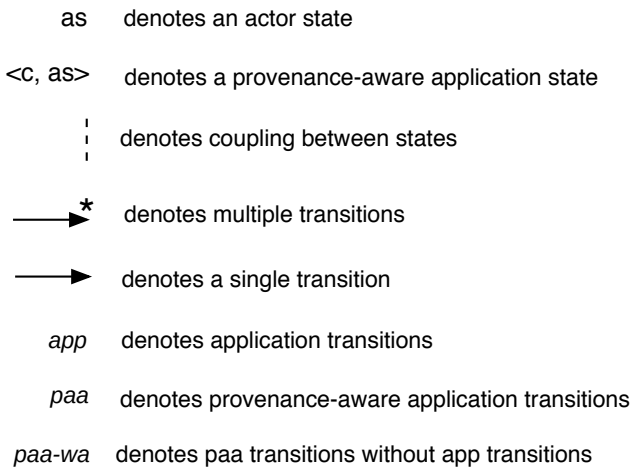


FIGURE 4.15: Legend for Figures 4.16, 4.17, and 4.18

Lemma 4.37 is defined as follows and is depicted in Figure 4.16.

Lemma 4.37 (Process Reflection). *For any application state, as, reachable from as_i, where $as_i \longmapsto^*_{app} as$; for all apc reachable from apc_i: $apc_i = \langle c_i, as_i \rangle \longmapsto^*_{paa}$ apc with $apc = \langle c, as \rangle$; there exists a final configuration $apc_2 = \langle c_2, as \rangle$ for some c_2, where there are no messages in transit and no messages to send, such that $apc \longmapsto^*_{paa} apc_2$ without application transitions, such that the provenance stores in apc_2 contain the description of $as_i \longmapsto^*_{app} as$.*

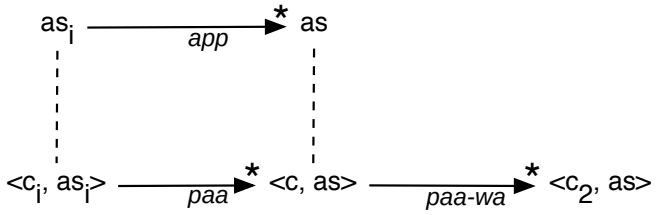


FIGURE 4.16: State transition diagram depicting Lemma 4.37

Intuitively, the application proceeds from an initial state as_i to some final state as where the application has finished executing. Because of system coupling (shown by the vertical hash lines), the provenance-aware application also proceeds from an initial state $\langle c_i, as_i \rangle$ to some state $\langle c, as \rangle$. However, there may be p-assertions remaining to be recorded that describe the application's execution, thus the provenance-aware application finishes recording those p-assertions without using application transitions (denoted by *paa-wa* in the figures). We now outline a proof of this lemma.

Proof. Our proof outline proceeds by induction on the length of the transition sequence from $as_i \longmapsto^*_{app} as$.

In the base case, as_i equals as, hence no execution has taken place and process documentation is empty and the lemma holds trivially.

In the inductive case, we assume if $as_i \longmapsto^*_{app} as_n$ then $apc_i \longmapsto^*_{paa} apc_n$ and process documentation describing $as_i \longmapsto^*_{app} as_n$ will be recorded in a set of provenance stores at some later configuration, $\langle c_x, as_n \rangle$. This is the inductive hypothesis and is shown in Figure 4.17.

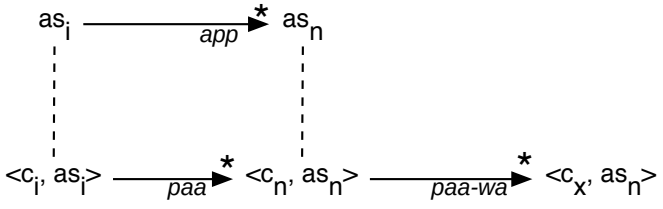


FIGURE 4.17: State transition diagram depicting the inductive hypothesis for proof of Lemma 4.37

We now show that for all possible transitions from $as_n \longmapsto_{app} as$ that process documentation describing that transition will be in a provenance store after $apc_n \longmapsto^*_{paa-wa} apc_2$, where $apc_2 = \langle c_2, as \rangle$ and one application transition.

This inductive step is depicted in Figure 4.18. The figure shows the application proceeding from as_i after any number of transitions to the state as_n. We assume that p-assertions describing this execution are recorded in the provenance store when the provenance-aware application state space reaches $\langle c_x, as_n \rangle$. Once the application has reached the state as_n, one more application transition occurs to the state as. Through system coupling, the provenance-aware application state will also proceed from $\langle c_x, as_n \rangle$ to $\langle c_x, as \rangle$ by one application transition. We note that this application transition can occur at any time after configuration $\langle c_n, as_n \rangle$. It does not have to wait for the recording of p-assertions to finish.

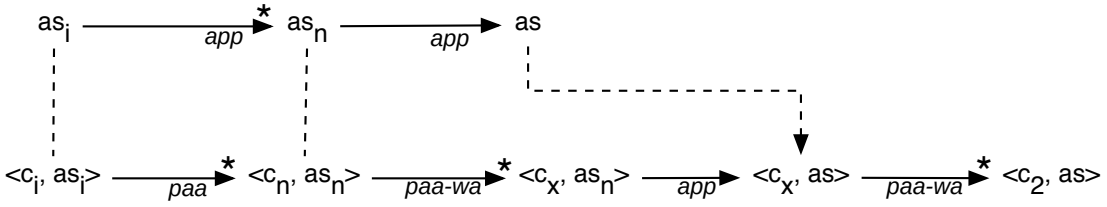


FIGURE 4.18: State transition diagram depicting the inductive step for proof of Lemma 4.37

We now consider the two possible application transitions from $as_n \longmapsto_{app} as$, *produce_msg* and *consume_msg*. When one of these rules executes, from Definition 4.35, the corresponding rule (*send_app_msg* and *receive_app_msg*) defined in Figure 4.13 will eventually execute. This eventual execution is based on our hypothesis that in the ASM any number of other rules can fire between the execution of the application rule and the corresponding PReP rule but that rule will fire.

Once the corresponding rule is fired, the *makeAssertion* function is called and a set of p-assertions, $\alpha_1, \ldots, \alpha_n$, is produced. These p-assertions are then placed in a table, $assert\_T$, with a key for the particular interaction, κ. Once α is in the $assert\_T$ table, from Invariants 4.30 and 4.31, it will end up in a message in $sent\_T$. From Lemma 4.29, the message will be recorded and acknowledged in a finite number transitions and the system will reach a final configuration, apc_2. Thus, in the inductive case the lemma holds.

□

Conclusion 4.38 (Process Oriented). *For a finite process, PReP can record documentation that reflects a process.*

If actors in an application execute as described by Definition 3.1 and use Definition 4.36

to create their p-assertions, then from Lemma 4.37 documentation reflecting application's process will eventually be found in provenance stores.

Through the above proofs, we have shown that PReP helps ensure that the process documentation recorded using it complies with the six characteristics necessary for accurate process documentation. We follow a systematic proof procedure based on mathematical induction. We believe that this systematic approach, while done by hand, is sufficient to provide confidence that the protocol does conform to the properties specified in Section 4.4.1. Furthermore, because of this systematic approach and the extensive use of invariants, we hypothesise that the proofs are at a stage where they could be translated into a format suitable for mechanical proof.

Beyond establishing these properties, the formalisation of PReP serves another equally important purpose, namely, to provide an accurate, implementation independent means of specifying the protocol. Because we consider multiple platforms, it is necessary to have a specification that is independent of any given programming language or implementation. Furthermore, this specification should be rigorous and well-defined to ensure that developers know exactly how to the protocol should perform. If we were to specify the protocol in a particular programming language, the protocol would very likely become dependent on the underlying features of the language and its execution semantics. In fact, it has been shown that problems arise from defining protocols using such an approach [132]. The abstract state machine we have defined provides the rigorous, independent, and well-defined specification that is required.

Finally, defining the protocol using an abstract state machine has helped us to better understand the ramifications of our design choices. This improved understanding led to extraneous messages and fields being removed from the protocol and clearer definitions of the required behaviour of actors. Without using formal modelling the protocol would be more complicated and its semantics would not have been as precise. In summary, the use of a formal approach has significantly improved the design of PReP.

4.5 Summary

Once p-assertions are created by actors, they need to be aggregated and stored such that queriers can access them to determine the provenance of results. In this chapter, we introduced the provenance store, an architectural element that caters for the long term storage of process documentation. It provides well defined interfaces for the recording and querying of process documentation. We defined the recording interface of the provenance store through the formal specification of the P-assertion Recording Protocol (PReP). The protocol is formalised using an abstract state machine notation, which is both code-like and rigorous enough for proof. Based on this formalisation, a number of proofs were given that guarantee that process documentation recorded using the protocol

will be high-quality. In addition to the protocol, we developed a series of patterns that provide rules for the deployment of provenance stores within applications. Moreover, we defined a solution, linking, for connecting process documentation distributed across provenance stores.

We now revisit the four contributions of this chapter. First, the introduction of the provenance store provides a mechanism to store and maintain process documentation and is supported by analysis conclusion five (see Section 2.5), namely, that provenance information should be separated from its collection point. This architectural approach has been successfully used to store data similar to process documentation in large scale Grid implementations. Second, the chapter detailed deployment patterns, which help to cater for systems that have multiple distributed components under different institutional authorities. Third, linking provides a way of connecting distributed process documentation by following an approach that has been successful for connecting information on the World Wide Web. Finally, PReP provides a well-specified, implementation independent protocol for recording process documentation in a manner that helps ensure that the documentation recorded will make for high-quality evidence that a process occurred.

In Chapter 6, an implementation of the protocol is discussed in more detail. This implementation directly reflects PReP. Here, we note that the client side side portion of the protocol has been implemented in the Java [89, 102] and Python [93] programming languages. Likewise, the server side of the protocol has been implemented by two different development teams [100, 89]. Additionally, the patterns defined in this dissertation were adopted verbatim by the EU Provenance project [92], a European Commission funded six member consortium.

A common thread that runs throughout this chapter is that distribution is critical to catering for multi-institutional applications and for scalability. Applications that span multiple institutions fundamentally require repositories owned and operated by different owners due to regulations, politics, and security concerns. Hence, our solution allows process documentation to be both distributed *and* connected. Furthermore, as the size, frequency, and number of p-assertions generated by applications grow, the mechanisms we provide allow provenance stores to be added in order to divide the load and thus scale. Distributed provenance stores are the final element of a blueprint for the creation, organisation and recording of process documentation so that the provenance of results produced by multiple institutions can be determined. In the next two chapters, we evaluate this blueprint from the perspective of a multi-institutional bioinformatics experiment.

Chapter 5

Case Study: The Amino Acid Compressibility Experiment

The previous chapters have presented a conceptual blueprint for the creation, organisation, and recording of process documentation. The question now arises as to how best to evaluate this blueprint. This dissertation began by motivating the need for provenance in multi-institutional scientific systems. Therefore, one possible form of evaluation would show that an instantiation of this blueprint can be used to answer questions about the provenance of data produced by such systems. However, we cannot feasibly test an instantiation of the blueprint in *every* application. Thus, we take the approach of conducting a range of provenance queries in an application that has the general properties of multi-institutional scientific systems. Furthermore, we adopt an application that is high performance and has fine grain parallelism, which implies that p-assertion recording may be difficult. Hence, showing that our approach performs in this difficult application provides support for the conclusion that the approach will work for a large set of applications with less demanding requirements. While we consider one case study within this dissertation, our blueprint has been used by ourselves and others in a number of other applications including organ transplant management [105], aerospace engineering [107], RSS feeds [118], trust calculations [149], fault tolerance in distributed systems [183], and biodiversity [193].

In this chapter, the Amino Acid Compressibility Experiment (ACE) is presented, which has the properties of being multi-institutional scientific system. In the next chapter, we evaluate whether an implementation of our blueprint can be used to answer the provenance questions arising in ACE. The majority of the chapter is background information necessary to understand the evaluation. Its main contribution comes in the form of six provenance use cases from ACE that also reflect use cases from other scientific applications.

It is important to note that ACE was designed by Dr. Klaus-Peter Zauner and Dr.

Stefan Artmann. The description of it here is based on personal communication with Dr. Zauner. Our implementation of the experiment for a Grid environment reflects his original implementation in the Tool Command Language (TCL). The work presented here follows on from a paper we co-authored with Dr. Zauner [88].

The rest of this chapter is organised as follows. We begin with two short introductions to the biochemistry and information theory that underlie ACE. We then give a description of its workflow. After which, we show how ACE has the properties of a multi-institutional scientific system. With this understanding of the experiment, a set of use cases is enumerated. Finally, we conclude.

5.1 A Short Introduction to Biochemistry and Information Theory

ACE attempts to find possible new relationships between amino acids, the basic building blocks of life, by investigating the information theoretic properties of their computational representations. To understand the functioning of the experiment, a brief introduction to both subjects is given. The aim with these brief introductions is to keep this dissertation as self contained as possible.

5.1.1 Biochemistry

Proteins, like the one pictured in Figure 5.1, are the essential functional components of all known forms of life and are linear chains of amino acids joined by peptide bonds. There are 21 different standard amino acids found in proteins, which are listed in Figure 5.2. These amino acids are assembled into protein sequences following a code sequence represented by a polymer (mature mRNA). During and following the assembly, the protein will fold under the electrostatic interaction of its thousands of atoms into a defined but flexible shape of typically 58 nanometres size. The resulting 3D-shape of the protein determines its function [142].

How a protein folds, and thus the function it takes, is affected by the amino acids that it is made up of. In many cases, amino acids can be substituted for one another in a protein sequence and the protein will still fold. This substitution can have both beneficial and adverse effects on protein function and thus impact the operation of the organism to which it belongs. For example, the substitution of the amino acid glutamate with the amino acid valine in haemoglobin proteins, which are part of red blood cells, causes sickle cell anaemia. Because of these possible effects, it is helpful to categorise possible substitutions of amino acids.

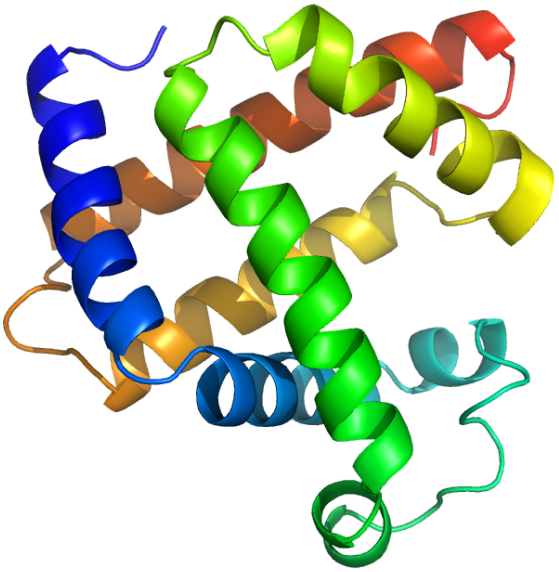


FIGURE 5.1: The 3D structure of the myoglobin protein.

Amino Acid	3-Letter Code	1-Letter Code
alanine	ala	A
arginine	arg	R
asparagine	asn	N
aspartic acid	asp	D
cysteine	cys	C
glutamic acid	glu	E
glutamine	gln	Q
glycine	gly	G
histidine	his	H
isoleucine	ile	I
leucine	leu	L
lysine	lys	K
methionine	met	M
phenylalanine	phe	F
proline	pro	P
selenocysteine	sec	U
serine	ser	S
threonine	thr	T
tryptophan	trp	W
tyrosine	tyr	Y
valine	val	V

FIGURE 5.2: The amino acids and their abbreviations

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
Ala A	13	6	9	9	5	8	9	12	6	8	6	7	7	4	11	11	11	2	4	9
Arg R	3	17	4	3	2	5	3	2	6	3	2	9	4	1	4	4	3	7	2	2
Asn N	4	4	6	7	2	5	6	4	6	3	2	5	3	2	4	5	4	2	3	3
Asp D	5	4	8	11	1	7	10	5	6	3	2	5	3	1	4	5	5	1	2	3
Cys C	2	1	1	1	52	1	1	2	2	2	1	1	1	1	2	3	2	1	4	2
Gln Q	3	5	5	6	1	10	7	3	7	2	3	5	3	1	4	3	3	1	2	3
Glu E	5	4	7	11	1	9	12	5	6	3	2	5	3	1	4	5	5	1	2	3
Gly G	12	5	10	10	4	7	9	27	5	5	4	6	5	3	8	11	9	2	3	7
His H	2	5	5	4	2	7	4	2	15	2	2	3	2	2	3	3	2	2	3	2
Ile I	3	2	2	2	2	2	2	2	2	10	6	2	6	5	2	3	4	1	3	9
Leu L	6	4	4	3	2	6	4	3	5	15	34	4	20	13	5	4	6	6	7	13
Lys K	6	18	10	8	2	10	8	5	8	5	4	24	9	2	6	8	8	4	3	5
Met M	1	1	1	1	0	1	1	1	1	2	3	2	6	2	1	1	1	1	1	2
Phe F	2	1	2	1	1	1	1	1	3	5	6	1	4	32	1	2	2	4	20	3
Pro P	7	5	5	4	3	5	4	5	5	3	3	4	3	2	20	6	5	1	2	4
Ser S	9	6	8	7	7	6	7	9	6	5	4	7	5	3	9	10	9	4	4	6
Thr T	8	5	6	6	4	5	5	6	4	6	4	6	5	3	6	8	11	2	3	6
Trp W	0	2	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	55	1	0
Tyr Y	1	1	2	1	3	1	1	1	3	2	2	1	2	15	1	2	2	3	31	2
Val V	7	4	4	4	4	4	4	4	5	4	15	10	4	10	5	5	5	72	4	17

FIGURE 5.3: An example mutation matrix

One method for categorising amino acid substitutions is by investigating what substitutions have occurred in nature. If a substitution occurred often during the course of evolution, then that substitution may be effective in a particular protein. Substitutions, which occurred in nature, are represented in so-called *mutation matrices* that describe the propensity for a particular substitution to occur [21]. Figure 5.3 shows an example matrix. The higher the number in the matrix the more often the substitution has happened in nature. For example, Figure 5.3, shows the substitution of phenylaline (Phe F) with theronine (Tyr) appears often in nature. These mutation matrices are built by computing how often a substitution occurs and comparing that to the random probability that the substitution would occur.

Mutation matrices provide a reasonable starting point for knowing whether a particular substitution is good overall. However, it does not give a precise description of how a substituted amino acid would function within a protein (i.e. whether it is chemically possible).

Therefore, another classification has been developed that is based on the various physical, chemical and structural properties of amino acids. This classification is known as the Taylor Categorisation and is shown as a Venn Diagram in Figure 5.4[1]. The letters in Figure 5.4 correspond to the 1-letter codes shown in Figure 5.2. The amino acids are clustered according to the properties enumerated below [21].

1. Size - The number of nucleotides making up a particular amino acid has an effect on possible substitution. Three categories are given in the diagram: tiny, small and large.

2. Hydrophobicity - Whether an amino acid prefers a water based environment or not.

[1]Used with the permission of the author, Robert Russell.

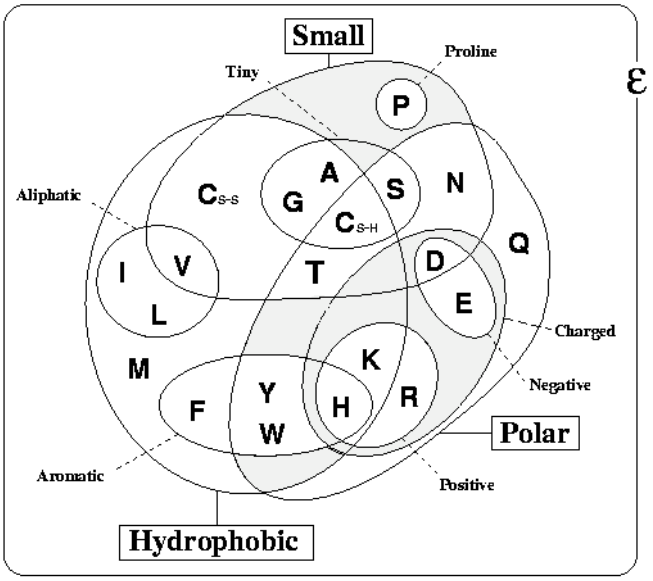


FIGURE 5.4: The Taylor Categorisation of amino acids.

3. Polarity - Whether the amino acid has a positive, neutral, or negative charge has an effect on substitution.

4. Aliphatic - Whether the "side chain" (a part of the molecule that does not belong to the core structure) of the amino acid contains mostly carbon and hydrogen atoms, which results in amino acids that tend to be non-reactive, and thus do not often effect the function of the protein they are in but instead act as a structural glue.

5. Aromatic - Whether the side chain of the amino acid contains a specific aromatic configuration of electrons. This configuration has an effect on the binding of the amino acid to other molecules.

The Taylor Categorisation shows what amino acids have similar properties. Those amino acids that share the same properties are more likely to be substitutable for one another. Thus, the Taylor classification provides another good starting point for determining whether a particular substitution is functionally feasible.

Mutation matrices and the Taylor classifications provide a basis for substitution studies, they do not and cannot provide definitive answers as to whether a substitution works. In the spirit of these classifications, ACE aims to find other possible groups of amino acids that make good candidates for substitution studies. However, instead of using the physical, chemical, or structural properties of amino acids to find possible substitutions, ACE measures the information content of proteins. This mechanism of classification is based on information theory, whose pertinent points we now discuss.

5.1.2 Information Theory

Information Theory is the mathematical study of information. It is also known as Communication Theory because it began with the study of idealised communication systems. In fact, the theory stems from the study of early telegraph and telephone networks. We begin with a description of classical information theory by briefly summarising the work of Claude Shannon [157, 41]. Figure 5.5 shows an idealised communication systems with the following components:

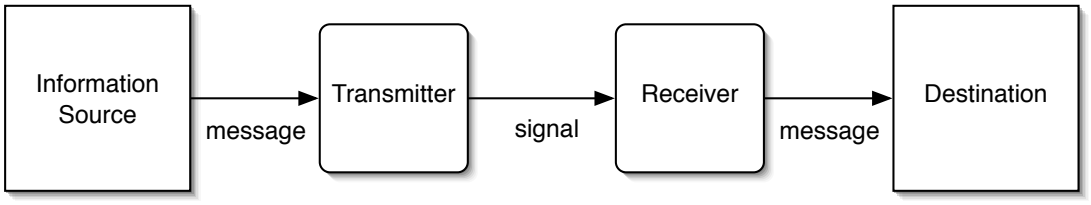


FIGURE 5.5: A basic communication system as defined by Shannon

1. An *information source* is an entity which produces messages to be communicated to a recipient.
2. A *transmitter* takes messages from the information source and produces a *signal*, which can be sent over a *channel*.
3. A *channel* is the medium over which signals are transmitted. The medium could be fiber optic cables, radio frequencies, coaxial cable, etc..
4. A *receiver* takes a signal and reconstructs a message from it.
5. A *destination* is the entity for which the message is intended.

An information source may provide messages of any type, text, sound, pictures, etc.. These messages are then *encoded* by the transmitter before being sent over a channel. Encoding takes one representation of a message and changes it into another representation. For a communication system builder, it is important to be able to determine an encoding that generates the most efficient representation for communication. Information Theory provides a precise mechanism for determining whether an encoding provides an efficient representation through the measurement of the amount of information sent in an encoded message.

By definition, an information source produces a message from a set of possible messages. It exercises a *choice* between those messages. If the destination can predict the message coming from a source, then there is no need to receive the message and thus the message does not convey any new information from the destination's point of view. However, the

smaller the chance the destination has of predicting the message the more information that message contains. Hence, the more uncertainty a destination has about an incoming message the more information is conveyed by that message. The goal of communication is then to resolve the uncertainty that a destination has about the knowledge of the source. Shannon Entropy is the name given to this uncertainty and thus is one way in which information quantity is measured. Mathematically, Shannon Entropy is a measure of the amount of uncertainty a receiver has about some discrete random variable and is measured in bits. Thus, classical information theory provides a *statistical* measure of information content.

In contrast to classical information theory, algorithmic information theory studies the information content using an *algorithmic* approach. The common measure provided by algorithmic information theory is Kolmogorov complexity [48], which is defined as the length of the shortest description of a given string. Descriptions must be specified in a fixed language such as a computer programming language. The longer the description of a given string is, the more information that string contains. One of the problems with Kolmogorov complexity is that in general it is non-computable [48, 162].[2] However, an estimate of it can be computed using a variety of mechanisms one of those being compression.

Compressors are designed to achieve the smallest possible data size given some predictable input. In terms of Kolmogorov complexity, one can view a compressor as generating a description that describes a string in the shortest possible manner. Essentially, one treats the loss-lessly compressed data as the description and the decompresser as the language of that description. Thus, the size of the compressed data is an estimate of the Kolmogorov complexity and hence can be used as an estimate for the information content of the string. Broadly, the less a data item can be compressed the more information it contains.

We now explain how both the statistical and algorithmic approaches are combined within ACE to find interesting amino acid substitutions.

5.2 ACE: The Amino Acid Compressibility Experiment

ACE starts from a basic assumption that proteins are information efficient, i.e. they use the least number of amino acids possible to obtain their function. The premise is that evolution results in the best and hence most efficient use of information, which seems reasonable given that, with just four nucleotides, DNA codes for incredibly complex

[2]Kolmogorov complexity is non-computable because of the halting problem. To find the shortest program that generates a string, one needs to test all possible programs that could generate that string. However, at any given time some of the programs may not have finished execution and there is no way to tell when they will finish. Therefore, one cannot be certain that the shortest program has been found and hence the Kolmogorov Complexity cannot be computed.

organisms. Based on this assumption, ACE tests whether particular substitutions of one amino acid for another result in high *information efficiency*. The intuition is that high information efficiency means is key to creating functioning proteins.

The experiment begins with a set of *codings* that specify the substitutions to test. Each coding consists of *groups*, which are analogous to the clusters shown in Figure 5.4. For example, a group could specify that isoleucine, leucine, and valine can be substituted for one another. Using a symbol assigned to each group, codings are specified using the following format:

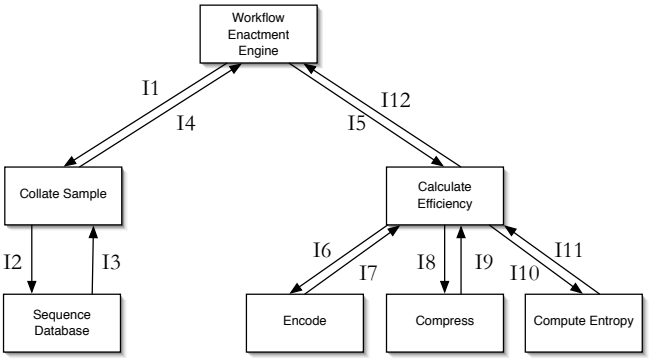
⟨GROUP SYMBOL⟩:⟨AMINO ACID 1-LETTER CODE⟩⟨AMINO ACID 1-LETTER CODE⟩..., ⟨GROUP SYMBOL⟩⟨AMINO ACID 1-LETTER CODE⟩....

Therefore using the above format, A:GST,B:ILV is an example of a coding. Codings contain multiple groups because the combination of different substitutions are more likely to produce a functional protein whereas a substitution on its own may not be effective. For example, substituting lecuine with valine may not work, however, substituting these along with glycine for alanine may result in a functional protein. Therefore, it is important to test a variety of group combinations. Currently, the set of codings is programatically produced in sequential order. However, future versions of the experiment will use genetic algorithms to generate efficient codings and thus reduce the search space of possible codings.

These codings are treated as input to the workflow shown in Figure 5.6. Steps in the workflow appear in parenthesis in the text. The first step of the workflow is to generate several samples to which the codings can be applied (Collate Sample). The samples must be of sufficient size so that the statistical methods used by the compression algorithms in the later portion of the workflow work appropriately. Therefore, a sample is generated from collating multiple randomly selected protein sequences from a protein sequence database such as UniProtKB and RefSeq [195]. To ensure an absence of repetition in a sample, sequences are selected that are highly dissimilar from each other. Dissimilarity is determined using sequence identities obtained by applying sequence alignment algorithms such as BLAST or PSI-BLAST [187]. The identification of sequences can either be performed during the run of the experiment or by acquiring a precomputed sequence list from an external culling services such as PISCES [185].[3] Thus, there are two mechanisms for building samples, either randomly selecting from a predetermined set of dissimilar sequences or randomly selecting a sequence from a database and checking for dissimilarity with sequences already in the sample.

After several samples are generated, the efficiency of the substitutions, enumerated in the list of codings, can be tested (Calculate Efficiency). The first step in the calculation of a coding's efficiency is its application to the samples (Encode). The encoding proceeds by substituting the group symbol specified in the coding for each amino acid that is part

[3] A culling service obtains subsets of proteins based on some fixed criteria.



I1: collate sample request
I2: database request
I3: sequences
I4: collated sample response
I5: calculate efficiency request
I6: sample
I7: encoded sample
I8: encoded sample
I9: compressed sample
I10: encoded sample
I11: entropy
I12: information efficiency value

FIGURE 5.6: The ACE workflow

of that group. For instance, after the application of the coding, A:GST,B:ILV, the sequence GSTILTVVSI would be AAABBABBAB.

Once a sample has been encoded, it is then compressed using a common loss-less compression algorithm such as GZIP, BZIP or PPM* (Compress). Typically, a high order PPM compression algorithm is used to significantly compress the encoded sample. The size of this compressed sequence is taken as an approximation of Kolmogorov complexity. The better the compression algorithm is, the better the approximation.

After the encoded sample is compressed, its Shannon Entropy is then computed using Equation 5.1 (Compute Entropy). The encoded sample is treated as a random variable S that has an alphabet $\mathcal{A}$. The function $p(x_i)$ determines the probability that a symbol, x_i, within $\mathcal{A}$ occurs in S. For example, given the string AAB, p(A) would be 2/3, where the alphabet used by the string is A and B.

$$H(S) = -\sum_{x_i \in \mathcal{A}} p(x_i) \log_2 p(x_i) \tag{5.1}$$

The Shannon Entropy provides a standard of comparison for the encoded sample. It removes the influence of two factors from the compression: the particular data encoding used to represent the sample, and the non-uniform frequency of groups. Once the Shannon Entropy and estimated Kolmogorov Complexity are computed, the information efficiency is calculated using Equation 5.2 (Calculate Efficiency). In this equation, $\eta_{\mathcal{C}}(s)$ is the information efficiency of a string, s, calculated using the compression algorithm $\mathcal{C}$. The length in bits of a given string is computed by the function $\ell()$ and the Kolmogorov

complexity approximated by compression is represented as $\mathcal{K}_{\mathcal{C}}()$. The encoded sample is treated as the input to $\eta_{\mathcal{C}}()$ (i.e. s).

$$\eta_{\mathcal{C}}(s) = \frac{\mathcal{K}_{\mathcal{C}}(s)}{\ell(s)H(s)} \tag{5.2}$$

Information efficiency values can be compared because their calculation takes into account both the compression method and group coding employed as well as the size of the sequence used. Once the information efficiency values for different groups are calculated they can then be plotted to find those codings that maximise efficiency and thus are good candidates for further substitution investigation.

5.3 ACE as a Multi-Institutional Scientific System

ACE relies on multiple institutions to be effective. First, the data necessary to run the experiment is held by another institution, namely, the European Bioinformatics Institute, which collates protein sequence information from hundreds of other institutions throughout the world. Second, the workflow's performance is improved when using a culling service provided by another institution. Third, the computational resources of multiple institutions are necessary in order to generate interesting results.

This need for other institution's computational resources is necessitated by the vast number of possible codings (roughly 474 trillion[4]) that could be investigated. Even with the introduction of genetic algorithms to prune the search space, for the experiment to investigate a sufficiently sized sample, large scale computational resources are necessary.

To obtain the resources necessary to run this single experiment at the scientist's site would be difficult as well as wasteful. Just to test one billion codings would require, roughly, the use of 32 computers non-stop for a year.[5] Already, this type of computing power requires a computer cluster to be built and maintained which is outside the bioinformatician's remit. Furthermore, during the analysis of results and modification of the experiment, the cluster would lay idle, wasting processor cycles.

The University of Southampton already recognises these difficulties and runs several shared computational resources for use by its child institutions. There is a separate institution, Information System Services[6], that is responsible for running these resources, which include clusters, high speed servers, and a desktop-based Grid. However, because

[4]The exact number of possible codings is 474,869,816,156,751, which is the Bell number for 21: this is the number of partitions of the set of 21 amino acids. If only 20 the standard amino acids are considered, the number of codings falls to a still impressive 51,724,158,235,372.

[5]It takes just over 1 second to process a coding on a 2.2 Ghz AMD Opteron Processor. Thus, it would take 31.7 years for one computer to process a billion codings.

[6]http://www.soton.ac.uk/iss/

these resources are shared by multiple departments (physics, computer science, chemistry, etc.), they cannot all be applied only to ACE. Therefore, to obtain even more computing power, ACE has been implemented, as described in the next chapter, to take advantage of resources made available by other institutions (i.e. other Universities, research labs and corporations) using the Grid. An example of the computational resources made available through the Grid is the Enabling Grids for E-Science (EGEE) Grid, which has 20,000 CPUs from 90 institutions available for shared use. This is 100 times larger than the largest cluster provided by the University of Southampton.

From the preceding discussion, we make the case that ACE fulfils the properties of a multi-institutional scientific system described in Section 2.1.

1. It requires the *sharing* of both data (protein sequences, dissimilarity information) and processing power.
2. It executes in a *heterogeneous* environment where information is provided from different sources and the software is executed on a range of machines with different software and hardware environments.
3. Both the resources and algorithms used by ACE are *dynamic*. For example, the resources available to the experiment changes as the shared facilities it requires have more or less demand.

In this context, we now enumerate the particular provenance use cases ACE presents.

5.4 Six Provenance Use Cases

The provenance use cases below are derived from discussions with the bioinformatician who developed ACE as well as use cases taken from a requirements gathering process [126]. Requirements gathering consisted of discussions with scientists conducting seven different projects ranging from high energy physics to computer security. During interviews with the scientists, a broad description of provenance and the goal of computational support for provenance was given. To aid understanding, some example use cases were presented to the scientist. The scientists then discussed their various needs (functional requirements) and the criteria with which any resulting software would be judged (non-functional requirements). After the interviews, the requirements were written in a consistent format and presented to the scientist for approval. In total 23 different use cases were gathered.

The following use cases cover the questions who, what, when, where, why, and how. We label each question with the interrogative word that it covers in parenthesis. Also after an explanation of each use case in terms of ACE, we add a use case from Miles et al.

that reflects a similar requirement [126]. These additional use cases are taken verbatim[7] from the paper and are formatted as quotations. Our aim is to show that the use cases from ACE case study are representative of a variety of use cases from different projects.

1. What were the sequences used in the production of a particular information efficiency value? (What)

When a coding is found that is of interest because of its information efficiency value, the bioinformatician would like to determine the causes of the particular result. A noteworthy piece of information is the list of sequences that were randomly chosen and then collated together to form the original sample. With this information, the bioinformatician can decide whether the sequences had any adverse or abnormal impact on the outcome of the experiment.

> USE CASE 3. (Intron Compressibility Experiment) A bioinformatician, B, performs an experiment on a set of chromosome data, from which the exon and intron sequences have been extracted. As a result of that experiment, B identifies a highly compressable intron sequence. B identifies which chromosome the intron originally came from.

2. What were the input figures used in the efficiency calculation that produced a particular information efficiency value? (How)

During the execution of ACE several intermediate values are produced in the generation of the final information efficiency figure. Of particular importance, are the values for $H(s)$, $\mathcal{K}_{\mathcal{C}}(s)$, and $\ell(s)$. These values can be reused in different equations for the calculation of information efficiency as well as in other related experiments. Keeping access to these values can save significant computation time.

> USE CASE 6. (Protein Identification Experiment) A biologist, B, sets the voltage of a mass spectrometer before performing an experiment to determine the mass-to-charge ratio of peptides. Later another biologist, R, judges the experiment results and considers them to be particularly accurate. R determines the voltage used in the experiment so that it can be set the same for measuring peptides of the same protein in future experiments.

3. Were there any conflicting views of an interaction in the production of a particular information efficiency value? (Who)

Because ACE is executed across multiple institutions, it is important to be able to determine if communicating parties are in agreement and have the same view of the

[7] Abbreviations for the experiments are expanded.

data they send and receive. If conflicting views are detected, it is important that the bioinformatician knows who is involved so that appropriate action can be taken.

> USE CASE 15. (Second Harmonic Generation Experiment) A chemist, C, performs an experiment finishing at a particular time. D later performs the same experiment and submits a patent for the result and the process that led to it to patent officer R. C claims to R that they performed the experiment before D. R determines whether C is correct.

4. Were references used in the process documentation created for a particular information efficiency value? (Where)

One method, previously described, for reducing the size of process documentation is the use of references (file paths or URLs) instead of storing data within the provenance store. Hence, when ACE creates process documentation, the bioinformatician often wants to know if it contains the actual application data or just references to those data items. If the documentation of the experimental run contains the actual data, less work is necessary to analyse the data because it is in one place. Furthermore, the bioinformatician might have moved, deleted or modified his original data mistakenly or on purpose. Thus, the knowledge, as to whether the documentation of a particular experiment contains original data, can inform the bioinformatician about the types of analysis that can be performed.

> USE CASE 21. (Particle Detection Experiment) A physicist, P, performs an experiment using detector data as input. The size of the detector data is in the order of petabytes. The process documentation of the experiment is recorded for later use without copying the data set.

5. What interactions are common for all information efficiency values produced by a particular job? (Why)

In Grid computing environments, it is often the case that a collection of computations are assembled together in what is referred to as a *job*. These jobs can then be executed on different computers in the Grid. In the case of ACE, multiple information efficiency calculations are put together in one job. In this use case, the bioinformatician wants to know if there were any interactions that were shared between jobs. This is used, for example, to determine if the same sequence database was used in the production of the sample that the job's computations operate on. Also, if the bioinformatician knows that a particular computation is producing incorrect output, he can determine whether the other computations in the same job are also producing incorrect output helping to explain why a particular error occurred.

> USE CASE 9. (Second Harmonic Generation Experiment) A batch of chemicals is received by a laboratory, and samples are distributed to chemists in that laboratory. A chemist, C, performs an experiment but then examines the results and finds them doubtful. C determines the source material used in the experiment and then which other recent experiments used material from the same batch. C examines the results of those experiments to determine whether the batch may have been contaminated and so should be discarded.

6. How long does it take to produce an information efficiency value from a particular sample? (When)

Some samples may take more or less time to encode and thus process. In this use case, the bioinformatician wishes to know the impact that using a particular sample has on performance. This can be used for comparisons between the computational resources provided by different institutions within a Grid. Moreover, this data can be used to determine the impact that different codings have on performance.

> USE CASE 12. (Security Testing Experiment) A service, X, is accessed by an intruder, I, that should not have rights to do so. Later, an administrator becomes aware of the intrusion and determines the time and the credentials used by the intruder to gain access.
>
> USE CASE 18. (Candidate Gene Experiment) Several bioinformaticians perform experiments using service X. Another bioinformatician, B, constructs a workflow that uses X. B can estimate the duration that the experiment might take on the basis of the average time X has taken to complete its tasks before.

These use case questions give examples of the who, what, when, where, why and how questions that can be asked about an application's process. Others have identified these as vital questions that a provenance system should address [150]. However, there are other questions beyond these six questions that our approach can address because of its process focus. Furthermore, ACE use cases represent a range of provenance questions that arise in applications from a variety of domains. In the next chapter, we demonstrate how these questions can be answered using queries performed over process documentation.

5.5 Summary

In this chapter, we presented the Amino Acid Compressibility Experiment, a multi-institutional scientific system that combines notions from the fields of information theory and biochemistry to help find possible amino acid substitutions for proteins. ACE

is an interesting application not only from a scientific perspective but also from the requirements it has in terms of computational resources. These requirements make it necessary that the experiment be implemented in a multi-institutional setting. Thus, ACE provides a basis for our evaluation of the conceptual blueprint described in preceding chapters. In the next chapter, we describe an implementation of both the provenance architecture and ACE and show that our approach is successful in terms of scalability, application performance impact, and functionality.

Chapter 6

Evaluation

In the previous chapter, we presented the Amino Acid Compressibility Experiment and six provenance use cases related to the experiment. Using these representative use cases, we now evaluate an implementation of the approach outlined in Chapters 3 and 4, P-assertion Recording for Services (PReServ). By evaluating a concrete implementation, we show that our conceptual blueprint can be realised and is effective under real world conditions.

Our evaluation is conducted at several levels. First, we analyse the scalability of PReServ in a controlled environment and demonstrate that it can handle up to 560 simultaneous connections recording p-assertions without reaching a plateau in throughput. Second, we analyse the performance impact on the execution time of ACE and find that their is a 13% overhead for p-assertion recording. Third, we demonstrate that the six provenance use cases can be answered practically by queries over process documentation. This multi-level analysis provides a characterisation of the use of process documentation in a practical application to answer provenance questions. Additionally, it provides a characterisation of the impact recording has on application performance. From the evaluation, we also learned a number of lessons on the use of PReServ and integration with applications. These lessons are summarised as recommendations.

From this evaluation, there are four key contributions:

1. Process documentation allows for provenance to be determined as demonstrated by answering the use case questions posed in Chapter 5.
2. The recording of process documentation is shown to be scalable both in controlled and real world experiments.
3. Recording process documentation has an acceptable overhead.
4. A set of recommendations for developers when creating and deploying provenance-aware applications.

The rest of this chapter is organised as follows. First, an implementation of the provenance store is detailed. Next, the environment in which the experiments were performed is described. Then, a series of performance tests that provide figures for p-assertion record time, throughput, and contention are presented. Then, an investigation of the overhead of p-assertion recording for ACE is described. After which, we demonstrate that the use case questions can be concretely answered. Finally, we discuss recommendations for the deployment of provenance stores and the integration of p-assertion recording with applications.

6.1 Implementation

In this section, we discuss the design and implementation of PReServ. First, we enumerate the non-functional requirements that are addressed by PReServ. Second, we describe the software design. Lastly, a justification and presentation of the technologies used in the implementation is given.

6.1.1 Non-functional requirements

PReServ is designed to address several non-functional requirements. Those we consider are enumerated below:

1. *Scalability:* The provenance store should be scalable in terms of data size and number of connections. One of the consequences of creating and recording process documentation is that a large amount of data must be stored. Therefore, any implementation needs to cope with large data sizes. Furthermore, the provenance store should degrade gracefully as more clients record p-assertions, which ensures that even if performance suffers, p-assertion recording still continues.

2. *Client Independence:* The provenance store should not prefer any client technology or implementation. The goal is to encourage the development of clients for a range of runtime environments. For example, the provenance store should be agnostic as to whether the client was implemented in C++, Java or Awk. This requirement impacts the choice of the underlying technology used for the instantiation of PReP.

3. *Ease of Installation:* The provenance store should be easy to install to facilitate adoption and integration into applications. The quicker that a provenance store can be installed, the more likely application developers and scientists will test the implementation and see if provenance concepts and the architecture can be used with their systems.

4. *Feature Integration:* The implementation should allow for the simplified integration of new features. The implementation should facilitate the testing of new functionality and research ideas without having to reengineer or recode large portions of the software.

These non-functional requirements influenced both the design and underlying technologies used in the implementation of PReServ.

6.1.2 Design

We now discuss the design of PReServ. PReServ consists of two components: a Provenance Service and a Provenance Store Client (PSC).

6.1.2.1 The Provenance Service

The Provenance Service is a wrapper for multiple provenance stores, which allows one host to support multiple institutions or applications each with their own provenance stores. The component diagram of the Provenance Service is shown in Figure 6.1. Input messages are received by the *Communication Mediator*, which is responsible for converting the communication medium's format into the internal format of the Provenance Service. The Communication Mediator is also responsible for extracting the operation specified in the input message and routing the converted message to the correct Provenance Store as well as the appropriate internal component to handle the message type. The benefit of a specific component for dealing with the communication medium is that it allows the Provenance Service to be ported to different communication mediums without effecting any other component. Furthermore, the Mediator could be implemented to support multiple communication mediums, which supports the non-functional requirement of Client Independence.

Depending on the operation specified, the input message will be sent either to an appropriate *Provenance Store* or to the *Provenance Store Manager*. The Communication Mediator uses the Provenance Store Manager to lookup the Provenance Store. If the message is intended to create, delete or manage Provenance Stores, the Manager handles the message directly. The Provenance Store Manager, supports the Ease of Installation requirement through the dynamic creation of Provenance Stores via a client interface (for example, a web page) without having to install software for each store.

Each Provenance Store has a *Dispatcher* that takes an input message and routes it to an appropriate *PlugIn* that implements the operation specified by the Communication Mediator. Each PlugIn registers the operation it supports with the Dispatcher. The Feature Integration requirement is supported by allowing multiple PlugIns. If a new feature needs to be added, a new PlugIn can be created for it.

Depending on the operation, the Dispatcher may choose to cache a message and at a later point, for example when the Provenance Store is less busy, the Dispatcher then retrieves the cached message and provides it to a PlugIn for processing. For example, when recording p-assertions there is processing overhead to create indexes for searching as well as data consistency checks. By delaying processing, greater throughput can be achieved. This caching mechanism supports the requirement for connection scalability by allowing a large number of messages to be accepted without the overhead of message processing. An important difference between the Provenance Service's approach to caching and other data stores is that it is done on a per operation basis and not for all operations. Thus, depending on the operation, caching can be enabled and disabled and thus a spectrum of qualities of service can be catered for.

A Provenance Store can be seen as a *thin layer* of provenance-specific operations over a data store (i.e. a relational database, an object database, a file system, etc.). To allow for various types of data stores, the Provenance Service provides a storage abstraction layer in the *Storage System* component. The Dispatcher, Provenance Store Manager, and PlugIns access data stores through the various provenance specific storage and retrieval functions provided by the Storage System. Different types of data stores are fitted into the Storage System through a *Backend Connector*, which maps a set of common storage functions (a Service Provider Interface) to the data stores native functions. For example, in the case of a relational database, the Backend Connector would convert a particular retrieval function to an SQL statement. The Storage System allows for storage and retrieval functions that may or may not be supported by the underlying data store. The Storage System abstraction allows for a choice of data stores that have different capabilities. For instance, a store may offer support for large data sizes or automatic inference. Multiple types of backend data stores helps fulfil the Feature Integration requirement, because new stores with different feature sets can be added. Also, the Storage System may implement features not directly supported by the underlying data store. This means that new provenance specific storage functions can be added without changing data stores and without impacting previously stored process documentation.

The design of the Provenance Service matches the non-functional requirements enumerated in Section 6.1.1.

6.1.2.2 The Provenance Store Client

The PSC provides an Application Programming Interface (API) to application developers to create and record p-assertions as well as query a Provenance Store. We focus on the creation and recording functionality. The PSC provides a set of classes that reflect the p-structure and PReP's messages. Applications then instantiate the appropriate classes for the p-assertions they need to create. The classes guarantee that the created p-assertions will be syntactically compatible with the p-structure. The application then

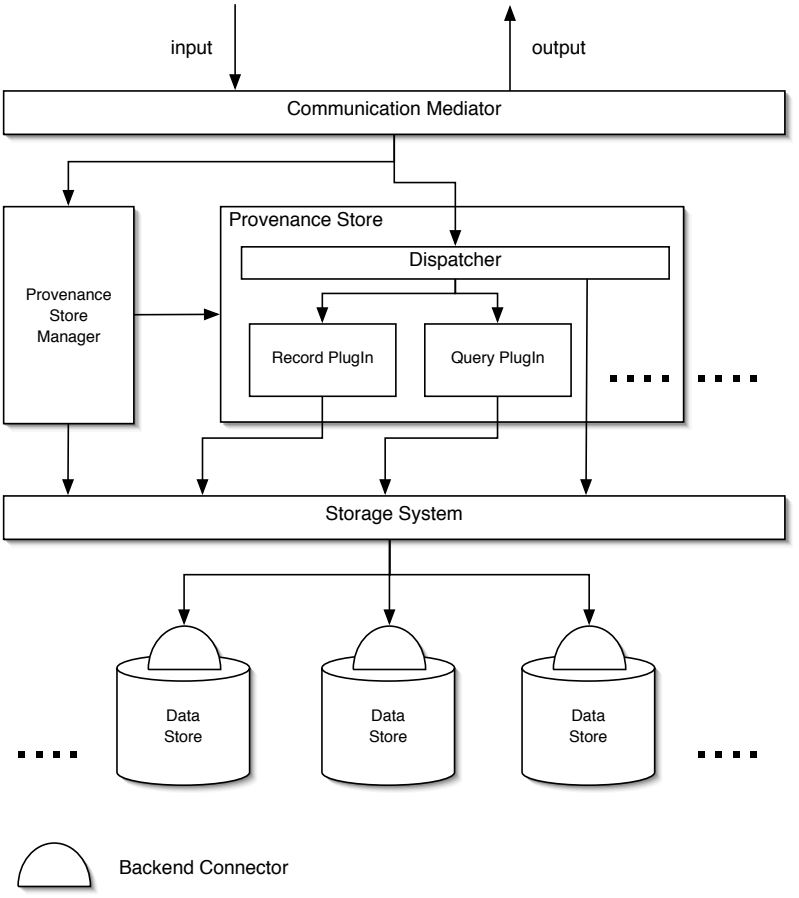


FIGURE 6.1: The Provenance Service Architecture

puts the p-assertions into a corresponding record message object, which can then be sent to the Provenance Store through the record function available in the API. A variant of the record function takes a list of record message objects and sends these in bulk to the Provenance Service. Applications can use bulk submission to delay recording until the network is less congested, to wait until the application is finished processing or to save on the overhead of establishing a network connection

The PSC also enables the concurrent recording of p-assertions while an application continues to execute. The API provides a *PSProxy* component, that acts as proxy to a particular Provenance Store. Instead of generating record message objects, an application enqueues its p-assertions into the PSProxy, which has a thread that handles the recording of the p-assertions. Therefore, once a p-assertion has been created and enqueued the application can continue execution without waiting for a response from the Provenance Store.

The PSProxy also provides several functions for the simplified generation of p-assertions. We list these functions below.

- Automatic generation of local p-assertions ids.
- Generation of interaction keys.
- Tracking of previously recorded p-assertions for the creation of relationship p-assertions.
- Generation of relationship p-assertions from p-assertion references.
- Functions for creating interaction and internal information p-assertions from string content.

The PSC provides a foundation for the integration of creation and recording into applications.

6.1.3 Technologies Used by PReServ

We now discuss the technologies used by PReServ and the rationale behind their selection. Web Services are one of the main ways of implementing SOAs. Furthermore, Web Services and its underlying document format, eXtended Markup Language (XML), are the focus of a number of standardisation efforts for SOAs. To integrate with this fabric of standards, we chose to instantiate PReP using these technologies. This choice supports Client Independence. XML is a platform independent textual format with a typing system, XML Schema. Furthermore, most platforms support the Hypertext Transport Protocol (HTTP), which is the common transport layer used by Web Services. For security, encrypted communication over secure HTTP (HTTPS) is also available.

PReServ is implemented in the Java programming language. Java allows for the software to run on multiple platforms without recompilation. PReServ has been successfully run on Mac OS X, Windows XP, and multiple versions of Linux both on the x86 and PowerPC architectures. The selection of Java supports the Ease of Installation requirement because the user does not need to compile the code for their specific platform and PReServ can be used on any platform that has a Java Virtual Machine. This selection also supports Feature Integration through the addition of classes with the Java Archive (JAR) mechanism. A developer can add a set of features by adding a JAR file to PReServ. Therefore, PReServ can be distributed with different feature sets depending on the included JAR files. For instance, different types of data stores can be supported by adding a JAR file that contains the appropriate Backend Connector.

Currently, PReServ integrates with three types of data stores: a Java specific embedded database (Oracle's Berkeley DB Java Edition), an in-memory data store, and a relational database (mySQL) [104].

The tests here were conducted with the Berkeley DB Java Edition database (BDB). The selection was made for a number of reasons. First, the in-memory store was not a

viable option because it could not handle the data sizes we consider nor could it persist data. Second, because of its embedded nature, BDB must not be installed as a separate service or application. The only requirement is that BDB must be provided with a directory where it can write its files to. This reduces the complexity of installation thus supporting the Ease of Installation requirement. Furthermore, BDB is an append-only database and thus is optimised for write performance. This work focuses on recording performance and thus BDB is a good fit. Finally, the BDB Backend Connector was the most mature of the three available. The BDB Connector has several optimisations designed to improve performance and scalability as enumerated below.

1. Updates are kept to a minimum when recording a p-assertion.
2. The keys used to identify a p-assertion are translated from their external XML format to an internal format that minimises the number of bytes used.
3. Data is automatically compressed by the BDB Connector using GZIP [58] before storage to disk, which increases the amount of data that can be stored. By its nature, XML repeats format elements and thus compresses well [40].

The PReServ Service is implemented as a Java Servlet [99] and deployed in the Apache Tomcat Servlet container [34]. The servlet supports both compressed and uncompressed communication using GZIP. Likewise, the Java PCS used in the experiments also supports both modes of communication. Therefore, applications using the Java PCS can make use of compression when recording large amounts of data; again, this supports the Scalability non-functional requirement.

Both the design and implementation of PReServ support the non-functional requirements enumerated in Section 6.1.1. We now evaluate PReServ.

6.2 Evaluation Environment

The experiments used in this evaluation were run on the Iridis Computing Cluster at the University of Southampton. Iridis contains several sets of nodes (i.e. computers). The set used in the experiments consisted of 237 nodes each with two AMD Opteron processors running at 2.2 GHz and 2 GB of RAM. A local storage of 25 GB is made available as scratch disk space for running jobs. Each node has access to a shared file system where results and executables can be stored. The Provenance Service runs on a node with 4 Dual Core AMD Opteron processors running at 2.4 Ghz and 2 GB of RAM. The database stores its files on a shared filesystem with 1.4 TB of space available for the storage of process documentation. All nodes are connected by Gigabit Ethernet.

Iridis is a job-based cluster. Shell scripts are written that call various executables, which are already available on the nodes or located on the shared file system. These shell scripts can then be submitted to the cluster as jobs and scheduled on nodes by a scheduler. Once a job controls a node no other job can use the node until the job finishes or is taken off the node by the scheduler because it has run beyond its requested time slot. On Iridis, jobs are submitted using PBS/Torque [1] or through a Globus [68] interface and scheduled using the Maui scheduler. Because the Iridis cluster is shared by the entirety of the University of Southampton's scientific community, there is a limit to the number of nodes that can be in simultaneous use by any one user's jobs. The policy on Iridis is that a user has a limit of 40 jobs running simultaneously. If enough resources are available, this limit may expand to around 60 jobs.

All applications used in the evaluation were written in Java and were run using the Java 1.5.0_05 64-bit Server Virtual Machine with HotSpot Just-In-Time compilation enabled. The Provenance Store in all experiments had caching enabled as this is the default deployment option for PReServ and the option that is designed for recording performance. We now evaluate provenance store performance in the Iridis environment.

6.3 Provenance Store Performance

The objective of this section is to characterise the performance of the provenance store from the perspective of a client in a controlled environment. First, we show how both data size and number of clients affect performance. We begin with data size.

6.3.1 Storage Size Impact

Figures 6.2(a) and 6.2(b) show the impact of store size on p-assertion record time for different recording scenarios and different payload sizes. The tests were run with one client recording internal information p-assertions into a provenance store. The provenance store's caching mechanism was disabled. Each test recorded 10000 p-assertions. The number of p-assertions is indicative of the amount of data held within a store, thus, it is used as the measure of store size on the x-axis of each graph. The y-axis of each graph shows the time it took to record the p-assertions measured in milliseconds. Measurements were taken after recording 100 p-assertions. The graphs display an average from ten trials.

Two p-assertion payload sizes, 10K and 100K, were considered. For each payload size, tests were performed using the following scenarios.

- (M Scenario) P-assertions are recorded without compression on communication channels and with a new interaction record created for each p-assertion (i.e each

p-assertion is recorded as if it is a part of a new interaction).

- (O Scenario) P-assertions are recorded without compression on communication channels and all p-assertions are recorded in the same interaction record (i.e. each p-assertion is part of the same view).
- (M & C Scenario) P-assertions are recorded with compression on communication channels and with a new interaction record created for each p-assertion.
- (O & C Scenario) P-assertions are recorded with compression on communication channels and all p-assertions are recorded in the same interaction record.

The M and O Scenarios are the two extremes when recording p-assertions. Most applications would have a combination of creating new interaction records and updating previously created interaction records. When a p-assertion is recorded in a new interaction record the Backend Connector must also record a new asserter identity in the database slightly hindering performance compared to the creation of a new interaction record.

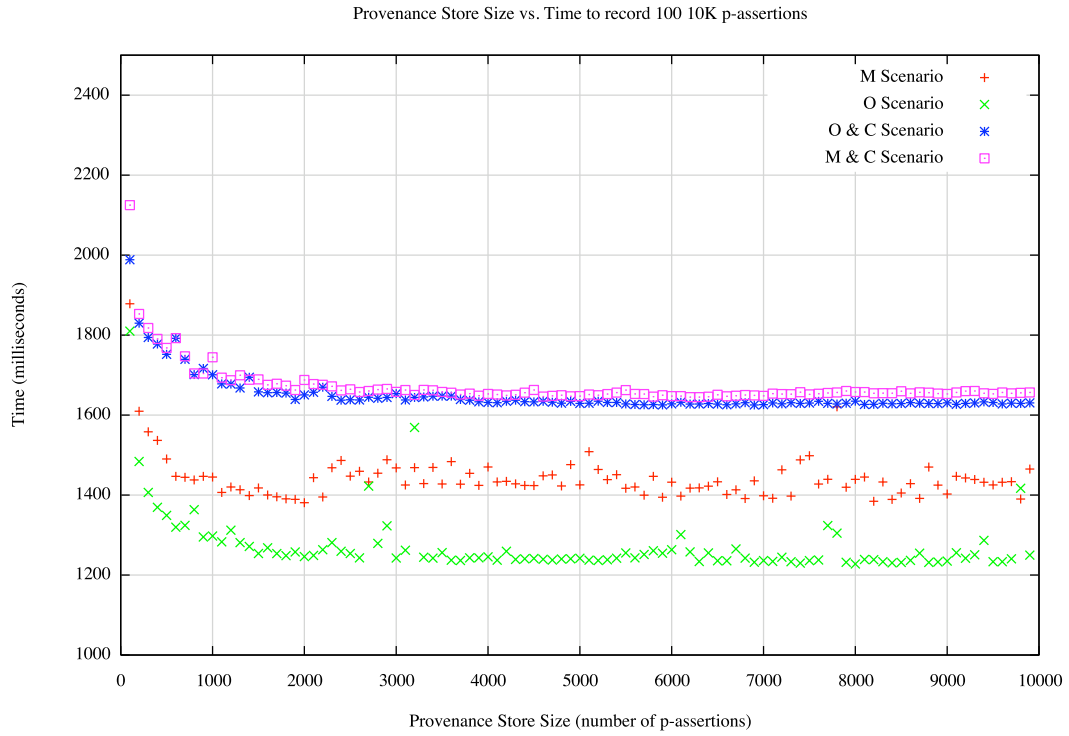
To give some more intuition as to the performance of PReServ, the following table gives the time to record one p-assertion with 10K and 100K payloads for the four different scenarios. These times take into account the variance across store sizes. Given the speed of the network, the majority of the time taken in these tests is the result of client and service processing.

Scenario	10K p-assertion		100K p-assertion	
M	14.4 ms	(±3.2 ms 95% confidence)	34.3	(±7.7 ms 95% confidence)
O	12.7 ms	(±2.9 ms 95% confidence)	31.7	(±7.1 ms 95% confidence)
M & C	16.7 ms	(±3.8 ms 95% confidence)	126.9	(±28.6 ms 95% confidence)
O & C	16.5 ms	(±3.7 ms 95% confidence)	126.7	(±28.5 ms 95% confidence)

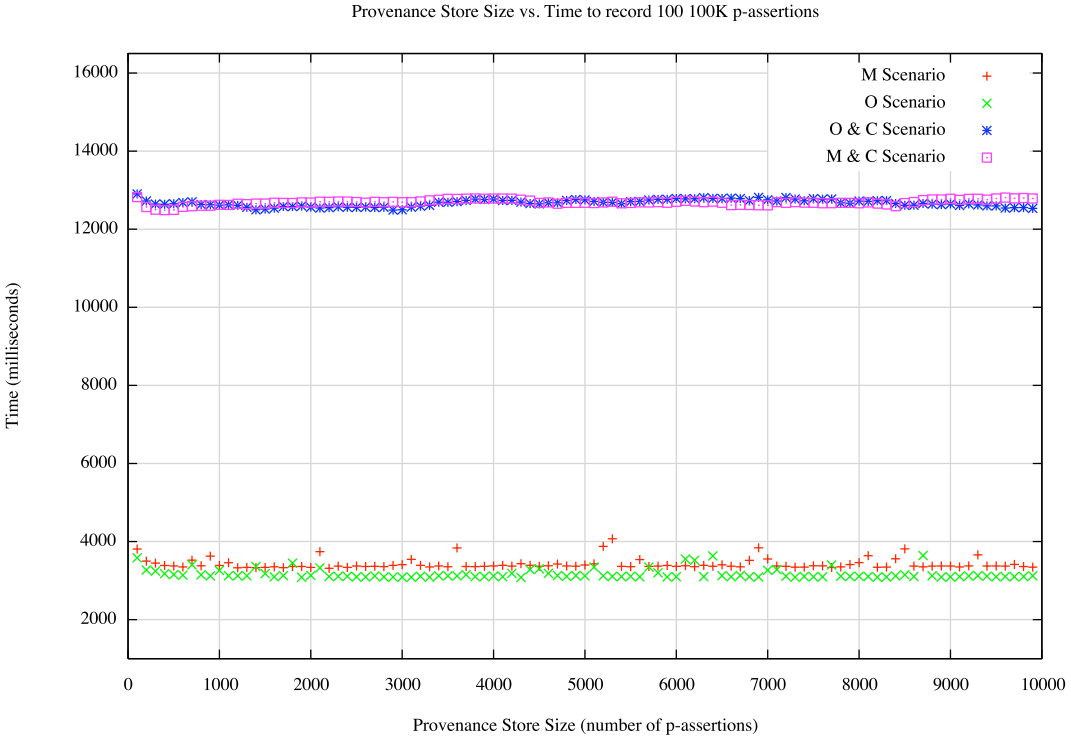
Overall, the effect of store size on recording times is flat. The curve at the beginning of each graph is explained by Java's just-in-time compilation. Compression has a clear impact on record times. In an environment with a slower network, compression may have resulted in better results. However, due to the speed of the test network, the time to compress and decompress a p-assertion adds significantly to p-assertion record time. The increase in compression record times between the 10K and 100K case is explained by the computational complexity of the Deflate algorithm [58] GZIP relies upon.

6.3.2 Multiple Client Connections Impact

The impact on provenance store performance as the number of client connections increases is now presented. In each experiment the number of nodes used in parallel is increased by powers of 2 up to a maximum of 32 nodes. Given the Iridis environment,



(a) Time to record 100, 10 kilobyte p-assertions as the Provenance Store size increases with compression enabled and disabled



(b) Time to record 100, 100 kilobyte p-assertions as the Provenance Store size increases with compression enabled and disabled

FIGURE 6.2: Provenance store size impact on p-assertion record times.

experiments using more than 32 nodes could not be performed. An MPI based test harness was used in the experiments to guarantee that all jobs were run in parallel. Each client in the experiments used the M Scenario with a 10K payload.

Figure 6.3 shows the impact of contention for the provenance store on record time for a single client. The figure was computed by taking the average record time from all participating clients. The graph shows that as the number of clients accessing a single provenance store increases, the time it takes for a single client to receive a response also increases.

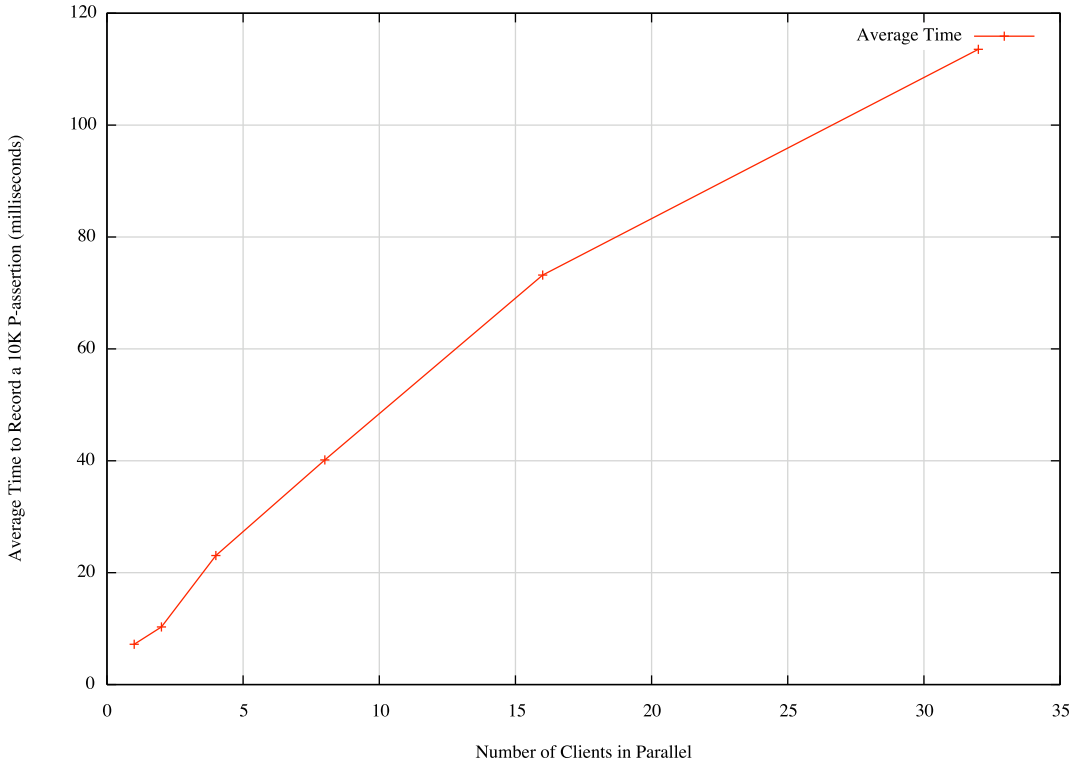


FIGURE 6.3: Contention

However, an increase in the number of clients improves the overall throughput of the provenance store (i.e. how many p-assertions can be recorded in a period of time). We come back to this trade-off between total throughput and record time performance later in the chapter.

In Figure 6.4, we see that as the number of clients increases the throughput also increases. Typically, a systems throughput will increase until a maxima and then throughput levels off and may slightly decrease. We attempted to find such a maxima for the provenance store by increasing the number of threads per node. However, throughput continued to increase up to the point where 32 nodes each had 16 threads recording p-assertions at the same time. At this point, 234025 p-assertions were recorded in a 10 minute period, which means that on average a 10K p-assertion was recorded every 2.6 milliseconds. Figure 6.5 provides a colour map representation of throughput as the number of nodes

and threads per nodes increases. The figure clearly shows an upward trend. Warmer colours represent higher throughput. In black and white, the lighter the shade the higher the throughput.

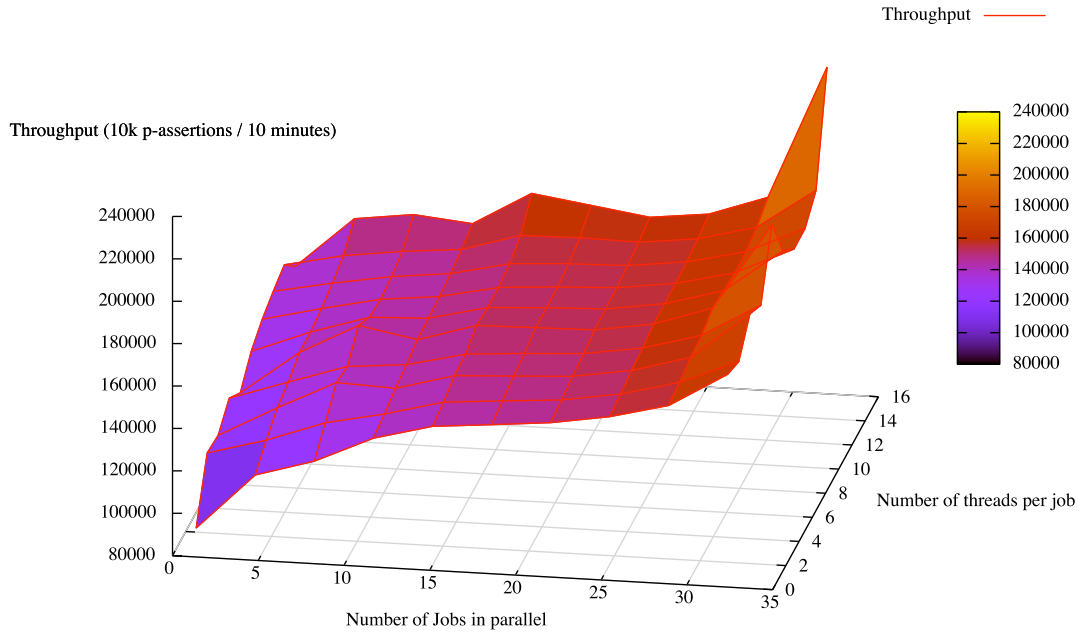


FIGURE 6.4: Throughput as the number of jobs and threads per jobs increases

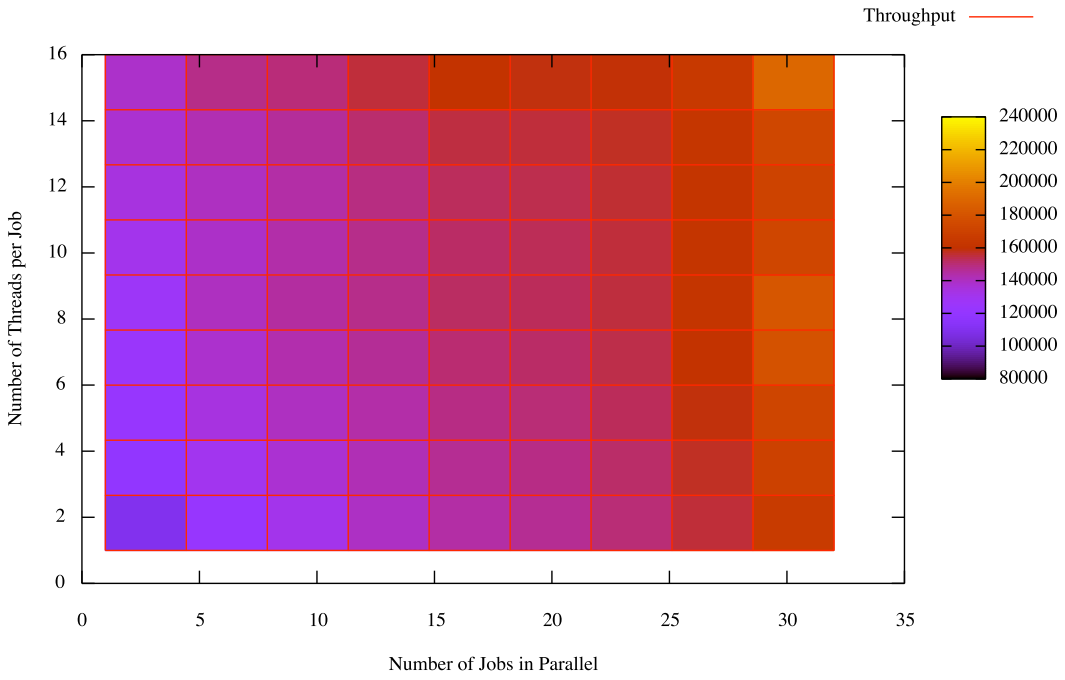


FIGURE 6.5: Colour map of throughput

These performance figures show that the provenance store is scalable with respect to data size and client connections. More importantly, this information can be used by applica-

tion developers to ascertain the impact p-assertion recording will have when integrated with their application. They can also use the figures to determine the deployment strategy that bests suits their performance requirements. We come back to this trade-off between total throughput and record time performance later in the chapter.

6.4 Case Study Performance

Having looked at the performance of the Provenance Store in a controlled environment, we now investigate the the impact of p-assertion recording on the example application ACE. Recall from Chapter 5 that ACE is a combination of multiple distributed components running across multiple institutions. Our implementation of ACE was designed to run in a Grid environment. Figure 6.6 shows the deployment workflow for the implementation of the ACE application. The different organisations involved are identified by grey boxes with dotted lines. Sequences are obtained from an external provider through either a Web Service or via FTP. These are then collated locally (i.e. on the bioinformatician's computer) into several sample sequences. The Jobs Creator then generates a series of jobs to be submitted to a Grid and executed. The executables used by the jobs are pre-staged on the Grid (i.e. they are already available on the Iridis cluster).

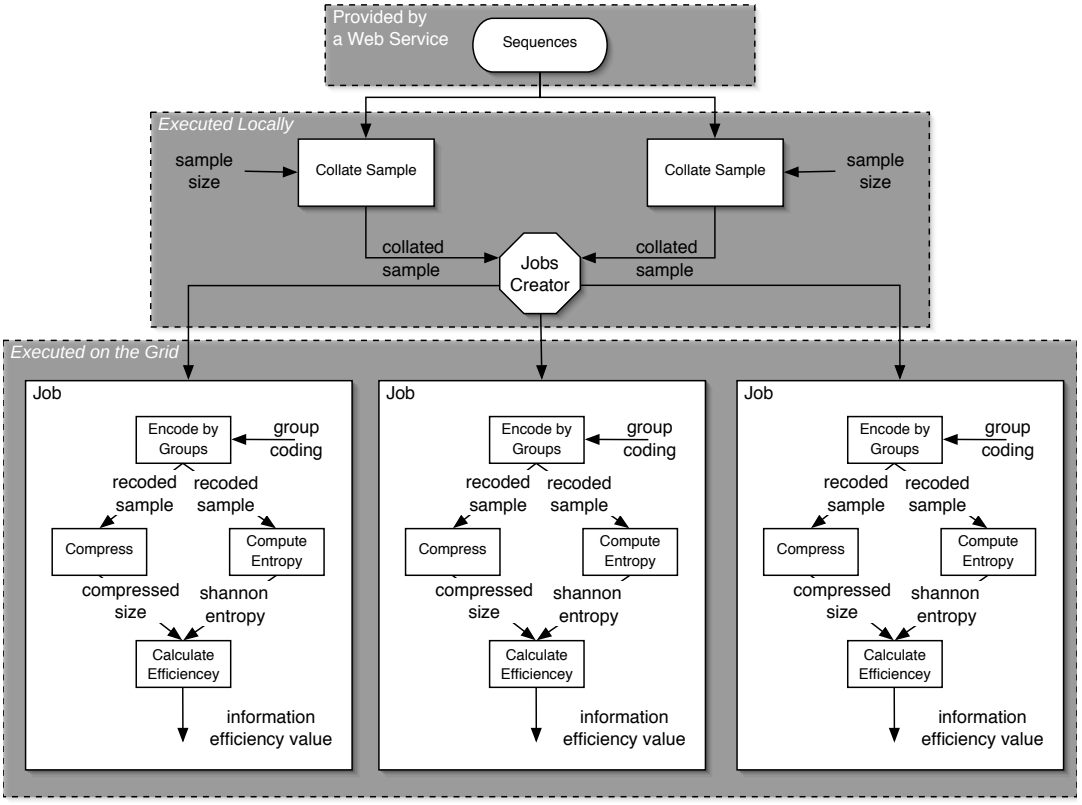


FIGURE 6.6: ACE deployment workflow

Our implementation generates Condor [181] compatible job descriptions, which are then

submitted to Iridis via a Globus interface using Condor-G [76]. Globus provides an abstraction over a wider variety of schedulers and jobs submission interfaces. Therefore, the jobs created by the Jobs Creator can run on any Globus enabled Grid that can run Java executables. This portability is important because it allows ACE to be run wherever computational resources can be procured. For example, ACE could be run on the United Kingdom's National Grid Service (http://www.grid-support.ac.uk/) or the Open Science Grid (http://www.opensciencegrid.org/). In fact, the certificates used by our implementation to access Iridis using Globus are issued by the National Grid Service.

The collate sample portion of the ACE workflow is typically run once and a number of jobs are generated to process these samples with different groups. In the experimental setup for these performance measurements, one run of ACE consists of 80 jobs. Each job analysed 900 unique groups on 5 different 100K collated samples, thus, a job generates 4500 information efficiency values. A set of 900 groups is a 50K file. Process documentation that represents the provenance of each information efficiency value is stored across two provenance stores. One store is deployed on the Grid infrastructure, the other is deployed on the same network where the local portion of ACE executes. The provenance store hosted locally contains documentation of the generation of the 80 jobs, which is 5 MB in size on disk.

The process documentation created by ACE is extremely detailed; the steps used to compute each information efficiency value are recorded. To prevent duplication of data and the creation of a larger than necessary provenance store, we make use of documentation styles to enable references to input data. Therefore, a collated sample is only documented once and not for every information efficiency value computation that takes it as input. Furthermore, intermediate data is not stored in the provenance store if it can be generated by a well-known and documented algorithm. For example, the output of the PPMZ compression algorithm is not stored because it can be regenerated accurately. After processing one run of ACE, the provenance store deployed on the Grid contains 14GB of data. In Section 6.6, we discuss the trade-offs between process documentation detail and performance. For ACE, our choice of documentation detail is a good compromise as it allows all our use case questions to be effectively answered while still achieving acceptable performance. We now calculate the effect of p-assertion recording on ACE. To provide an average, the data used was collected from three runs of ACE where the same jobs were submitted for each run.

The most pertinent measure of application performance for the scientist is the duration of an application run measured in wall clock time. Therefore, we calculate the slowdown of ACE when recording p-assertions in terms of wall clock time. Because we run ACE in an uncontrolled environment, we now show that the average difference in duration between recording and non-recording jobs provides a reasonable approximation for the impact of p-assertion recording on application performance.

We start by noting that the time to collate samples is constant and is small when compared to the time necessary to run jobs. Discounting sample collation time, we can approximate the time to perform one run of the ACE application by summing the runtimes of all the application's jobs and dividing that by the average number of jobs run in parallel. When jobs actually get scheduled and run is beyond our control and is representive of the load on Iridis and not our experiment.

This approximation is reasonable because all jobs run on a standard computational environment and there are no dependencies between jobs. Furthermore, we observe that in ACE, parameter variation in terms of groups has very little effect on job duration, as we have 95% confidence that a job will last 22 minutes ± 30 seconds. Figure 6.7 shows that job times follow a normal distribution and the majority of job times fall within the stated confidence interval. Thus, from Figure 6.7 and our reasoning, we conclude that the average job runtime is representative of overall application runtime. Thus, a slowdown in job runtime is a good predictor of the slowdown in application runtime.

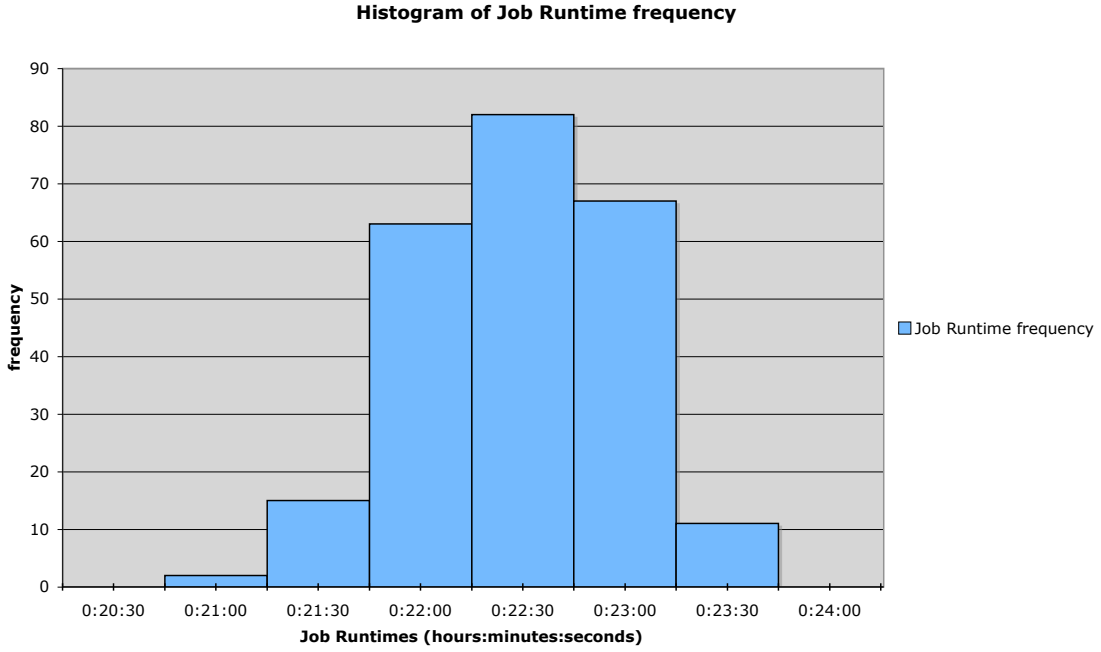


FIGURE 6.7: Frequency distribution of job times

We now consider the scenario in which jobs record p-assertions. In this scenario, the runtime is affected by two additional factors: the p-assertion creation time and the p-assertion record time. We note that the record time for p-assertions can be influenced by contention for the provenance store from other jobs as shown in Figure 6.3. However, the influence of contention is negligible because, although the distribution of job parallelism is broad ranging from 0 to 60 jobs in parallel (see Figure 6.8), a majority of p-assertion recording job times fall within two standard deviations of the average (i.e. either plus/minus a minute away from the average job run time). This clustering of job times is shown in Figure 6.9 and mirrors the result from Figure 6.7. Thus, contention is

not a factor influencing p-assertion record time within ACE.

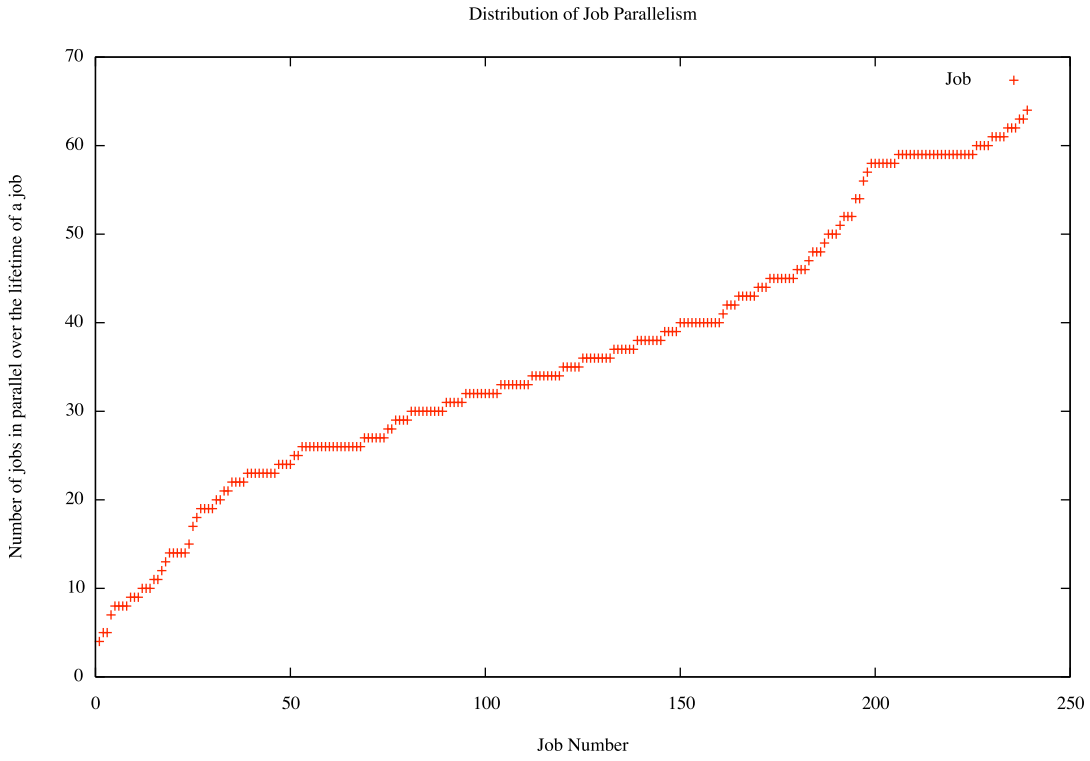


FIGURE 6.8: Distribution of job parallelism

Having discounted both contention as an influence on p-assertion record time and parameter variation as influence on job runtime, we conclude that the difference between the runtime of ACE and the runtime of ACE with p-assertion recording is the time it takes to create and record p-assertions. Figure 6.10 shows the maximum, minimum and average job record times over all application runs and the difference between times with and without p-assertion recording. Taking the average, there is a 13% overhead on job runtime for p-assertion recording. From our previous reasoning, we conclude that there is 13% overhead for recording on *application runtime*.

We believe this value is acceptable in light of the functionality gained from having an accurate representation of the process by which the results of ACE are produced. We now analyse how the process documentation recorded can be used to satisfy the provenance questions posed in Chapter 5.

6.5 Use Case Satisfaction

The process documentation recorded when running ACE was detailed enough to answer the six use case questions presented in Section 5.4. We now discuss the query algorithms used to answer the questions and give examples of the results produced. The common

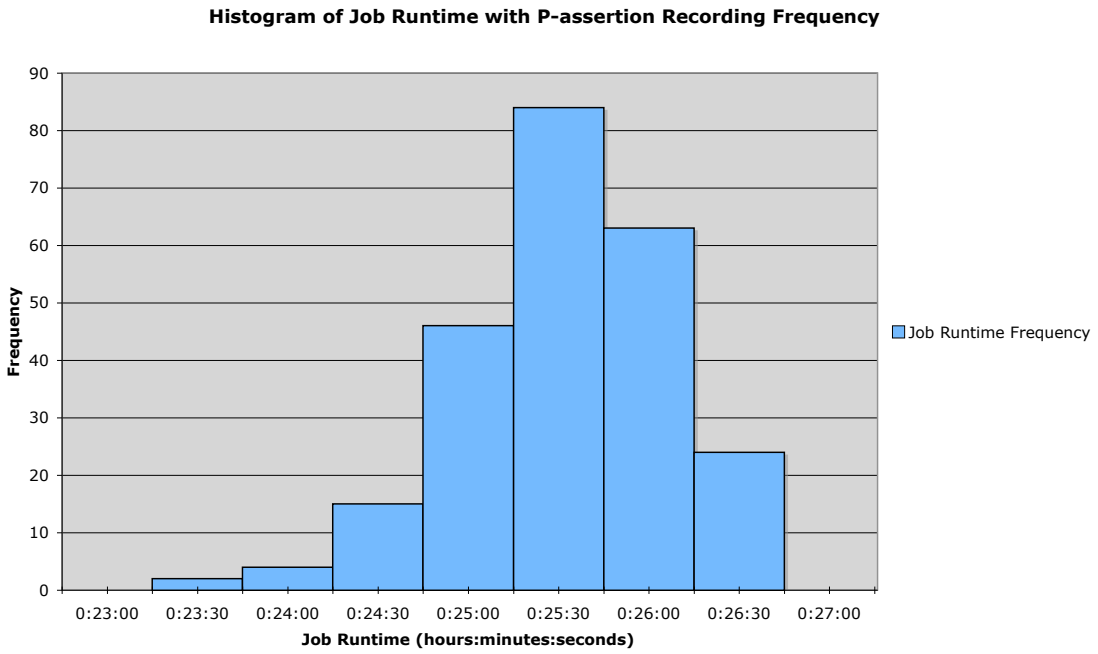


FIGURE 6.9: Frequency distribution of p-assertion recording job times

	Job Runtime	Job Runtime with Recording	Difference
Maximum	23:20	26:24	3:04
Average	22:24	25:17	2:53
Minimum	20:39	23:09	2:30

FIGURE 6.10: Maximum, Minimum and Average job record times both with and without p-assertion recording

thread that runs through all the algorithms is that they all relate to the provenance of the results produced by the ACE. Figure 6.11 shows a plot of the results produced by one run of the ACE. The x-axis shows the number of the group coding and the y-axis shows the information efficiency value. The bioinformatician generally wants information related to the provenance of the largest information efficiency values as they identify possible groups of interest.

The determination of the provenance of a particular information efficiency value is determined by using the provenance query functionality provided by PReServ. A simplified version of the provenance query algorithm [125] implemented by the provenance query functionality is shown as PROVENANCE(x)[1]. The algorithm determines the provenance of a data item x. In this case, x would be a particular information efficiency value. PROVENANCE(x) determines the relationship p-assertions describing the provenance of x and returns them as a set G.

[1]We adopt the pseudocode style from the second edition of the book Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein [46]

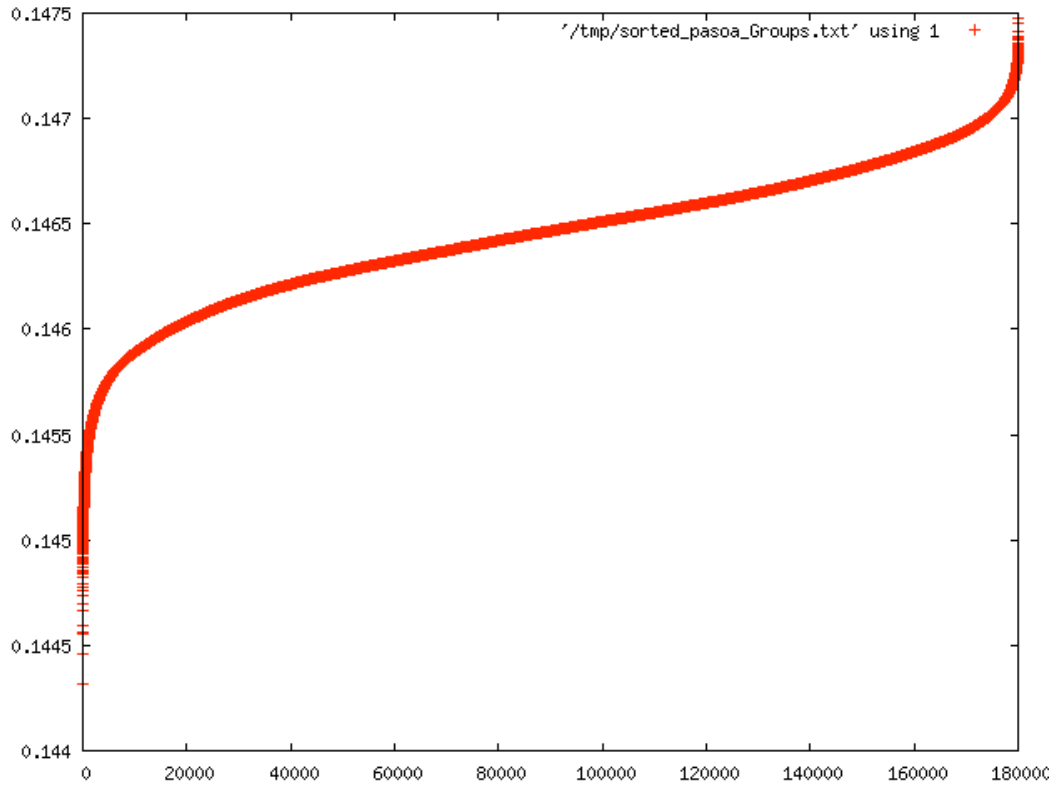


FIGURE 6.11: Graph of groupings sorted by their ACE information efficiency values

PROVENANCE(x)

```
1  G = ∅
2  find relationship p-assertions, R, where x is an effect
3  for each r ∈ R
4        do
5           G := G ∪ {r}
6           find causes, C, of r
7           for each c ∈ C
8                 do G := G ∪ PROVENANCE(c)
9  return G
```

Each of the algorithms used to answer the use case questions makes use of the PROVENANCE(x) algorithm. PReServ also offers process documentation query functionality. The p-structure is presented as an XML Document Object Model (DOM) whose hierarchy can be traversed using XQuery [22] and various portions of the structure can be extracted. Furthermore, a client can traverse the results returned by PReServ to extract particular information using XPath [43]. Results are provided as an XML DOM. To improve pseudocode readability, we introduce extra syntax that mirrors the functionality provided by PReServ's XQuery interface as well as the client's XPath functionality.

- **retrieve** This command describes the acquisition of an entity from the provenance store. This models querying PReServ through the XQuery interface.
- **extract** This command describes the extraction of an entity from another local entity. This models performing an XPath operation over a local XML DOM document.

When PROVENANCE(x) is shown in pseudocode it can be interpreted as being a call to the provenance store to perform a provenance query. We now proceed through each use case, in turn, presenting the query algorithm used and an example of the output produced.

6.5.1 Use Case 1

What were the sequences used in the production of a particular information efficiency value?

In this use case, the bioinformatician wants to find the original sequences that were collated together and then used to produce a given information efficiency value, x. The file paths of these sequences have been recorded as internal information p-assertions during the execution of ACE. There exists a structural relation between these sequences and the collated sequence, namely, the collated sequence is a concatenation of the various randomly selected original sequences. This structural relationship is documented as a relationship p-assertion where each sequence is a cause with a parameter name of "sequence path". Figure 6.12 depicts this relationship; the contents of two sequence files identified by their file paths have been collated together to form a larger sequence.

The relationship p-assertion in question is also part of the annotated causality graph that represents the provenance of x. Therefore, the original sequences used in the production of x can be found through the determination of the provenance of x as shown in line 1 of ORIGINALSEQUENCES(x).

```
ORIGINALSEQUENCES(x)
1  G := PROVENANCE(x)
2  extract causes, C, from G, where parameter name = "sequencePath"
3  for each c ∈ C
4       do extract gpak from c
5          retrieve p-assertion, α, with gpak
6          extract sequence, s, from α
7          print s
```

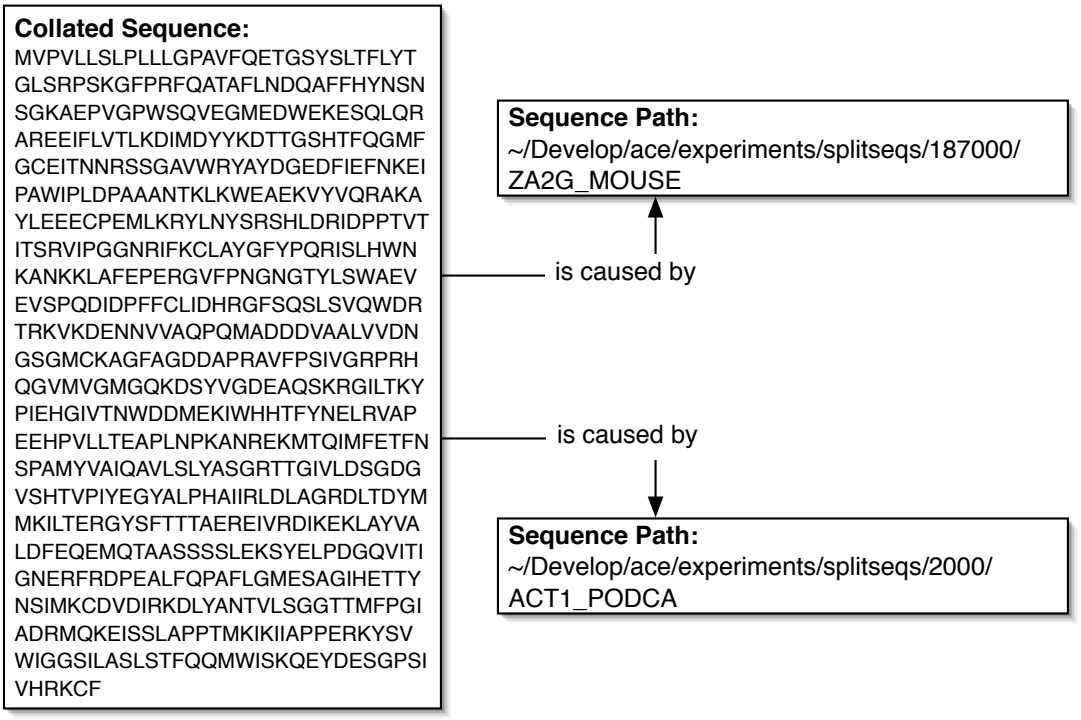


FIGURE 6.12: A collated sequence as the product of two sequences identified by their file paths

```
/Users/pgroth/Develop/ace/experiments/splitseqs/58000/H2B_TRYCR
/Users/pgroth/Develop/ace/experiments/splitseqs/104000/PEX2_HUMAN
/Users/pgroth/Develop/ace/experiments/splitseqs/89000/MURE_XYLFA
/Users/pgroth/Develop/ace/experiments/splitseqs/77000/LGRC_BREPA
/Users/pgroth/Develop/ace/experiments/splitseqs/88000/MTLD_STRPN
/Users/pgroth/Develop/ace/experiments/splitseqs/24000/COAT_MUMIM
/Users/pgroth/Develop/ace/experiments/splitseqs/95000/NRAM_IAHSO
/Users/pgroth/Develop/ace/experiments/splitseqs/180000/YCF3_PORPU
/Users/pgroth/Develop/ace/experiments/splitseqs/139000/SLYD_HELPY
/Users/pgroth/Develop/ace/experiments/splitseqs/63000/HISX_METCA
```

FIGURE 6.13: Example of the results produced for Use Case 1

After retrieving the provenance of x, the ORIGINALSEQUENCES(x) algorithm finds the global p-assertion key that identifies the internal information p-assertions containing the paths to the original sequences. The algorithm then prints the sequence paths to standard output. An example of the results produced is shown in Figure 6.13.

6.5.2 Use Case 2

What were the input figures used in the efficiency calculation that produced a particular information efficiency value?

In this use case, the bioinformatician wants to obtain values that were computed during

the calculation of an information efficiency value, x. These values can then be used as part of different experiments or used to recalculate the information efficiency value using a different algorithm. In the case of the ACE, the bioinformation wants to obtain the input values for Equation 5.2, which is the last step in the production of x.

For x, the precise input values can be found using the algorithm EfficiencyCalculationInputs(x), which first determines the causality graph, G, that represents the provenance of x. From this graph, it extracts the interaction key identifying the request made to the EfficiencyCalculation actor, which performs a programmatic version of Equation 5.2. Using this interaction key, the interaction p-assertion containing a representation of the specific request to the EfficiencyCalculation actor can be obtained. Once the interaction p-assertion has been retrieved, the input figures ($\mathcal{K}_{\mathcal{C}}(s)$, $\ell(s)$, $H(s)$) can be printed to standard output.

EfficiencyCalculationInputs(x)

1 $G :=$ Provenance(x)
2 **extract** the interaction key, ik, from G, where ik contains
  a messageSource = "EfficiencyComputer" and
  a messageSink = "EfficiencyCalculation"
3 **retrieve** the interaction p-assertion, α, from
  the sender view of the interaction record with ik
4 **extract** input figures, F, from α
5 **for** each $f \in F$
6 **do** print f

The correct input figures are obtained because of the causal connection between the input figures and x. An example set of results for this use case is shown in Figure 6.14.

```
compressed recoded sequence length: 4570.0 bytes
original recoded sequence length: 8704.0 bytes
recoded sequence entropy: 4.082372081011276
```

Figure 6.14: Example of the results produced for Use Case 2

6.5.3 Use Case 3

Were there any conflicting views of an interaction in the production of a particular information efficiency value?

In multi-institutional scientific systems, it is important to be able to detect disagreements about the data transmitted between parties. In this use case, the bioinformatician wants to see if there are any such disagreements in the production of an information

efficiency value, x. The algorithm ConflictingViews(x) determines if there were any interactions in which the two participating actors created documentation that differed. First, the algorithm retrieves the provenance of x from which it extracts the interaction keys for all the interactions involved in the production x. Using these interaction keys, the algorithm retrieves and compares corresponding interaction p-assertions. If the interaction p-assertions from two corresponding views (i.e. a sender and receiver view in the same interaction record) disagree, the interaction key identifying the interaction is printed to standard output.

ConflictingViews(x)

```
1  G := Provenance(x, G)
2  extract interaction keys, K, from G
3  for each ik ∈ K
4        do retrieve the interaction p-assertion, α_s, from
               the sender view of the interaction record with ik
5           retrieve the interaction p-assertion, α_r, from
               the receiver view of the interaction record with ik
6           extract p-assertion content, d_s, from α_s
7           extract p-assertion content, d_r, from α_r
8           if d_s ≠ d_r
9              then print ik
```

A set of example results produced by ConflictingViews(x) is shown in Figure 6.15. Two interactions are identified as having conflicting views. Using this information, we found that, in the case of the first interaction, the CondorJobsCreator actor fails to record provenance specific information it provides via an argument list to the Driver actor, thus, causing the conflict. In the case of the second interaction, the views differ because the actors use slightly different Java to XML serialisation processes when creating p-assertions. In this instance, the conflicting views are not of any concern, however, it does show that disagreements can be detected and subsequent action can be taken to find the source of conflict.

```
InteractionId: acc89d74-1925-4161-b975-b4eeadffc1f32
    Message Source:
        http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/CondorJobsCreator
    Message Sink:
        http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/Driver
InteractionId: c2fdea59-face-43e8-aba1-a8355c6affb01
    Message Source:
        http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/RandomSequenceBuilder
    Message Sink:
        http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/AceEnactor
```

Figure 6.15: Example of the results produced for Use Case 3

6.5.4 Use Case 4

Were references used in the process documentation created for a particular information efficiency value?

One mechanism to reduce the size of process documentation in a provenance store is to use references to data items instead of recording actual data items themselves. However, as we will later discuss, the data store that is referred to must be as permanent as the provenance store itself. Furthermore, this data store must guarantee that the data items referred to will not change. Because of these issues, it is important to be able to determine whether references were used when creating documentation for the process that led to a result. In this use case, the result in question is, again, an information efficiency value, x.

```
ReferencesUsed(x)
1  D := ∅                                  ▷ Set of Documentation Style URLs
2  G := Provenance(x)
3  extract global p-assertion keys, I, from G
4  for each gpak ∈ I
5        do retrieve p-assertion, α, identified by gpak
6           extract documentation style, d, from α
7           D := D ∪ {d}
8  for each d ∈ D
9        do print d
```

The ReferencesUsed(x) algorithm determines whether references were used by retrieving the documentation styles present in the process documentation pertaining to x. As with the previous algorithms, the provenance of x is determined. All of the global p-assertion keys in relationship p-asssertions returned by Provenance(x) are extracted. Using these keys, the documentation style for each identified p-assertion can be obtained from the provenance store. Instances of the same documentation style are collapsed. The URL of each documentation style is then printed to standard output. An example of the output is given in Figure 6.16

```
http://www.pasoa.org/docstyle/AceVerbatim
http://www.pasoa.org/docstyle/AceSequenceRef
```

Figure 6.16: Example of the results produced for Use Case 4

Documentation styles explicitly define how process documentation was created. ACE uses the http://www.pasoa.org/docstyle/AceSequenceRef documentation style to identify where sequences have been replaced by references to those sequences. The output contains this URL, therefore, references have been explicitly used in the documentation

of the process that led to the information efficiency value used as input. Such an explicit definition is preferable to an approach where the use of references must be inferred from the underlying representation. For example, one may infer the use of references by the presence of URLs in the documentation of an interaction, however, those URLs may have been the actual data transmitted between actors.

6.5.5 Use Case 5

What interactions are common for all information efficiency values produced by a particular job?

Often scientists want to know what is in common between different experiments. This use case is an example of such a situation. The bioinformatician needs to determine what steps were in common for the information efficiency values produced by the same job. The bioinformatician provides an information efficiency value, x from the job under consideration to the CommonInteractions(x) algorithm. In ACE, tracers are used to demarcate both a particular job and a particular experimental run. The algorithm makes use of these tracers to find the set of information efficiency values produced by a job. It proceeds as follows.

```
CommonInteractions(x)
 1  T := ∅                                          ▷ Set of Tracers
 2  S := ∅                                ▷ Set of common interaction keys
 3  G := Provenance(x)
 4  retrieve tracers, T, where the tracers are in p-assertions identified in G
 5  retrieve efficiency values, E, where
        the parent interaction p-assertion contains T
 6  for each e ∈ E
 7        do G_e := Provenance(e)
 8           extract interaction keys, K, from G_e
 9           for each k ∈ K
10                do if k ∈ S
11                      then tally(k) := tally(k) + 1
12                      else  S := S ∪ {k}
13                            tally(k) := 1
14  for each k ∈ S
15        do if tally(k) = |E|
16              then print k
```

First, the algorithm retrieves the tracers used in the production of information efficiency value x. Then all the information efficiency values are retrieved where both the value

```
InteractionId: c2fdea59-face-43e8-aba1-a8355c6affb04
  Message Source:
    http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/AceEnactor
  Message Sink:
    http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/CondorJobsCreator
InteractionId: c2fdea59-face-43e8-aba1-a8355c6affb03
  Message Source:
    http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/GroupFileDivider
  Message Sink:
    http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/AceEnactor
InteractionId: acc89d74-1925-4161-b975-b4eeadffc1f32
  Message Source:
     http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/CondorJobsCreator
  Message Sink:
     http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/Driver
InteractionId: c2fdea59-face-43e8-aba1-a8355c6affb01
  Message Source:
    http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/RandomSequenceBuilder
  Message Sink:
    http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/AceEnactor
InteractionId: c2fdea59-face-43e8-aba1-a8355c6affb02
  Message Source:
    http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/AceEnactor
  Message Sink:
    http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/GroupFileDivider
```

FIGURE 6.17: Example of the results produced for Use Case 5

and the tracers are contained within the same interaction p-assertion. Here, we make use of the fact that ACE transmits tracers along with the contents of messages and thus are documented in the same p-assertion. Once the information efficiency values produced by a job have been retrieved, the provenance of each value is then found. From the returned causality graphs, interaction keys are extracted. A tally is kept of whether an interaction key occurs for an information efficiency value. If an interaction key occurs in the causality graphs of every value, then the interaction key is printed to standard output. Figure 6.17 shows an example set of results. It lists formatted interaction keys that specify the interactions that were in common for the all the information efficiency produced by the job that also produced the information efficiency value 0.12861295968697928. These results are in line with what is expected given the ACE workflow.

6.5.6 Use Case 6

How long does it take to produce an information efficiency value from a particular collated sequence?

In this use case, the bioinformatician wants to determine the time it takes to calculate information efficiency for a particular collated sequence. The start time of an efficiency calculation is recorded as an internal information p-assertion in the context of the request

interaction from the Driver actor to the EfficiencyComputer. In the response interaction, the corresponding end time is recorded, again, as an internal information p-assertion. The ProcessingDuration(s) algorithm makes use the notion that the request and response interactions are causally connected in order to associate the start and end times. The algorithm takes a sequence, s as input. It then finds the interaction keys of the interaction records where s was documented. After this step, the provenance of each information efficiency value is found. If the returned causality graph contains any of the previously found interaction keys, then the start time and end time as well as the grouping are retrieved. The grouping along with the difference between start and end time are printed to standard output. An example set of results is shown in Figure 6.18.

ProcessingDuration(s)

```
1  retrieve interaction keys, K, where
       the interaction records contain s
2  retrieve efficiency values, E
3  for each e ∈ E
4        do G := Provenance(e)
5           for each k ∈ K
6                 do
7                    if G contains k
8                       then retrieve the internal information p-assertion, α_s, where
                                α_s is in the sender view
                                of the interaction record identified by k
9                            extract start time, st, from α_s
10                           extract the interaction key, ik_e, from G where
                                message source = "EfficiencyComputer" and
                                message sink = "Driver"
11                           retrieve the internal information p-assertion, α_e, where
                                α_e is in the receiver view
                                of the interaction record identified by ik_e
12                           extract end time, et, from α_e
13                           retrieve the interaction p-assertion, pa_g, where
                                pa_g is in the the sender view
                                of the interaction record identified by k
14                           extract grouping, grp, from pa_g
15                           print grp
16                           print (et − st)
```

This use case shows that process documentation can be created at multiple levels of abstraction. Furthermore, we can connect, through a causal chain, documentation created at different times and by different components within a job.

```
for group: b:A,c:R,d:N,e:D,f:C,g:Q,h:E,i:G,j:H,
           k:I,l:L,m:K,n:M,o:F,p:P,q:S,r:T,s:W,t:Y,u:V
  duration is 804
for group: b:A,c:R,d:N,e:D,f:C,g:Q,h:E,i:G,j:H,
           k:I,l:L,m:K,n:M,o:F,p:P,q:S,r:T,s:WY,t:V
  duration is 547
```

FIGURE 6.18: Example of the results produced for Use Case 6

In this section, we have shown how the use cases presented in Chapter 5 can be solved, practically, through provenance queries performed over recorded process documentation. In the next section, we analyse the results of our evaluation and make a number of recommendations for provenance store deployment and the creation of provenance-aware applications.

6.6 Analysis

Section 6.5 provides clear evidence that it was worthwhile to record process documentation for the ACE. Similar questions to those answered in Section 6.5 arise in a wide variety of applications [126]. Thus, our approach to solving the provenance problem should be considered for such applications. When considering a process documentation based solution, developers face two major decisions:

1. How detailed should the process documentation created by the application be?
2. What is the provenance store infrastructure required to provide the quality of service expected for an application?

The answers to these questions are clearly application dependent. However, based on our evaluation, we offer a series of recommendations for developers to use when making their applications provenance-aware. These recommendations are presented in terms of trade-offs. A developer can then decide what costs to bare for what benefits.

6.6.1 More detail vs. more time

Detailed process documentation is useful because it allows more questions to be asked about the execution of an application. Furthermore, queries can be answered in a more comprehensive fashion while allowing users to drill down to find out what exactly happened during an application run. With enough detail, process documentation could be used to rerun a process without the original programs or executables. However, the more detailed process documentation is, the greater the impact recording it will have on application performance. Therefore, developers face a trade-off between recording

detailed process documentation and overall application performance. From our evaluation, one way to determine how detailed process documentation should be is to define a set of example use cases for the process documentation created by an application. The application should, at a minimum, record process documentation at a level of detail great enough to answer these questions.

Recommendation: *Design the application so that it records enough process documentation to answer a set of example use cases.*

This recommendation is part of a methodology for designing provenance-aware applications specified in Munroe et al. [139]. It is, however, beyond the scope of this dissertation to define a such a methodology specifying what needs to be recorded by applications.

6.6.2 Confidence and longevity vs. space and time

A provenance store has a number of attributes to enable the data it maintains to be held over a long period of time and with confidence by its users. First, a provenance store follows a Write-Once-Read-Many (WORM) policy. Once data is written to the store, it should not be deleted or changed, such that a user can be confident that the p-assertions in the store will be the same as when they were recorded. Furthermore, provenance stores cater for longevity through an explicit fixed data structure. Because of these attributes, it is beneficial to record all the elements that make up the documentation of a process in a provenance store.

However, in many cases data is large enough to make this approach unfeasible. For example, in the case of CERN, the data transferred between actors is on the order of terabytes [79], which would be expensive and time consuming to duplicate. Therefore, the p-structure supports the use of references to data stored outside the provenance store. In ACE, references were used when recording sequence data, which led to a decrease in record time and storage overhead. Unfortunately, data stored outside a provenance store may not have the same longevity and confidence guarantees that provenance store. To address this trade-off between the provenance store's guarantees and the storage of large data, we make the following recommendation based on our evaluation:

Recommendation: *Store referenced data items on storage infrastructure that has the same qualities of service as the infrastructure used by provenance stores.*

6.6.3 Throughput vs. contention

As shown in Figure 6.4, the greater the number of clients accessing the provenance store the more throughput the provenance store achieves. Therefore, to maximise the use of computational resources, a single provenance store should be employed. However,

from Figure 6.3, as the number of clients to the provenance store increases the response time as perceived by individual clients also increases. Thus, to maximise client response time there should be a provenance store per client, which is prohibitive in terms of computational and system administration resources. There exists, then, a trade-off between maximising the usage of computational resources and minimising the response time for clients. Based on our evaluation of ACE, contention has little perceived impact on application performance because application components do not typically access the provenance store at exactly the same time. Therefore, our recommendation is as follows:

Recommendation: *Initially, deploy a provenance store per application and increase the number of provenance stores deployed if application performance is not acceptable.*

6.6.4 Space vs. time

Process documentation takes up storage space. However, because PReServ's implementation of PReP and the p-structure make use of XML, significant space can be saved by using compression. However, as shown in Figures 6.2(a) and 6.2(b), compression on communication channels can increase record time substantially. However, if p-assertions are bundled together in large chunks, compression may have a positive impact on p-assertion recording time, especially on slower speed networks. This bundling approach is taken in ACE, which had a runtime overhead of 13%. Based on this, we make the following recommendation:

Recommendation: *For applications which can bundle p-assertions together in large record messages, use compression.*

These four recommendations provide a guide to developers on how their applications should generate process documentation and how that documentation can be most effectively recorded into provenance stores. Together they provide reasonable initial configurations and suggest alternatives to improve specific performance factors.

6.7 Confidence Revisited

In the introduction to this dissertation, we discussed the problem of confidence in multi-institutional scientific systems. One reason for this lack of confidence is the inability to reproduce results produced by these systems. Using our approach, the results of experiments can be reproduced given enough information is stored in the provenance store. In the case of ACE, the experiment could be reproduced based on the information stored in the provenance store, which includes the algorithms used, the ordering of algorithms, pointers to original data, and important intermediate data items. However, this

reproduction would have to be programmed but such a program to perform automatic reproduction is beyond the scope of this thesis.

Reproduction is one important factor in scientists' confidence in experimental results. However, as listed in Section 1.1, there are several other factors that contribute. We now revisit these factors and discuss how our approach to the provenance problem addresses them.

- *The ability to interpret and understand a result.*

  To interpret and understand a result one needs to know the process that led to it (i.e. its provenance). We have shown that for results produced by ACE, their provenance can be determined using a query over process documentation. An example of this is Use Case 2, where intermediate data that led to the production of an information efficiency value was extracted by performing a provenance query. This intermediate data helps the scientist to understand why the efficiency calculation was performed in a particular way.

- *The ability to understand the experiment and chain of reasoning that was used in the production of a result.*

  Again, the core element in understanding the chain of reasoning that led to a result is understanding the provenance of that result. Each use case that has been discussed relies on being able to determine the provenance of a result. Thus, our approach to the provenance problem provides confidence to scientists and users because they can understand the causal chain that was followed in an experiment.

- *The ability to verify that the experiment responsible for a result was performed according to acceptable procedures.*

  While not discussed here, process documentation stored in PReServ has been used to verify that an experiment was performed according to an acceptable plan [129].

- *The ability to identify what the inputs to an experiment were and where they came from.*

  In Use Case 1, a query was performed that determined the sequences that were inputs to ACE and the paths to them on the file system demonstrating that process documentation can be used to address this confidence factor.

- *The ability to know who performed an experiment and who is responsible for its results.*

  It was shown in Use Case 3 that discrepancies between the sender and receiver views of an interaction can be detected. Once such a discrepancy is found, the p-structure contains the identity of the party responsible for creating the documentation (the asserter identity). By the notion of observation by participation,

as codified in the factual characteristic, a party is responsible for the portion of the experiment they documented. Hence, using process documentation organised via the p-structure, a scientist can know who is responsible for an experiment.

Because these factors are addressed by our conceptual blueprint, scientists who have integrated PReServ with their multi-institutional scientific systems will have greater confidence in those systems' results.

6.8 Related Work and Other Applications

This evaluation follows on from previous evaluations of PReServ. We previously described the integration of provenance in an experiment, the Protein Compressibility Experiment, similar to the Amino Acid compressibility Experiment (ACE) and evaluates the performance impact of p-assertion recording on that experiment [88]. The impact of recording p-assertions in this setting was found to be 15% of application runtime, which is comparable to the overhead measurement presented here. The evaluation here differs in that overhead is studied across a Grid-enabled cluster environment. Previously, it was conducted using a small number networked personal computers running virtual machine software.

PReServ's performance was also compared with the provenance collection system Karma [165]. In terms of recording process documentation, the performance of PReServ was similar to, or outperformed, Karma. The version of PReServ studied in this evaluation addresses several deficiencies noted in the comparison study with Karma. First, PReServ no longer demonstrates a super-linear trend as the size of the provenance store increases. Second, through the use of compression, PReServ has reduced the amount of disk space it requires for the storage of process documentation. Lastly, while not in the scope of this dissertation, PReServ has increased its query scalability dramatically through the use of lazy loading techniques.

Additionally, a PReP compatible Provenance Store has been implemented by IBM [100]. PReServ differs with the IBM implementation in its underlying design philosophy and goals. The IBM Provenance Store was designed to consider security and standards conformance. Thus, they adopted the Globus Toolkit 4 [66] container for development. While this container enables certificate based authentication and security, it also results in a significantly more difficult installation procedure than PReServ. Furthermore, PReServ was designed to achieve optimal recording performance by adopting Berkeley DB JE and append-only database. While IBM's approach was to use an XML database to ease the development of query support. A benefit to using the common PReP interface is that applications have been able to move between the two implementations depending on their requirements.

As noted on page 116 a variety of applications have used our approach and in particular the PReServ software. This gives us confidence that our approach is generally applicable to a variety of applications that use different technologies and design approaches. For example, Townend et al. used our approach in the context of a web service environment where the goal was to find services that were responsible for common faults [183]. In this case, the application was modelled at a much less detailed level than ACE. Kloss et al. shows how an aerospace engineering example can be mapped to our view of process and what process documentation can be generated. While this example is in a Grid environment, it uses completely different underlying technologies, namely, CORBA and WebDav [107]. In another completely different context, organ transplant management, our approach of using process documentation to answer provenance questions, again, proved effective [105]. The management application was designed using an agent-based approach and was implemented using Web Services [105]. Finally, in our own work we were able to capture process documentation for an earlier version of ACE, which was entirely based on shell scripts and UNIX utilities [88].

6.9 Summary

In this chapter, we presented a design for a Provenance Service, PReServ, that meets the non-functional requirements of scalability, ease of installation, feature integration, and client independence. The implementation was shown to be scalable in two controlled performance tests. Specifically, PReServ achieved a throughput of 234,025 10k p-assertions in 10 minutes or, from a different perspective, 2.2 GB of p-assertions were created and recorded in a 10 minute period. Thus, every 2.3 milliseconds a p-assertion was recorded.

PReServ was also used as the provenance store in an evaluation of the effect of p-assertion recording in a real world multi-institutional experiment, the ACE. This evaluation showed that p-assertion recording has a 13% overhead in terms of ACE application runtime. We believe this performance is acceptable given that the process documentation recorded by the application was used to answer six different use case questions, which would otherwise be unanswerable. The algorithms used for answering these questions were also presented in this chapter. Finally, four recommendations were given as to how developers could best create and deploy provenance-aware applications.

In this chapter, we have shown that recording process documentation can be done with reasonable overhead and that this recorded process documentation enables a variety of provenance questions to be answered. Because process documentation is organised using the p-structure and recorded using PReP, users can be confident that it is high-quality, accurate evidence of a system's processes. Furthermore, questions that were not conceived of at the time of execution can be posed by querying over process documentation

stored within PReServ. This querying is enabled by the p-structure, which explicitly represents the causal connections between occurrences. These explicit causal connections allow causal graphs describing the provenance of multiple results to be extracted from process documentation. Overall, our conceptual blueprint, as realised by PReServ, enables the provenance problem to be addressed in a principled manner.

Chapter 7

Conclusion

A bioinformatician downloads the sequences for hundreds of thousands of proteins from databases created by a worldwide community of biologists. Using these sequences, he harnesses the power of billions of computer cycles made available by institutions and individuals to run a novel algorithm to better study the relationships between those proteins. These results are then used by another group of scientists to create new and better pharmaceuticals. While this is an example from bioinformatics, similar methodologies are being applied in a variety of domains including physics and economics. This is the new face of science: experiments that span multiple locations, multiple institutions, and multiple domains. By combining resources (computers, software, equipment, people, etc.), such multi-institutional scientific systems have already had a powerful impact in fields ranging from astronomy to earthquake engineering.

However, because of their complexity, it is difficult to repeat, reproduce, understand, interpret, and verify the results produced by these multi-institutional scientific systems. Hence, the confidence which scientists, policy makers, and the public have in these results is decreased. On the other hand, if these stake-holders understood how these results were produced, their history, their origins, their *provenance*, they would have greater confidence in them.

This lack of the provenance of results produced by such systems is the fundamental problem addressed by this dissertation. We have shown that through the autonomous creation, scalable recording, and principled organisation of documentation of multi-institutional scientific systems' processes, this *provenance problem* can be solved.

We now revisit the core contributions of this dissertation.

7.1 Contributions

7.1.1 Process Documentation and Provenance

In this dissertation, we have made an explicit distinction between documentation and provenance. From a brief review of the use of provenance in art, we ascertained that evidence of the provenance of a physical or digital object is shown through documentation. Thus, a representation of the provenance of an object can be obtained by querying over a set of documentation. Specifically, such a query is attempting to find the set of documentation that represents the particular process by which the object came to be.

Hence, by providing comprehensive documentation of a system's processes, the provenance of the results produced by that system can be determined. This was shown in Section 6.5 where six provenance use cases for the Amino Acid Compressibility Experiment were solved using a query over process documentation. These use cases were not only important for this particular bioinformatics experiment, but they also exemplify a range of provenance questions that arise from various domains including physics, chemistry, and computer security.

The distinction between process documentation and provenance also results in better system design. Concretely, it enables the *separation of concerns* between functionality for creating, recording querying process documentation. For example, developers can design software that specialises in analysing, querying and making use of process documentation without ever having to worry about how it ended up in a provenance store. Likewise, other developers can design tools to assist with the integration of recording in applications without concerning themselves with how queries will be catered for.

To enable this clear distinction to be manifested in systems, we specified a common data model shared between creators of process documentation and queriers of it. This data model is called the p-structure.

7.1.2 The P-Structure

The p-structure is the shared understanding between those who create process documentation and those who use it to answer provenance queries. A shared model that is also generic has a number of benefits, which include future proofing, easier sharing of documentation between institutions, empowering the development of common tools, and platform independence. These are powerful reasons to adopt such a shared model.

However, these benefits are subsidiary to the primary objective of a data model for process documentation, namely, the support for the *accurate* determination of *provenance*. To achieve this objective, the p-structure is based on two fundamental principles: causality and accuracy.

Understanding the origins of scientific results is fundamentally about letting users understand what *caused* these results to be as they are. Thus, a core part of our model of process documentation is the *explicit* representation of causal connections between data. The causal connections or relationships present in the data model are based on a well-defined notion of causality as defined in Section 2.4.3. Without these explicit causal relationships, the determination of the provenance of results would be significantly harder.

To ensure that the results of provenance queries conducted over process documentation are accurate, the p-structure as well as the P-assertion Recording Protocol support and when possible enforce process documentation to have the characteristics of being *factual*, *attributable*, *autonomously creatable*, *process oriented*, *immutable* and *finalizable*. Taken together these characteristics define high-quality process documentation. Each characteristic provides a different and important assurance to the user that the answers they receive from a provenance system are accurate and something which they can be confident in.

7.1.3 Recording process documentation

To cater for the recording and storage of high-quality process documentation, we introduce the notion of a specialised architectural element called the provenance store. Once an actor has created process documentation, it records the documentation into a provenance store so that other users can query it. The provenance store frees actors from the burden of individually maintaining process documentation.

Because there are many different institutions and sites within a multi-institutional scientific system, provenance stores can be distributed. Distribution helps to ensure scalability by spreading load over multiple repositories. Furthermore, it lets institutions preserve control over their process documentation. To help developers correctly use distributed provenance stores, we have developed patterns that show how provenance stores can be deployed for use in their applications. In addition to these deployment patterns, we introduced a mechanism to connect process documentation stored across multiple provenance stores, which is modelled after World Wide Web hyperlinks. These links allow process documentation to be queried across multiple, distributed provenance stores.

To ensure that the process documentation recorded in provenance stores has high-quality characteristics, we have specified the P-assertion Recording Protocol (PReP). The protocol is designed to cater for scalable implementations by being asynchronous and stateless. Using an abstract state machine formalisation, we established, through a series of proofs, that each high-quality characteristic was addressed by the protocol. The formalisation also provides an implementation independent specification, which devel-

opers can then use to correctly implement PReP on the various platforms that exist in multi-institutional systems.

7.1.4 Performance Impact

Through an evaluation of an implementation of our approach, we demonstrated that recording process documentation is an effective solution to the provenance problem. The implementation, PReServ, is Web Services based and designed to address four non-functional requirements: scalability, client independence, ease of installation, and feature integration. Using PReServ, the evaluation performed consisted of both controlled and real world experiments.

In the controlled experiments, we considered the impact on performance of increasing provenance store size and an increasing number of clients using the provenance store concurrently. We showed that store size had some impact on performance but when compression was enabled storage size impact was not detectable. Additionally, we demonstrated that the provenance store was capable of handling up to 512 concurrent clients without having a plateau in throughput. These performance results led to a recommendation that applications record process documentation in bundles using compression.

In the real world experiment, we analysed the impact on of p-assertion recording on the Amino Acid Compressibility Experiment presented in Chapter 5. This bioinformatics experiment was executed on the University of Southampton's Iridis Cluster computer and produced 14 gigabytes of process documentation for each run of the experiment. Over 360,000 unique results were produced during a run. Using statistical reasoning, we found that the overhead of the p-assertion recording on experiment runtime was an acceptable 13%.

For this 13% overhead, six provenance use cases were answered that had previously been unanswerable. Each algorithm used to answer the use cases was dependent on being able to find the provenance of some data item. These algorithms along with example results were presented in Section 6.5. The use cases selected are representive of various other use cases from domains ranging from high energy physics to chemistry. Additionally, they cover the questions of who, what, when, where, why, and how. We have shown that in a particular multi-institutional scientific system the approach of creating, recording and querying process documentation orgranised using a shared model is effective for solving highly relavent provenance questions. Furthermore, the use of our approach in a number of other applications [105, 107, 118, 149, 183, 193] including organ transplant management and fault tolerance in distributed systems shows that it is applicable to a hetereogenous mix of applications.

7.2 Support for High-Quality Documentation

In general, users want to be able to have an accurate representation of the provenance of their digital objects. Using the approach described here, this representation is obtained via a query over process documentation. Thus, the accuracy of provenance is directly associated with the accuracy of process documentation. Therefore, throughout this dissertation, we have discussed characteristics that help to ensure that process documentation is accurate. We termed process documentation that has these characteristics, high-quality.

These characteristics are based on an analogy between process documentation and evidence. We have argued that process documentation can be seen as evidence that a process occurred in a particular fashion. This view is supported by our review of various use cases [126] where documentation is often used as evidence. This analogy is at the heart of our notion of what makes high-quality process documentation. We now review these high-quality characteristics and discuss briefly how our protocol and data model support them.

Characteristic 1: Factual

It is important for evidence and thus process documentation to be factual. It should reflect what has actually occurred. The data model supports this characteristic by adopting and enforcing, where possible, the notion of observation by participation that is only actors who participate in a process should provide process documentation about it. PReP supports this characteristic by ensuring that only correctly typed p-assertions end up in the provenance store, which means that queriers can interpret documentation within the provenance store as being factual (i.e. from actors who directly participated in the process).

Characteristic 2: Attributable

Just as evidence in a court of law is often judged by its provider, process documentation is also judged by the actor, person, or institution responsible for it. Furthermore, if evidence or process documentation is false, knowing who created it allows remedial action to be taken against the responsible party. The p-structure supports attribution through the asserter identity associated with each p-assertion. Likewise, PReP ensures that the asserter identity is the identity of the actor who creates the p-assertion and that the identity is not modified in transit to or within the provenance store.

Characteristic 3: Autonomously Creatable

Gathering evidence or creating and recording process documentation should be conducted by the appropriate person, institution, or actor at the most appropriate time. The p-structure supports this characteristic through the interaction key, which enables

process documentation created and recorded by independent actors to be collated together. The interaction key is also designed so that it can be generated without external dependencies. To ensure that interaction keys are generated and used correctly, PReP enforces three actor behaviour rules. Additionally, to allow process documentation to be recorded when most appropriate or convenient, the protocol is asynchronous.

Characteristic 4: Process Oriented

Process documentation, like evidence, is only useful if it can be put together to show that a process occurred in a particular fashion. Understanding processes is inherently about understanding the causal connections between occurrences. The interaction and relationship p-assertions of the p-structure are specifically designed to capture just such causal connections. Furthermore, the p-structure is based on a view of process derived from the service-oriented architectural style. We have also shown that, using PReP, process documentation reflecting a process will eventually be stored in the provenance store after the process has completed.

Characteristic 5: Immutable

Just as it is important not to tamper or delete evidence, it is important not to modify or delete process documentation. We have shown, via formal proof, that process documentation stored within the provenance store is immutable.

Characteristic 6: Finalizable

To make a judgement about how a process, experiment, or crime has occurred, it is helpful to have all the evidence available. To ensure that users know when all the process documentation from a particular party is available, we introduced the notion of a complete view of an interaction. Using PReP's submission finished message, an actor can tell the provenance store when they have finished recording all the p-assertions about a particular interaction. PReP, then, prevents any additional p-assertions from being recorded about that view.

By analogy to evidence, these characteristics are crucial for having accurate process documentation. This dissertation has specifically described mechanisms for supporting the creation and recording of process documentation with these high-quality characteristics.

7.3 Future Work

The approach we have outlined centered around a common model for process documentation provides a foundation for the comprehensive support of provenance in both virtual and physical systems. Based on the ideas and concepts presented in this dissertation, a complete middleware architecture for provenance systems has been developed, which

extends the concepts here to deal with issues of security, querying, and the management of process documentation [92]. This architecture is partially implemented by both the PReServ provenance store and an IBM produced implementation [100]. This provides a basis for users of the software to integrate our approach with more applications, to improve the software's performance and scalability and most importantly start building applications based on provenance.

We see two directions for future work based on this foundation: integration and usage.

7.3.1 Integration

One of the significant hurdles for any provenance system is how to integrate with applications. While other systems advocate capturing data by modifying the operating system, this does not deal with distributed applications nor does it capture the high level meaning of application execution. In other work, we have proposed the use of generic wrappers designed for specific execution environments [89]. However, these wrappers may not be able to capture some application specific data. Therefore, it is still an open question as to the appropriate mechanisms necessary to *automatically* integrate with applications. Particularly, the question as to what is necessary to not only capture application execution but also the high level meaning of the process in an automatic fashion. Furthermore, work needs to be done to integrate with the variety of desktop and web-based applications that people use for their every day information processing activities.

Beyond specific application integration tasks, the methodology [139] used in making systems provenance-aware needs to be improved through integration with common development methods such as those based on the Unified Modelling Language like the Rational Unified Process [109]. To assist in the usage of the methodology, tools for integrated development environments need to be developed. Taken together both improvements could engender an environment where making applications provenance-aware is a central part of any application development effort.

Outside of the digital world, work is necessary in understanding how to integrate with physical systems. There are many technologies such as RFID and bar codes that allow physical objects to be tracked. However, the key is to be able to create documentation for all the processes that effect physical objects across multiple zones of control. For example, it would be useful to determine the provenance of a physical object in a home where the provenance included what has happened to it in the home, at the retailer where it was sold, in the ship where it was transported, and at the factory where it was manufactured.

In this dissertation, work has focused primarily on the ability to capture the information necessary to determine provenance. In the future, we would like to exploit this new

source of data.

7.3.2 Usage

As we have already seen in Chapter 5, an assortment of questions can be answered by the determining the provenance of results. However, there are still numerous research questions as to how best to make use of this new source of information. One question that arises is how do we make provenance accessible for user analysis. While a number of systems provide for the visualisation and navigation of provenance [200, 13, 57], we believe there is a significant amount of research that remains to be done in the area of visualisation for analysis. This includes questions such as how to display provenance graphs that have thousands of relationships, are there novel mechanisms to display the provenance of a result, and are there novel user interfaces to cater for the navigation of process documentation at multiple levels of abstraction.

Another area of interest is how to effectively use the provenance of results as input to new experiments. Can understanding the history of results change experimental processes? Can provenance be used to automatically improve experiments or even generate new derivative experiments? How can we most effectively reason about past processes and validate them against current plans? We imagine that future experiments will rely not only on input data but also on the provenance of that data. To assist in the usage of provenance for experiments, published papers deposited in institutional repositories will provide direct access to the process documentation that describes the provenance of the paper. This process documentation will including information ranging from the experimental process used to generate the paper's result to the programs used to edit the paper.

Finally, as the public demand increases for greater transparency by corporations and governments, the ability to determine the provenance of all products whether digital or physical will need to be provided. This raises interesting research issues in terms of security, privacy, and trust. Additionally, research needs to be done in how to manage and analysis all this data in scalable fashion, hopefully, by making use of some of the intrinsic qualities of process documentation like causal connections. The key is to obtain transparency while not being overwhelmed with extraneous or superfluous information.

7.4 Concluding Remarks

This dissertation has shown that the problem of determining the provenance of results produced by complex multi-institutional scientific systems can be solved through the autonomous creation, scalable recording, and principled organisation of documentation of these systems' processes. While we have emphasised the application of this

approach to scientific systems, we believe that same ideas can be applied to any system whether consumer, business, or scientific. Globalisation, out-sourcing, and the rise of loosely coupled organisations emphasise the need to be able to track and understand the provenance of objects (both physical and digital) produced by systems composed from multiple, independent institutions and organisations. Additionally, the trend towards ever greater transparency will require that the provenance of documents and products be made available. In the future, we will not only be able to understand how the report we received in our email was produced but also how, where, and by whom our t-shirt was made and the exact chain of events that led to us sitting in Starbucks drinking a coffee. By understanding the provenance of all the things that we deal with in our daily lives, we will have greater confidence and knowledge about our world. This dissertation is a small building block towards the grand vision of being able to know the provenance of everything.

Bibliography

[1] Portable Batch System, May 2007. http://www.openpbs.org.

[2] A. Abramovici, W. E. Althouse, R. W. P. Drever, Y. Grsel, S. Kawamura, F. J. Raab, D. Shoemaker, L. Sievers, R. E. Spero, K. S. Thorne, R. E. Vogt, R. Weiss, S. E. Whitcomb, and M. E. Zucker. LIGO: The laser interferometer gravitational-wave observatory. *Science*, 256(5055):325–333, April 1992.

[3] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: A System for Data, Uncertainty, and Lineage. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 1151–1154, Seoul, Korea, September 2006.

[4] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proceedings of 19th ACM Symposium on Operating Systems Principles*, October 2003.

[5] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, second edition, 2006.

[6] C. Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[7] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language*. Oxford University Press, 1977.

[8] G. Alonso and A. E. Abbadi. GOOSE: Geographic object oriented support environment. In *Proc. of the ACM workshop on Advances in Geographic Information Systems*, pages 38–49, Arlington, Virginia, November 1993.

[9] G. Alonso and C. Hagen. Geo-Opera: Workflow Concepts for Spatial Processes. In *Proc. 5th Intl. Symposium on Spatial Databases (SSD '97)*, Berlin, Germany, June 1997.

[10] I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance Collection Support in the Kepler Scientific Workflow System. In Moreau and Foster [135], pages 118–132.

[11] M. G. Anja Ebersbach and and R. Heigl. *Wiki: Web Collaboration*. Springer, 2005.

[12] K. Arnold and J. Gosling. *The Java Programming Language*. ACM Press/Addison-Wesley Publishing Co., second edition, 1998.

[13] T. Assandri. An Investigation into Provenance Visualisation (Summer Project Report). Technical report, University of Southampton, 2006.

[14] M. Atkinson, D. DeRoure, A. Dunlop, G. Fox, P. Henderson, T. Hey, N. Paton, S. Newhouse, S. Parastatidis, A. Trefethen, P. Watson, and J. Webber. Web Services Grids: An Evolutionary Approach. Technical report, Open Middleware Infrastructure Institute UK, July 2004.

[15] R. S. Barga and L. A. Digiampietri. Automatic Generation of Workflow Provenance. In Moreau and Foster [135], pages 1–9.

[16] R. S. Barga and L. A. Digiampietri. Automatic Capture and Efficient Storage of eScience Experiment Provenance. *Concurrency and Computation: Practice and Experience*, 2007.

[17] P. C. Bates. Debugging heterogeneous distributed systems using event-based models of behavior. *ACM Transactions on Computer Systems*, 13(1):1–31, 1995.

[18] R. A. Becker and J. M. J. M. Chambers. Auditing of data analyses. *SIAM Journal of Scientific and Statistical Computing*, 9(4):747–760, 1988.

[19] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). Technical report, The Internet Engineering Task Force, December 1994. RFC 1738.

[20] D. Bernholdt, S. Bharathi, D. Brown, K. Chancio, M. Chen, A. Chervenak, L. Cinquini, B. Drach, I. Foster, P. Fox, J. Garcia, C. Kesselman, R. Markel, D. Middleton, V. Nefedova, L. Pouchard, A. Shoshani, A. Sim, G. Strand, and D. Williams. The Earth System Grid: Supporting the Next Generation of Climate Modeling Research. *Proceedings of the IEEE*, 93(3):485–495, March 2005.

[21] M. J. Betts and R. B. Russell. *Bioinformatics for Geneticists*, chapter 14, pages 289–314. John Wiley & Sons, 2003. Amino acid properties and consequences of subsitutions.

[22] S. Boag, D. Chamberlin, M. F. Fernndez, D. Florescu, J. Robie, and J. Simon. XQuery 1.0: An XML Query Language. Technical report, World Wide Web Consortium, 2006.

[23] G. Booch. UML in action. *Communications of the ACM*, 42(10):26–28, 1999.

[24] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. Working group note, World Wide Web Consortium, 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[25] R. Bose. A Conceptual Framework for Composing and Managing Scientific Data Lineage. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, pages 15–19, Edinburgh, Scotland, July 2002.

[26] R. Bose and J. Frew. Composing lineage metadata with XML for custom satellite-derived data products. In *16th International Conference on Scientific and Statistical Database Management*, pages 275 – 284, Santorini Island, Greece, June 2004.

[27] R. Bose and J. Frew. Lineage retrieval for scientific data processing: a survey. *ACM Computing Surveys*, 37(1):1–28, 2005.

[28] D. Bourilkov, V. Khandelwal, A. Kulkarni, and S. Totala. Virtual Logbooks and Collaboration in Science and Software Development. In Moreau and Foster [135], pages 19–27.

[29] S. Bowers, T. McPhillips, and B. Ludaescher. A Provenance Model for Collection-Oriented Scientific Workflows. *Concurrency and Computation: Practice and Experience*, 2007.

[30] S. Bowers, T. M. McPhillips, B. Ludäscher, S. Cohen, and S. B. Davidson. A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows. In Moreau and Foster [135], pages 133–147.

[31] D. Box and D. Shukla. WinFX Workflow: Simplify Development With The Declarative Model Of Windows Workflow Foundation. *MSDN Magazine*, 21(1), January 2006.

[32] U. Braun, S. L. Garfinkel, D. A. Holland, K.-K. Muniswamy-Reddy, and M. I. Seltzer. Issues in Automatic Provenance Collection. In Moreau and Foster [135], pages 171–183.

[33] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan. Extensible Markup Language (XML) 1.1. W3C Recommendation, World Wide Web Consortium, August 2006. http://www.w3.org/TR/xml11.

[34] J. Brittain and I. F. Darwin. *Tomcat: The Definitive Guide*. O'Reilly, 2003.

[35] P. Buneman, A. Chapman, J. Cheney, and S. Vansummeren. A Provenance Model for Manually Curated Data. In Moreau and Foster [135], pages 162–170.

[36] P. Buneman, S. Khanna, K.Tajima, and W. Tan. Archiving scientific data. In *Proc. of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM Press, 2002.

[37] P. Buneman, S. Khanna, and W. Tan. Why and Where: A Characterization of Data Provenance. In *International Conference on Databases Theory (ICDT)*,

volume 1973 of *Lecture Notes in Computer Science*, page 316. Springer-Verlag, 2001.

[38] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM Press.

[39] K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.

[40] J. Cheney. Tradeoffs in XML Database Compression. In *Data Compression Conference (DCC'06)*, pages 392–401, 2006.

[41] C. Cherry. *On Human Communication: a review, a survey, and a criticism.* The M.I.T. Press, 1966.

[42] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, March 2001.

[43] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3c recommendation, World Wide Web Consortium, 1999. http://www.w3.org/TR/xpath.

[44] S. Cohen-Boulakia, O. Biton, S. Cohen, and S. Davidson. Addressing the provenance challenge using ZOOM. *Concurrency and Computation: Practice and Experience*, 2007.

[45] R. Conradi and B. Westfechtel. Version models for software configuration management. *ACM Computer Suveys*, 30(2):232–282, 1998.

[46] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press and McGraw-Hill, second edition, 2001.

[47] B. Cornell, P. A. Dinda, and F. Bustamante. Wayback: A user-level versioning file system for linux. In *In Proceedings of USENIX 2004*, 2004.

[48] T. M. Cover and J. A. Thomas. *Elements of Information Theory*, chapter 7, pages 144–162. John Wiley & Sons, Inc., 1991.

[49] J. Crowcroft, T. Moreton, I. Pratt, and A. Twigg. *The Grid 2: Blueprint for a New Computing Infrastructure - Peer-to-Peer Technologies*, chapter 29, pages 593–622. Morgan-Kaufmann, second edition, 2004.

[50] Y. Cui. *Lineage Tracing in Data Warehouses.* PhD thesis, Stanford University, December 2001.

[51] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'00)*, San Diego, California, February 2000.

[52] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *The VLDB Journal*, 12(1):41–58, 2003.

[53] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.

[54] T. Dean. Automated planning. *ACM Computing Surveys*, 28(1):85–87, 1996.

[55] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Cahi, and M. Livny. Pegasus: Mapping Scientific Workflows onto the Grid. In *Across Grids Conference*, 2004.

[56] E. Deelman and Y. Gil, editors. *Workshop on the Challenges of Scientfic Workflows*. National Science Foundation, May 2006.

[57] V. Deora, A. Contes, O. F. Rana, S. Rajbhandari, I. Wootten, K. Tamas, and L. Z.Varga. Navigating Provenance Information for Distributed Healthcare Management. In *IEEE/WIC/ACM Web Intelligence Conference*, pages 859–865, 2006.

[58] P. Deutsch. GZIP file format specification version 4.3. Technical report, Internet Engineering Task Force, 1996. RFC 1952.

[59] L. Ding and T. Finin. Characterizing the Semantic Web on the Web. In *Proceedings of the 5th International Semantic Web Conference*, 2006.

[60] F. Dvorák, D. Kouril, A. Krenek, L. Matyska, M. Mulac, J. Pospísil, M. Ruda, Z. Salvet, J. Sitera, and M. Vocu. gLite job provenance. In Moreau and Foster [135], pages 246–253.

[61] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, 2002.

[62] H. Fan and A. Poulovassilis. Tracing data lineage using schema transformation pathways. In B. Omelayenko and M. Klein, editors, *Knowledge Transformation for the Semantic Web*, pages 64–79. IOS Press, 2003.

[63] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Technical report, Internet Engineering Task Force, June 1999. RFC 2616.

[64] I. Foster. What is the Grid? A Three Point Checklist. *GridToday*, July 2002.

[65] I. Foster. Service-oriented science. *Science*, 308(5723):814–817, May 2005.

[66] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *IFIP International Conference on Network and Parallel Computing*, number 3779 in LNCS, pages 2–13, 2006.

[67] I. Foster, J. Gieraltowski, S. Gose, N. Maltsev, E. May, A. Rodriguez, D. Sulakhe, A. Vaniachine, J. Shank, S. Youssef, D. Adams, R. Baker, W. Deng, J. Smith, D. Yu, I. Legrand, S. Singh, C. Steenberg, Y. Xia, A. Afaq, E. Berman, J. Annis, L. A. T. Bauerdick, M. Ernst, I. Fisk, L. Giacchetti, G. Graham, A. Heavey, J. Kaiser, N. Kuropatkin, R. Pordes, V. Sekhri, J. Weigand, Y. Wu, K. Baker, L. Sorrillo, J. Huth, M. Allen, L. Grundhoefer, J. Hicks, F. Luehring, S. Peck, R. Quick, S. Simms, G. Fekete, J. vandenBerg, K. Cho, K. Kwon, D. Son, H. Park, S. Canon, K. Jackson, D. E. Konerding, J. Lee, D. Olson, I. Sakrejda, B. Tierney, M. Green, R. Miller, J. Letts, T. Martin, D. Bury, C. Dumitrescu, D. Engh, R. Gardner, M. Mambelli, Y. Smirnov, J. Voeckler, M. Wilde, Y. Zhao, X. Zhao, P. Avery, R. Cavanaugh, B. Kim, C. Prescott, J. Rodriguez, A. Zahn, S. McKee, C. Jordan, J. Prewett, T. Thomas, H. Severini, B. Clifford, E. Deelman, L. Flon, C. Kesselman, G. Mehta, N. Olomu, K. Vahi, K. De, P. McGuigan, M. Sosebee, D. Bradley, P. Couvares, A. D. Smet, C. Kireyev, E. Paulson, A. Roy, S. Koranda, B. Moe, B. Brown, and P. Sheldon. The Grid2003 Production Grid: Principles and Practice. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC-13'04)*, pages 236–245, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[68] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[69] I. Foster and C. Kesselman. Scaling system-level science: Scientific exploration and it implications. *Computer*, 39(11):31–39, November 2006.

[70] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In *International Journal of Supercomputer Applications*, pages 15–18, 2001.

[71] I. Foster, J. Vckler, M. Wilde, and Y. Zhao. Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In *14th International Conference on Scientific and Statistical Database Management (SSDBM'02)*, 2002.

[72] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In Moreau and Foster [135], pages 10–18.

[73] J. Frew and R. 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, Fairfax, VA, July 2001.

[74] J. Frew, D. Metzger, and P. Slaughter. Automatic capture and reconstruction of computational provenance. *Concurrency and Computation: Practice and Experience*, 2007.

[75] J. Frey, M. Bradley, J. Essex, M. Hursthouse, S. Lewis, M. Luck, L. Moreau, D. D. Roure, M. Surridge, and A. Welsh. *Grid Computing - Making the Global Infrastructure a Reality - Combinatorial Chemistry and the Grid*, chapter 42, pages 945–962. John Wiley and Sons, 2003.

[76] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proc. of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*. IEEE Press, 2001.

[77] J. Futrelle. Harvesting RDF Triples. In Moreau and Foster [135], pages 64–72.

[78] J. Futrelle and J. Myers. Tracking Provenance Semantics in Heterogeneous Execution Systems. *Concurrency and Computation: Practice and Experience*, 2007.

[79] F. Gagliardi, B. Jones, F. Grey, M.-E. Bgin, and M. Heikkurinen. Building an infrastructure for scientific Grid computing: status and goals of the EGEE project. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 363(1833):1729–1742, August 2005.

[80] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley Professional, 1995.

[81] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. Yang, and J. Zhang. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5(10):R80, August 2004.

[82] Y. Gil, E. Deelman, J. Blythe, C. Kesselman, and H. Tangmunarunkit. Artificial Intelligence and Grids: Workflow Planning and Beyond. *IEEE Intelligent Systems*, 19(1):26–33, Jan-Feb 2004.

[83] C. Goble. Position Statement: Musings on provenance, workflow and (semantic web) annotations for bioinformatics. In *Workshop on Data Provenance and Derivation*, October 2002.

[84] J. Golbeck and J. Hendler. A semantic web approach to tracking provenance in scientific workflows. *Concurrency and Computation: Practice and Experience*, 2007.

[85] G. Graham, R. Cavanaugh, P. Couvares, A. D. Smet, and M. Livny. *The Grid 2: Blueprint for a New Computing Infrastructure - Distributed Data Analysis: Federated Computing for High-Energy Physics*, chapter 10, pages 135–146. Morgan Kaufmann, 2004.

[86] P. Groth, M. Luck, and L. Moreau. A Protocol for Recording Provenance in Service-Oriented Grids. In T. Higashino, editor, *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04)*, volume 3544 of *Lecture Notes in Computer Science*, pages 124–139, Grenoble, France, December 2004. Springer-Verlag.

[87] P. Groth, M. Luck, and L. 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.

[88] P. Groth, S. Miles, W. Fang, S. C. Wong, K.-P. Zauner, and L. Moreau. Recording and Using Provenance in a Protein Compressibility Experiment. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, pages 201–208, July 2005.

[89] P. Groth, S. Miles, and L. Moreau. PReServ: Provenance Recording for Services. In *Proceedings of the UK OST e-Science Fourth All Hands Meeting (AHM05)*, September 2005.

[90] P. Groth, S. Miles, and L. Moreau. A Shared Model for Documentation of Processes Enabling the Determination of Provenance. *ACM Transactions on Internet Technology*, 2007. Under Review.

[91] P. Groth, S. Miles, and S. Munroe. Principles of High Quality Documentation for Provenance: A Philosophical Discussion. In Moreau and Foster [135], pages 278–286.

[92] P. Groth, S. Miles, V. Tan, and L. Moreau. An Architecture for Provenance Systems. Technical report, University of Southampton, October 2006. http://eprints.ecs.soton.ac.uk/11310/.

[93] R. Gude and M. Oster. Provenance-CSL: A provenance client side library. Technical report, Fachhochschule Bonn-Rhein-Sieg, Fachbereich Informatik, 2007.

[94] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. Soap version 1.2 part 1: Messaging framework (second edition). Technical report, World Wide Web Consortium, 2007. http://www.w3.org/TR/2007/REC-soap12-part1-20070427.

[95] D. Gunter, B. Tierney, K. Jackson, J. Lee, and M. Stoufer. Dynamic Monitoring of High-Performance Distributed Applications. In *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC-11)* , July 2002.

[96] J. Hollingsworth and B. Tierney. *The Grid 2: Blueprint for a New Computing Infrastructure - Instrumentation and Monitoring*, chapter 20, pages 319–351. Morgan Kaufmann, 2004.

[97] http://www.cvshome.org. Concurrent versions system.

[98] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34:729–732, April 2006.

[99] J. Hunter. *Java Servlet Programming.* O'Reilly, 2nd edition edition, 2001.

[100] J. Ibbotson, N. Hardman, and A. Biller. Provenance Store (Server) Implementation Design. Technical report, IBM, Feb 2006. D9.3.2: Functional Prototype (Public Release).

[101] M. Jaffe. Provenance. *Groove Art Online (The Dictionary of Art)*, 1996. Accessed Online (March 27, 2007).

[102] S. Jiang, L. Moreau, P. Groth, S. Miles, S. Munroe, and V. Tan. Client Side Library Design and Implementation. Technical report, University of Southampton, November 2006.

[103] J. Joyce, G. Lomow, K. Slind, and B. Unger. Monitoring distributed systems. *ACM Transactions on Computer Systems*, 5(2):121–150, 1987.

[104] A. Kazakov. An investigation on the performance of storing process documentation in a relational database. Technical report, University of Southampton, 2006.

[105] T. Kifor, L. Z. Varga, J. Vzquez-Salceda, S. lvarez, S. Willmott, S. Miles, and L. Moreau. Provenance in Agent-Mediated Healthcare Systems. *IEEE Intelligent Systems*, 21(6):38–46, Nov.-Dec. 2006.

[106] R. Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, Inc., second edition, 2002.

[107] G. K. Kloss and A. Schreiber. Provenance Implementation in a Scientific Simulation Environment. In L. Moreau and I. Foster, editors, *International Provenance and Annotation Workshop (IPAW)*, volume 4145 of *Lecture Notes in Computer Science*, pages 37–46. Springer-Verlag, 2006.

[108] D. Kranzlmuller. Dewiz - event-based debugging on the grid. In *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (EUROMICRO-PDP 2002)*, pages 162–169, 2002.

[109] P. Kruchten. *The Rational Unified Process: An Introduction.* Addison-Wesley Professional, third edition, 2003.

[110] L. Lamport. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Enginnering*, 3(2):125–143, 1977.

[111] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[112] D. Lanter. Design of a lineage-based meta-data base for GIS. *Cartography and Geographic Information Systems*, 18(4):255–261, 1991.

[113] D. Lanter. Lineage in GIS: The problem and a solution. Technical Report 90-6, National Center for Geographic Information and Analysis (NCGIA), UCSB, Santa Barbara, CA, 1991.

[114] D. Lanter and R. Essinger. User-centered graphical user interface design for GIS. Technical Report 91-6, National Center for Geographic Information and Analysis (NCGIA). UCSB, 1991.

[115] H. Lavana, A. Khetawat, F. Brglez, and K. Kozminski. Executable workflows: a paradigm for collaborative design on the internet. In *DAC '97: Proceedings of the 34th annual conference on Design automation*, pages 553–558, New York, NY, USA, 1997. ACM Press.

[116] D. Lewis. Causation. *Journal of Philosophy*, 70:556–67, 1973.

[117] B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience*, 18(10):1039–1065, 2005. Special Issue on Scientific Workflows.

[118] D. Luna. Creation of a Provenance-Aware RSS System. Part III project. Technical report, University of Southampton, May 2007.

[119] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[120] A. P. Marathe. Tracing Lineage of Array Data. *Journal of Intelligent Information Systems*, 17(2-3):193–214, 2001.

[121] T. Margaria and B. Steffen. Service Engineering: Linking Business and IT. *Computer*, 39(10):45–55, October 2006.

[122] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30:817–840, 2004.

[123] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. W3c recommendation, World Wide Web Consortium, 2004. http://www.w3.org/TR/owl-features/.

[124] R. Menday. The Web Services Architecture and the UNICORE Gateway. In *Proceedings of International Conference on Internet and Web Applications and Services (ICIW 2006)*, 2006.

[125] S. Miles. Electronically Querying for the Provenance of Entities. In Moreau and Foster [135], pages 37–46.

[126] S. Miles, P. Groth, M. Branco, and L. Moreau. The Requirements of Using Provenance in e-Science Experiments. In *Journal of Grid Computing*, 2006.

[127] S. Miles, P. Groth, S. Munroe, S. Jiang, T. Assandri, and L. Moreau. Extracting Causal Graphs from an Open Provenance Data Model. *Concurrency and Computation: Practice and Experience*, 2007.

[128] S. Miles, P. Groth, S. Munroe, M. Luck, and L. Moreau. AgentPrIMe: Adapting MAS designs to build confidence. In *Proceedings of 8th Internation Workshop on Agent Oriented Software Engineering*, 2007.

[129] S. Miles, S. C. Wong, W. Feng, P. Groth, K.-P. Zauner, and L. Moreau. Provenance-based Validation of e-Science Experiments. *Journal of Web Semantics*, 5(1):28–38, 2007.

[130] N. Mitra. Soap version 1.2 part 0: Primer. http://www.w3.org/TR/soap12-part0/, 2007.

[131] L. Moreau. Distributed directory service and message router for mobile agents. *Science of Computer Programming*, 39(2-3):249–272, 2001.

[132] L. Moreau, P. Dickman, and R. Jones. Birrell's Distributed Reference Listing Revisited. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 27(6):1344–1395, November 2005.

[133] L. Moreau and J. Duprat. A construction of distributed reference counting. *Acta Informatica*, 37:563–595, 2001.

[134] L. Moreau and I. Foster, editors. *Provenance and Annotation of Data — International Provenance and Annotation Workshop, IPAW 2006*, volume 4145 of *Lecture Notes in Computer Science*. Springer-Verlag, May 2006.

[135] L. Moreau and I. T. Foster, editors. *Provenance and Annotation of Data, International Provenance and Annotation Workshop, IPAW 2006, Chicago, IL, USA, May 3-5, 2006, Revised Selected Papers*, volume 4145 of *Lecture Notes in Computer Science*. Springer, 2006.

[136] L. Moreau, B. Ludäscher, I. Altintas, R. S. Barga, S. Bowers, S. Callahan, G. Chin Jr., B. Clifford, S. Cohen, S. Cohen-Boulakia, S. Davidson, E. Deelman, L. Digiampietri, I. Foster, J. Freire, J. Frew, J. Futrelle, T. Gibson, Y. Gil, C. Goble, J. Golbeck, P. Groth, D. A. Holland, S. Jiang, J. Kim, D. Koop, A. Krenek, T. McPhillips, G. Mehta, S. Miles, D. Metzger, S. Munroe, J. Myers, B. Plale, N. Podhorszki, V. Ratnakar, E. Santos, C. Scheidegger, K. Schuchardt, M. Seltzer, Y. L. Simmhan, C. Silva, P. Slaughter, E. Stephan, R. Stevens, D. Turi, H. Vo, M. Wilde, J. Zhao, and Y. Zhao. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience*, 2007.

[137] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-Aware Storage Systems. In *Proceedings of the 2006 USENIX Annual Technical Conference*, June 2006.

[138] S. Munroe, P. Groth, S. Jiang, S. Miles, V. Tan, and L. Moreau. Data Model for Process Documentation. Technical report, University of Southampton, 2006.

[139] S. Munroe, S. Miles, L. Moreau, and J. Vazquez-Salceda. PrIMe: A software engineering methodology for developing provenanceaware applications. In *ACM Digital Proceedings of the Software Engineering and Middleware Workshop (SEM'06)*, 2006.

[140] J. D. Myers, C. Pancerella, C. Lansing, K. L. Schuchardt, and B. Didier. Multi-scale science: supporting emerging practice with semantically derived provenance. In *ISWC 2003 Workshop: Semantic Web Technologies for Searching and Retrieving Scientific Data*, Sanibel Island, Florida, USA, October 2003.

[141] National Gallery of Art Website. Woman Holding a Balance - Provenance. World Wide Web, March 2007. http://www.nga.gov/collection/gallery/gg51/gg51-1239.0-prov.html.

[142] D. L. Nelson and M. M. Cox. *Lehninger Principles of Biochemistry*. W.H. Freeman, fourth edition edition, 2004.

[143] J. D. Novak. *Learning, Creating, and Using Knowledge: Concept Maps As Facilitative Tools in Schools and Corporations*. LEA, Inc, 1998.

[144] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[145] J. Pearl. *Causality: models, reasoning and inference*. Cambridge University Press, 2001.

[146] L. Pearlman, C. Kesselman, S. Gullapalli, B. S. Jr., J. Futrelle, K. Ricker, I. Foster, P. Hubbard, and C. Severance. Distributed Hybrid Earthquake Engineering Experiments: Experiences with a Ground-Shaking Grid Application. In *Proceedings of the 13th IEEE Symposium on High Performance Distributed Computing (HPDC-13)*, 2004.

[147] C. Pilato, B. Collins-Sussman, and B. W. Fitzpatrick. *Version Control with Subversion*. O'Reilly Media, 2004.

[148] B. Plale, D. Gannon, J. Brotzge, K. Droegemeier, J. Kurose, D. McLaughlin, R. Wilhelmson, S. Graves, M. Ramamurthy, R. D. Clark, S. Yalda, D. A. Reed, E. Joseph, and V. Chandrasekar. CASA and LEAD: Adaptive cyberinfrastructure for real-time multiscale weather forecasting. *Computer*, 39(11):56–64, November 2006.

[149] S. Rajbhandari, A. Contes, O. F.Rana, V. Deora, and I. Wootten. Establishing Workflow Trust Using Provenance Information. In *1st IEEE International Workshop on Modelling Autonomic Communications Environments (MACE 2006)*, October 2006.

[150] S. Ram and J. Liu. Understanding the Semantics of Data Provenance to Support Active Conceptual Modeling. In *ACM-L Workshop, ER 2006*, 2006.

[151] C. F. Reilly and J. F. Naughton. Exploring Provenance in a Distributed Job Execution System. In Moreau and Foster [135], pages 237–245.

[152] J. C. Sancho, F. Petrini, G. Johnson, J. Fernandez, and E. Frachtenberg. On the Feasibility of Incremental Checkpointing for Scientific Computing. In *18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, page 58b, 2004.

[153] D. S. Santry, M. J. Feeley, N. C. Hutchinson, A. C. Veitch, R. W. Carton, and J. Ofir. Deciding when to forget in the Elephant file system. In *SOSP '99: Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, pages 110–123, New York, NY, USA, 1999. ACM Press.

[154] C. Scheidegger, D. Koop, E. Santos, H. Vo, S. Callahan, J. Freire, and C. Silva. Tackling the provenance challenge one layer at a time. *Concurrency and Computation: Practice and Experience*, 2007.

[155] C. Scheidegger, D. Koop, H. Vo, J. Freire, and C. Silva. Querying and creating visualizations by analogy. *IEEE Transactions on Visualization and Computer Graphics*, 2007. To appear.

[156] M. Seltzer, D. A. Holland, U. Braun, and K.-K. Muniswamy-Reddy. PASS-ing the provenance challenge. *Concurrency and Computation: Practice and Experience*, 2007.

[157] C. E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27:379–423, 623–656, October 1948.

[158] A. Shieh, A. C. Myers, and E. G. Sirer. Trickles: A Stateless Network Stack for Improved Scalability, Resilience, and Flexibility. In *Second Symposium on Networked Systems Design and Implementation (NSDI'05)*, May 2005.

[159] C. Silva, J. Freire, and S. P. Callahan. Provenance for visualizations: Reproducibility and beyond. *IEEE Computing in Science & Engineering*, 2007. To appear.

[160] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable Techniques for Mining Causal Structures. In *Procceedings of the 24th VLDB Conference*, pages 594–606, 1998.

[161] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.

[162] Y. L. Simmhan, B. Plale, and D. Gannon. A Framework for Collecting Provenance in Data-Centric Scientific Workflows. In *International Conference on Web Service (ICWS'06)*, 2006.

[163] Y. L. Simmhan, B. Plale, and D. Gannon. Towards a quality model for effective data selection in collaboratories. In *Proceedings of the 22nd International Data Engineering Workshops*, 2006.

[164] Y. L. Simmhan, B. Plale, and D. Gannon. Querying Capabilities of the Karma Provenance Framework. *Concurrency and Computation: Practice and Experience*, 2007.

[165] Y. L. Simmhan, B. Plale, D. Gannon, and S. Marru. Performance Evaluation of the Karma Provenance Framework for Scientific Workflows. In Moreau and Foster [135].

[166] P. M. Sloot, A. Tirado-Ramos, I. Altintas, M. Bubak, and C. Boucher. From Molecule to Man: Decision Support in Individualized E-Health. *Computer*, 39(11):40–46, November 2006.

[167] C. D. Snow, H. Ngyen, V. S. Pande, and M. Gruebele. Absolute comparison of simulated and experimental protein-folding dynamics. *Nature*, 420:102–106, 2002.

[168] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. The MIT Press, second edition, 2000.

[169] L. Stein. Creating a bioinformatics nation. *Nature*, 417:119–120, May 2002.

[170] L. Stein. Integrating biological databases. *Nature Reviews Genetics*, 4(5):337–345, May 2003.

[171] R. D. Stevens, H. J. Tipney, C. J. Wroe, T. M. Oinn, M. Senger, P. W. Lord, C. A. Goble, A. Brass, and M. Tassabehji. Exploring williams-beuren syndrome using myGrid. *Bioinformatics*, 20(1):303–310, March 2004.

[172] T. J. Strader, F.-R. Lin, and M. J. Shaw. Information infrastructure for electronic virtual organization management. *Decision Support Systems*, 23(1):75–94, 1998.

[173] A. Streit, D. Erwin, T. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and P. Wieder. *UNICORE - From Project Results to Production Grids*, pages 357–376. Elsevier, 2005.

[174] R. Strom and S. Yemini. Optimistic recovery in distributed systems. *ACM Transactions on Computer Systems*, 3(3):204–226, 1985.

[175] N. Suri, J. Bradshaw, M. R. Breedy, P. T. Groth, G. A. Hill, and R. Jeffers. Strong Mobility and Fine-Grained Resource Control in NOMADS. In F. M. David Kotz, editor, *Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents, ASA/MA 2000*, volume 1882 / 2004 of *Lecture Notes in Computer Science*, pages 2–15, Zurich, Switzerland, 2000. Springer-Verlag.

[176] A. S. Szalay. The Sloan Digital Sky Survey. *Computing in Science & Engineering*, 1(2):54–62, 1999.

[177] A. S. Szalay and J. Gray. *The Grid 2: Blueprint for a New Computing Infrastructure - Scientific Data Federation: The World-Wide Telescope*, chapter 7, pages 95–108. Morgan Kaufmann, 2004.

[178] M. Szomszor and L. Moreau. Recording and Reasoning over Data Provenance in Web and Grid Services. In *International Conference on Ontologies, Databases and Applications of Semantics*, volume 2888 of *LNCS*, 2003.

[179] V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau. Security Issues in a SOA-Based Provenance System. In Moreau and Foster [135], pages 203–211.

[180] V. H. K. Tan. *Interaction tracing for mobile agent security*. PhD thesis, University of Southampton, 2004.

[181] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[182] The Message Passing Interface Forum (MPIF). MPI: A Message-Passing Interface Standard. Technical report, June 1995. http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html.

[183] P. Townend, P. Groth, and J. Xu. A Provenance-Aware Weighted Fault Tolerance Scheme for Service-Based Applications. In *Proceedings of the 8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2005)*, pages 258–266. IEEE Computer Society, May 2005.

[184] A. Vahdat and T. Anderson. Transparent Result Caching. In *Proceedings of the 1998 USENIX Technical Conference*, New Orleans, Louisiana, June 1998.

[185] G. Wang and R. L. Dunbrack. PISCES: a protein sequence culling server. *Bioinformatics*, 19(12):1589–1591, 2003.

[186] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture*. Prentice Hall, 2005.

[187] D. L. Wheeler, C. Chappey, A. E. Lash, D. D. Leipe, T. L. Madden, G. D. Schuler, T. A. Tatusova, and B. A. Rapp. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Research*, 28(1):10–14, 2000.

[188] J. Widom. Trio: a system for integrated management of data, accuracy, and lineage. In *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, Calif., January 2005.

[189] M. Wilkinson, H. Schoof, R. Ernst, and D. Haase. BioMOBY successfully integrates distributed heterogeneous bioinformatics web services.the planet exemplar case. *Plant Physiology*, 138:5–17, May 2005.

[190] J. Woodfill and M. Stonembraker. An implementation of hypothetical relations. *Proceedings of the 9th Internationl Conference on Very Large Databases*, pages 157–166, October 1983.

[191] A. Woodruff and M. Stonebraker. Supporting Fine-grained Data Lineage in a Database Visualization Environment. In *Proceedings of the 13th International Conference on Data Engineering*, pages 91–102, Birmingham, England, April 1997.

[192] A. G. Woodruff. *Data Lineage and Information Density in Database Visualization.* PhD thesis, University of California at Berkeley, 1998.

[193] I. Wootten, S. Rajbhandari, O. Rana, and J. Pahwa. Actor Provenance Capture with Ganglia. In *Proceedings of the 6th IEEE Internation Symposium on Cluster Computing and the Grid (CCGrid'06)*, 2006.

[194] C. Wroe, C. Goble, M. Greenwood, P. Lord, S. Miles, J. Papay, T. Payne, and L. Moreau. Automating Experiments Using Semantic Data on a Bioinformatics Grid. *IEEE Intelligent Systems*, 19(1):48–55, Jan-Feb 2004.

[195] C. H. Wu, R. Apweiler, A. Bairoch, D. A. Natale, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, R. Mazumder, C. O'Donovan, N. Redaschi, and B. Suzek. The Universal Protein Resource (UniProt): an expanding universe of protein information. *Nucleic Acids Research*, 34:187–191, January 2006.

[196] J. Yang and M. P. Papazoglou. Web Component: A Substrate for Web Service Reuse and Composition. In *Advanced Information Systems Engineering: 14th International Conference, CAiSE 2002*, volume 2348 of *LNCS*, pages 21–36, 2002.

[197] J. Yu and R. Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing*, 3(3-4):171–200, September 2005.

[198] J. Zhao, C. Goble, M. Greenwood, C. Wroe, and R. Stevens. Annotating, linking and browsing provenance logs for e-Science. In *Proceedings of the Workshop on*

Semantic Web Technologies for Searching and Retrieving Scientific Data, October 2003.

[199] J. Zhao, C. Goble, R. Stevens, and D. Turi. Mining Taverna's Semantic Web of Provenance. *Concurrency and Computation: Practice and Experience*, 2007.

[200] J. Zhao, C. Wroe, C. Goble, R. Stevens, D. Quan, and M. Greenwood. Using Semantic Web Technologies for Representing e-Science Provenance. In *Proceedings of the 3rd International Semantic Web Conference*, volume 3298, pages 92–106, Hiroshima, Japan, 2004.

[201] Y. Zhao, M. Wilde, and I. Foster. A Virtual Data Provenance Model. In Moreau and Foster [135].

[202] Y. Zhao, M. Wilde, I. Foster, J. Voeckler, J. Dobson, E. Gilbert, T. Jordan, and E. Quigg. Virtual data Grid middleware services for data-intensive science. *Concurrency and Computation: Practice and Experience*, 18(6):595–608, May 2006.