UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

A nine month progress report

Supervisor: Professor Luc Moreau
Examiner: Professor Peter Henderson

Recording Provenance in Service-Oriented Architectures

by Paul T. Groth

February 21, 2005

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

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

A nine month progress report

by Paul T. Groth

Provenance is the documentation of process for some result. This report addresses provenance recording in Service-Oriented Architectures, specifically for Grids and Web Services. The document begins by motivating the need for provenance recording. It then presents background information for Service-Oriented Architectures, Grids, Web Services and provenance software. Given this background, an architecture and protocol for recording provenance are presented along with an implementation of a provenance service. Finally, a direction for future work is outlined.

esse sequitur operari
being follows functioning

Contents

1 Introduction **1**

2 Background **3**
2.1 Service-Oriented Architectures . . . 3
2.2 Grids . . . 3
2.2.1 Grid Definition . . . 3
2.2.2 Grid Applications . . . 4
2.2.3 Grid Technologies . . . 5
2.3 Web Services . . . 6
2.3.1 Web Service Definition . . . 6
2.3.2 Web Services and Grids . . . 6
2.4 Provenance . . . 8
2.4.1 Low-level Provenance . . . 8
2.4.2 Domain Specific Provenance Systems . . . 9
2.4.3 Provenance in Database Systems . . . 10
2.4.4 Middleware to Support Provenance . . . 11
2.4.5 Conclusion . . . 12

3 Current Work **13**
3.1 The Problem . . . 13
3.2 Requirements . . . 14
3.3 SOA Provenance Recording . . . 15
3.4 Provenance Recording Protocol . . . 19
3.5 Actors . . . 22
3.6 Properties . . . 27
3.7 Conclusion . . . 29

4 Future Work **30**
4.1 A Timeline . . . 31

Bibliography **32**

Chapter 1

Introduction

Who? What? When? Where? Why? How?

An artist applies a bit of titanium white to his canvas, takes a soft brush and ever so slightly smears it to form the shape of a pearl. Two hundred years later, the painting is being studied by a student who would love to know how the pearl was made, what paint was used, where it was painted, who painted it, when exactly was it painted, and why the painter did it that way? Luckily, for the student this artist was one of the few that kept an accurate record (a journal) of the process of his painting, allowing his techniques to be studied, reused, and improved upon. With this record, the student had access to the provenance of the painting. Provenance is the documentation of process for some result, whether that result is a painting, drug, or integer value. Its necessity is apparent across a wide variety of fields. For example, the American Food and Drug Administration requires that the provenance of a drug's discovery be kept as long as the drug is in use (up to 50 years sometimes). In chemistry, provenance is used to detail the procedure by which a material is generated, allowing the material to be patented. In aerospace, simulation records as well as other provenance data are required to be kept up to 99 years after the design of an aircraft. In financial auditing, the American Sarbanes-Oxley Act requires public accounting firms to maintain the provenance of an audit report for at least seven years after the issue of that report (United States Public Law No. 107-204). In medicine, the provenance of an organ is vital for its effective and safe transplantation. These are just some examples of the requirements for provenance in science and business.

The requirements for provenance in these diverse fields arise because provenance has many different uses. These uses can be divided into the following general categories, which have been adapted from Goble (2002):

1. Attribution - Provenance can be used to determine who contributed to a result, allowing the appropriate contributors to be credited whether through payment, citation or some other mechanism.

2. Quality - Knowing the process by which a result was generated can give insight into the quality of that result. A result may seem high-quality but the provenance may cast doubt on it by showing that low-quality data was used.

3. Audit - Provenance can be used both to verify by users (possibly third parties) that a process was done according to guidelines and to check for anomalies in a process.

4. Reusability - Provenance can be used to repeat, reproduce, or improve processes.

Essentially, provenance helps answer the questions enumerated at the beginning of this report.

Now imagine that the student, who was lucky enough to have the artist's journal, decides to do some computational image analysis on the painting to see whether other artists have similar paintings. He scans the painting and submits the image along with some date ranges to a painting comparison portal on the web. Behind the scenes the portal accesses several museum databases of stored images, uses a service in Taiwan that determines colour composition, uses several different image comparison services available from different universities, collates the results locally, and finally sends an email to the student with a list of similar paintings. The student is quite pleased goes off and looks at the various suggested paintings. Later, the student returns and resubmits the same image to the portal. Surprisingly, the list of paintings returned is somewhat different. What happened? The web portal is completely automated. It finds the services it uses dynamically based on parameters like cost and availability. The list of paintings could have changed because one of the databases was unavailable, a new database was added, or a cheaper colour composition service in Arizona was used. Unfortunately for the student, there is no provenance available from the portal and therefore, he cannot determine why the results changed or more particularly what databases, algorithms or services contributed to the results he received.

This scenario leads to the overarching question that we intend to address with our research, namely, how to record provenance in service-oriented architectures? This report begins to address this question. The usage of provenance motivates our research but is beyond its scope. Instead, our research will be combined with the work of the Provenance Aware Service Oriented Architecture project to address usage scenarios. The next section presents background information on service-oriented architectures, Web Services, Grid Computing and provenance. This section is followed by a presentation of work completed in the last nine months, including a proposed architecture and protocol for recording provenance. Finally, we present a direction for future work.

Chapter 2

Background

2.1 Service-Oriented Architectures

A service-oriented architecture (SOA) is one in which the architecture consists of loosely-coupled services communicating via a common transport. A service, in turn, is defined as a well-defined, self-contained, entity that performs tasks which provide coherent functionality. Typically, a service is only available through an interface represented in some standard format. SOAs provide several benefits. First, they hide implementation behind an interface allowing implementation details to change without affecting the user of the service. Secondly, the loosely-coupled nature of services allows for their reuse in multiple applications. Because of these properties, SOAs are particularly good for building large scale distributed systems. Later in this chapter, we will address two technologies that use the SOA approach, Web Services and Grids. Although, we focus on these two technologies any techniques for recording provenance should also be transferable to other SOA-like systems such as Common Object Request Broker Architecture (CORBA) [Pope (1997)] or Jini [Waldo (2000)].

2.2 Grids

2.2.1 Grid Definition

The scenario described in the introduction is an example of Grid computing. The definition of a Grid has developed over time. It has its basis in the idea of utility computing, where computing resources are available "on-demand" like electricity is available from the power grid [Foster and Kesselman (1999)]. This definition has been adjusted over time to embrace more of a policy perspective. Foster, Kesselman and Tuecke took this approach by defining a Grid in terms of the problem it solves, namely, coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations

[Foster et al. (2001b)]. A virtual organisation (VO) is defined by the rules that govern the sharing of resources between a set of individual and/or institutions [Foster et al. (2001b)]. The application running behind our web portal from the above scenario is an example of a virtual organisation, multiple institutions/individuals (museums, universities, the student) share their resources (images, algorithms) to solve a problem (identifying similar paintings). VOs provide a useful computational model for coordinating the interactions of diverse resources controlled by multiple participants. It allows diverse computational 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.

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. The goal, then, of a Grid is to provide infrastructure for the assembly, management, and execution of VOs [Foster et al. (2002a)]. Foster further refined the Grid definition by stating that a Grid is a system that coordinates computational resources, not subject to centralized control, using standard, open, general-purpose protocols and interfaces to deliver non-trivial qualities of service [Foster (2002)]. These two definitions have become the point of reference for the Grid community.

2.2.2 Grid Applications

Grids are being used for a number of different applications. For example, Grids are used to study earthquakes [Pearlman et al. (2004)], link supercomputing sites (http://www.teragrid.org), study data from the Large Hadron Collider (http://lcg.web.cern.ch/LCG/), run experiments in surface chemistry [Frey et al. (2003)], study global climate [Foster et al. (2001a)], simulate aircraft and to improve financial recommendations (http://www.ibm.com/grid/). These applications can be divided into five different categories as outlined by Foster and Kesselman [Foster and Kesselman (1999)]. The categories are as follows:

1. Distributed supercomputing. Used to solve problems that require large amounts of computational resources. (e.g. link supercomputing sites)
2. High throughput. Used to aggregate idle resources to solve problems composed of independent tasks. (e.g. SETI at home)

3. On demand. Used when an application needs resources that are not available locally on a time constrained basis. (e.g. financial recommendations)

4. Data intensive. Used to process large, distributed data sets. (e.g. Large Hadron Collidor)

5. Collaborative. Enhances communication between people across physical and organisational boundaries. (e.g. study earthquakes)

2.2.3 Grid Technologies

Grid applications are based on several different technologies. The most basic being Grid middleware, which provides the basic foundation of a particular Grid. Each node that is a member of a Grid is normally required to speak the common language of that Grid by having a particular middleware stack installed. The most common stack is the Globus Toolkit [Foster and Kesselman (1997)]. There are, however, other middleware stacks available such as the Grid Application Toolkit [Seidel et al. (2002)] or the UNiform Interface to Computing Resources (UNICORE) software [Huber (2001)]. It is important to note that such middleware does not make a Grid by itself but instead forms the basis of Grid applications much like TCP/IP is the basis for Internet applications. Such middleware is usually an implementation of various protocols and specifications. One such protocol is GridFTP [Allcock (2003)] for transferring large amounts of data. The protocols and specifications that Grid middleware should implement have yet to be completely standardised. The goal is to eventually have standards such as TCP/IP for Grids such that individual Grids could be linked into one larger Grid just as today's local area networks are connected to make one large wide area network, the Internet.

Based on this middleware, a Grid would typically be built using an SOA approach. Each resource on a Grid is wrapped as a service that is accessed using one of the protocols defined by the middleware. Using these services, a VO can be assembled either dynamically or through an off-line process. SOAs map well to virtual organisations because of their loosely-coupled nature. In order to tie services together, one technique that is often used is workflow and workflow enactment. 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 [Foster et al. (2001b)].

A workflow is usually represented as a Directed Acyclic Graph (DAG) and can be divided into two types, abstract and concrete. 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 [Deelman et al. (2004)] takes abstract workflows and generates concrete workflows taking advantage of available resources. Users can also create concrete workflows directly either specifying a DAG by hand or by using workflow editing software such as Taverna (http://taverna.sourceforge.net/). These workflows are then executed with a workflow enactment engine. The enactment engine then produces job files that are submitted to various nodes for processing. These jobs can also be managed by scheduling software such as Condor-G [Frey et al. (2001)].

2.3 Web Services

2.3.1 Web Service Definition

A Web Service is a service whose interface, and the protocols that are used to communicate with that interface, are defined by Internet standards. These standards are either de facto or developed by one of the many standards bodies including the W3C and OASIS. Web services arose out of the need for application to application communication over the World Wide Web. Because a Web Service is defined only by its interface and uses common standards to communicate with other services, Web Services can be used to build distributed systems in a SOA fashion. There are roughly sixty different Web Service specifications either in development or standardised. There are two central specifications that define a Web Service, the Web Service Definition Language (WSDL) [Christensen et al. (2001)] and SOAP [Mitra (2003)]. WSDL specifies the interface to a service. Using an XML based format, a service defines the ports that it accepts messages through. These ports map to functionality that a service provides. The use of XML provides an extensible type system for WSDL allowing types to be described in a platform independent manner. WSDL can specify any number of transports or data formats that a service might support. Therefore, a transport could be anything from HTTP to SMTP and the message could be represented in SOAP, MIME or a completely different format. In practice, Web Services normally communicate over HTTP using SOAP as a message format. SOAP is a simple XML message format consisting of an envelope containing a body and an optional header. The promise of Web Services is that it provides a standardised language for applications to communicate without specifying the architecture, programming language, or system that the application must use.

2.3.2 Web Services and Grids

The Grid community has embraced Web Services as a basis for implementing Grids. Many of the specifications that Web Services provide are a good fit to the requirements of Grid computing in that they provide a common language for Grid middleware to implement and Grid services to use. As evidence of the integration of Grids and Web

Services, the third release of the Globus Toolkit has a Web Service implementation. Many new Web Service specifications are being driven by the needs of the Grid community.

One such need is in the area of state, i.e. data values that persist and change across multiple service interactions [Czajkowski et al. (2004)]. Both web services and Grids require state. For example, with a compute service, a user might want to enquire and modify a job after submission or in the case of a travel service a user might want to change a booking after creating it. Currently, many web services allow for the manipulation of these sorts of stateful resources but that manipulation is implicit. It is not always semantically clear how to manage a resource and the facilities available differ from service to service. Therefore, it would be useful to provide a common explicit mechanism for managing stateful resources. The Web Services Resource Framework (WSRF) [Czajkowski et al. (2004)] provides such a mechanism by specifying common methods for accessing, creating, deleting and modifying stateful resources.

Another technology required for Grids is service level agreements (SLA). Services negotiate with each other about what functionality they will provide each other finally reaching an SLA. These agreements usually revolve around the quality of service that a service provides to an invoker. Two proposed specifications that address this area are WS-Agreement [Czajkowski et al. (2003)] and WS-Policy [Hondo and Kaler (2003)]. These specifications provide a syntax and protocol for exchanging and expressing SLAs. A specification also being driven by the Grid community is Web Services Distributed Management, which will deal with the problems of monitoring services, enforcing service level agreements, and governing the lifecycle of services. The ability to monitor services and control the lifecycle of aggregated services is essential for creating VOs.

The last technology useful for Grids is notification. The notification specifications detail how to implement a publish-and-subscribe model for Web Services. This model allows a service to listen for events published by other services allowing, for example, the sending of messages to multiple recipients. The two proposed specifications in this area are the family of documents under the WS-Notification [Graham and P.Niblett (2004)] and WS-Eventing [Schlimmer (2004)] headings.

There are other specifications already in the Web Services stack that are useful for Grid computing including specifications for addressing security (WS-Trust, WS-Security, SAML), interface definition (WSDL) and message format (SOAP). To help the adoption of interoperable specifications the Web Services Interoperability Organisation publishes profiles that define sets of specifications that work together. The paths of Grid computing and Web Services are converging as Web Services becomes the mechanism of choice for constructing Grids.

2.4 Provenance

At the beginning of this report, we defined provenance as the documentation of process. Prior research has referred to this concept using several other terms including audit trail, lineage [Lanter (1991b)], dataset dependence [Alonso and Hagen (1997)], and execution trace [Tan (2004)]. We use these terms interchangeably to refer to the documentation of process. However, the literature that does refer directly to provenance is in some disagreement about the definition of the various types of provenance. Based on the survey below, we have defined three types of provenance:

- Data Provenance - Given a piece of data, x, data provenance refers to the set of data used in the creation of x. For example, in a database the data provenance of a tuple is all the tuples used in creating that tuple.
- Execution Provenance - Given a piece of data, x, execution provenance refers to the transformations and the input and output data of those tranformations used in the creation of x. For example, execution provenance for an enacted workflow would be the services invoked and their input/output data.
- Knowledge Provenance - Given a piece of data, x, knowledge provenance refers to metadata about the creation of x. For example, this could be a note by an author of a document explaining why the document was written.

It is worth noting that our definition of knowledge provenance does not require the use of semantically marked up metadata nor does it require that this metadata carry proof-like information based on reasoning techniques. The metadata can be anything from annotations from the users to information derived from data and execution provenance. Our definition is much broader than the one proposed by da Silva et al. (2003), which defines knowledge provenance as source metadata plus the reasoning process used to generate a result.

The literature below can be divided into four rough categories: low-level provenance, domain specific provenance systems, provenance in database systems, and middleware for provenance systems. This review gives a brief summary of the literature pertaining to each of these categories.

2.4.1 Low-level Provenance

Low-level provenance refers to provenance recording at the level of script or program execution. These systems differ from workflow centric systems because of the higher granularity of provenance these systems achieve. One example, is the Transparent Result Caching (TREC) prototype [Vahdat and Anderson (1998)]. TREC uses the Solaris

UNIX proc system to intercept various UNIX system calls in order to build a dependency map. Using this map, a trace of a program's execution can be transparently captured, which can be used to keep Web page caches current and to provide an 'unmake' function. Although TREC has several limitations, including high overhead, it is interesting case study for determining the level of granularity at which provenance should be captured.

Another technique for capturing low-level provenance is Marathe (2001) sub-pushdown algorithm. This algorithm can only be used to capture lineage for array operations in the Array Manipulation Language and was implemented in a prototype database system, ArrayDB. A more comprehensive system is the audit facilities designed for the S language by Becker and J. M. Chambers (1988). S is an interactive system for statistical analysis. The result of users command are automatically recorded in an audit file. These results include the modification or creation of data objects as well as the commands themselves. The AUDIT utility can then be used to analyse the audit file. This utility can also create script to reexecute a series of commands from the audit file.

Low-level provenance capture has also been used in security for mobile agent systems. Using a technique called interaction tracing, a user sending a mobile agent can verify that it has correctly executed on the host platform. Interaction tracing treats a mobile agent as a black box, recording all the inputs/outputs of the executing agent [Tan (2004)].

2.4.2 Domain Specific Provenance Systems

Much of the research into provenance recording has come in the context of domain specific applications. Some of the first research in provenance was in the area of geographic information systems (GIS)[Lanter (1991b)]. Provenance is critical in GIS applications because it allows one to determine the quality of derived map products [Lanter (1991b)]. Lanter developed two systems for tracking provenance in a GIS, a meta-database for tracking execution provenance and a system for tracking Arc/Info GIS operations from a graphical user interface with a command line [Lanter (1991a); Lanter and Essinger (1991)]. The workflow centric user interface was integrated into a software product called Geolineus, which was one of the few lineage tracking systems to be incorporated into a commercial software product [Bose (2002)]. Another GIS system that includes provenance tracking is Geo-Opera, which is based on a non-domain specific software [Alonso and Hagen (1997)]. Many of the ideas in Geo-Opera are extended from GOOSE, which uses data attributes to point to the latest inputs/outputs of a data transformation. All inputs/outputs must be stored in GOOSE and data transformations are programs or scripts [Alonso and Abbadi (1993)]. Both GOOSE and Geo-Opera are workflow based systems.

Related to GIS is the satellite image processing domain. The Earth System Science Workbench (ESSW) is designed for processing satellite imagery locally. It provides

a lab notebook service for tracking processing steps and a No-Duplicate Write Once Read Many storage service for storing files. Essentially, as a workflow is run inside ESSW the results of each metadata described step is stored in the lab notebook service. In chemistry, the CMCS project has developed a system for managing metadata in a multi-scale chemistry collaborations [Myers et al. (2003a)]. The CMCS project is based on the Scientific Application Middleware project [Myers et al. (2003b)], which we discuss in greater detail later in this review. Another domain where provenance tools are being developed is bioinformatics. The myGrid project has implemented a system for recording provenance in the context of in-silco experiments represented as workflows aggregating Web Services [Greenwood et al. (2003)]. In myGrid, provenance is gathered about workflow execution and stored in the user's personal repository along with any other metadata that might be of interest to the scientist [Zhao et al. (2003)]. The focus of myGrid is personalising the way provenance is presented to the user.

The needs of particular domains has led to the development of specific systems for provenance recording. Many of these systems have been designed to be used in more general cases but have not been used outside their specific domain. A common thread connecting these systems is their workflow-centric nature.

2.4.3 Provenance in Database Systems

Provenance in database systems has focused on the data lineage problem [Cui et al. (2000)]. This problem can be summarised as given a data item, determine the source data used to produce that item. Woodruff and Stonebraker (1997) look at solving this problem through the use of the technique of weak inversion. Given some output data and a weak inversion function f^{-w}, f^{-w} 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 an inversion function for that view. This technique has been used to improve database visualization [Woodruff (1998)]. Cui et al. (2000) formalises the data lineage problem and presents algorithms to generate lineage data in relational databases. 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 [Cui and Widom (2000)]. This work was also extended to deal with general transformations of data sets inside a data warehouse [Cui and Widom (2003)]. Another system that looks at the data lineage problem in a data warehouse context is AutoMed [Fan and Poulovassilis (2003)]. Data lineage is tracked in AutoMed by recording schema transformations. A series of schema transformations is termed a schema transformations pathway. The granularity of this approach depends on the granularity and number of schemas defined in the system.

Buneman et al. (2001) redefines the data lineage problem as "why-provenance" and defines 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. Based on this terminology a formal model of provenance was developed applying to both relational and XML databases. In other work, Buneman et al. (2002) argue for a time-stamped based archiving mechanism for change tracking in contrast to diff-based mechanisms. These mechanisms may not capture the complete provenance of a database because there may be multiple changes between each archive of the database.

2.4.4 Middleware to Support Provenance

There have been several systems developed to provide middleware provenance support to applications. These systems aim to provide a general mechanism for recording and querying provenance for use with multiple applications across domains and beyond the confines of a local machine.

Ruth et al. (2004) presents a system based around the concept of an e-notebook. Each user is required to have an individual e-notebook which can record data and transformations either through connections directly to instruments or via direct input from the user. Data stored in an e-notebook is represented as a DAG and can be shared with other e-notebooks via a peer-to-peer mechanism. A DAG may span multiple e-notebooks to take in account multiple individuals participation in a process. To enable support of trust views and credential tracking each node in a DAG must be digitally signed by the node's creator.

Another system supporting provenance is Scientific Application Middleware (SAM) [Myers et al. (2003b)]. SAM provides facilities for storing and managing records, metadata and semantic relationships and is built on the WebDav standard. Support for provenance is provided through adding metadata to files stored in a SAM repository. SAM is of interest because it does not specify the format of the data or metadata that it handles instead it acts as an open repository.

The Chimera Virtual Data System is a virtual data catalogue, which is defined by a virtual data schema and accessed via a query language [Foster et al. (2002b)]. The schema is divided into three parts a transformation: a derivation and a data object. A transformation represents an executable, a derivation represents the execution of a particular executable, and a data object is the input or output of a derivation. The virtual data language provided by Chimera is used to both describe schema elements and query the data catalogue. Using the virtual data language a user could query the catalogue to retrieve the DAG of transformations that led to a result. The benefit

of using a common description language is that relationships between entities can be extracted without understanding the underlying data. Chimera is designed for Grid applications but does not consider how data is submitted to its catalogue.

Szomszor and Moreau (2003) argued for infrastructure support for recording provenance in Grids and presented a trial implementation of an architecture that was used to demonstrate several mechanisms for handling provenance data after it had been recorded. Their system is based around a workflow enactment engine submitting data to a provenance service. The data submitted is information about the invocation of various web services specified by the executing workflow script.

2.4.5 Conclusion

The literature makes a convincing case for the need to record provenance in computer systems. The domain specific nature of much of the literature is encouraging because it points to user's need for the provenance of computations. Other common threads are that workflow enanctment lends itself to provenance capture and that provenance belongs at the middleware level. The low-level provenance literature provides us a reasonable baseline for captured provenance granularity. Finally, research into database level provenance shows that formal representations can be successfully transferred to working systems.

However, there are some gaps in the literature. A major gap is in the area of security. Most systems fail to address trust and security issues an exception is the e-notebook system [Ruth et al. (2004)]. Although, it fails to address the verifiability of an interaction between actors. Another gap is the emphasis on specific systems to record provenance (mostly for a domain) instead of an implementation-independent specification.

Chapter 3

Current Work

This chapter presents the work accomplished in the first nine months of research. We first define the provenance recording problem and then we present a system architecture and protocol for recording provenance adapted from a published paper [Groth et al. (2004)].

3.1 The Problem

At the beginning of this report, we outlined the need for provenance in a variety of different areas as well as the necessity for provenance recording in service-oriented architectures, specifically in a Grid scenario. Although some of the literature has begun to address provenance recording in Grids, the systems presented are either bespoke or ad-hoc solutions. Unfortunately, this means that such provenance systems cannot interoperate. Therefore, incompatibility of components prevents provenance from being shared. Furthermore, the absence of components for recording provenance makes the development of applications requiring provenance recording more complicated and onerous. Typically, applications developers must re-implement provenance components.

Another drawback to current bespoke solutions is the inability for provenance to be shared and trusted by different parties. Even with the availability of provenance-related software components, the goal of sharing trusted provenance information will not be achieved. To address this problem, standards should be developed for how provenance information is recorded, represented, and accessed. Such standards would allow provenance to be shared across applications, provenance components, and Grids, making provenance information more accessible and valuable. In summary, the paucity of standards, components, and techniques for recording provenance is a problem that needs to be addressed by the Grid community. This work is a first step towards addressing these problems.

3.2 Requirements

The first step in determining the requirements placed on a provenance recording system is defining what kind of provenance information the system should support. In the context of a service-oriented architecture, we have identified two kinds of provenance information that a system should record. These kinds of provenance arose from basic requirements gathering in the Provenance-Aware Service Oriented Architecture project (http://www.pasoa.org): (1) provenance about the interaction between actors, which supports execution and data provenance, and (2) provenance information from each actor, which supports knowledge provenance.

With these two kinds of provenance information as background, there are a number of requirements that a provenance recording system needs to address, which we now enumerate.

1. Trust It is vital that a provenance recording system be trustworthy for a wide range of users. In the case of recording provenance about a client-service interaction, both the client and service must trust that the system maintains an accurate representation of their interaction. In the case where an entity is recording provenance about itself with the system, the entity must trust that the system will faithfully maintain that data. Users of provenance information captured by the system must trust the system to provide them with correct information. In contrast, providers of provenance information must trust that the system only supply recorded information to authorised users.

2. Preservation A provenance recording system should have the ability to maintain provenance information for an extended period of time. This is vital for applications run in the VO context because even after a VO disbands, provenance will typically need to be maintained. A good example of the need for preservation is the aircraft simulation application discussed earlier. A VO may be employed to run an aircraft simulation but the provenance about that simulation needs to be preserved much longer than the VO's existence.

3. Security It is natural expectation that a provenance recording system be secure. There are three aspects to security that a system should address. First, the recording process itself must be secure: there should be no way for a malicious entity to effect the provenance recorded by the system. This type of security could be provided through mutual-authentication techniques. The second aspect of security is non-repudiation: the system must be sure the entities recording provenance information cannot deny the fact that they are responsible for that information. This type of security is vital in allowing for dynamic VOs, especially when members of a VO may have a competitive relationship, such as in the case where provenance is used for billing purposes. Lastly, the stored provenance data must be kept confidential. Only those entities authorized to

access provenance should be have access to it. Even though this level of security should be provided by a system, it is not necessary for all scenarios.

4. Scalability Given the large amounts of data that Grid applications handle, such as in the processing of data from the Large Hadron Collider, it is necessary that a provenance recording system be scalable. Another reason for scalability is that provenance information may be larger than the output data of an application. For example, in a service-oriented architecture with provenance recording, it may be useful to maintain the input and output data of each service invocation, whereas in an architecture without provenance recording this data can normally be discarded. Finally, a system must be scalable in terms of the number of invocations it can handle.

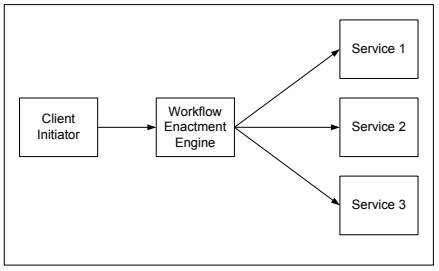
5. Generality Grids are designed to support a wide variety of applications, therefore, a provenance recording system should be general enough to record provenance from these varying applications. From the recording perspective, this requirement is fairly straightforward, the real challenge arises when trying to reason over and use stored provenance. It is important that a provenance recording system takes into account the challenges faced by the users of provenance while still keeping its generality.

6. Customisability To allow for more application specific use of provenance information, a provenance recording system should allow for customisation. For example, the system should allow for constraints to be placed on the type of provenance recorded, maybe based on the schema of the provenance. Other aspects of customisability include security level, time constraints on when recording can take place, and the granularity of provenance to be recorded.

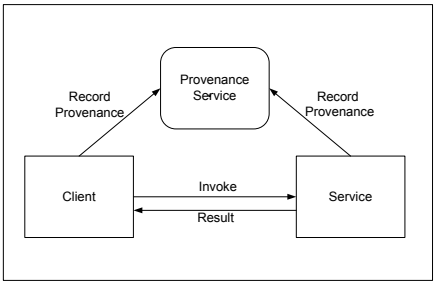
With these requirements in mind, we now detail our system for recording provenance in a SOA.

3.3 SOA Provenance Recording

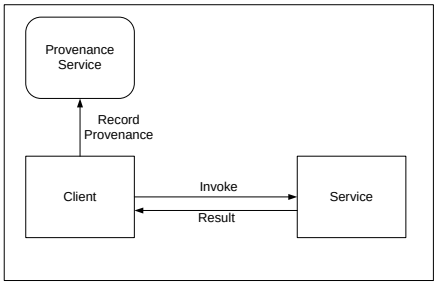
Figure 3.1(a) shows a typical workflow based service-oriented architecture. A client initiator invokes a workflow enactment engine which, in turn, invokes various services based on the workflow specified by the initiator. In essence, the architecture can be broken down into two types of actors: clients who invoke services and services that receive invocations and return results. An example of such an architecture is myGrid [Zhao et al. (2003)]. Provenance is captured in myGrid via the workflow enactment engine, acting as a client, recording provenance information (inputs and outputs of services, time stamps, mime type information, etc.) in a provenance repository. The repository is controlled by the same organisation as the workflow enactment engine. Figure 3.1(c) represents myGrid's approach to provenance recording.



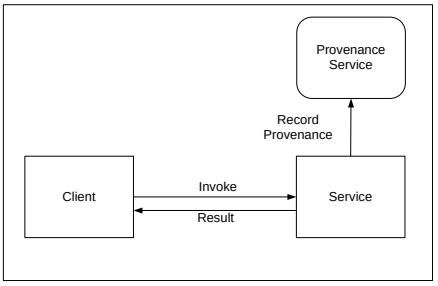
(a) Typical workflow based architecture



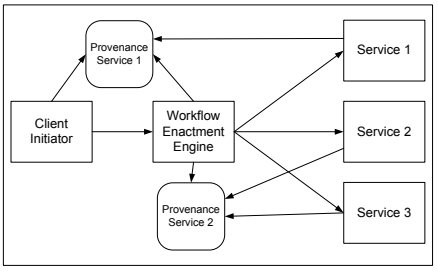
(b) The interaction between a client, service and provenance service



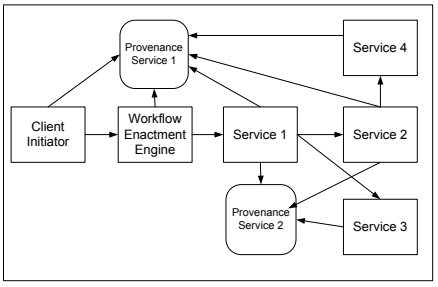
(c) Provenance recording from the client



(d) Proveance recording from the service



(e) Workflow based architecture with third party provenance services



(f) Architecture with third party provenance services and services invoking other services

FIGURE 3.1: Architecture diagrams

There are several problems associated with such an architecture in terms of provenance recording. One problem is that invoked services and users of provenance must trust that the information maintained in the provenance repository is accurate or complete. Because the client's organisation is in total of control of what provenance is stored and how it is maintained, the client's repository could expose, modify, or delete provenance at any time. Related to this problem is the requirement for preservation. In a myGrid like architecture, preservation falls completely to the client's provenance repository. It is easy to imagine a situation where the client's organisation deletes provenance, for example, because of space concerns, that may have been vital to an outside user at some later date.

Third Party Provenance Services To address these concerns our architecture introduces third party provenance services. In the case of the trust problem, neither the client nor the service need to trust the other to maintain accurate provenance information about an interaction when recording provenance in a third party. They only need to trust the third party provenance service that they mutually agree upon. Third party provenance services are also better suited to support preservation: by placing the burden of preservation on the provenance service, neither clients nor services have to maintain provenance information beyond the scope of any given application run.

The main difference between an architecture with third party provenance services and myGrid-like architectures is ownership. The shortcomings that arise in our architecture are due to the provenance information not being owned by the actors that produce it. For example, a client and service might not be able to agree on a common provenance service. The provenance service also may not be able to provide performance characteristics that actors need when recording provenance or that users need when accessing it. Finally, neither the client nor the service has direct control over the maintenance of possible valuable provenance information. However, our architecture could be adapted to support a myGrid-like scenario by placing the provenance service under the organisational control of the client. We now discuss our architecture for capturing client-service interactions using third party provenance services.

A Triangle of Interaction In contrast to the myGrid architecture, where only the client submits provenance information, our architecture requires that for each interaction between a client and service, consisting of a negotiation between parties, an invocation and a result, each party is required to submit their view of the interaction to a common provenance service. This 'triangular' pattern of provenance recording is shown in Figure 3.1(b). Later in this report, we describe the Provenance Recording Protocol that governs the interaction of the actors in this triangle.

The case for recording two views The triangular nature of the interaction between the client, service and provenance service stems from the requirement that both the client and service submit their view of their interaction to the provenance service. At its most basic, this view consists of the input and output data of the service. Each party submits to the provenance service all the data that it sends and receives during an interaction. This requirement is vital for recording an accurate picture of a client-service interaction because it allows the provenance service to verify an interaction by checking that the views of the two parties agree. Without this requirement, several problems could arise, which are related to the problem of trust in myGrid-like architectures.

For example, should the client be the only party recording the interaction in the provenance service, see Figure 3.1(c), the service would completely be dependent on the client to submit provenance. In fact, without the submission of provenance from the service, there would be no evidence that the client invoked the service should the client choose

not to record the interaction. This is a major problem for provenance based billing because there exists no record of a service invocation, therefore, a client could use a service for free. In our system, the provenance service would know that a service was invoked because the service submits that information. The same problem would also exist in the case where the service was the only party submitting to the provenance service, see Figure 3.1(d). For example, a service could hide the fact that it was invoked, which, in an auditing scenario, would allow a service to deny its involvement in a particular VO being investigated. We note that the requirement that both parties submit their views does not prevent collusion between parties, but it does allow the provenance service to detect when the two parties disagree about the record of an interaction.

Multiple provenance servers Although a client and service are required to share a common provenance service for an interaction, different provenance services can be used for different interactions even between the same client and service. Figure 3.1(e) shows a typical workflow based architecture with multiple provenance services. This architecture is assembled from the 'triangle' pattern pictured in Figure 3.1(b). One benefit of multiple provenance services is the elimination of a central point. Another benefit is that demand is spread across multiple services increasing the architecture's robustness. Lastly, the use of multiple provenance services also promotes a competitive environment, in which clients and services can choose which provenance service best suits their needs in terms of factors such as trust, reliability and possibly cost. There are some disadvantages in storing provenance across multiple services, including the possibility of losing some provenance information due to a failed provenance service and the need to query multiple service to rebuild the complete provenance of an enacted workflow.

Advanced Architecture Support As well as supporting typical workflow enactment based architectures, our system supports more advanced architectures like the one shown in Figure 3.1(f). In this architecture, services invoke other services to produce a result, in contrast to the previous architectures where the workflow enactment engine was the only actor invoking services. In order to maintain provenance across provenance services in such an architecture, a client needs to inform the original provenance service when it uses a new provenance service. For example, in Figure 3.1(f), Service 1 must inform Provenance Service 1 that it has used Provenance Service 2 when invoking Service 3. This creates a link between provenance services that can be followed in order to provide the entire provenance trace for an application started by a client initiator. The support for advanced architectures is enabled because they can be reduced to a series of the simple 'triangular' patterns of Figure 3.1(b).

Actor Provenance We have mainly discussed how our system supports the recording of information about the interaction between actors in a service-oriented architecture. Our system also supports the submission of provenance information about each actor. This information could include anything from the workflow that an enactment engine runs to the disk and processing power a service used in a computation. Typically,

this information can only be provided by the actor itself, so it cannot be verified like interaction provenance. We use a simple mechanism to store actor-centric provenance by tying it to interaction provenance. The basis for our provenance recording system is the interaction between one client, one service and one provenance service. This interaction is specified by the Provenance Recording Protocol, which is presented next.

Name	Notation	Fields
propose	pro	ActivityId, PSAllowedList, Extra
reply	reply	ActivityId, PSAccepted, Extra
invoke	inv	ActivityId, Data, Extra
result	res	ActivityId, Data, Extra
record negotiation	rec_neg	ActivityId, PSAllowedList, PSAccepted, Extra
record negotiation acknowledgement	rec_neg_ack	ActivityId
record invocation	rec_inv	ActivityId, Extra, Data
record invocation acknowledgement	rec_inv_ack	ActivityId
record result	rec_res	ActivityId, Data
record result acknowledgement	rec_res_ack	ActivityId
submission finished	sf	ActivityId, NumOfMessages
submission finished acknowledgement	sf_ack	ActivityId
additional provenance	ap	ActivityId, Extra
additional provenance acknowledgement	ap_ack	ActivityId

Figure 3.2: Protocol messages, their formal notation and message parameters.

3.4 Provenance Recording Protocol

In order to capture both interaction provenance and actor provenance in a third party, we have developed a novel protocol for recording provenance. The Provenance Recording Protocol (PReP) is a four phase protocol consisting of negotiation, invocation, provenance recording and termination phases. The negotiation phase allows a client and service to agree on a provenance service to store a trace of their interaction. After this phase, the protocol enters the invocation phase, during which a client invokes a service and receives a result. Asynchronously, in the provenance recording phase, both the client and service submit their input and output data to the provenance service. When all data has been received by the provenance service, the termination phase occurs. After discussing the messages and their parameters used by PReP, we consider the four phases in detail.

We model the protocol as an asynchronous message-passing system, in which all communication is expressed by an outbound message followed by a return message. The return message is either a result of the service invocation, a reply from the service during negotiation, or an acknowledgement that the provenance service has received a particular message. Figure 3.2 lists the fourteen messages in our protocol. Although we detail each individual message, an implementation could merge them for performance or other reasons.

These messages can be divided into two groups: those that are between a client and service; and those that are used to interact with the provenance service. Figure 3.3 shows what messages between the different actors. The *propose*, *reply*, *invoke* and *result* messages belong to the first group, while the *record*, *submission finished* and *additional provenance* messages belong to the second. The usage of each message is described in more detail when we present the phases of the protocol. The message parameters shown in Figure 3.2 are detailed below.

The ActivityId parameter identifies one exchange between a client and server. It contains: NonceId, an identifier generated by the client to distinguish between other exchanges with the called service; SessionId, comprising all invocations that pertain to one result (the client originator of Figure 3.1(e) generates this identifier, which must be unique); ThreadId, which allows clients to parse multiple interactions with the same service; Client, which identifies the client; and Service, which identifies the service.

Other parameters are: Data, which contains data exchanged between a client and service; Extra, which is an envelope that can contain other messages related or not to the protocol allowing it to be extended; NumOfMessages, which indicates the total number of messages an entity sends to the provenance service; PSAllowedList, which is a list of approved provenance services; and PSAccepted, which contains a reference to a provenance service that an entity accepts, or a rejection token.

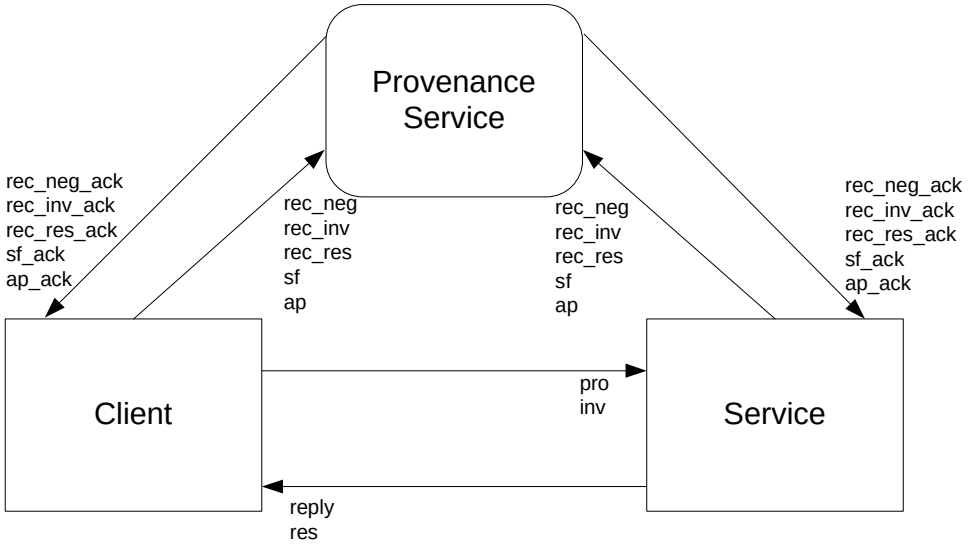


Figure 3.3: The messages exchanged between the client, service and provenance service.

PReP is divided into four phases: negotiation, invocation, provenance recording, and termination, which we now discuss in detail.

Negotiation is the process by which a client and service agree on a provenance service to use. Typically, a client presents a list of provenance services it trusts to the service via a *propose message*. The service then extracts the PSAllowedList from the propose message and selects a provenance service from the list. The service then replies with a *response message* containing the selected provenance service or a rejection in the PSAccepted parameter. Although the negotiation modelled here is simple, with only one request-response, the protocol is extensible through the use of the Extra parameter. Entities can encode more complicated messages into this envelope, providing a means for complex negotiations to take place allowing for more custom provenance recording and advanced negotiations [Lawley et al. (2003)]. A client and service that have already negotiated and agreed on a provenance service might like to skip the negotiation phase of the protocol. Therefore, a message informing the service of the use of a previously agreed provenance service can be enclosed in the Extra envelope of the *invoke message*. However, the provenance service still needs to be informed of the agreement between the service and client via the *record negotiation message*.

Invocation If a client has successfully negotiated with a service, it can then invoke the service and receive a result via the *invoke message* and *result message*. We have tried to limit the impact of PReP on normal invocation, the only extra parameters required to be sent are the ActivityId and the Extra envelope. The ActivityId is necessary to identify the exchange in relation to the provenance stored in the service, while the Extra envelope allows the protocol to be used without a negotiation phase and for later protocol extension.

Provenance Recording is the key phase of the protocol. As discussed previously, the client and service are required to submit copies of all their sent and received messages to the provenance service. Submission is done through the various record messages with both the client and service sending *record negotiation*, *record invocation* and *record result* messages. Acknowledgement messages then inform the sender that each message has been received by the provenance service. The *record negotiation message* contains the list of provenance services (PSAllowedList), the client proposed, and the provenance service accepted (PSAccepted) by the service. The *record invocation* and *record result* messages together contain the entire data transmitted between the client and service from the perspective of both entities. The requirement that all data be submitted allows the provenance service to have a complete view of the exchange. In order not to delay service invocation, the submission process can be done in a totally asynchronous fashion; for example, the client could send a *record invocation message* to the provenance service before or after receiving a *result message* from the service. Although the protocol requires two copies of an invocation to be sent to the provenance service, it minimises

any performance penalties through the use of asynchronous submission while adding the benefit of capturing the complete provenance of an exchange.

We cater for actor provenance instead of interaction provenance by the *additional provenance message*. With this message, an actor can record provenance about itself or other actors in the architecture by enclosing in the EXTRA envelope whatever information is pertinent. An important use of this capability is the linking of provenance records across provenance services as described in Section 3. We note that there are no constraints on the data that can be submitted to the provenance service, allowing a wide variety of applications to be supported.

Termination The final phase of the protocol is termination. The protocol terminates when the provenance service has received all expected messages from both the client and the service. The client and service are notified of termination through the acknowledgement to the *submission finished message*, which is returned after all expected messages are received from the client and service. The number of expected messages is determined by the NUMOFMESSAGES parameter in the *submission finished message*. Because of the asynchronous nature of the protocol, the *submission finished message* can be sent any time after the negotiation phase.

3.5 Actors

We now consider how the provenance service, service and client act in response to the messages they send and receive. To understand the actions of these actors, we use complementary formalisation techniques, chosen because of the nature of the actors involved. First, we represent the provenance service as an abstract state machine (ASM). Second, we use a 3D state diagram to show the possible responses of the client and service. Both techniques assume asynchronous message passing. The importance of the internal functionality of the provenance service lends itself to an ASM formalisation whereas, given the importance of the external interactions of the client and service, a state transition diagram formalisation is more appropriate. We begin with the provenance service.

The Provenance Service plays the central role in PReP. As far as recording is concerned, its interaction with the outside world is simple: it receives messages and sends acknowledgements. It does not initiate any communication and its purpose is to simply store messages. By formalising the provenance service, we can explain how the accumulation of messages dictates its actions.

To detail these actions, we model the provenance service as an ASM whose behaviour is governed by a set of transitions it is allowed to perform. The notation allows for any form of transition with no limits on complexity or granularity and has been used previously to describe a distributed reference counting algorithm [Moreau and Duprat

$A = \{a_1, a_2, \ldots, a_n\}$ (Set of Actors)
CLIENT $\subset A$ (Set of Clients is a subset of Actors)
SERVICE $\subset A$ (Set of Services is a subset of Actors)
ACTIVITYID = SESSIONID $\times$ NONCEID $\times$ THREADID $\times$ CLIENT $\times$ SERVICE (Activity Identification)

rec_neg:ACTIVITYID $\times$ PSALLOWEDLIST $\times$ PSACCEPTED $\times$ EXTRA $\rightarrow RN$ (Negotiation Messages)
rec_inv:ACTIVITYID $\times$ EXTRA $\times$ DATA $\rightarrow RI$ (Invocation Messages)
rec_res:ACTIVITYID $\times$ EXTRA $\times$ DATA $\rightarrow RR$ (Result Messages)
sf:ACTIVITYID $\times$ NUMOFMESSAGES $\rightarrow SF$ (Submission Finished Messages)
ap:ACTIVITYID $\times$ EXTRA $\rightarrow AP$ (Additional Provenance Messages)

$\mathcal{M} = RN \cup RI \cup RR \cup SF \cup AP$ (Messages)
Each message has a corresponding acknowledgement message, which is also a part of $\mathcal{M}$.

$\mathcal{K} = A \times A \rightarrow Bag(\mathcal{M})$ (Set of Message Bags)

Charateristic Variables:
$a \in Actor, k \in \mathcal{K}$, $ai \in$ ACTIVITYID, $rec\_neg \in RN, rec\_inv \in RI$, $rec\_res \in RR, sf \in SF, ap \in AP$, $e \in$ EXTRA, $psal \in$ PSALLOWEDLIST, $psa \in$ PSACCEPTED, $d \in$ DATA, $nid \in$ NONCEID, $tid \in$ THREADID, $client \in$ CLIENT, $service \in$ SERVICE, $nm \in$ NUMOFMESSAGES

If $ai = \langle sid, nid, tid, ts, client, service\rangle$ then
$ai.sid = sid, ai.nid = nid, ai.tid = tid, ai.ts = ts, ai.client = client, ai.service = service$
If $sf = \langle ai, nm\rangle$ then $sf.ai = ai, sf.nm = nm$

FIGURE 3.4: System State Space

(2001)] and a distributed directory service and message router for mobile agents [Moreau (2001)].

APL	=	$\mathbb{P}(AP)$	(Set of Sets of Additional Provenance Messages)
CN	=	RN	(Client Negotiation Messages)
CI	=	RI	(Client Invocation Messages)
CR	=	RR	(Client Result Messages)
CSF	=	SF	(Client Submission Finished Messages)
SN	=	RN	(Service Negotiation Messages)
SI	=	RI	(Service Invocation Messages)
SR	=	RR	(Service Result Messages)
SSF	=	SF	(Service Submission Finished Messages)
CS	=	ACTIVITYID $\rightarrow CN \times CI \times CR \times CSF \times APL$	(Client Records, a Client Message Store)
SS	=	ACTIVITYID $\rightarrow SN \times SI \times SR \times SSF \times APL$	(Service Records, Service Message Store)
PS	=	$CS \times SS$	(Set of Provenance Services)
		$PS \subset A$	(Set of Provenance Services is a subset of Actors)

Characteristic variables:
$p = \langle client\_T, service\_T\rangle, ps \in PS$, $apl \in APL, client\_T \in CS, service\_T \in SS$,

If $service\_T[ai] = \langle rec\_neg, rec\_inv, rec\_res, sf, apl\rangle$ then
$service\_T[ai].rec\_neg = rec\_neg$,
$service\_T[ai].rec\_inv = rec\_inv$,
$service\_T[ai].rec\_res = rec\_res, service\_T[ai].sf = sf$,
$service\_T[ai].apl = apl$
The same notation applies for $client\_T[ai]$.

Initial State:
$p_i = \langle client\_T_i, service\_T_i, k_i\rangle$, $client\_T_i = ai \rightarrow \emptyset$, $service\_T_i = ai \rightarrow \emptyset, k_i = \emptyset$

FIGURE 3.5: Provenance Service State Space

The ASM State Space The state space of the provenance service's ASM is shown in Figure 3.5 and Figure 3.4. The System State Space models the space of messages and message channels that actors in the system use to communicate, whereas the Provenance Service State Space models the internal state space of provenance services. We first describe the System State Space.

The System State Space considers a finite number of actors, A, which exchange messages. The set of messages is defined as the union of the sets RN, RI, RR, SF, and AP. All of these sets, excluding AP, are in turn defined by inductive types, whose constructors are named according to the messages in Figure 3.2. Communication between actors is modelled as a set of communication channels represented as bags of messages between pairs of actors.

An instance of a provenance store, p, is a tuple that consists of an element from the Client Message Store, CS, and an element from the Service Message Store, SS. These two tables are defined as functions whose argument is of type ActivityId and consist of sets of messages that are from either the client or the service. On the other hand, AP is a set that contains all of the *additional provenance messages*. Note that SS and CS are not defined using AP but with APL, the power set of AP. Informally, this shows that any number of *additional provenance messages* can be stored per ActivityId.

Given the state space, the ASM is described by an initial state and a set of transitions. Figure 3.5 contains the initial state space, which can be summarised as empty client and service message stores. We use an arrow notation for a function taking an argument and returning a result. Therefore, $client\_T_i$ and $service\_T_i$ take an ActivityId as an argument and return an empty state.

The ASM Rules The transitions of the ASM are described through rules with the following form:

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

Rules are identified by their name and a number of parameters that the rule operates over. Any number of conditions must be met in order for a rule to be fireable. A new state is achieved after applying all the pseudo-statements and functions to the state that met the conditions of the rule. The execution of a rule is atomic, so that no other rule may interrupt or interleave with an executing rule. This maintains the consistency of the ASM. A rule may contain *send*, *receive* or 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 $receive(a_1, 1_2, m)$ removes the message. A rule may also contain the *complete* function, which checks that none of the fields accessed by an ActivityId are null. Formally, the pseudo-statements are defined as follows.

$receive\_neg(ps, a, ai, psal, psa, e)$:
  $\mathsf{rec\_neg}(ai, psal, psa, e) \in \mathcal{K}(ps, a)$
$\rightarrow$ {
  $receive(ps, a, \mathsf{rec\_neg}(ai, psal, psa, e))$;
  $if(a = ai.client), then$
    $client\_T[ai].rec\_neg := \mathsf{rec\_neg}(ai, psal, psa, e)$;
  $elif(a = ai.service), then$
    $service\_T[ai].rec\_neg := \mathsf{rec\_neg}(ai, psal, psa, e)$;
  $send(ps, a, \text{rec\_neg\_ack}(ai))$;
  if $complete[ai]$, then
    $send(ps, a, \mathsf{sf\_ack}(ai))$;
}

$receive\_inv(ps, a, ai, e, d)$:
  $\mathsf{rec\_inv}(ai, e, d) \in \mathcal{K}(ps, a)$
$\rightarrow$ {
  $receive(ps, a, \mathsf{rec\_inv}(ai, e, d))$;
  $if(a = ai.client), then$
    $client\_T[ai].inv := \mathsf{rec\_inv}(ai, e, d)$;
  $elif(a = ai.service), then$
    $service\_T[ai].inv := \mathsf{rec\_inv}(ai, e, d)$;
  $send(ps, a, \mathsf{rec\_inv\_ack}(ai))$;
  if $complete[ai]$, then
    $send(ps, a\mathsf{sf\_ack}(ai))$;
}

$receive\_res(ps, a, ai, e, d)$:
  $\mathsf{rec\_res}(ai, e, d) \in \mathcal{K}(ps, a)$
$\rightarrow$ {
  $receive(ps, a, \mathsf{rec\_res}(ai, e, d))$;
  $if(a = ai.client), then$
    $client\_T[ai].res := \mathsf{rec\_res}(ai, e, d)$;
  $elif(a = ai.service), then$
    $service\_T[ai].res := \mathsf{rec\_res}(ai, e, d)$;
  $send(ps, a, \mathsf{rec\_res\_ack}(ai))$;
  if $complete[ai]$, then
    $send(ps, a, \mathsf{sf\_ack}(ai))$;
}

$additional\_provenance(ps, a, ai, e)$:
  $\mathsf{ap}(ai, e) \in \mathcal{K}(ps, a)$
$\rightarrow$ {
  $receive(ps, a, \mathsf{ap}(ai, e))$;
  $if(a = ai.client), then$
    $client\_T[ai].apl := client\_T[ai].apl \cup \{\mathsf{ap}(ai, e)\}$;
  $elif(a = ai.service), then$
    $service\_T[ai].apl := client\_T[ai].apl \cup \{\mathsf{ap}(ai, e)\}$;
  $send(ps, a, \mathsf{ap\_ack}(ai))$;
  if $complete[ai]$, then
    $send(ps, a, \mathsf{sf\_ack}(ai))$;
  }

$submission\_finished(p, x, ai, nm)$:
  $\mathsf{sf}(ai, nm) \in \mathcal{K}(ps, a)$
$\rightarrow$ {
  $receive(ps, a, \mathsf{sf}(ai, nm))$;
  $if(a = ai.client), then$
    $client\_T[ai].sf := \mathsf{sf}(ai, nm)$;
  $elif(a = ai.service), then$
    $service\_T[ai].sf := \mathsf{sf}(ai, nm)$;
  $send(ps, a, \mathsf{sf\_ack}(ai))$;
  if $complete[ai]$, then
    $send(\mathsf{sf\_ack}(ai), x, p)$;
}

FIGURE 3.6: The abstract state machine's rules

- If k is the set of message channels of a state $\langle \ldots, k \rangle$, then the expression $send(a_1, a_2, m)$ denotes the state $\langle \ldots, k' \rangle$, where [1] $k'(a_1, a_2) = k(a_1, a_2) \oplus \{m\}$, and $k'(a_i, a_j) = k(a_i, a_j), \forall (a_i, a_j) \neq (a_1, a_2)$.

- If k is the set of message channels of a state $\langle \ldots, k \rangle$, then the expression $receive(a_1, a_2, m)$ denotes the state $\langle \ldots, k' \rangle$, where $k'(a_1, a_2) = k(a_1, a_2) \ominus \{m\}$, 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 $table\_T[ai].y := V$ denotes the state $\langle \ldots, table\_T', \ldots \rangle$, where $table\_T[ai].x = table\_T'[ai].x$ if $x \neq y$, and $table\_T'[ai].y = V$.

Likewise, the function *complete* is defined as follows:

- If $client\_T$ and $service\_T$ are components of a state $\langle client\_T, service\_T, \ldots \rangle$, then the expression $complete[ai]$ evaluates to true if $client\_T[ai].rec\_neg \neq \bot$, $client\_T[ai].rec\_inv \neq \bot$, $client\_T[ai].rec\_res \neq \bot$, $client\_T[ai].sf \neq \bot$, $client\_T[ai].sf.nm - 4 = |client\_T[ai].apl|$ and $service\_T[ai].rec\_neg \neq \bot$, $service\_T[ai].rec\_inv \neq \bot$, $service\_T[ai].rec\_res \neq \bot$, $service\_T[ai].sf \neq \bot$, $service\_T[ai].sf.nm - 4 = |service\_T[ai].apl|$.

Figure 3.6 shows the ASM's transition rules. *receive_neg* is the transition rule for the receipt of a record negotiation message. It specifies the behaviour of the provenance service when receiving, from actor a, a rec_neg message containing: an ActivityId, a PSAllowedList, a PSAccepted parameter and an Extra envelope. The condition placed on the rule states that for the rule to fire there must be a rec_neg message, which is part of the communication channel ($\mathcal{K}$) between a provenance service, ps, and a. If this condition is satisfied, the message is consumed using the *receive* pseudo-statement. The rule then determines whether a is a client or service and puts the rec_neg message in the correct field of the appropriate table. After this table update, an rec_neg_ack is sent using the *send* pseudo-statement, which places the given message onto the communication channel between the specified entities. Finally, the *complete* functions tests to see if all messages have been received from both the client and the service. If all messages have been received, the *submission finished acknowledgement message* can be sent. The other four transitions listed follow the same pattern as the *receive_neg* rule, consuming a message and placing it into the the correct field of the appropriate table.

The Client and Service We now formalise the actions of the client and the service. In this case, we have chosen not to use the ASM formalism because we have no knowledge of the decision algorithm a service would use when selecting a provenance service from the list proposed by the client. Furthermore, we want developers to be free to experiment

[1] We use the operators $\oplus$ and $\ominus$ to denote union and difference on bags.

with any sort of algorithm they deem best. However, we still want to formally investigate the actions of the client and service in response to PReP, so we represent the two entities with a 3D state transition diagram, which offers an intuitive yet rigorous means to describe the actions of the client and service based on sent and received messages.

Figure 3.7 shows the state transition diagram for both the client and service. It contains all the possible states of a client or service with regard to the PReP. The diagram can either be read from the point of view the client or that of the service. Depending on which view is chosen, one chooses the corresponding transition key to read the diagram. The transitions numbered six through fourteen are common to both the client and service. Transitions between states are only permitted when messages are sent or received by the actor. These transitions are identified by the transition keys in the diagram. For example, transition (4) is the receipt of a *result message* and transition (5) is the sending of an *invoke message* in the case of the client. The diagram shows all possible ways that a client or service could send and receive messages. To make make the formalisation clearer we have developed a 3D model of the diagram using the Virtual Reality Modelling Language, which allows the reader to change perspectives when reading the diagram.

We believe that these formalisations provide a firm basis for developers to implement the protocol. The ASM and 3D state transition diagram allow developers to understand the interaction of the client, service, and provenance service without prescribing a particular implementation technique. This gives developers the opportunity to choose the implementation mechanisms that fit their needs.

3.6 Properties

Given the above formal representations of the client, service and provenance service, we now can show an important property of PReP, namely, liveness. In distributed systems, it is common to refer to safety and liveness properties, to denote, respectively, that nothing bad will happen and that something good will eventually happen. In the case of PReP, liveness is that, ultimately, the *submission finished acknowledgement* message will be sent to both the client and the service.

To show that the protocol is indeed live, we first make some assumptions about the system implementing PReP. We assume that the client and service are live i.e. that they will eventually send and receive all the messages designated in the protocol. This entails that for any given invocation a service will always respond. Finally, we assume that all communication channels are live. Therefore, all sent messages will be delivered to the addressed party.

Given these assumptions, we now show that both the client and service will eventually end their interaction with the provenance service for one invocation of the service.

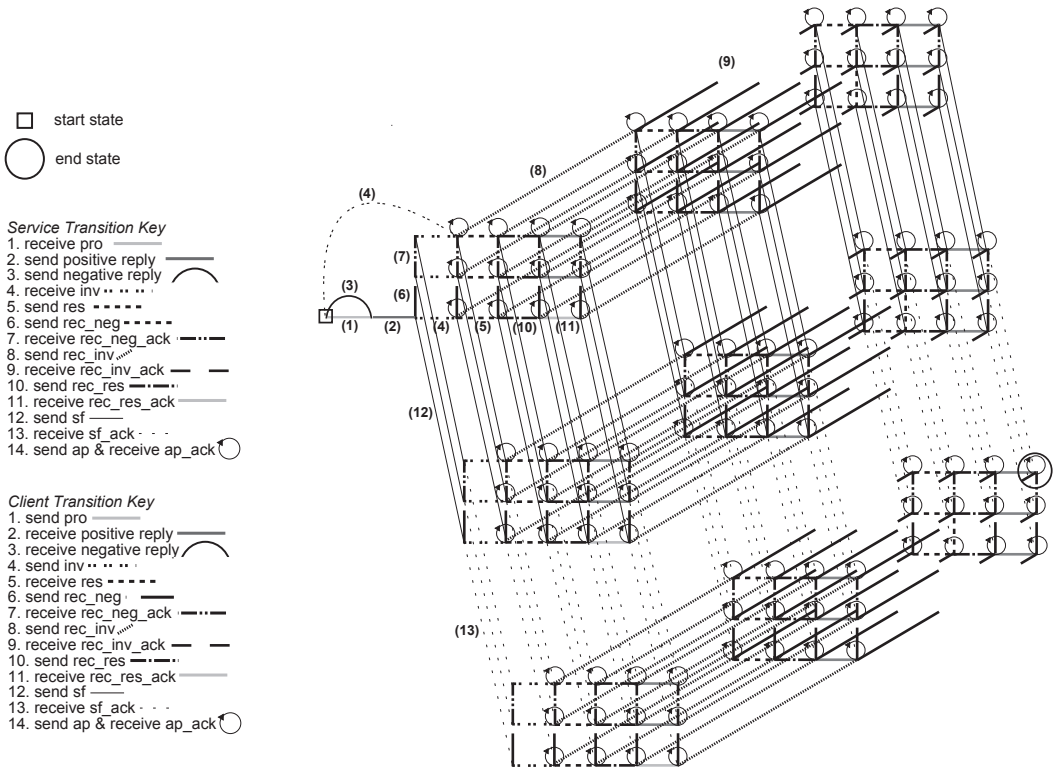


FIGURE 3.7: State transition diagram for both the client and service

Lemma 1 (Termination)
Given a finite number of exchanged messages, the actions of the client and service in relation to PReP will terminate for one invocation of a service.

Proof
Figure 3.7 shows, by definition, the actions of the client and service in relation to PReP for one invocation of a service. We then derive the assumption that there are a finite number of *additional provenance messages*, because the *submission finished message* requires that a finite number of messages be specified. Next, we can determine a bound on the number of messages a client or service will exchange. Excluding *additional provenance messages*, we calculate this bound by enumerating all paths from the start state to the end state in the graph and selecting the longest, which is twelve transitions i.e. messages long. Given this fixed bound and a finite number of *additional provenance messages*, the client and service will reach the end state shown in the graph and terminate.

Lemma 2 (Completeness)
A provenance service can determine when it has a complete record of a service invocation.

Proof
We define a complete record as the function *complete* evaluating to true. An invocation

is identified by an ActivityId. Therefore, by definition, the provenance service can determine when it has a complete record for a service invocation.

Lemma 3 (PReP satisfies the liveness property)
The *submission finished acknowledgement* message will be sent to both the client and the service.

Proof
Given that both the client and service will terminate (Lemma 1), both actors will send all their messages to the provenance service, which, as represented by the state machine, will fire the appropriate rule corresponding to the receipt of each message. These rules in turn update the state of the record referenced by an ActivityId, *ai* and check for a complete record (Lemma 2) and, if it exists for *ai*, the *submission finished acknowledgement* is sent.

3.7 Conclusion

This section presented the work accomplished in our first nine months of research. Our contributions include a series of requirements, a system architecture, a formalisation of a provenance recording protocol and a proof of liveness for that protocol. These contributions have begun to address the problem of a lack of components, standards and techniques for recording provenance.

Chapter 4

Future Work

The main objective of this work is to address provenance recording in SOAs. There are several possible areas for future work in addressing this objective. Because of the lack of background literature in security for provenance recording, we intend to focus on this area in the upcoming six months. In addition to security, we would like to document the overhead that recording provenance adds to a SOA. Such documentation is vital for the future acceptance of provenance recording. Therefore, our two main objectives for the next six months are: one, to specify a secure version of PReP and two, to benchmark PReP.

There are several tasks that need to be accomplished in order to achieve these two objectives. They are as follows:

Benchmarks:

- Finish implementation of PReP.
- Create a series of test suites that may include scalability and speed tests.
- Run these benchmarks on the PReP implementation.

Security:

- Identify properties that a secure version of PReP should support such as non-repudiation and mutual authentication.
- Revisit PReP to support these properties.
- Describe secure alternatives to PReP and compare these alternatives to our protocol.
- Implement the secure version of PReP.

- Run the same benchmarks on the secure version of PReP.

We project that there will be three main deliverables for this period, an implementation of PReP, a paper on security in PReP and finally the mini-thesis.

4.1 A Timeline

This timeline places the tasks above on a six month time span.

Date	Target
October	Finish implementation of PReP.
	Create test suite and evaluate provenance service.
November	Identify security properties. Describe alternatives to PReP and compare.
December	Revisit PReP to support identified security properties.
January	Begin writing a paper on security in PReP.
	Integrate security into the implementation.
February	Finish writing paper. Re-evaluate the implementation with the test suite.
	Begin writing mini-thesis.
March	Finish writing mini-thesis.

Bibliography

W. Allcock. Gridftp protocol specification. Global Grid Forum Recommendation GFD.20, March 2003.

G. Alonso and A. El 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.

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.

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.

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.

P. Buneman, S. Khanna, K.Tajima, and W.C. Tan. Archiving scientific data. In *Proc. of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM Press, 2002. ISBN 1-58113-497-5.

P. Buneman, S. Khanna, and W.C. Tan. Why and where: A characterization of data provenance. In *Int. Conf. on Databases Theory (ICDT)*, 2001.

E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl, March 2001.

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.

Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *The VLDB Journal*, 12(1):41–58, 2003. ISSN 1066-8888.

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. ISSN 0362-5915.

K. Czajkowski, A. Dan, J. Rofrano, S. Tuecke, and M. Xu. Ws-agreement: Agreement-based grid service management. In *Global Grid Forum*, 2003.

K. Czajkowski, D.F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The ws-resource framework 1.0. http://www.globus.org/wsrf/specs/ws-wsrf.pdf, 2004.

P. P. da Silva, D. L. McGuinness, and R. McCool. Knowledge provenance infrastructure. *Data Engineering Bulletin*, 26(4):26–32, December 2003.

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.

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.

I. Foster. What is the grid? a three point checklist. *GridToday*, July 2002.

I. Foster, E. Alpert, A. Chervenak, B. Drach, C. Kesselman, V. Nefedova, D. Middleton, A. Shoshani, A. Sim, and D. Williams. The earth system grid ii: Turning climate datasets into community resources. In *Proc. of the American Meterologcal Society Conference*, 2001a.

I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.

I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, June 2002a.

I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. In *Int. J. Supercomputer Applications*, pages 15–18, 2001b.

I. Foster, J. Voeckler, M. Wilde, and Y.Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proc. of the 14th Conf. on Scientific and Statistical Database Management*, July 2002b.

Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1st edition edition, 1999.

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.

J.G. Frey, M. Bradley, J.W. Essex, M.B. Hursthouse, S.M. Lewis, M.M. Luck, L. Moreau, D.C. De Roure, M. Surridge, and A. Welsh. *Grid Computing - Making the Global Infrastructure a Reality*, chapter 42, pages 945–962. John Wiley and Sons, 2003.

C. Goble. Position statement: Musings on provenance, workflow and (semantic web) annotations for bioinformatics. In *Workshop on Data Provenance and Derivation*, October 2002.

S. Graham and P.Niblett. Publish-subscribe notification for web services. http://www.ibm.com/developerworks/library/ws-pubsub/, March 2004.

M. Greenwood, C. Goble, R. Stevens, J. Zhao, M. Addis, D. Marvin, L. Moreau, and T. Oinn. Provenance of e-science experiments - experience from bioinformatics. In Simon J Cox, editor, *Proc. UK e-Science All Hands Meeting 2003*, pages 223–226, September 2003. ISBN 1-904425-11-9.

P. Groth, M. Luck, and L. Moreau. Formalising a protocol for recording provenance in grids. In *Proc. of the UK OST e-Science second All Hands Meeting 2004 (AHM'04)*, Nottingham, UK, September 2004.

M. Hondo and C. Kaler. Web services policy framework (wspolicy). http://www.ibm.com/developerworks/library/ws-polfram/, May 2003.

V. Huber. Unicore: A grid computing environment for distributed and parallel computing. In V. Malyshkin, editor, *Parallel Computing Technologies : Proc. 6th International Conference, PaCT 2001*, LNCS, 2001.

D.P. Lanter. Design of a lineage-based meta-data base for gis. *Cartography and Geographic Information Systems*, 18(4):255–261, 1991a.

D.P. 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, 1991b.

D.P. 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.

R. Lawley, K. Decker, M. Luck, T. Payne, and L. Moreau. Automated negotiation for grid notification services. In *Ninth Int. Europar Conf.*, volume 2790 of *LNCS*, pages 384–393. Springer-Verlag, 2003. ISBN ISBN 3-540-40788-X.

A. P. Marathe. Tracing lineage of array data. *J. Intell. Inf. Syst.*, 17(2-3):193–214, 2001. ISSN 0925-9902.

N. Mitra. Soap version 1.2 part 0: Primer. http://www.w3.org/TR/soap12-part0/, 2003.

L. Moreau. Distributed directory service and message router for mobile agents. *Science of Computer Programming*, 39(2-3):249–272, 2001.

L. Moreau and J. Duprat. A construction of distributed reference counting. *Acta Informatica*, 37:563–595, 2001.

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 2003a.

J.D. Myers, A.R. Chappell, M. Elder, A. Geist, and J. Schwidder. Re-integrating the research record. *IEEE Computing in Science & Engineering*, pages 44–50, 2003b.

L. Pearlman, C. Kesselman, S. Gullapalli, B.F. Spencer 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 *Proc. 13th IEEE Symposium on High Performance Distributed Computing (HPDC-13)*, 2004.

Alan Pope. *The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture*. Addison Wesley Publishing Company, December 1997.

P. Ruth, D. Xu, B. K. Bhargava, and F. Regnier. E-notebook middleware for acccountability and reputation based trust in distributed data sharing communities. In *Proc. 2nd Int. Conf. on Trust Management, Oxford, UK*, volume 2995 of *LNCS*. Springer, 2004. ISBN 3-540-21312-0.

J. Schlimmer. Web services eventing (ws-eventing). http://msdn.microsoft.com/webservices/understanding/specs/default.aspx?pull=/library/en-us/dnglobspec/html/ws-eventing.asp, January 2004.

E. Seidel, G. Allen, A. Merzky, and J. Nabrzyski. Gridlab - a grid application toolkit and testbed. *Future Generation Computer Systems*, 18:11, 2002.

M. Szomszor and L. Moreau. Recording and reasoning over data provenance in web and grid services. In *Int. Conf. on Ontologies, Databases and Applications of Semantics*, volume 2888 of *LNCS*, 2003. ISBN 3-540-20498-9.

V. H. K. Tan. *Interaction tracing for mobile agent security*. PhD thesis, University of Southampton, 2004.

A. Vahdat and T. Anderson. Transparent result caching. In *Proc. of the 1998 USENIX Technical Conference*, New Orleans, Louisiana, June 1998.

Jim Waldo. *The Jini Specifications*. Addison-Wesley Professional, 2nd edition, December 2000.

A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Proc. of the 13th International Conference on Data Engineering*, pages 91–102, Birmingham, England, April 1997.

Allison Gyle Woodruff. *Data Lineage and Information Density in Database Visualization.* PhD thesis, University of California at Berkeley, 1998.

J. Zhao, C. Goble, M. Greenwood, C. Wroe, and R. Stevens. Annotating, linking and browsing provenance logs for e-science. In *Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*, October 2003.