

Formalising a protocol for recording provenance in Grids

Paul Groth

Michael Luck

Luc Moreau

School of Electronics and Computer Science
University of Southampton
{pg03r, mml, l.moreau}@ecs.soton.ac.uk

Abstract

Both the scientific and business communities are beginning to rely on Grids as problem-solving mechanisms. These communities also have requirements in terms of provenance. Provenance is the documentation of process and the necessity for it is apparent in fields ranging from medicine to aerospace. To support provenance capture in Grids, we have developed an implementation-independent protocol for the recording of provenance. We describe the protocol in the context of a service-oriented architecture and formalise the entities involved using an abstract state machine or a three-dimensional state transition diagram. Using these techniques we sketch a liveness property for the system.

1 Introduction

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 [3]. By coordinating diverse, distributed computational resources, Grids can be used to address large-scale problems that might otherwise be beyond the scope of local, homogenous systems. The scientific and business communities are beginning to rely on Grids as problem-solving mechanisms. These same communities also have requirements in terms of *provenance*, which is the documentation of the *process* that leads to a result. The necessity for provenance is apparent in a wide range 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. Provenance is even more

important when there is no physical record as in the case of a purely *in-silico* scientific process.

Given the need for provenance information and the emergence of Grids as infrastructure for running major applications, a problem arises that has yet to be fully addressed by the Grid community, namely, how to record provenance in Grids? Some bespoke and ad-hoc solutions have been developed to address the lack of provenance recording capability in Grid infrastructure. Unfortunately, this means that such provenance systems cannot interoperate. This 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.

Another drawback to current bespoke solutions is the inability for provenance to be shared among different parties and trusted by those parties; for example, in the case of auditing of an application after it has run. 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.

Given the length of this paper, we assume the reader is familiar with Grids, Virtual Organisations (VO), Web Services, and service-oriented architectures (SOA). The rest of the paper is organised as follows: Section 2 presents a set of requirements that a provenance recording system should address. Then, Section 3 outlines a design for a provenance recording system in the context of service-oriented architectures. The key element of our system is the Provenance Recording Protocol described in Section 4. In Section 5, the actors in the system are formalised, and the formalisations are then used, in Section 6, to derive some important properties of the system. Finally, Section 7 discusses related work, followed by a conclusion.

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 types of provenance information that a system should record through basic requirements gathering: (1) provenance about the interaction between actors and (2) provenance about each actor.

With these two types of provenance information as background, there are a number of requirements that a provenance recording system needs to address. These requirements include trust, preservation, security, scalability, generality and customisability. 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. They also must trust that the system will be able to preserve provenance for an extended period of time. Given the importance of provenance to some organisations, any system should be secure against internal and external threats. A provenance recording system should also have the ability to deal with both a large amount and a wide variety of provenance information, therefore, it needs to be scalable, general, and customizable.

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

3 SOA Provenance Recording

Figure 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. This is the minimum architecture for which our system should record provenance. In the requirements section, we identified two types of provenance information that a provenance recording system should support, namely, provenance about the interaction between actors and provenance about the actors themselves. To support the gathering of these types of provenance information in the aforementioned architecture, our system introduces trusted third party provenance services as shown in Figure 1(b).

Third Party Provenance Services We see third party provenance services as key to fulfilling the requirements outlined above. In the case of the trust requirement, neither the client nor the service need to trust the other to maintain accurate provenance information about an interaction. They only need to trust the provenance service that they mutually agree upon. Provenance services also 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. Recording provenance across multiple provenance services also eliminates a central point of failure as well as lowering the demand placed on particular provenance services. We imagine federations of provenance services actively managing provenance information in order to maximise storage and network resources, improving scalability. 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.

Advanced Architecture Support As well as supporting typical workflow enactment based architectures, our system supports more advanced architectures like the one shown in Figure 1(d). In order to maintain provenance across provenance services, a client needs to inform the original provenance service when it uses a new provenance service. For example, in Figure 1(d), 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.

A Triangle of Interaction Provenance services are central to recording the interactions between clients and services in our system. 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. Even though our architecture considers multiple actors, the inter-

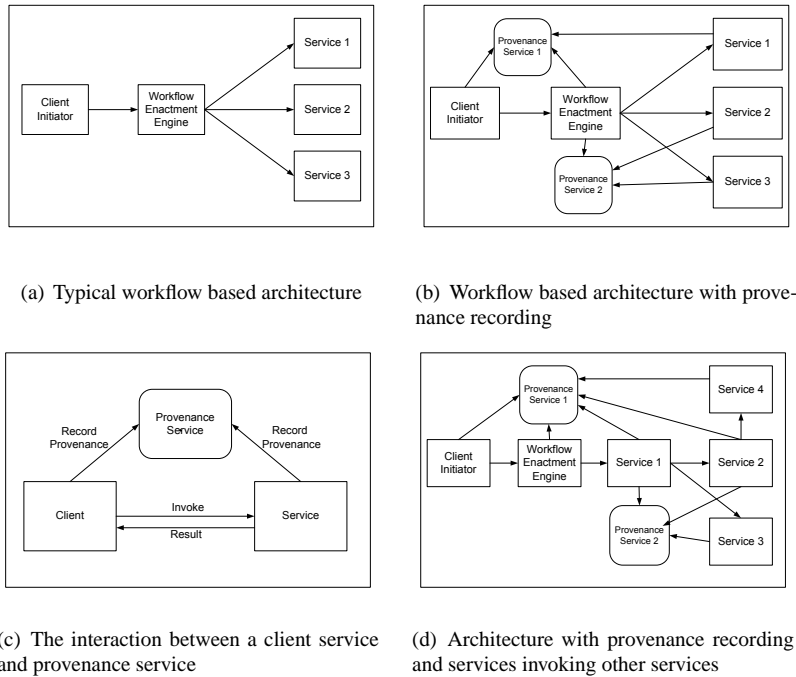


Fig. 1: Architecture diagrams

action between all these actors can be reduced down to a common "triangular" pattern of interaction described above and shown in Figure 1(c). This reduction is possible because our system is based on the simple one-to-one interaction between a client and service that is the foundation of service-oriented architectures. The only additional requirement is that this interaction be recorded in a third party. The interaction of these three actors is governed by the Provenance Recording Protocol, which we will detail later in the paper.

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.

For example, in the case where the client is the only party recording the interaction in the provenance service, the service is completely 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. In our system, the provenance service would know that a service was invoked because the service submits that information. The same problem also exists in the case where the service is the only party submitting to the provenance service. 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.

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.

4 Recording Protocol

PRP is a four phase protocol consisting of negotiation, invocation, provenance recording and termina-

Name	Notation	Fields
<i>propose</i>	pro	ACTIVITYID, PSALLOWEDLIST, EXTRA
<i>reply</i>	reply	ACTIVITYID, PSACCEPTED, EXTRA
<i>invoke</i>	inv	ACTIVITYID, DATA, EXTRA
<i>result</i>	res	ACTIVITYID, DATA, EXTRA
<i>record negotiation</i>	rec.neg	ACTIVITYID, PSALLOWEDLIST, PSACCEPTED, EXTRA
<i>record negotiation acknowledgement</i>	rec.neg_ack	ACTIVITYID
<i>record invocation</i>	rec.inv	ACTIVITYID, EXTRA, DATA
<i>record invocation acknowledgement</i>	rec.inv_ack	ACTIVITYID
<i>record result</i>	rec.res	ACTIVITYID, DATA
<i>record result acknowledgement</i>	rec.res_ack	ACTIVITYID
<i>submission finished</i>	sf	ACTIVITYID, NUMOFMESSAGES
<i>submission finished acknowledgement</i>	sf_ack	ACTIVITYID
<i>additional provenance</i>	ap	ACTIVITYID, EXTRA
<i>additional provenance acknowledgement</i>	ap_ack	ACTIVITYID

Fig. 2: Protocol messages, their formal notation and message parameters.

tion 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. Synchronously, 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 2 lists the fourteen messages in our protocol.

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. 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 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 1(b) generates this identifier, which must be unique); THREADID, which allows clients to parse multiple interactions with the same service; TIMESTAMP, the time when a message was sent, based on the clock of

the sender; 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 unrelated 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.

PReP is divided into four phases: negotiation, invocation, provenance recording, and termination.

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 [4]. 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 storing 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 a blocking call that waits until 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.

5 Actors

ACTIVITYID = SESSIONID × NONCEID × THREADID × TIMESTAMP × CLIENT × SERVICE (Activity Identification)
 Elements of RN, RI, RR and SF are constructed as follows:
 rec_neg : ACTIVITYID × PSALLOWEDLIST × PSACCEPTED × EXTRA → RN (Negotiation Messages)
 rec_inv : ACTIVITYID × EXTRA × DATA → RI (Invocation Messages)
 rec_res : ACTIVITYID × EXTRA × DATA → RR (Result Messages)
 sf : ACTIVITYID × NUMOFMESSAGES → SF (Submission Finished Messages)

AP = {ap₁, ap₂, ..., ap_n} (Additional Provenance Messages)
 M = RN ∪ RI ∪ RR ∪ SF ∪ PU (Messages)
 APL = 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 → CN × CI × CR × CSF × APL (Client Records, a Client Message Store)
 SS = ACTIVITYID → SN × SI × SR × SSF × APL (Service Records, Service Message Store)
 PS = CS × SS (Set of Provenance Services)

Characteristic variables:

p = ⟨client.T, service.T⟩, ai ∈ ACTIVITYID, rec_neg ∈ RN, rec_inv ∈ RI, rec_res ∈ RR, sf ∈ SF, ap ∈ AP, apl ∈ APL, client.T ∈ CS, service.T ∈ SS, e ∈ EXTRA, psal ∈ PSALLOWEDLIST, psa ∈ PSACCEPTED, d ∈ DATA, nid ∈ NONCEID, tid ∈ THREADID, ts ∈ TIMESTAMP, client ∈ CLIENT, service ∈ SERVICE, nm ∈ NUMOFMESSAGES

If ai = ⟨sid, nid, tid, ts, client, service⟩ then
 ai.sid = sid, ai.nid = nid, ai.tid = tid, ai.ts = ts,
 ai.client = client, ai.service = service
 If sf = ⟨ai, nm⟩ then sf.ai = ai, sf.nm = nm
 If service.T[ai] = ⟨rec_neg, rec_inv, rec_res, sf, apl⟩ 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 = ⟨client.T_i, service.T_i⟩, client.T_i = ai → ∅,
 service.T_i = ai → ∅

Fig. 3: Provenance Service State Space

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.

The ASM State Space The state space of the machine is shown in Figure 3. An instance of a provenance service, 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. 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 2. 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 an `ACTIVITYID`.

Given the state space, the ASM is described by an initial state and a set of transitions. Figure 3 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:

$$\begin{aligned}
 &rule_name(v_1, v_2, \dots) : \\
 &\quad condition_1(v_1, v_2, \dots) \wedge condition_2(v_1, v_2, \dots) \wedge \dots \\
 &\rightarrow \{ \\
 &\quad pseudo_statement_1; \\
 &\quad \dots \\
 &\quad pseudo_statement_n; \\
 &\}
 \end{aligned}$$

Rules are identified by their name and a number of variables 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 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.

Figure 5 shows two of 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 entity x , 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 x and p . A communication channel is some point-to-point communication mechanism provided by the implementation. If this condition is satisfied, the message is consumed using the $receive$ pseudo-statement. The rule then determines whether x is a client or service and puts the `rec_neg` message in the correct field of the appropriate table. After this table update, a `rec_neg_ack` is sent using the $send$ pseudo-statement. Finally, the $notify$ rule is called, which 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 test is achieved by invoking the function $complete[ai]$, which checks that none of the fields accessed by ai are null. The other four transitions not 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 entire set of rules can be found at <http://www.pasoa.org/protocol/rules.htm>.

```

receive_neg(p, x, ai, psal, psa, e) :
  rec_neg(ai, psal, psa, e) ∈ K(x, p)
→ {
  receive(rec_neg(ai, psal, psa, e), x, p);
  if(x = ai.client), then
    client.T[ai].rec_neg := rec_neg(ai, psal, psa, e);
  elif(x = ai.service), then
    service.T[ai].rec_neg := rec_neg(ai, psal, psa, e);
  send(rec_neg_ack(ai), x);
  notify(p, ai);
}

notify(p, ai) :
  complete[ai];
→ {
  send(sf_ack(ai), x);
}

```

Fig. 4: The abstract state machine's rules

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 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 formal means to describe the actions of the client and service based on sent and received messages.

Figure 5 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. Transitions between states are only permitted when messages are sent or received by the actor. 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.

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.

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 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. We also assume that the communication channel, \mathcal{K} , is 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. Using the state transition diagram in Figure 5, we can show that this termination property holds. First, we make the assumption that there are a finite number of *additional provenance messages*, although these are represented by cycles

in the graph. With this assumption, 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 send and receive a finite number of messages and then terminate.

In order for the provenance service to be able to send a *submission finished acknowledgement*, it must determine when it has a complete record of the interaction between a client and service. It does this via the *complete* function defined earlier.

We now show that PReP satisfies the liveness property. Given that both the client and service will terminate, 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 invoke the *notify* rule. *Notify* checks for a complete record and, if it exists for ai , the *submission finished acknowledgement* is sent.

7 Related Work

Provenance recording also been investigated in the myGrid (www.mygrid.org.uk) project. The goal of myGrid is to provide a personalised "workbench" for bioinformaticians to perform *in-silico* experiments [5]. Although myGrid allows users to capture provenance data [8], it does not address general architectures or protocols for recording provenance.

Ruth *et al.* present a system for recording provenance in the context of data sharing by scientists [6]. Each scientist has an e-notebook which records and digitally signs any input data or manipulations of data. When the data is shared via peer-to-peer communication, a scientist cannot refute the provenance of the data because of the digital signature process. The goal of the system is to generate a virtual community where scientists are accountable for their data. [6] focuses mainly on the trust aspect of the e-notebook system, rather than the protocols for distributing and storing provenance data.

Some work has focused on data provenance in databases. In [2], Buneman *et al.* make the distinction between *why* (which tuples in a database contribute to a result) provenance and *where* (the location(s) of the source database that contributed to a result) provenance. In [1], a precise definition of provenance is given for both XML hierarchy and re-

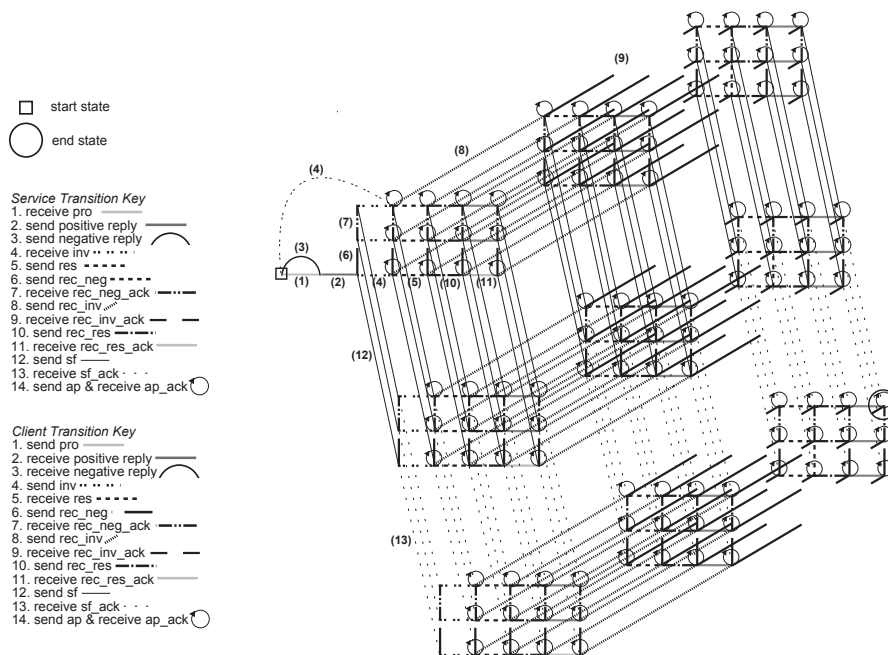


Fig. 5: State transition diagram for the client and service

lational databases.

Szomszor and Moreau in [7] 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. Our work extends [7] in several important ways. First we consider an architecture that allows for provenance services as well as composite services. Secondly, we model an implementation-independent protocol for recording provenance within the context of a service-oriented architecture, whereas, Szomszor and Moreau present an implementation specific service-oriented architecture.

Conclusion The necessity for storing, maintaining and tracking provenance is evident in fields ranging from biology to aerospace. As science and business embrace Grids as a mechanism to achieve their goals, recording provenance will become an ever more important factor in the construction of Grids. The development of common components, protocols, and standards will make this construction process faster, easier, and more interoperable. In this paper, we presented a stepping stone to the development of a common provenance recording system, namely, an implementation-independent protocol for recording provenance, PReP.

Acknowledgements This research is funded in part by EPSRC PASOA project GR/S67623/01.

References

- [1] P. Buneman, S. Khanna, and W.-C. Tan. Data provenance: Some basic issues. In *Foundations of Software Technology and Theoretical Computer Science*, 2000.
- [2] P. Buneman, S. Khanna, and W.-C. Tan. Why and where: A characterization of data provenance. In *Int. Conf. on Databases Theory (ICDT)*, 2001.
- [3] I. Foster. What is the grid? a three point checklist., July 2002.
- [4] 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.
- [5] L. Moreau and et. al. On the use of agents in a bioinformatics grid. In S. Lee, S. Sekguchi, S. Matsuoka, and M. Sato, editors, *Proc. of the 3rd IEEE/ACM CC-GRID'2003 Workshop on Agent Based Cluster and Grid Computing*, pages 653–661, Tokyo, Japan, 2003.
- [6] P. Ruth, D. Xu, B. K. Bhargava, and F. Regnier. E-notebook middleware for accountability 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.
- [7] 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.
- [8] 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.