A Model of Process Documentation to Determine Provenance in Mash-ups

Paul Groth
University of Southern California
Simon Miles
Kings College London
Luc Moreau
University of Southampton

Through technologies such as RSS (Really Simple Syndication), Web Services, and AJAX (Asynchronous JavaScript And XML), the Internet has facilitated the emergence of applications that are composed from a variety of services and data sources. Through tools such as Yahoo Pipes, these "mash-ups" can be composed in a dynamic, just-in-time manner from components provided by multiple institutions (i.e. Google, Amazon, your neighbour). However, when using these applications, it is not apparent where data comes from or how it is processed. Thus, to inspire trust and confidence in mash-ups, it is critical to be able to analyse their processes after the fact. These *trailing analyses*, in particular the determination of the provenance of a result (i.e. the process that led to it), are enabled by *process documentation*, which is documentation of an application's past process created by the components of that application at execution time. In this paper, we define a generic conceptual data model that supports the autonomous creation of attributable, factual process documentation for dynamic multi-institutional applications. The data model is instantiated using two Internet formats, OWL and XML, and is evaluated with respect to questions about the provenance of results generated by a complex bioinformatics mash-up.

Categories and Subject Descriptors: E.1 [**Data**]: Data Structures—*Distributed Data Structures*; J.3 [**Computer Applications**]: Life and Medical Sciences—*Biology and genetics*; D.2.12 [**Software Engineering**]: Interoperability—*Data mapping*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Distributed debugging*; H.2.1 [**Database Management**]: Logical Design—*Data models*

General Terms: Design, Documentation, Standardization

Additional Key Words and Phrases: process, process documentation, provenance, data model, concept maps, mash-ups

1. INTRODUCTION

In both scientific and commercial endeavours, *processes* have become of critical importance. Businesses wish to understand both their physical and software processes better, so as to improve the efficiency, effectiveness and timeliness of their design, manufacturing, or commercial activities. Likewise, understanding processes is fundamental to science: scientists must be able to repeat other scientists' or their own experiments, and they need to analyse and conceive new processes in order to validate their hypotheses.

The Oxford English Dictionary defines a process as a continuous and regular action or succession of actions, taking place or carried on in a definite manner, and leading to the accomplishment of some result. Two different and complementary perspectives can be taken on the notion of process. On the one hand, when con-

sidering a future, prospective process, we may make use of tools to model such a process so that it can be analysed and simulated, in order to understand its properties, and decide whether it meets requirements [Booch 1999]. Being satisfied with a process model, we may then make use of notations that describe the model and that are processable by computers [Lavana et al. 1997]. Such notations can also be optimised or compiled to ensure that computers can execute them efficiently. In other cases, we may not be interested in identifying all the different steps that must be followed, but instead we may specify broad goals, from which we expect a computer system to infer the necessary actions to take, so that goals become satisfied [Dean 1996]. All these process-related activities — modeling, simulating, verifying, analysing, planning, optimising — all have in common that they are undertaken *without ever needing the process to actually take place*.

On the other hand, there is a whole set of activities that pertain to *past processes*, such as understanding how a carcinogen dye ended up in the food chain, finding evidence and building the case to arrest a criminal suspect, and understanding the cause of a natural phenomenon. In computer systems, it is critical to understand what files were modified, stolen, or deleted after a computer was hacked or infected with a virus to allow users to recover and ensure the validity of their information. Likewise, it is important to be able to validate that both credit card transactions and computer-based scientific simulations were conducted according to what the business regulations or experiment design expected.

In this article, instead of looking at future, prospective processes, we describe mechanisms to analyse processes *after they have taken place*. Hence, in such a context, we define a process as a set of past events that led to a result; we also use the term *trailing analysis* to denote the analysis of a process, after the fact, based on the evidence left by its execution. Such trailing analysis is to be contrasted to the kind of analysis performed on programs or workflows, aiming at predicting the properties of future processes.

Trailing analyses are particularly suited to processes that are dynamic, in which actions are executed at runtime based on variable parameters or possible changes in the environment. Such dynamic processes are the essence of a new set of Internet applications that combine data, storage, and computation from a range of sources provided by multiple institutions. Examples of these "mash-ups" include:

—BBC News Maps (www.benedictoneill.com/content/newsmap/) combines RSS feeds provided by the BBC and Google Maps to show the location of news items.

—iSpecies.org combines data from Google Scholar, genome processing from the National Center For Biotechnology Information and images from Yahoo to automatically generate a web page for a given species.

—Salesforce.com now integrates with Google AdWords to allow business to use their customer relationship data in their Web-based ad campaigns.

—Virtual Van (www.thevirtualvan.com) provides interoperability between businesses in a supply chain using Amazon storage, processing, and queuing web services (www.amazon.com/aws/).

These examples show how a variety of domains are using Internet infrastructure (RSS, Web Services, AJAX, XML, and HTTP) to achieve new functionality

with a minimum of duplication. Furthermore, workbenches such as OpenKapow (open kapow.com) and Yahoo Pipes (pipes.yahoo.com) are encouraging the rapid development of these complex dynamic applications. As users become reliant on these applications, they would like to understand where, why, and how their results were produced. Such questions can be answered by performing a trailing analysis that would determine the *provenance* of the result in question, where provenance is defined as the process that led to a result.

To facilitate trailing analysis and particularly the ability to answer questions related to the provenance of results, so called *provenance questions*, we introduce the notion of *process documentation*. Process documentation is information that describes a process that has occurred. To ensure that process documentation can be created by multiple software components, across multiple institutions, in multiple domains and then used in trailing analyses, we have developed a generic data model for process documentation. This model facilitates the sharing of process documentation between institutions; it allows for the development of tools that work across domains and applications and allows for the creation of future-proof process documentation by independent application components running on a variety of platforms. These properties have been evidenced by the model's use in a variety of domains, including healthcare [Kifor et al. 2006], fault tolerant systems [Townend et al. 2005], aerospace engineering [Kloss and Schreiber 2006], and bioinformatics [Groth et al. 2005; Miles et al. 2007].

This paper focuses on describing this generic data model, its contributions are twofold:

(1) A precise conceptual definition of a generic data model for process documentation.

(2) An evaluation of the model with respect to a use case from the bioinformatics domain.

Our previous work has focused on the architecture, performance, and uses of provenance systems not on a detailed description of the data model that makes these systems possible. Furthermore, while previous work has discussed an XML instantiation of the data model, it has not described fundamental principles and design rationale behind the data model. This paper describes a conceptual data model for process documentation that is independent from any one implementation or instantiation.

The rest of the paper is organised as follows. First, we motivate a generic data model for process documentation and enumerate a number of non-functional requirements that such a model should take into account. We then describe a bioinformatics mash-up that foreshadows the direction of future scientific and business applications and enumerate a number of provenance questions that arise from it [DeRoure 2007]. Next, the data model and the concepts that underpin it are presented with various examples given from the use case. A qualitative evaluation of the data model with respect to the enumerated requirements and the use case's provenance questions is then given. Finally, we discuss related work and conclude.

2. DATA MODEL MOTIVATION

In the study of fine art, provenance refers to the documented history of a piece of art. This documented history provides weight and authority to the piece and gives scholars, collectors, and viewers a context with which they can understand the value of the work. Without this documented history, the piece would have less value in terms of its importance within the art community. This concept of provenance is of practical use in computer systems. If the provenance of digital objects could be determined like it can for some art pieces, then users could understand how documents were produced, how simulation results were generated, and why business decisions were made.

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

Take the example of a mathematical calculation performed on some data, I, from a database that produces a result R. This process is the composition of two sub-processes: Process A, the mathematical calculation, and Process B, the means by which I came into the database. The process documentation for Process A would document actions of addition, subtraction, and formula evaluation. On the other hand, the process documentation for Process B would document the entry of I into the database, for example, by documenting who was responsible for storing I in the database, what institution I was from, and whether I was produced experimentally or was synthesised from publications. The documentation for these two sub-processes differ in their level of detail and the kind of information included. However, using a generic model, we can still obtain the provenance of R that includes the whole of the process.

A model that is applicable to multiple processes could be generated on a case-by-case basis. However, a number of benefits arise from a data model that is not only generic but also shared across applications and application components. The benefits of a generic, shared model of process documentation are enumerated below.

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

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

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

(4) **Independent creation**: Using it, process documentation created independently by application components can be integrated together, which allows

trailing analyses to be performed over unanticipated groupings of process documentation.

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

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

In this paper, we present a generic, shared model of process documentation termed the *p-structure*.

3. REQUIREMENTS

To develop a data model that is both general and usable across applications, we took into account a number of non-functional requirements derived from our analysis of use cases from several domains [Miles et al. 2007]. These requirements will be revisited in the evaluation section of the paper. These requirements are as follows:

(1) **Factual** Process documentation must be factual and not based on inferences, so that it is interpreted as what happened in a system. Indeed, if process documentation were to be based on guesses or inferences, a user could not ascertain whether the process characterised in the documentation actually occurred as described.

(2) **Attributable** Process documentation must be attributable. If a user deems that process documentation is somehow erroneous, the user must know who is responsible for the creation of the documentation so that the party can be held accountable.

(3) **Autonomously Creatable** The systems we consider consist of multiple components in distributed settings. Therefore, each component should be able to create process documentation at a convenient point in time without having to synchronise with any other component. Furthermore, process documentation created in an autonomous manner should be able to be collated together to present a complete representation of processes that occur across multiple components.

Before detailing the p-structure, we introduce a concrete use case from the bioinformatics domain that will help illustrate the model's concepts.

4. A BIOINFORMATICS USE CASE

4.1 Biology

Proteins are the essential functional components of all known forms of life; they are linear chains of typically a few hundred building blocks taken from the same set of about 20 different amino acids. Protein sequences are assembled following

a code sequence represented by a polymer (mature messenger RNA). During and following the assembly, the protein will curl up under the electrostatic interaction of its thousands of atoms into a defined but flexible shape of typically 58 nm size. The resulting 3D-shape of the protein determines its function.

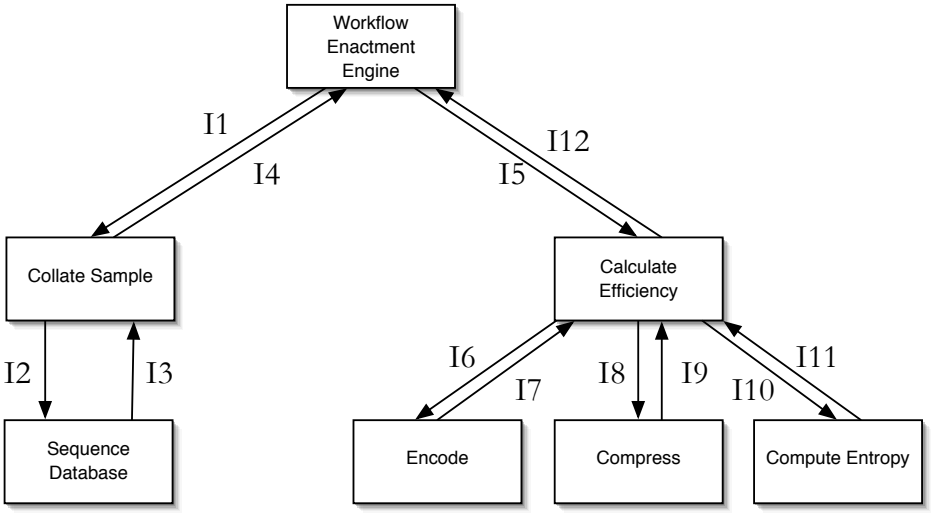
Amino acids can be *grouped* together by their chemical or physical properties. Those in the same group can often be substituted for one another in a protein sequence and the sequence will, in many cases, fold in the same manner. The ability to substitute amino acids is useful when trying to change or modify protein function. The aim of the Amino acid Compressibility Experiment (ACE) is to find other possible groups of amino acids that can be substituted for one another.

4.2 Experiment Description

In this experiment, it is assumed that protein sequences that occur in nature are efficient, i.e. they use the least number of amino acids possible to represent their function. Based on this assumption, a group is tested for interest by substituting the amino acids specified by the group with a symbol representing the group and then measuring the efficiency of the recoded sequence.The efficiency of a protein sequence can be quantified in a computational setting through compression. If a sequence compresses well then it is not efficient, whereas if the compression causes little reduction then the sequence is efficient.The ACE, therefore, uses compression to attempt to find possible groups of interest.

The workflow for the experiment is shown in Figure 1. It starts with the creation of a sample, which is composed from individual sequences obtained from sequence databases made available on the Web (see www.ebi.uniprot.org). This collation provides enough data for the statistical methods employed by the compression algorithms. The experiment requires that the samples be composed from dissimilar sequences. This dissimilarity is determined by using a culling service such as PISCES [Wang and R. L. Dunbrack 2003]. Once a sample is created, the symbols in it are substituted with those of a given group (Encode). This recoded sequence is then compressed with compression algorithms, e.g., gzip, bzip2 or ppmz, to obtain the length of the compressed sequence (Compress). The Shannon entropy is then computed on the recoded sequence to provide a standard for comparison (Compute Entropy). This standard removes the influence of two factors from the calculation of compressibility: the particular data encoding used to represent the groups, and the non-uniform frequency of groups. From the results, an information efficiency value is computed for the sample that is relative to both the compression method and group coding employed and takes into account the size of the sample (Calculate Efficiency). The information efficiency values for different groups can then be plotted to find those that are largest and thus are good candidates for further investigation.

Because there are a vast number of possible groups (roughly 474 trillion), for the experiment to investigate a sufficiently sized sample, large scale computational resources are necessary. Thus, the experiment is implemented using Grid technology [Foster and Kesselman 1999] and is run using resources from multiple institutions. Furthermore, scientific analysis is increasingly done using mash-ups [Butler 2006]. Thus, the ACE is a *good example mash-up* for the evaluation of the p-structure because its reliance on services provided by multiple institutions where those insti-



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

Fig. 1. ACE workflow.

tutions may dynamically change and the increasingly use of mash-up . For example, data is provided by two different Web Services and computing is dynamically allocated using the Grid. Whereas today's consumer mash-ups rely on one or two Web Services, future applications especially in business and science will require resources from a variety of sources, which may in turn rely on other institutions to provide their functionality. With its multi-step workflow, the ACE is an example of a future mash-up.

4.3 Use Case Questions

The bioinformatician who developed the ACE posed several provenance questions, which are enumerated below. These questions will be revisited in the evaluation of our data model.

(1) What were the original sequences used in generating an information efficiency value?
(2) Which institutions were involved in the production of a particular information

efficiency value?

(3) What were the common steps in the production of two information efficiency values?

(4) Were references or pointers used when documenting this experiment run?

5. CONCEPT MAPS

The above use case questions, as we will later show, can be answered using our generic model of process documentation, the p-structure. We now present the various concepts that underpin the p-structure using examples from the ACE. First, we define a specific notion of process represented by the p-structure. Next, we detail the data model and its constituent parts. After which, requirements on component behaviour are defined. Finally, we describe how provenance can be determined from process documentation organised using the model.

For each set of concepts, we provide a concept map [Novak 1998] that provides an overview of the concepts and the relationships between them. Concept maps were chosen for this article because they are designed for human consumption. Computer parsable representations are available as XML [Munroe et al. 2006] and OWL (http://www.pasoa.org/schemas/ontologies/pstruct025.owl). The concept maps, shown in Figures 2, 3, 7 and 9, contain concepts represented by shaded rounded rectangles and relationships linked by lines between concepts. The words in the middle of a line denote the kind of relationship between the linked concepts. Maps are read downward, or if an arrow is present, in the direction which the arrow points. For example, the top portion of Figure 2 can be read as "Actors play a role", "Actors have communication endpoints". In the text, we italicise the first occurrence of concepts that appear in a concept map.

6. PROCESS

The concepts discussed in this section are summarised by Figure 2. Applications are developed to address a variety of problems using different programming languages, design approaches and execution environments. To represent this dynamic range of situations, we take a particular perspective on all applications, which embraces the principles of encapsulation and abstraction to enable process documentation to be created at varying levels of detail while still preserving coherence across applications and their components.

The perspective we take is to view applications as composed of entities, called *actors*, each of which represents a set of functionality within the application and interact with other actors by the *sending* and *receiving* of messages through well-defined *communication endpoints*. Such a view naturally fits with service oriented architectures, one of the primary software engineering approaches for complex multi-institutional applications [Foster and Kesselman 2006]. The view also works well with object-oriented, functional and process calculi approaches. Our decomposition of applications is conceptual and is not restricted to applications already based on message passing. For example, as we view messages as information exchanged by actors, two threads communicating by a shared memory can also be viewed as actors.

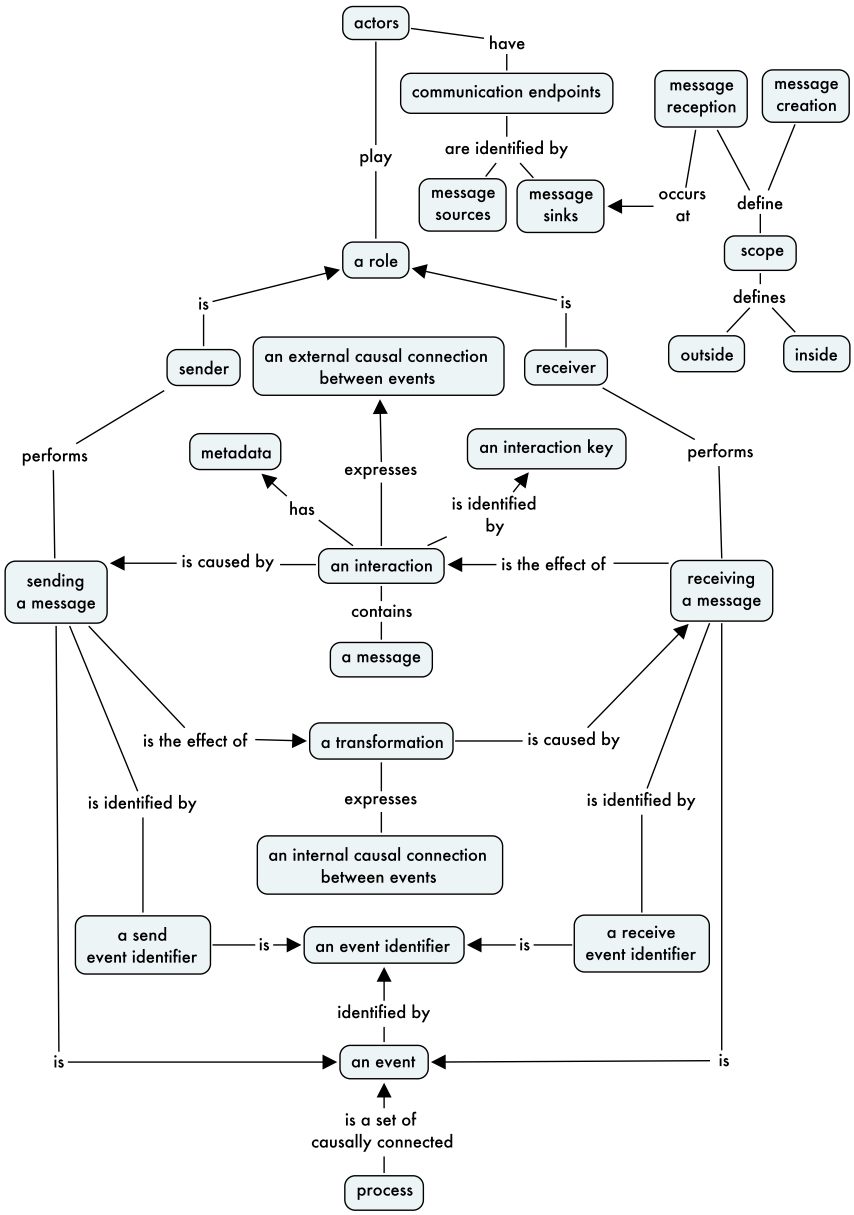


Fig. 2. Concept Map describing process.

To aid developers in the mapping of their applications to this conceptual perspective, a software engineering methodology for the decomposition of applications into actors has been created [Munroe et al. 2006]. For example, using this methodology, the ACE application was decomposed into actors that map to each step in the workflow i.e. Encode, Collate Sample, Compress etc.. Actors represent a set of functionality at some level of abstraction. This means that through the addition of actors application functionality can be represented at a greater level detail. For instance, some of the functionality encapsulated by the Calculate Efficiency actor is represented in more detail by the Encode, Compute Entropy, and Compress actors.

When decomposing an application, specific communication endpoints are pinpointed and given an identifier which is called a *message sink* or *message source* that respectively denote where messages enter and exit the actor. An actor may have any number of endpoints; the only restriction is that they are clearly identified. For example, in a Web Services context, a message sink would be the endpoint reference of an actor. Using communication endpoints, we define the boundary of an actor, or its *scope*. The scope of an actor is all data reachable by an actor, where reachability is derived from the eponym concept in memory management [Jones and Lins 1996]. Concretely, data is reachable by an actor after it is received through a message sink, before it is sent through a message source, after it is created by the actor, and when it is being processed by the actor. We note that once data transits through a message source (i.e. data is sent in a message), it is no longer reachable by the actor. Thus, the scope defined here is not a static scope defined by software components but a dynamic notion of scope in the space of execution. As an illustration, an actor that represents a procedure would contain within its scope not only all the inputs to the procedure and all the output data the procedure returns, but also all the data it may read or store in memory.

From scope, we define the notion of *being inside* an actor. Data is said to be *inside* an actor when it is part of a message that is an element of an actor's scope. Symmetrically, *being outside* an actor is defined as not being inside it.

We define inside and outside occurrence for an event in terms of the data the event acts upon. Thus, if an event happens to data inside an actor, then the event is said to occur inside the actor. Correspondingly, if an event happens to data outside an actor, then the event said to occur outside the actor. Thus, both receive and send events are inside an actor when the message being sent or received is also inside that actor. For example, if we defined a Java Web Service, JWS, as an actor, all of the Java executing code and data being accessed (such as messages received) by that code would be inside JWS. On the other hand, data and code making up other Web Services or being transmitted to JWS would be outside JWS.

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

From distributed systems theory [Lynch 1996], the receipt of a message is caused by the sending of that same message. According to Lynch, this relationship is

referred to as the "cause" relation and is part of Lamport's "happened-before" relation [Lamport 1978], which also considers the dependency between events within an actor. Later, we will revisit causal connnections within actors. The combination of a send and receive event along with the message exchanged is termed an *interaction*. Therefore, an interaction expresses both a causal connection between a sending event and a receiving event as well as the contents of the message being exchanged. Specifically, an interaction describes an *external causal connection*, a logical connection formed where, for a given actor, the event inside the actor is caused by an event outside the actor. An interaction matches this definition because the internal receive event is caused by a send event outside the receiving actor. An example of an interaction is the sending of a database request from the Collate Sample actor to the Sequence Database actor.

With respect to interactions, actors can also play different *roles*. An actor may have the role of a *sender* of messages in one interaction and may play the role of a *receiver* of messages in another. An interaction can have *metadata* associated with it to help with the documentation of process. This metadata is usually embedded within the message being exchanged typically in the message header. However, it may be associated in some other manner. For example, metadata associated with a message could be stored in a database keyed on a hash of the message. Actors could then use a hash of the message to find and store metadata. Using metadata associated with an interaction, a sender can share information with a receiver that enables process documentation produced by these separate actors to be collated together.

Specifically, a sender can generate a unique key for an interaction, termed an *interaction key*, and send it to the receiver. Thus, the sender and receiver share the same identifier for an interaction, which allows process documentation created separately by these actors to be easily combined. An interaction key contains the sender and receiver in the interaction identified by their message source and message sink respectively as well as a field that distinguishes this interaction from any other interaction happening between the same message source and sink. Together these three pieces of information allow an interaction to be uniquely identified. Based on the interaction key both the sending and receiving events can be identified by *event identifiers*. As we later show, these identifiers are crucial to organising process documentation as they provide the central mechanism by which process documentation is referenced (i.e. indexed) in the data model. Therefore, actors must exchange interaction key metadata in some manner whether it is through embedding within messages or some other mechanism. We assume that implementations sharing a common instantiation of the data model (e.g. XML, OWL, or Java Objects) will be able to extract interaction keys from metadata. We do not consider how metadata defined outside the p-strucuture can be processed and emphasise that the metadata we do consider is not arbitrary but is there to support the documentation of processes.

Interactions express the causal connection between the sending and receiving of a message. However, they do not express the causal connection between the receiving of a message and the sending of another message within the scope of an actor. While Lamport's "happened-before" relationship enables the expression of a partial order

of events (i.e. sending and receiving) within an actor, it is not designed to provide the causal information necessary to provide a detailed description of the provenance of data [Lamport 1978]. First, it does not describe the causal connections between data. Second, it does not provide the information as to how the sending of a message by an actor is caused by the execution of some functionality within the actor and this, in turn, is caused by the receipt of a set of messages.

We term the causal connection between the receiving of messages and the sending of messages caused by an actor's functionality a *transformation*.[1] This causal connection is termed an *internal causal connection* because the events being connected are both inside the actor.

Consequently, in an application, the receiving of data by an actor may cause a transformation to occur. This transformation may cause the sending of data, which itself causes the receipt of data in another actor, which in turn may cause a transformation and so on. Thus, a causally connected set of interactions and transformations defines a *process* for which precise organised process documentation can be created. We now describe a model, the p-structure, tied to this definition of process.

7. PROCESS DOCUMENTATION

The concepts discussed in this section are summarised by Figure 3. Thus far, we have discussed process documentation in the abstract. We now distinguish between the whole of process documentation and its individual parts. A *p-assertion* is an assertion that is made by an actor and pertains to a process. *Process documentation* then consists of a set of p-assertions. Actors that create p-assertions are termed *asserters*. We place a restriction on all asserters that they only create p-assertions for events and data that are inside their scope. Thus, asserters only create process documentation about what they know to be the case, which supports the requirement of factuality. Furthermore, every p-assertion contains an *asserter identity*. This is the identity of the responsible party or parties for the p-assertion. Thus, the asserter identity will always contain the identity of the actor but may also contain, for example, the identity of the owner of the actor.

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

7.1 Interaction P-assertions

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

(1) An asserter identity.

(2) An event identifier.

(3) A representation of the message exchanged in the interaction.

(4) A *documentation style* describing how the representation was generated.

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

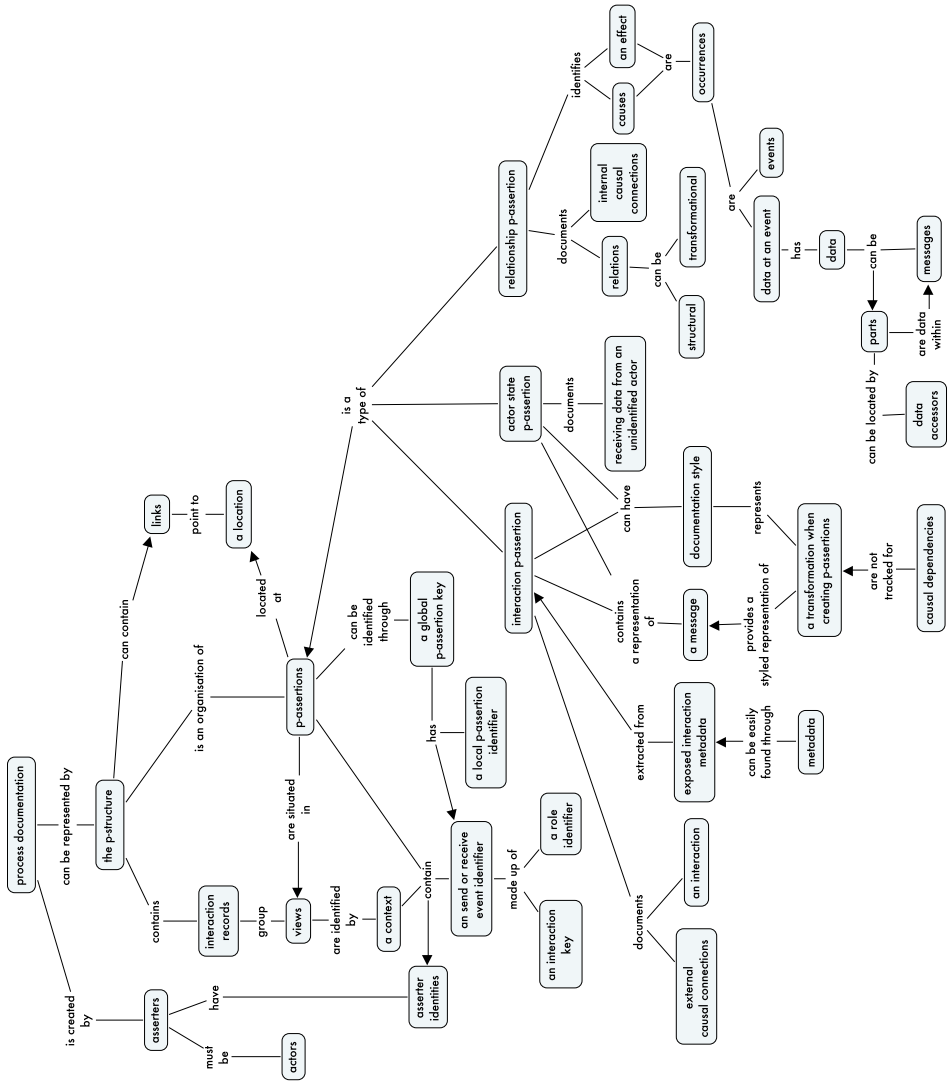


Fig. 3. Concept Map describing process documentation

As with all p-assertions the interaction p-assertion contains an assertor identity. The event identifier uniquely identifies an event through a combination of a role identifier and an interaction key. The role identifier denotes whether the actor creating the p-assertion was the sender or receiver in the interaction and thus whether the event being documented is the sending or receiving of a message.

The representation of the message contained in the interaction p-assertion often contains an exact duplicate of the message, but, in some instances it may not be feasible to have such a representation, for example, when the data being transferred needs to remain anonymous to users of process documentation or is of a large

size. In the ACE, this occurs when sequence sample data generated by the Collate Sample actor is replaced with a reference to save storage space. To allow for these cases while still preserving an accurate representation, we allow a message to be transformed in a well-defined manner during the generation of a p-assertion, which is termed styling the p-assertion. The styling that is performed is defined explicitly by a documentation style. Causal dependencies are not tracked for these styling transformations because they pertain to the creation of process documentation as opposed to the production of application results. Likewise, the created p-assertions are not seen as application data and are not in the scope of an actor.

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

7.2 Relationship P-assertions

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

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

(1) *Structural.* Relationships of this type describe the composition of a data item from its constituent parts. They do not describe how a data item was composed from other data items, only that there exists a causal dependency between the data item and its parts. For example, one might want to express that a particular sequence database is causally dependent on the sequences it contains. If one of the sequences in the database changed then the database itself would change. Relationships of this type can only be created between data items at the same event.

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

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

7.3 Levels of Abstraction

The combination of interaction p-assertions and relationship p-assertions provide the information necessary to document processes. Furthermore, they allow process documentation to be created at different levels of abstraction. Figure 4 shows the documentation for the collation process of the ACE application at two different levels of abstraction. Relationship p-assertions are shown by dotted arrow-capped lines with labels. Interaction p-assertions are shown with solid-arrow capped lines with labels. The arrows represent the direction of causation from cause to effect.

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

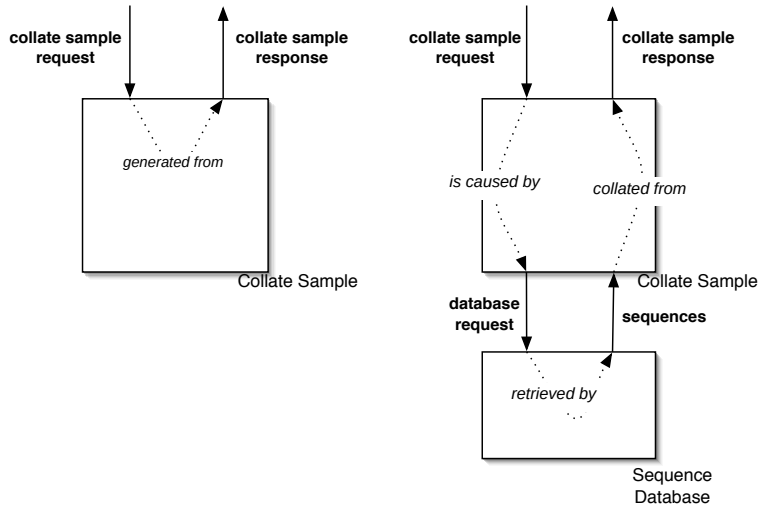
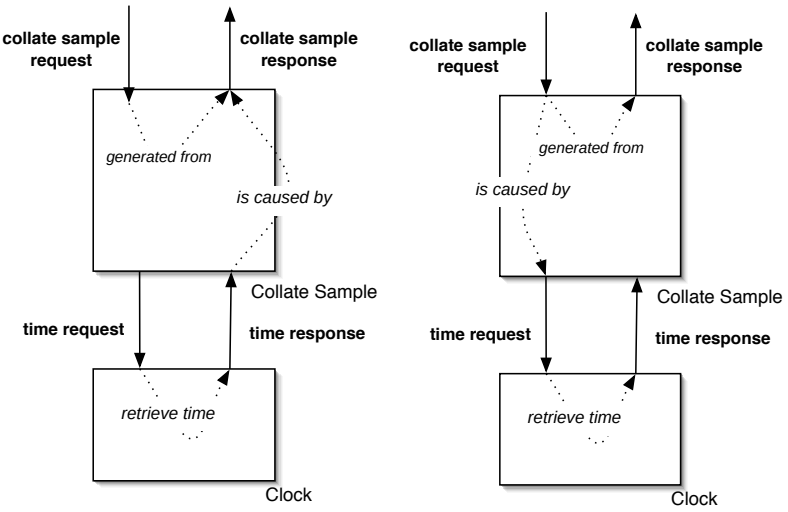


Fig. 4. An example of documenting process at different levels of abstraction.

If even further detail is required, more actors can be exposed that model progressively more detailed functional components within the application. No matter what level of abstraction is required, process documentation for each level can coexist and supplement one another to facilitate trailing analyses.

7.4 Internal Information P-assertions

We now discuss one final type of p-assertion that facilitates abstraction. It is often the case that a piece of data plays an important role in a process but the manner of its generation is not of interest. Examples of this include, the time, the memory usage of an actor or the original configuration of an actor. All of these data items can be represented using relationship p-assertions and interaction p-assertions. For example, Figure 5(a) represents a time stamp on the sending of the collate sample response. The retrieval of the time by the Collate Sample actor is one of the causes of the sending of the response. Likewise, Figure 5(b) represents a time stamp on the receipt of the collate sample request because the request causes the retrieval of the time from the clock actor.



(a) Modelling the time when a message was sent.

(b) Modelling the time when a message was received.

Fig. 5. Modelling message send and receive times.

Using *internal information p-assertions*, the detail of how the time was obtained can be abstracted away and we are left with just the time and its basic causal connection to the process. An internal information p-assertion represents the receipt of data by an actor from some other unidentified actor and is causally connected to the sending or receiving of a message by the former actor. The causal connections represented by internal information p-assertions are different for sending and receiving events and are as follows:

(1) **Sending:** The sending of a message is caused by the receipt of the data within the internal information p-assertion.

(2) **Receiving:** The receipt of the data documented by an internal information p-assertion is caused by the sending of a message to some unidentified actor which

is in turn caused by the receiving of another message. In this case, an internal information p-assertion can only be made where the actor can determine that such a causal chain exists.

We note that because internal information p-assertions represent the receipt of messages they can be used to represent causes within relationship p-assertions. Also, an actor can style the data during the creation of the internal information p-assertion. Thus, an internal information p-assertion consists of three parts: the data, the documentation style of the data, and the event identifier of the event to which the data is causally connected. Internal information p-assertions allow data items to be made explicit without, the sometimes unnecessary, overhead of creating documentation for their generation. This is exemplified by Figure 6, which shows how Figure 5 can be abstracted using internal information p-assertions.

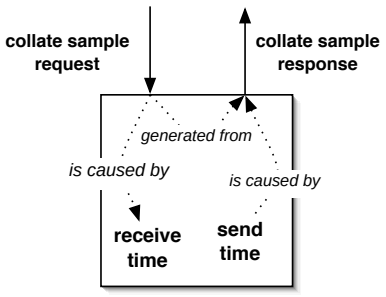


Fig. 6. Example of 5 abstracted using two internal information p-assertions.

7.5 The P-Structure

P-assertions contain the elements necessary to represent a process. However, without some organisation it would be difficult to discover distinct processes within process documentation. Therefore, we introduce the *p-structure* which is an organisation of p-assertions that provides several elements to help isolate, find, and understand sets of p-assertions. The p-structure situates each p-assertion in a container termed a *view*, which is identified by an event identifier. This means that p-asssertions are grouped together by the event that they are most closely associated with. The associations for the three types of p-assertions are as follows.

—Interaction p-assertions are associated with the event they document as given by event identifier within the interaction p-assertion.

—Relationship p-assertions are associated with the event identified in the effect occurrence of the relationship p-assertion. This allows users of process documentation to find the causes of a particular occurrence; once the occurrence is known the relationship p-assertion documenting its causes can be found in the same view.

—Internal information p-assertions are associated with the event that they are causally related to. Again, this provides a way to easily find data that is fundamentally about a given occurrence because the data is in the same view as the occurrence.

Each p-assertion within a view is given a local p-assertion id that, when combined with the event identifier for the view, allows the p-assertion to be uniquely identified within the p-structure. This combination is termed a *global p-assertion key*. Because actors are only allowed to create p-assertions about events in their scope, all the p-assertions in a given view will have been asserted by the same actor. Therefore, the common asserter identity shared between a view's p-assertions is also placed within the view.

Views also have a place for *links*, which point to the location of a p-assertion. Typically, an actor creates a p-assertion and stores the p-assertion in some repository. Through links, the p-structure can reference p-assertions stored across multiple storage devices, which aids in their finding and retrieval. For example, each component in ACE may store process documentation in a different repository.

Finally, the metadata that is associated with interactions is typically documented using interaction p-assertions. Because this interaction metadata is helpful for isolating and demarcating processes within process documentation, the p-structure allows it to be extracted from p-assertions and placed inside a view as *exposed interaction metadata*. Once there, the metadata can be easily found. We now look briefly at one mechanism, *tracers*, that is metadata used to demarcate processes. See Figure 7 for an overview.

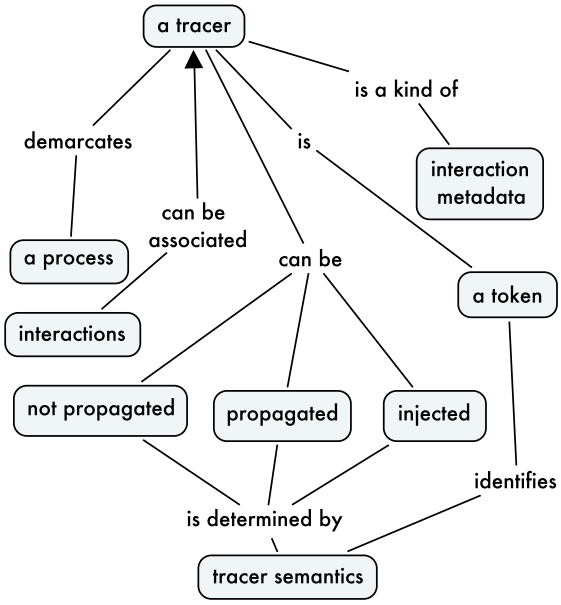


Fig. 7. Concept Map describing tracers.

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

Given our interaction-centric perspective, the p-structure logically groups together the two views that document an interaction into what we term an *interaction record*. By a logical grouping, we mean that two views are associated with each other by a shared interaction key. The views themselves may be stored in different repositories or locations. The grouping of views into interaction records has the added benefit of collating together process documentation created by independent actors. The p-structure then contains a set of these interaction records.

Figure 8 shows an example of a p-structure. The portions of the interaction records shown contain the documentation for the encoding of the collated sample. Each large square is an interaction record labelled with the interaction key and label from Figure 1 of the interaction it represents. The label is in the upper-right hand corner of the square. The interaction key is shown by the name of the actor sending the message (acting as a message source) followed by an arrow pointing to the name of the actor receiving the message (acting as a message sink). Each p-assertion in Figure 8 is also followed by its local p-assertion identifier in parenthesis. The following abbreviations are used in the Figure: pa denotes p-assertion, ik denotes interaction key, lpid denotes local p-assertion identifier, da denotes data accessor.

8. ACTOR BEHAVIOUR

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

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

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

Interaction Key Transmission Rule *A sender asserting actor must make the interaction key it assigns to an interaction available to the receiver in that*

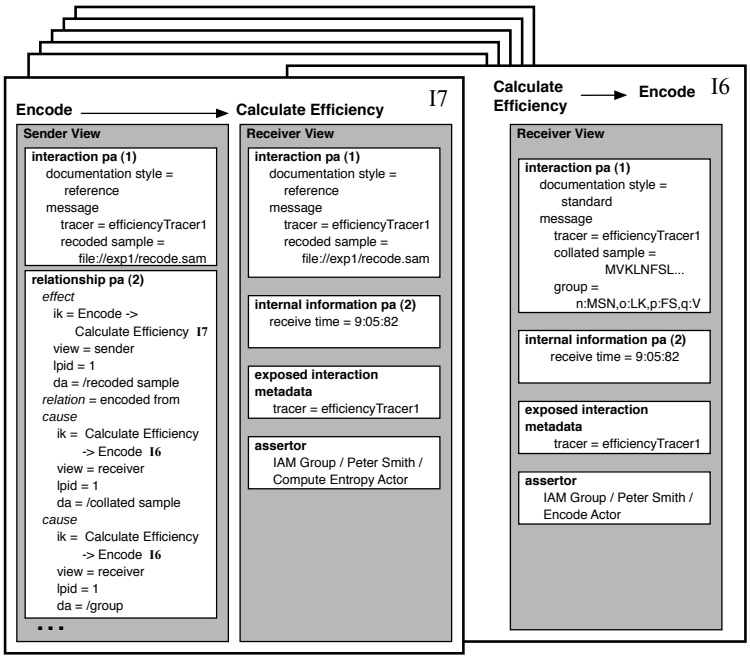


Fig. 8. An example of the contents of a p-structure that documents the interactions I6 and I7 from Figure 1.

interaction.

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

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

These rules are simple and have been kept to a minimum so that actors can create process documentation as independently as possible.

9. PROVENANCE

The concepts discussed in this section are summarised by Figure 9. Thus far we have discussed how to create and organise documentation of an application's past processes. We now discuss how that documentation can be used to find the provenance of occurrences.

The documentation generated by an application will encompass many different events and data items. However, in the case of provenance, only the process leading to a selected event or data item is of interest. Therefore, it is necessary to extract a particular set of p-assertions from the p-structure which together describe that process (i.e. the provenance of that occurrence). In Section 6, we described a process as a set of causally connected interactions and transformations. These interactions and transformations are described by p-assertions. Starting from a

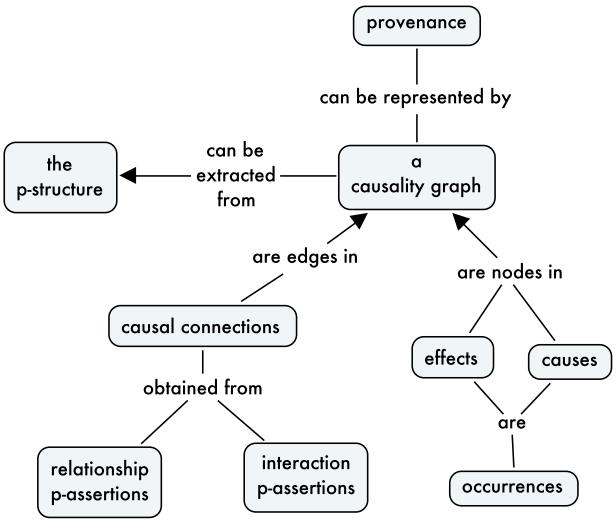


Fig. 9. Concept Map describing provenance.

particular occurrence, its causes can be obtained from the p-assertions where it is documented. Once these causes have been obtained, their causes can also be obtained from the process documentation and so on until the extraction mechanism is finished. Thus, a graph of causal connections leading back to a particular occurrence can be extracted from the p-structure. Such a *causality graph* describes the provenance of an occurrence. Different forms of causality graphs can be extracted from the p-structure depending upon usage. Here, we use a simple form for illustration purposes.

Figure 10 shows the provenance of an information efficiency value. The nodes in the graph are occurrences, in the role of causes, effects or both. The edges in the graph are hyper edges and represent the causal connections extracted from relationship p-assertions. All arrows on the edges point from effect to cause. The external causal connections represented by interaction p-assertions are collapsed into the numbers shown to the bottom right of each node. These numbers map to the interactions shown in Figure 1. Internal information p-assertions are shown as annotations connected to the interactions by double-arrow headed lines. To save space, not all of these p-assertions are shown. The relationship p-assertions shown in Figure 8 are found in this figure.

Figure 10 evinces the claim that the provenance of an occurrence can be explained at different levels of detail. For example, the edge labelled as *efficiency calculation from* abstracts the six nodes and seven hyper edges to its left.

10. EVALUATION

In prior work, we identified three core components of an architecture for determining the provenance of data produced by dynamic systems [Miles et al. 2007]. The first component is the recording of process documentation into the a separate repository, called a provenance store. The second component is the querying of stored process

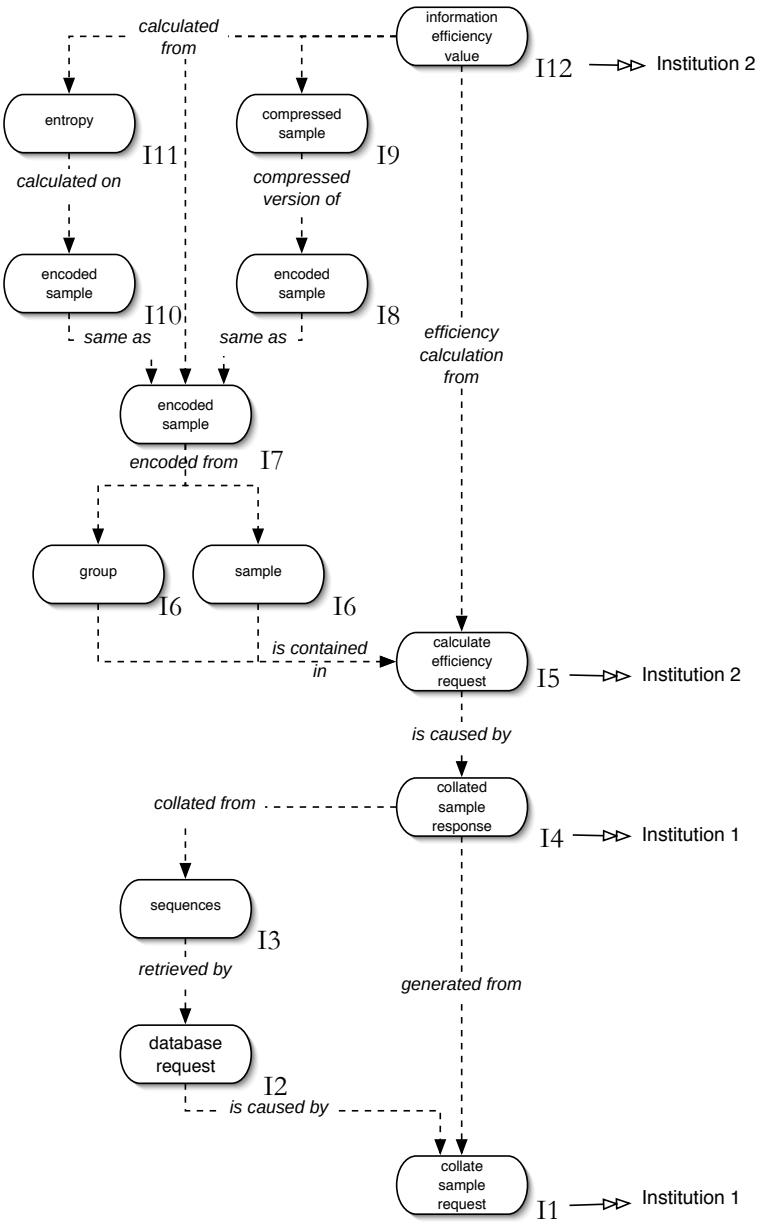


Fig. 10. Causal graph describing the provenance of an information efficiency value.

documentation to determine the provenance of a data item. The third component is the data model that enables the separation of concerns between the recording and querying components of the architecture.

Prior work presented the design, implementation and evaluation of both the recording and querying components. The recording component's implementation and performance has also been presented elseswhere [Groth et al. 2005; Simmhan et al. 2006]. Likewise, the querying component has been evaluated in a number of domains including including organ transplant management [Kifor et al. 2006], aerospace simulation [Kloss and Schreiber 2006], fault tolerant systems [Townend et al. 2005], and bioinformatics [Miles et al. 2007].

In this paper, we detail and evaluate the data model component of the architecture. Our evaluation of the data model consists of analysis of how the data model fulfils the initial requirements presented in Section 3 and a description of how the data model can be used to address the four use case questions from the ACE mash-up. We begin by addressing the initial requirements.

(1) **Factual** This requirement is the basis of our notion of actors and thus is at the core of our data model. If application developers follow the data model specification, actors will only assert what is in their scope and thus process documentation will be factual. Furthermore, we can detect when actors have conflicting views of the messages they exchange in interactions. This encourages actors to create factual documentation.

(2) **Attributable** The requirement of attribution is supported through each p-assertion containing an asserter identity. Therefore, users of process documentation can identify who is responsible for a particular p-assertion and hold them to account for any information contained within the p-assertion. Cryptographic techniques such as digital signatures can be used to ensure that asserter identities are accurate and correctly associated with p-assertions. A more detailed discussion of the use of such security primitives is outside the scope of this paper. However, a complete discussion can be found in Tan et al. [Tan et al. 2006].

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

Having shown how our original requirements are addressed by the p-structure, we now show how the use case provenance questions can be answered using process documentation organised by the p-structure. First, we briefly discuss how process documentation is created in ACE and how it can then be retrieved.

Each workflow component (Collate Sample, Compress, Calculate Efficiency) represents an actor and is responsible for creating the interaction, relationship, and internal information p-assertions necessary to document its portion of the process. Collate Sample was implemented as a Java based Web Service. Compress and Calculate Efficiency were run as jobs on the Grid. The actors create p-assertions for

incoming messages either from external or internal components and then create relationship p-assertions connecting incoming and outgoing messages. After creation, process documentation can be recorded into a provenance store. There are two independent but compatible provenance stores [Ibbotson and Jiang 2006; Groth et al. 2005] that make use of an XML instantiation of the p-strucuture [Munroe et al. 2006] . Both implementations of provenance stores currently available offer XQuery [Boag et al. 2006] and a provenance specific graph traversal query function [Miles 2006]. Using these query mechanisms, the use case questions were answered. We used the PReServ provenance store in our tests [Groth et al. 2005].

(1) What were the original sequences used in generating an information efficiency value? To find the answer to this question, the provenance of the particular information efficiency value needs to be known. The following is a simplified algorithm for finding the provenance of a given occurrence, in this case it would be a particular information efficiency value.

(1) All the relationship p-assertions where a given occurrence, X, was an effect are found.

(2) From these relationship p-assertions, all the occurrences that were the causes of the effect can be found.

(3) For each of the occurrences found in Step 2, find the provenance of those items using this algorithm.

This algorithm runs until there are no more relationship p-assertions and the entire causality graph for the information efficiency value is generated as shown in Figure 10. Then it is a matter of locating the relationship p-assertion of interest, which in this case is the p-assertion that has the relation "collate sample from", the causes identified in the p-assertion would be the original sequence. Even though the process documentation that contains the provenance of the information efficiency value was created by different actors, the p-structure allows the process documentation to be brought together enabling the provenance of the value to be determined.

(2) Which institutions were involved in the production of a particular information efficiency value? Because the ACE is performed in a dynamic multi-institutional environment, the bioinformatician would like to know what institutions participated. The p-structure allows p-assertions produced by multiple institutions to be collated together. Once process documentation has been assembled, this question can be answered in the same manner as the previous question, however, in this instance, the data held within internal information p-assertions is used. At every node in the causal graph, the internal information p-assertion can be found that contains the institution name. This is made possible because each internal information p-assertion is contained in a view, which connects the p-assertion to an interaction and to the actor who created it. The institution name was modelled as an internal information p-assertion because the data is important but not how the data was obtained by the actor.

(3) What were the common steps in the production of these two information efficiency values? To answer this question, a notion of a process that can be differentiated from other processes is required. For example, computations that collate sequences into a sample are often part of the production of several

information efficiency values. Knowing the provenance of the results of multiple experimental runs and being able to differentiate and compare them is fundamental to answering this question.

Therefore, the solution to this question involves finding the provenance of each information efficiency value, which generates two causal graphs. The common interactions between these two causal graphs are then found by comparing the interaction keys of the interactions. These interactions are the common steps asked for in the use case. Using tracers, we can then determine if the interactions belong to larger processes.

(4) Were references or pointers used when documenting this experiment run? When ACE creates process documentation, the bioinformatician often wants to know if it contains the actual application data or just references (file paths or URLs) to those data items. If the documentation of the run contains the actual data, less work is necessary to analyse the data because it is in one place. Furthermore, the bioinformatician might have moved, deleted or modified his original data mistakenly or on purpose. Thus, the knowledge, as to whether the documentation of a particular experiment contains original data, can inform the bioinformatician about the types of trailing analysis that can be performed. Our data model for process documentation takes this concern into account by using documentation styles to label p-assertions that contain references.

Therefore, the solution to this use case relies on documentation styles. First, the provenance of a particular experiment run is found, then for every interaction p-assertion contained in the returned causality graph, the documentation style is retrieved. If the style retrieved is a reference documentation style, an identifier for the corresponding interaction p-assertion can be returned demonstrating that references were used in the run.

Each of these provenance questions was answered through the extraction of a causality graph from process documentation enabled by the p-structure. Moreover, the extra facilities provided by the p-structure such as documentation styles were necessary to answer some of the provenance questions. Lastly, the various approaches used to answer these questions are reusable in applications across domains and have been integrated into common web-based tools [Deora et al. 2006].

11. RELATED WORK

The subject of provenance has not gone without notice in the literature. Under the heading of lineage, Bose and Frew present a comprehensive overview of provenance related systems [Bose and Frew 2005]. Likewise, Simmham et al. give a survey of provenance in the domain of e-Science [Simmhan et al. 2005]. A compilation of the current state of the art is given by Moreau and Foster [Moreau and Foster 2006]. From an analysis of these works, we assert that the focus of provenance research has been on the implementation of concrete systems for provenance in the context of either specific domains (i.e. geographic information systems, chemistry, biology) or technologies (i.e. databases). In contrast, this work focuses on a conceptual organisation of process documentation independent of technology or domain.

Work in the database community has focused on the data lineage problem, which can be summarised as: given a data item, determine the source data used to produce

that item. Cui et al. present a number of algorithms for determining the lineage of data in relational databases [Cui et al. 2000] and data warehouse environments [Cui and Widom 2003]. Buneman et al. also develop a formal model of provenance for database systems that applies to both hierarchical and relational databases [Buneman et al. 2001]. Our data model differs from these approaches because it can be used to represent processes that occur both inside and outside database environments.

In the e-Science community, work has focused on provenance for workflow-based environments. For example, Zhao et al. present a model for provenance in the Virtual Data System (VDS), which uses the workflow graph to tie together various data elements recorded during the execution of the workflow [Zhao et al. 2006]. The benefits of this approach over the p-structure is that information stating causal dependencies is inferred from the workflow and thus must not be created by the actors within the workflow. However, because a workflow is a plan and not what actually occurred, there exists the possibility that users will make use of causal connections that were not actually present in the running system. The p-structure also differs from workflow centric systems like VDS, myGrid [Zhao et al. 2003], and Kepler [Altintas et al. 2006] in that it supports any type of execution environment. For example, process documentation compatible with the p-structure can be generated by Java programs, shell scripts or workflow enactment engines.

Developments in the Semantic Web community have concentrated on adding annotations to Resource Description Framework (RDF) graphs that describe the provenance of the nodes of the graph [Carroll et al. 2005; Futrelle 2006]. As with the other systems presented, these approaches are technology dependent as they rely wholly on RDF. Comparatively, the p-structure, because of its conceptual definition, can be represented using multiple technologies including RDF.

Our approach differs from all of these systems in that it conceptually separates the documentation of a process and finding the provenance of an item from that documentation. This distinction is vital because it allows the results returned form provenance queries to evolve and become more comprehensive as more documentation is created.

The notion of solving problems through the analysis of a documentation of execution in a distributed environment resembles work in distributed debugging [Bates 1995; Ho and Hand 2005]. However, these systems are designed to help debug distributed programs not to answer questions at a user level. The p-structure's support of multiple levels of abstraction means that the answers to questions are not restricted to a programatic or code level. Aguilera et al. present a distributed debugging solution that divides distributed systems into black-boxes and infers the causality between the receiving and sending of messages [Aguilera et al. 2003]. This approach is similar to ours with its emphasis on causality and the division of applications into functional components. However, unlike our approach, theirs relies on inferences. This is beneficial in that applications do not need to be modified but makes the enforcement of attribution difficult. Furthermore, they focus on improving the performance of distributed systems whereas our approach is designed for finding the provenance of data and events.

Finally, our approach is designed to work in an environment where applications

can be dynamically composed from a variety of services. Thus, we briefly describe work in service composition. One technique to compose services is to explicitly combine them using a workflow description language such as BPEL4WS [Curbera et al. 2003]. Another mechanism to support service composition is the markup of Web Services with semantic annotations of their interfaces, which specify both the capabilities of the service as well as the functioning of the service (i.e. data flow, preconditions and effects) in a machine understandable fashion [Martin et al. 2007]. These annotations can be used for the translation between heterogeneous services [Szomszor et al. 2006] or for the automated construction of service workflows to meet a high-level goal [McIlraith and Son 2002]. Recent work has looked at the flexible provisioning of Web Services in unreliable environments [Stein et al. 2008]. The above work focuses on what we referred to at the beginning of this article as prospective processes. They are designed to facilitate the construction or execution of processes. Our work, on the other hand, is designed to represent past processes to facilitate their validation and analysis after the fact. However, we conjecture that process documentation could be used as a valuable source of information for the construction of workflows as well as the description of services.

12. CONCLUSION

As more complex mash-ups are developed that use resources that are dynamically allocated across multiple cooperating institutions, the ability to perform analyses after the fact becomes increasingly important. In such dynamic environments, users cannot predict how an application will execute. Therefore, users need a mechanism by which they can perform trailing analyses which enable them to understand how their results were produced.

In this paper, we proposed that such trailing analyses should be performed over process documentation organised using a generic data model to facilitate interoperability and sharing. Therefore, we have defined the concepts that comprise a generic data model for process documentation, the p-structure. This description included several concept maps which unambiguously summarise the various concepts underpinning the data model and the relationships between them. The p-structure was qualitatively evaluated with respect to four provenance questions from a bioinformatics use case. These four questions were answered from process documentation organised by the p-structure. The data model supports the autonomous creation of factual, attributable process documentation by separate, distributed application components.

In the future, we aim to interoperate with other systems through the expansion of the number of p-structure representations available and by continuing our involvement with the Provenance Challenge [Miles et al. 2007]. One outcome of this interoperability is that provenance represented in our model could used as input to trust calculation algorithms [Golbeck and Hendler 2006; Golbeck 2006]. We have already developed a provenance-aware RSS prototype, which we will combine with these trust algorithms to provide trust measurements for RSS feeds. Finally, we will continue to expand the use of the p-structure in a variety of applications including more consumer focused mash-ups.

REFERENCES

Aguilera, M. K., Mogul, J. C., Wiener, J. L., Reynolds, P., and Muthitacharoen, A. 2003. Performance debugging for distributed systems of black boxes. In *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. ACM Press, New York, NY, USA, 74–89.

Altintas, I., Barney, O., and Jaeger-Frank, E. 2006. Provenance collection support in the kepler scientific workflow system. In *International Provenance and Annotation Workshop, IPAW 2006*, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer-Verlag, 118–132.

Bates, P. C. 1995. Debugging heterogeneous distributed systems using event-based models of behavior. *ACM Transactions on Computer Systems 13,* 1, 1–31.

Boag, S., Chamberlin, D., Fernndez, M. F., Florescu, D., Robie, J., and Simon, J. 2006. Xquery 1.0: An xml query language. Tech. rep., World Wide Web Consortium.

Booch, G. 1999. Uml in action. *Communications of the ACM 42,* 10, 26–28.

Bose, R. and Frew, J. 2005. Lineage retrieval for scientific data processing: a survey. *ACM Computing Surveys 37,* 1, 1–28.

Buneman, P., Khanna, S., and Tan, W. 2001. Why and where: A characterization of data provenance. In *Int. Conf. on Databases Theory (ICDT)*. Lecture Notes in Computer Science, vol. 1973. Springer-Verlag, 316.

Butler, D. 2006. Mashups mix data into global service. *Nature 439*, 6–7.

Carroll, J. J., Bizer, C., Hayes, P., and Stickler, P. 2005. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*. ACM Press, New York, NY, USA, 613–622.

Cui, Y. and Widom, J. 2003. Lineage tracing for general data warehouse transformations. *The VLDB Journal 12,* 1, 41–58.

Cui, Y., Widom, J., and Wiener, J. L. 2000. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst. 25,* 2, 179–227.

Curbera, F., Khalaf, R., Mukhi, N., Tai, S., and Weerawarana, S. 2003. The next step in web services. *Communications of the ACM 46,* 10, 29–34.

Dean, T. 1996. Automated planning. *ACM Computing Surveys 28,* 1, 85–87.

Deora, V., Contes, A., Rana, O. F., Rajbhandari, S., Wootten, I., Tamas, K., and Z.Varga, L. 2006. Navigating provenance information for distributed healthcare management. In *IEEE/WIC/ACM Web Intelligence Conference*. 859–865.

DeRoure, D., Ed. 2007. *Web 2.0 and Grids Workshop at OGF19*. http://www.semanticgrid.org/OGF/ogf19/.

Foster, I. and Kesselman, C., Eds. 1999. *The Grid: Blueprint for a New Computing Infrastructure*, 1st Edition ed. Morgan Kaufmann Publishers, Inc.

Foster, I. and Kesselman, C. 2006. Scaling system-level science: Scientific exploration and it implications. *IEEE Computer 39,* 11 (Nov.), 31–39.

Futrelle, J. 2006. Harvesting rdf triples. In *International Provenance and Annotation Workshop, IPAW 2006*, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer-Verlag, 64–72.

Golbeck, J. 2006. Combining provenance with trust in social networks for semantic web content filtering. In *International Provenance and Annotation Workshop, IPAW 2006*, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer-Verlag, 101–108.

Golbeck, J. and Hendler, J. 2006. Inferring binary trust relationships in web-based social networks. *ACM Transactions on Internet Technology 6,* 4, 497–529.

Groth, P., Miles, S., Fang, W., Wong, S. C., Zauner, K.-P., and Moreau, L. 2005. Recording and using provenance in a protein compressibility experiment. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC'05)*.

Groth, P., Miles, S., and Moreau, L. 2005. PReServ: Provenance Recording for Services. In *Proceedings of the UK OST e-Science second All Hands Meeting 2005 (AHM'05)*. Nottingham,UK.

Ho, A. and Hand, S. 2005. On the design of a pervasive debugger. In *AADEBUG'05: Proceedings of the sixth international symposium on Automated analysis-driven debugging*. ACM Press, New York, NY, USA, 117–122.

Ibbotson, J. and Jiang, S. 2006. D9.3.3: Final Functional Prototype. Tech. rep., IBM United Kingdom. Nov.

Jones, R. and Lins, R. 1996. *Garbage Collection. Algorithms for Automatic Dynamic Memory Management*. Wiley.

Kifor, T., Varga, L. Z., Vzquez-Salceda, J., lvarez, S., Willmott, S., Miles, S., and Moreau, L. 2006. Provenance in agent-mediated healthcare systems. *IEEE Intelligent Systems*.

Kloss, G. K. and Schreiber, A. 2006. Provenance implementation in a scientific simulation environment. In *International Provenance and Annotation Workshop (IPAW)*, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer-Verlag, 37–46.

Lamport, L. 1978. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM 21,* 7 (July), 558–565.

Lavana, H., Khetawat, A., Brglez, F., and Kozminski, K. 1997. Executable workflows: a paradigm for collaborative design on the internet. In *DAC '97: Proceedings of the 34th annual conference on Design automation*. ACM Press, New York, NY, USA, 553–558.

Lynch, N. 1996. *Distributed Algorithms*. Morgan Kaufmann Publishers. page 460.

Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D., Sirin, E., and Srinivasan, N. 2007. Bringing semantics to web services with owl-s. *World Wide Web Journal 10,* 3 (September), 243–277. Special Issue: Recent Advances in Web Services.

McIlraith, S. and Son, T. 2002. Adapting golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*. Toulouse, France, 482–493.

Miles, S. 2006. Electronically querying for the provenance of entities. In *International Provenance and Annotation Workshop (IPAW)*, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer-Verlag, 37–46.

Miles, S., Groth, P., Branco, M., and Moreau, L. 2007. The requirements of using provenance in e-science experiments. *Journal of Grid Computing 5,* 1, 1–25.

Miles, S., Groth, P., Munroe, S., Jiang, S., Assandri, T., and Moreau, L. 2007. Extracting causal graphs from an open provenance data model. *Concurrency and Computation: Practice and Experience*. to appear.

Miles, S., Wong, S. C., Fang, W., Groth, P., Zauner, K.-P., and Moreau, L. 2007. Provenance-based validation of e-science experiments. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web 5*, 28–38.

Moreau, L. and Foster, I., Eds. 2006. *Provenance and Annotation of Data — International Provenance and Annotation Workshop, IPAW 2006*. Lecture Notes in Computer Science, vol. 4145. Springer-Verlag.

Munroe, S., Groth, P., Jiang, S., Miles, S., Tan, V., and Moreau, L. 2006. Data model for process documentation. Tech. rep., University of Southampton. http://eprints.ecs.soton.ac.uk/13200/.

Munroe, S., Miles, S., Moreau, L., and Vazquez-Salceda, J. 2006. PrIMe: A Software Engineering Methodology for Developing Provenance-Aware Applications. In *Proceedings of Sixth International Workshop on Software Engineering and Middleware (SEM 06)*. ACM Digital, Portland, Oregon, 8. Published electronically by ACM Digital at http://portal.acm.org/toc.cfm?id=1210525.

Novak, J. D. 1998. *Learning, Creating, and Using Knowledge: Concept Maps As Facilitative Tools in Schools and Corporations*. LEA, Inc.

Simmhan, Y. L., Plale, B., and Gannon, D. 2005. A survey of data provenance in e-science. *SIGMOD Record 34,* 3, 31–36.

Simmhan, Y. L., Plale, B., Gannon, D., and Marru, S. 2006. Performance evaluation of the karma provenance framework for scientific workflows. In *International Provenance and*

Annotation Workshop, IPAW 2006, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer-Verlag.

Stein, S., Payne, T. R., and Jennings, N. R. 2008. Flexible provisioning of web service workflows. *ACM Transactions on Internet Technology 8*.

Szomszor, M., Payne, T. R., and Moreau, L. 2006. Automated syntactic medation for web service integration. In *Proceedings of IEEE International Conference on Web Services (ICWS'06)*. Chicago, USA.

Tan, V., Groth, P., Miles, S., Jiang, S., Munroe, S., Tsasakou, S., and Moreau, L. 2006. Security issues in a soa-based provenance system. In *Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*. Springer-Verlag, Chicago, Illinois.

Townend, P., Groth, P., and Xu, J. 2005. A provenance-aware weighted fault tolerance scheme for service-based applications. In *Proc. of the 8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2005)*. IEEE Computer Society, 258–266.

Wang, G. and R. L. Dunbrack, J. 2003. Pisces: a protein sequence culling server. *Bioinformatics 19*, 1589–1591.

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

Zhao, Y., Wilde, M., and Foster, I. 2006. A virtual data provenance model. In *International Provenance and Annotation Workshop, IPAW 2006*, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer-Verlag.