PrIMe: A Methodology for Developing Provenance-Aware Applications

SIMON MILES
Department of Computer Science, King's College London
PAUL GROTH
Information Sciences Institute, University of Southern California
STEVE MUNROE and LUC MOREAU
Electronics and Computer Science, University of Southampton

Provenance refers to the past processes that brought about a given (version of an) object, item or entity. By knowing the provenance of data, users can often better understand, trust, reproduce, and validate it. A provenance-aware application has the functionality to answer questions regarding the provenance of the data it produces, by using documentation of past processes. PrIMe is a software engineering technique for adapting application designs to enable them to interact with a provenance middleware layer, thereby making them provenance-aware. In this article, we specify the steps involved in applying PrIMe, analyse its effectiveness, and illustrate its use with two case studies, in bioinformatics and medicine.

Categories and Subject Descriptors: D2.10 [**Software Engineering**]: Design; D2.2 [**Software Engineering**]: Tools and Techniques; H1 [**Information Systems**]: Models and Principles

General Terms: Design

Additional Key Words and Phrases: Provenance, Methodology

1. INTRODUCTION

Often, the claims made about a given entity can be verified by simple observation, or measurement. However, claims may also refer to the *history* (or pedigree or lineage) of such entities. Verifying such history-dependent claims involves understanding the *provenance* of the entity, where this is understood as the documented record of past events, actions or processes as they refer to and affect the entity in question. Given such a documented history, claims about the entity can be verified or corroborated by asking *questions* about the objects's provenance — referred to in this document as *provenance questions*. Entities that do not have a trusted, proven history, i.e. a record of their provenance, may be treated with some scepticism by those who study and view them. This same concept of provenance, although usually referred to in the context of physical objects, such as works of art or manufacturing [Kloss and Schreiber 2006], may also be applied to data and information generated within computer applications. Knowing the provenance of data enables its consumers to be more confident of its validity and fitness-for-purpose, and provides understanding of how the data came to be as it is, thus creating confidence that the application containing and using the data is performing as expected.

In general, understanding the provenance of data involves documenting *the processes that led to that data* [Groth et al. 2006]. *Provenance-aware* applications produce such documentation about their processes as they execute, which later allows users to ask *provenance questions* to obtain all or part of the data's provenance. Some examples of gen-

eralised, non application-specific provenance questions are: "what was the process that produced a given piece of data?", "Two processes, thought to be performing the same steps on the same inputs, have been run and produced different data. Was this difference caused by a change in the inputs, the steps making up the process or the configuration of the process?", "Did the process that produced this data use the correct types of information at each stage?" or "Did the process that produced this data follow the original plan?"

Answering these queries involves analysing the provenance of the data item(s) of concern by examining documentation on the processes that produced it. Such documentation is called *process documentation* and is comprised of a set of *assertions* about processes. One difficulty that remains is to ensure that the *right kind* of process documentation is captured so provenance questions can be answered.

This article presents PrIMe, a software engineering technique for making applications provenance-aware, i.e. for identifying what documentation is needed to answer provenance questions, analysing application designs to determine at what points process documentation will be recorded, and adapting application designs to produce process documentation. It requires the developers to consult the users of the application in order to understand what kind of provenance is required, and at what level, and provides a general approach not tied to any particular application domain or technology and can be used to adapt already existing applications or used during the development of new applications.

In the remainder of this document, PrIMe and how it can be applied to computational systems is described in detail. The following structure is followed. In the next section, we describe the motivations behind our development of a specific technique for adapting applications to be provenance-aware and, in Section 3, PrIMe is introduced and we outline the assumptions adopted by PrIMe as well as its overall structure. In Section 4, an example application from bioinformatics is introduced in order to ground the subsequent discussion and explanation. The data model in which process documentation is stored is specified in Section 5. In Section 6, the process for identifying provenance questions is described. Section 7 then describes how to decompose applications and how to map out the flow of information within them. Section 8 goes on to describe different kinds of adaptations that can be made to applications in order to facilitate the recording of process documentation. In order to assess the benefit of our new approach, we perform a comparison with an existing, aspect-oriented approach in Section 9, and a more general assessment of PrIMe is given in Section 10. Section 11 discusses some related work and, finally, Section 12 offers some concluding remarks and discusses future work.

The work described in this article was undertaken in the context of a major research investigation on provenance, which resulted in a novel approach to provenance that it is not constrained to a single technology or application domain: instead, this provenance vision and its software realisation are independent of execution environments, and thus allows the provenance of electronic data to be found even though it may be produced by multiple, distributed programs implemented using different technologies [Moreau et al. 2008]. This technology-independent design and its implementation was rolled-out to multiple applications bringing very different requirements: bioinformatics [Groth et al. 2005; Wong et al. 2005], aerospace engineering, medical domains such as Organ Transplant Management [Moreau et al. 2008] and functional magnetic resonance imaging [Moreau et al. 2007]. These wide-ranging applications have led an extensive requirement capture [Miles et al. 2007], comprising well over 30 different use cases related to Provenance, followed

by a provenance model [Groth et al. 2008], a provenance recording protocol [Groth and Moreau 2008], a provenance architecture that met applications' needs [Groth et al. 2006], and an open specification, based on Web services technology [Miles et al. 2006].

This article extends and gives a more detailed analysis of a preliminary discussion of PrIMe presented at the Sixth International Software Engineering and Middleware Workshop 2006 [Munroe et al. 2006]. Specifically, in this paper, we expand upon the key steps of the methodological approach, we discuss their novelty in the context of alternative approaches, we apply them to two applications, which allows us to develop an analysis of the effectiveness of PrIMe.

2. MOTIVATION

We undertook a study of provenance-related requirements in a range of science-based applications [Miles et al. 2007], and it is helpful to draw examples from these to understand the motivation for specialised techniques for engineering provenance-awareness. First, we can note that provenance questions, by their nature, concern processes potentially a long time after they have occurred. For instance, a team of biologists performing experiments to identify proteins in samples presented the following case. When conducting a new experiment, they wished to re-use machine settings (voltage etc.) for successful past experiments of a similar kind. Given that the success or otherwise of identifying a protein was based on several experiments, the connection between the final result and the initial settings used in experiments is not straightforward to capture and, without having considered the issue in advance, the biologists were not capturing these connections at the time of running the experiment. A long enough period, in which machine settings would be changed multiple times, would elapse between experiments so the biologists could not fulfil the use case.

One reason that ensuring provenance-awareness is non-trivial is that past processes may be remote and independent from the user asking the provenance question. For example, the Atlas experiment at CERN will produce (and has been simulating the production of) vast amounts of data from high-energy particle collisions. Scientists around the world use this data but, due to the size and breadth of the data, it must be filtered, annotated and otherwise pre-processed centrally before it can be used for analyses. The provenance of the particle collision data is important for the physicists to understand how the data they receive has been pre-processed. In particular, a physicist may not wish to rely on data processed using a given library that they believe to have bugs [Branco and Moreau 2006]. In this distributed application, the provenance questions concern (at least in part) remote, independent processes which must have been adequately documented centrally for the remote scientist to receive an answer.

Issues of trust and reliability can add to the difficulties in obtaining suitable answers to provenance questions. For example, we discussed, with bioinformaticians, a case in which a biologist attempts to patent a new drug. As part of the check on this application, reviewers ensure that the experiments determining the drug's potential function was not discovered using any databases that are free only for non-commercial use. Where the documentation about the experiment process exists but can be modified by the scientist, its reliability may be called into question. Ideally, the reviewers would wish to see the accounts of both scientist and databases regarding the experiment process. To connect the records in such a way that the relevant provenance questions can be answered is, again, non-trivial, and guidance in the form of an engineering technique can be usefully applied.

Finally, we have to consider what is recorded about a process. There are a potentially large number of factors that can influence some processes, even in a purely software-based process (concurrently running processes, faults that have occurred, the full contents of a database etc.) Any of this data could be the subject of a provenance question but not all of it can be recorded. We need a principled way of setting out what should be captured and what should not. The results of this analysis will necessarily be highly dependent on the application, and on the users influencing the design process, but it is still valuable to provide generic guidance for how to relate provenance-related requirements to the data about a process included or excluded for capture.

In summary, from our analysis of provenance-related requirements in a wide range of applications, there is a clear need for guiding techniques to know what to record about a process, and how to adapt the application to do so, in order to answer future provenance questions. Futher, the major issues that such techniques must address go beyond those of generic methodologies, i.e. in considering access to attributes of long-past, independently-controlled processes with non-trivial connections to the data available to those users trying to obtain access.

3. OVERVIEW OF PRIME

PrIMe is a guided approach for making applications *provenance-aware*. This is achieved by exposing application information that is then documented through a series of *analysis steps* and well-specified *adaptations*, which are application modifications (supported in implementation by elements of a *provenance architecture* [Groth et al. 2006]) to record documentation of processes as they execute.

As with software engineering techniques in general, in developing PrIMe, we aimed for criteria such as *effectiveness*, that PrIMe should enable the recording of *useful* process documentation, *usability*, that PrIMe should be easy to apply, *traceability*, that all design decisions made using PrIMe should be traceable back to one or more use case requirements, and *applicability*, that it should be possible to successfully apply PrIMe to a wide range of applications. Considering these criteria, we assess the value of PrIMe in Section 10.

3.1 Overview of the Provenance Architecture

The *provenance architecture* [Groth et al. 2006] is a technology-neutral data model for provenance and specifications of a *provenance store*. A provenance store is a (database-backed) component providing functionality for recording and storing process documentation, and for performing queries over this documentation in order to obtain the provenance of data items. The architecture more generally addresses concerns such as scalability and security. It adopts the following definition of provenance:

DEFINITION 1 PROVENANCE OF A PIECE OF DATA. *The provenance of a data item is the process that led to that data item.*

Here, we take 'process' to mean 'a succession of actions' as defined in the Oxford English Dictionary. Note that, while the provenance of a piece of data is itself conceptual, the methodology is concerned with the recording and extracting of documentation representing that provenance.

The architecture has been strongly influenced by the service-oriented architectural style [Booth et al. 2004; MacKenzie et al. 2006], according to which services, also referred to as *actors*,

interact with each other by exchanging messages. We have examined a range of applications, including those discussed in the preceding section, with very different structures and implementations, and found each to be mappable to a service-oriented style. This is because, as the service-oriented approach argues, all processing of data can be described as the processing of inputs to produce outputs at an abstract level, i.e. the level of a service description, and that further detail about such processing can be expressed by decomposing the abstract description into a set of other input/output processing, i.e. one service's operation can be described as the interaction of multiple other services. This description can be applied to workflows (where data flows in and out of workflow steps), databases (where queries and updates apply to the database or a relation or a value, depending on the level of description), distributed system communication (where each node in a network acts as a service taking input and producing output) etc.

The provenance store supports two operations: recording and querying process documentation. Querying the provenance of a given data item involves identification of the data item at a specific point during execution (e.g. when it was created, stored or presented to the user), and *scoping* of the query to specify what aspect of that item's history is relevant for the querier (there may be a vast amount of documentation somewhat connected to the item's production). The output of a query is a subset of process documentation representing a portion of the data flow graph, denoting the processes leading up to the specified point in execution, i.e. the data item's provenance.

Process documentation is structured as a set of *assertions* about processes at particular instants, which we call *p-assertions*. P-assertions are asserted by, i.e. recorded into a provenance store by, the actors involved in the process about which assertions are being made, during (or shortly after) execution. Briefly, three types of p-assertions are defined as follows. *(i) Interaction* p-assertions record the messages sent between application components, and so provides a model of information flow between components. *(ii) Relationship* p-assertions record the relationships between a component's incoming and outgoing messages, and so provide an abstracted model of information flow within a component. *(iii) Actor state* p-assertions record the state of actors as the time at which a message is sent or received, and so provide additional data on the components involved in a process. We note that the combination of interaction and relationship p-assertions provide an integrated account of an application's data flow. Technical details and examples of the data model are described in Section 5.

Recording copies of application data in p-assertions, means that data which exists *transiently* is now not only stored, but is stored in a way which places it in the context of application processes. Together this enables us to perform queries of a particular kind, *provenance queries*, and so answer questions about the history of data produced in those processes. Note that the data which is recorded in p-assertions does not *have* to be transient: for a permanently kept file, for example, the data in a p-assertion will be a snapshot of that file at a given instant, i.e. a version of that file.

3.2 The Effects of Applying PrIMe

In order for an application to use implementations of the provenance architecture, developers apply PrIMe to the application's design and, by doing so, enable the integration of their application with the provenance architecture.

Applying PrIMe to an application produces two distinct *views* of the application as we shall now discuss. Recall that the provenance architecture assumes that an application can

be seen as being composed of *actors* that *interact* with one another through the exchange of *messages*. If the application is not designed from this viewpoint or with this terminology, the process of applying PrIMe essentially re-conceptualises it in these terms. Therefore, the first effect of applying PrIMe is to produce an actor-based view of the application (see Section 7). The key aspects of this actor-based view are described below.

—All changes in an application state are produced by the actions of *application actors*.

—All actions by an application actor are triggered by the receipt of new information/data by that actor.

—All information received by an application actor is sent by one or more other application actors.

—All process documentation is created by application actors that have direct access to the information being asserted, i.e.

(1) If an actor receives or sends data, it can assert and record that data.
(2) At the time of receiving/sending, an actor has a given state and so can assert and record details about that state.
(3) An actor can know, and so assert and record, how the information in two messages it has sent/received are related, e.g. one is in response to another.

For example, a *data compression service* (an actor) is sent uncompressed data (an interaction) by a client (another actor), compresses it (relationship between compressed and uncompressed data), and returns the compressed data (another interaction). At the time of returning the data, the compression service is using a particular compression algorithm and has a particular time on its local clock (both examples of actor state).

The second view of the application that results from applying PrIMe is generated once all of PrIMe has been applied and the application has been made provenance-aware. Here, once the provenance-aware application has finished execution, descriptions of its execution will have been recorded into a provenance store, which effectively represents an abstract description of the execution of the application, and thus provides another view of the application. Applying PrIMe to an application produces an actor-based view, or model of the application which, in turn, leads to a process documentation model of the application being stored in a provenance store.

3.3 The Structure of PrIMe

The overall structure of PrIMe is shown in Figure 1. Each oval in the diagram corresponds to a distinct step within the methodology and the lines between each step indicate how they are related. The dashed ovals delimit three different phases of the methodology, comprising, from top to bottom: Phase 1, the identification of use cases termed *provenance questions*, which should specify the data items for which provenance is sought as well as the form, or kind, of provenance sought (termed the *scope* of the question); Phase 2, the decomposition of the application into a set of actors and their interactions (this phase is iterative for reasons described in Section 7.5) and; Phase 3, applying a set of principled adaptations to the application to expose the information necessary to obtain answers to provenance questions.

Traversing this process, PrIMe starts from the application design itself, and it is assumed that the structure and purpose of the application is known beforehand. This does not mean that the application design must already exist completely, but that the overall functionality

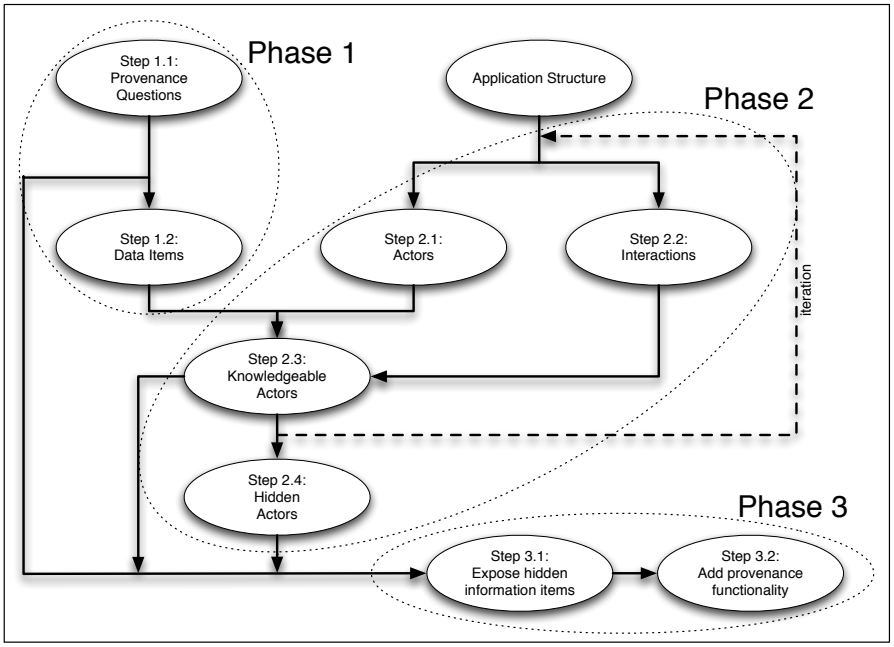


Fig. 1. Overall structure of PrIMe

has been identified and the general structure has been determined. Given this assumption, a designer applying PrIMe performs the following steps (which we expand on later in the paper).

First, an analysis is performed to identify the set of *provenance questions* that are to be answered by the provenance architecture (Step 1.1), then the *data items* (pieces of information) for which provenance is sought are identified as well as a *scope* for the answer (Step 1.2). The application structure is then examined to identify the *application actors* (Step 2.1), and from here the *interactions* between application actors are mapped out and the relationships between interactions are detailed, thus revealing the information flow through the application (Step 2.2). Once this is done, it is then possible to determine which application actors are potentially implicated in the provenance of a given data item: these actors are called *knowledgeable actors* (Step 2.3). At this point, it may become clear that the decomposition of the application into actors has not been at the right level of granularity; this is apparent when it is found that it is not possible to locate either the data item for which provenance is sought and/or part or all of its provenance within the current model. In this case, the process of identifying actors and interactions is repeated until either the data item in question and/or part, or all, of the provenance can be located within the interactions of identified actors and the relationships between those interactions (in Section 10, we describe an procedure for this iterative process that guarantees that such data items and provenance will be discovered in the application if they exist). Once the correct level of

granularity necessary to answer provenance questions has been discovered, it is important to consider which actors cannot be adapted to record documentation (Step 2.4). Finally, *adaptations* are introduced into the application in order to expose information necessary to obtain the provenance (Step 3.1), and to add recording functionality to the identified actors (Step 3.2). This last step involves giving actors the capability to record *process documentation* so that it can be produced and stored in provenance stores to allow actors to perform queries on the documentation in order to answer provenance questions.

PrIMe explicitly only addresses one factor of an application's design, provenance, allowing it to build in assumptions which aid the designer in satisfying use cases related to this factor. This means that every other factor of an application must be addressed by other techniques. As we cannot predict by what methods a designer will address these other factors, or whether provenance-related requirements will be addressed at the same time or later than others, we have endeavoured to ensure PrIMe is as independent of other techniques as possible. What this means in practice is that, in analysis, applying PrIMe involves building models of the application and performing analysis on the models particular to PrIMe, rather than relying on the prior existence of an appropriate design. Clearly, where there are existing design models which closely match those used in PrIMe (particularly a service-oriented view of the application), they can be re-used to the designer's benefit. In later phases, PrIMe causes the adaptation of the application design to address the provenance use cases, but again we try to ensure that the effect on the prior design is minimal by relying on *wrappers* around existing components.

This section has provided a brief overview of PrIMe. In Sections 6 to 8, we provide the steps by which PrIMe can be applied in detail. In particular, we describe how PrIMe is used to identify what kinds of information within an application should be the focus of such p-assertions, and how they can be captured. However, first we introduce an example application that we will use later to ground the discussion.

4. THE AMINO ACID COMPRESSIBILITY EXPERIMENT (ACE)

We applied PrIMe to an existing bioinformatics application [Groth et al. 2005] for which the primary user wished to be able to ask a number of provenance questions regarding their data. The experiment is outlined below, then used as a running example to illustrate PrIMe in the following sections.

4.1 The 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 exible 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 workow for the experiment is shown in Figure 2. 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 (Collate Sample), the information efficiency of the sample can be calculated (Calculate Efficiency).

The efficiency calculation can be broken down into a smaller workflow shown in Figure 2(b). First, the symbols in the sample are substituted with those of a given group (Encode). This recoded sample is then compressed with compression algorithms, e.g., gzip, bzip2 or ppmz, to obtain the length of the compressed sample sequence (Compress). The Shannon entropy is then computed on the recoded sample to provide a standard for comparison (Compute Entropy). This standard removes the inuence 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 (Information Efficiency Calculator).

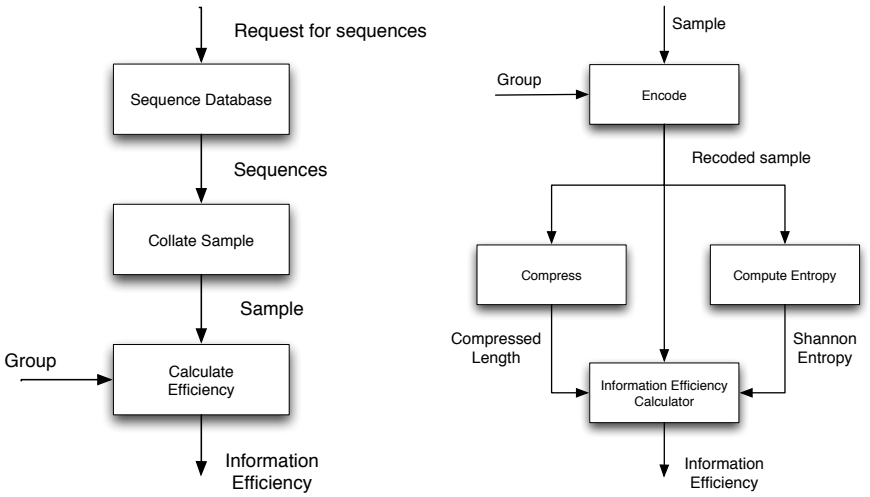
After the information efficiency values for different groups have been produced by the workflow, they can then be plotted to find those that are largest and thus are good candidates for further investigation.

4.3 Experiment Enactment

The experiment is run as a workflow script, calling each of the services in Figure 2, passing outputs from one service onto the next as depicted by the arrows in the diagrams. The first act of the workflow is to send a request for sequences to the Sequence Database, which triggers the rest of the experiment enactment. The user provides the Group argument for encoding sequences, and the final result, Information Efficiency, is returned to the user on completion. The inputs to the Calculate Efficiency sub-workflow (Sample, Group) are provided by the Calculate Efficiency service, which invokes each service in the sub-workflow in turn, and receives the Information Efficiency value in response.

5. A DATA MODEL FOR PROVENANCE

Earlier, we introduced the notion of documenting a system's execution and defined three kinds of assertion, which comprise process documentation. Specifically, documentation about the interaction between actors (interaction p-assertions), documentation about the internal processing of actors (relationship p-assertions), and documentation of the state



(a) Main Workflow (b) Calculate Efficiency Expanded

Fig. 2. The ACE application

⟨ps:pstruct⟩
+⟨ps:interactionRecord⟩⟨/ps:interactionRecord⟩
+⟨ps:interactionRecord⟩⟨/ps:interactionRecord⟩
+⟨ps:interactionRecord⟩⟨/ps:interactionRecord⟩
+⟨ps:interactionRecord⟩⟨/ps:interactionRecord⟩
⟨/ps:pstruct⟩

Fig. 3. The top level of the p-structure.

of an actor (actor state p-assertions). To ensure that documentation can be cohesively organized when provided by a multitude of different actors, we introduce the *p-structure*, a data model for such documentation and the schema for every provenance store. The data model itself has been described elsewhere along with the design decisions involved [Groth et al. 2008]. Here, we provide an overview of the data model with examples from the documentation that ACE records. The examples are provided as XML. We use the following notation within the XML.

—A '+' before an element means that it contains additional content.

—Elements preceded by ps: are those defined by the p-structure schema.

—Elements preceded by ns1: are defined by some other external schema.

—Elements preceded by ace: are those defined by the ACE's own schema.

The p-structure consists of a set of *interaction records* as shown in Figure 3. Each interaction record groups together p-assertions related to the communication of a single message from one actor to another.

The structure of an interaction record is shown in Figure 4. Each interaction record is identified by an interaction key, which consists of an identity for the actor that sends the message (i.e. sender), the identity of the actor that receives the message (receiver), and

```
⟨ps:interactionRecord⟩
    +⟨ps:interactionKey⟩⟨/ps:interactionKey⟩
    +⟨ps:sender⟩⟨/ps:sender⟩
    +⟨ps:receiver⟩⟨/ps:receiver⟩
⟨/ps:interactionRecord⟩
```

Fig. 4. The structure of an interaction record.

```
⟨ps:interactionKey⟩
    ⟨ps:messageSource⟩
        ⟨ns1:EndpointReference⟩
            ⟨ns1:Address⟩
                http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/CollateSample
            ⟨/ns1:Address⟩
        ⟨/ns1:EndpointReference⟩
    ⟨/ps:messageSource⟩
    ⟨ps:messageSink⟩
        ⟨ns1:EndpointReference⟩
            ⟨ns1:Address⟩
                http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/CalculateEfficiency
            ⟨/ns1:Address⟩
        ⟨/ns1:EndpointReference⟩
    ⟨/ps:messageSink⟩
    ⟨ps:interactionId⟩
        e71acbcc-6e05-46d0-b2ab-a65c1d1d453d2
    ⟨/ps:interactionId⟩
⟨/ps:interactionKey⟩
```

Fig. 5. An example interaction key

a field that distinguishes this interaction (ps:interactionId) from all other interactions between the same two actors. Figure 5, presents an example of interaction key that identifies the sending of a message from the Collate Sample actor to the CalculateEfficiency actor. Each actor is identified by an end point reference.

Actors may describe a particular interaction at varying levels of detail, to capture any such differences in documentation that the sender and receiver create for a particular interaction. Each interaction record contains two views grouping p-assertions from the sender and receiver, labeled a Sender View and Receiver View. Each p-assertion within a view is given a local p-assertion id that distinguishes it from other p-assertions in the same view. Figure 6 shows an example interaction p-assertion. Along with the local p-assertion id, it contains a documentation style that tells queriers how to interpret the p-assertion's content as well the content itself. In this case, the content is a path to a file, which contains the sample produced by Collate Sample.

Actor state p-assertions are often used to represent data that provides context to a given interaction. For example, in Figure 6, the actor state p-assertion documents the group provided by the user to the Calculate Efficiency actor.

Figure 8 shows an example relationship p-assertion, which consists of an effect, possibly several causes and a relation between those causes and the effect. The relation name

```
⟨ps:interactionPAssertion ⟩
    ⟨ps:localPAssertionId⟩1⟨/ps:localPAssertionId⟩
    ⟨ps:documentationStyle⟩
        http://www.pasoa.org/docstyle/AceVerbatim
    ⟨/ps:documentationStyle⟩
    ⟨ps:content⟩
        ⟨ace:filePath⟩
            /Users/ace/collatedseqs/sample1
        ⟨/ace:filePath⟩
    ⟨/ps:content⟩
⟨/ps:interactionPAssertion⟩
```

Fig. 6. An example interaction p-assertion.

```
⟨ps:actorStatePAssertion⟩
    ⟨ps:localPAssertionId⟩3⟨/ps:localPAssertionId⟩
    ⟨ps:content⟩
        ⟨ace:group⟩
            b:A,c:R,d:N,e:D,f:C,g:Q,h:E,i:G,j:H,k:I,l:L,m:K,n:M,o:F,p:PT,q:SV,r:W,s:Y
        ⟨/ace:group⟩
    ⟨/ps:content⟩
⟨/ps:actorStatePAssertion⟩
```

Fig. 7. An example actor state p-assertion.

```
⟨ps:relationshipPAssertion⟩
    ⟨ps:localPAssertionId⟩2⟨/ps:localPAssertionId⟩
    +⟨ps:effect⟩⟨/ps:effect⟩
    ⟨ps:relation⟩
        http://www.pasoa.org/schemas/ace/relationships/isCollatedFrom
    ⟨/ps:relation⟩
    +⟨ps:cause⟩⟨/ps:cause⟩
⟨/ps:relationshipPAssertion⟩
```

Fig. 8. An example relationship p-assertion.

provides an abstraction of the internal process the actor performed to generate one data item from others. In this case, "isCollatedFrom" described the process of the Collate Sample actor. In practice, these relation names are URIs, which point to descriptions of the algorithms used by the actor.

To identify both causes and effects in a relationship p-assertion, the hierarchical nature of the p-structure is used. Figure 9 shows how the cause is identified by specifying the interaction key, the view (as denoted by a view kind), and the local p-assertion id of the p-assertion in question. Note that the effect is identified similarly, however, it is assumed that the effect is in the same view as the relationship p-assertion; thus, the interaction key and view kind can be omitted as shown in Figure 10.

In addition to identifying the p-assertion in question, each cause and effect has a ps:parameterName field, which labels the *role* that the data documented by the p-assertion played with respect

```
⟨ps:cause⟩
    +⟨ps:interactionKey⟩⟨/ps:interactionKey⟩
    ⟨ps:viewKind xsi:type="pasoa:SenderViewKind"⟩
        ⟨ps:description⟩isSender⟨/ps:description⟩
    ⟨/ps:viewKind⟩
    ⟨ps:localPAssertionId⟩2⟨/ps:localPAssertionId⟩
    ⟨ps:parameterName⟩sequences⟨/ps:parameterName⟩
⟨/ps:cause⟩
```

Fig. 9. An example cause.

```
⟨ps:effect⟩
    ⟨ps:localPAssertionId⟩2⟨/ps:localPAssertionId⟩
    ⟨ps:parameterName⟩sample⟨/ps:parameterName⟩
⟨/ps:effect⟩
```

Fig. 10. The representation of an effect in a relationship p-assertion.

to the relation identified in the relationship p-assertion. Thus, taken together Figures 8, 9, and 10 document that a sample was collated from a set of sequences.

The p-structure enables each actor involved in an interaction to create documentation for their portion of an application and have that documentation be assembled together in a fashion that allows for easy reference and query. Examples of queries will be presented in the following section.

6. PRIME PHASE 1: PROVENANCE QUESTION CAPTURE AND ANALYSIS

In the following three sections (including this one), we specify the actions involved in applying PrIMe. The ACE bioinformatics case study is used as a running illustration in these sections, but we also provide a second case study concerning a medical scenario (referred to as the *organ transplant application*) in Appendix A.

Aim: The aim of the first phase is to identify questions related to the provenance of data in the application, and express them in a consistent specification structure. These will then act as use cases in subsequent analysis and adaptation of the application design.

6.1 Description

As would be expected of a methodology, the results of applying PrIMe ultimately depend on the particular uses to which the system will be put, i.e. the use cases. By the term 'use case', we refer to the same form as found in UML, i.e. descriptions of scenarios in which users interact with an application defining the system functionality [Harman and Watson 1998]. However, PrIMe is dedicated to adapting designs for a very particular kind of use case, in which the system answers a question regarding the provenance of data which it itself produced. This means that the use cases all take a particular simple form because each provenance question is always a user query with a system response, and the only applicable scenarios are success or failure on the system's part. Given that each of these use cases is equivalent to a question about the provenance of application data, we will refer to PrIMe use cases as *provenance questions*.

A provenance question informs the results of applying the methodology in two ways. First, there is the straightforward influence of the system being adapted to be able to answer the question when asked by a user. Primarily, this means determining for which data exchanged within the system we can and need to keep a copy of, stored with the context of the process in which it was produced. Second, provenance questions are used to inform 'future-proofing' of the application, i.e. not just answering provenance questions known at the time of applying PrIMe, but also questions that users may want to ask in the future. This is an issue of particular importance for provenance because in order to answer a provenance question, the historical data answering that question must have already been recorded: when a user considers asking about the provenance of a data item for the first time, we cannot go back in time and record the data which would answer that question. The way that PrIMe is designed to handle future proofing is discussed in Section 10.4.

The first phase of PrIMe concerns specifying provenance questions in a form which can be used for further analysis in the following phases, and also to generate queries to a provenance store with the data schema described in the previous section. The phase is delimited into two steps:

(1) Elicit provenance questions from users.
(2) Translate provenance questions into a structured form suitable for analysis and for generating provenance queries.

6.2 Concept Definitions

The following concepts will be used, and illustrated with the case study, in the steps described below.

DEFINITION 2 DATA ITEM. *A piece of data produced in the running application.*

DEFINITION 3 PROVENANCE QUESTION. *A question which can only be answered from the provenance of an application data item.*

DEFINITION 4 PROVENANCE QUERY. *A formulation of a provenance question sent to a provenance store to retrieve the answer to the question.*

DEFINITION 5 START ITEM. *A data item that is the subject of a provenance question/query.*

DEFINITION 6 SCOPE. *A specification of what is relevant to retrieve from a provenance store by a provenance query for a given question.*

6.3 Step 1.1: Provenance Questions

Aim: The aim of this step is to elicit provenance questions from users.

Actions:

(1) Provide an explanation and definition of provenance in computational systems.
(2) Give examples of generalised provenance questions that can be answered by a provenance-aware application.
(3) Explain how to express provenance questions.

Documentation Produced:

—A list of textual provenance questions.

6.3.1 *Description.* The starting point for making an application provenance aware is for the designer to identify the set of provenance questions that are to be answered. Note that the reason for including this requirements capture stage is not to suggest it is novel, but to provide a complete cycle for the methodology and make explicit the approach that has worked in our experience.

As in most applications, it is not always obvious to users what provenance questions they could expect the provenance architecture to support. To overcome this, PrIMe advocates a simple requirements elicitation process, similar to many software engineering approaches (e.g. [Harman and Watson 1998]) to help designers collect the known provenance questions. This process aims to inspire users to identify provenance questions that they may wish to ask in the course of using the application.

The elicitation process comprises three simple steps. First, we provide an explanation and definition of provenance in computational systems. In order to provide examples of the kinds of provenance questions a provenance-aware application can support, several generalised, non-application specific questions can be supplied. These questions attempt to expose information relating to processes within an application, and try to identify various aspects of process such as the data used within a process and the adherence of a process to regulatory rules or plans, as well as questions relating to data, its use and transformation. Here is a small sample taken from a larger list collated from discussions with users published elsewhere [Miles et al. 2007].

(1) What was the process that produced a given piece of data?
(2) Two processes, thought to be performing the same steps on the same inputs, have been run and produced different data. Was this because of a change in the inputs, the steps making up the process or the configuration of the process?
(3) Did the process that produced this data use the correct types of information at each stage?

The first question above is a purely a request for the provenance of a data item, answered by querying a provenance store with an appropriate query. The subsequent questions require not only querying a provenance store, to retrieve data on the provenance of a data item, but also processing the results of that query.

6.3.2 *Case Study.* In ACE, we elicited a set of provenance questions including the following, which we will refer to in the rest of the methodology description.

(1) What were the sequences used in the production of a particular information efficiency value?
(2) From what sequences was this recoded sample derived?
(3) How long does it take to produce an information efficiency value from a particular collated sample?

The application of this step to the organ transplant management application is described in Section A.1.1.

6.4 Step 1.2: Analysing Provenance Questions

Aim: The aim of this step is to translate provenance questions into a structured form suitable for analysis and for generating queries to a provenance store, called *provenance queries*.

Actions: For each provenance question, the following actions are performed.

(1) Identify the *start item* of the provenance query, i.e. the data item of which the provenance will be sought.
(2) Identify the *scope* of the provenance query, i.e. the relevant part(s) of the application process which led to the start item.
(3) Identify the additional processing steps, if any, required to extract or determine the question's answer from provenance query results.

Documentation Produced:

—For each question, a tabular form containing the details of the provenance query as identified in the steps above.

6.4.1 *Description.* In order to determine the provenance of a data item, a query is sent to a provenance store. A provenance store contains data of the form shown in Section 5, and so the queries sent to the store reflect this form. For each provenance question, we need to identify the *start item* of the query, the *scope* of the query and any additional processing which must be performed on the query results to answer the provenance question. The start item is the data item we are asking for the provenance of, while the scope is a specification of what in that item's history is relevant to the provenance question. This information is documented in a tabulated form for use in later stages of PrIMe.

We will not give a comprehensive definition of how to identify start items or encode scopes here, as it would have to be extensive to be complete and is adequately documented in existing documents [Groth et al. 2006; Miles 2006]. However, we will briefly examine the key points below.

6.4.1.1 *Identifying the start item.* In order to query a provenance store for the provenance of a data item, that item must be identified. In some cases, the data item will be uniquely identified as part of the application, e.g. it may have a filename with a version number, URL, database table key etc. Where a data item is not given a unique identifier in the original (unadapted) application, data items must be defined *intensionally* in the provenance query. For example, a start item may be identified as "the sequence downloaded at time 13:51" or "the version of the file named A at time of having data X appended to it". Such intensional definitions also apply to data items which do have a unique identifier, e.g. "the file named A-version23".

These definitions can be converted into queries over the provenance store data structure, to find records of a data item which match those criteria. As we have used an XML data structure in our implementation, as discussed above, we use XPath to define the provenance queries. For example, an XPath identifying instances of a file with a particular path being exchanged in a message between actors, can be encoded as follows.

```
//ps:interactionPAssertion/ps:content
  [ace:filePath='/Users/ace/collatedseqs/sample1']
```

6.4.1.2 *Specifying the scope.* Depending on how an application is adapted, the provenance of a data item may include a vast amount of information. For example, the efficiency value output by our case study workflow was derived from, amongst other data, the biological sequences downloaded from a public database; but those sequences were uploaded,

potentially with multiple versions to correct errors, following lab-based experiments, and so the efficiency value also derives from the inputs to those experiments; the sequence data was identified and checked using information on comparable sequences provided by others from prior experiments, possibly also stored in public databases, and so the efficiency value also derives from those comparable sequences; and so on.

Any piece of this information may be relevant to answering some provenance question, e.g. if you doubt the integrity controls of the database, you may want to know which sequences influenced your efficiency value. However, it would be, from the perspective of giving a coherent answer to a provenance question in a timely manner, impractical to return all this data in answer to every provenance store query. Therefore, the *scope* of the query is provided along with start item identification, to ensure the query results are bounded to that relevant for the question being asked. For example, if the question can be answered using information about the provenance of an efficiency value from the collation of sequences into a single sample, then the scope will exclude all occurrences prior to collation occurring. In our current provenance store implementation, the scope is also specified as an XPath, so the above example would take the following form (where we are saying that the query results should not include data prior to the point where sequences were collated into a sample, denoted by a particular relationship type).

```
/ps:relationshipTargetFilter
  [not(ps:relation='http://www.pasoa.org/schemas/ace/
  relationships/isCollatedFrom')]
```

We will not discuss the meaning of the scope terms or the expressivity of the scoping mechanism here, as it goes beyond the scope of this paper, but refer readers elsewhere [Groth et al. 2006; Miles 2006].

6.4.2 *Case Study.* We now work through each of the provenance questions identified in step 1.1 for the ACE example. For each we determine the start item, scope and any processing which must be done on the provenance store query results to obtain the answer to the question.

For the question "What were the sequences used in the production of a particular information efficiency value?", the data item for which we need to find the provenance is a recoded sample, and the only documentation of relevance to the question is the input sequences and their connection to the recoded sequence, so the scope of the provenance query is limited to that which occurred from download of sequences onwards. After the provenance has been found, we need to extract the sequences from the query results. The query is summarised in Table I.

Table I. Provenance question 1

Provenance question	What were the sequences used in the production of a particular information efficiency value?
Start item	The information efficiency value
Scope	From input of sequences onwards
Processing step	Extract sequences from provenance

The question "From what sequences was this recoded sample derived?" is of a similar form to the above, but now the question concerns a recoded sample rather than an efficiency

value. Therefore, there is a change of start item, as shown in the summary in Table II.

Table II. Provenance question 2

Provenance question	From what sequences was this recoded sample derived?
Start item	The recoded sample
Scope	From input of sequences onwards
Processing step	Extract sequences from provenance

The final question, "How long does it take to produce an information efficiency value from a particular collated sample?" illustrates that some questions require more post-processing to answer than others. In this case, the information efficiency value is the start item, and we only need to know the provenance back to where the sequences were collated to a sample. However, the processing of the query results is a little more involved (and application-specific): we need to extract time-stamps from the p-assertions documenting the collated sample being passed between actors (the start time) and the information efficiency value being output (the end time). The query is summarised in Table III.

Table III. Provenance question 2

Provenance question	How long does it take to produce an information efficiency value from a particular collated sequence?
Start item	The information efficiency value
Scope	From the collated sequence being produced
Processing step	Calculate difference in time-stamps between production of collated sequence and efficiency value

The application of this step to the organ transplant management application is described in Section A.1.2.

7. PRIME PHASE 2: ACTOR BASED DECOMPOSITION

Aim: The aim of this phase is to map the application design into a form composed of actors and interactions, within which we can identify the data items necessary for answering provenance questions.

7.1 Description

The purpose of this phase of PrIMe is to map the application to a model of *actors* and *interactions* from which we can start to define adaptations which will allow provenance questions to be answered. It is important to recognise that the resulting model of the application may or may not coincide with the primary componentisation of the original application. Actors identified using PrIMe may cut across the functional specification of the application, at times mirroring its underlying structure, while at other times bringing together disparate levels of functionality for the purposes of answering provenance questions. Components may be grouped together into one actor or a component may be sub-divided into a set of smaller actors.

In essence, the particular actor-based model that is superimposed on the original application design is a function of the provenance questions that are to be answered. In order

to identify the correct level of granularity, PrIMe takes an *iterative* approach, in which the following steps are carried out until the correct level of decomposition has been achieved.

(1) Identify an initial set of high-level actors that correspond to the major functional subdivisions of the application (Step 2.1).
(2) Map out the interactions between these actors (Step 2.2).
(3) Identify the causal relationships between actors, so that the process by which data items are produced is made explicit (Step 2.3).
(4) Identify the start items (and data items referred to in query scopes) within the interactions, or else begin another iteration of decomposition into finer-grained actors (Step 2.4).

This process stops when all of the data items salient to answering the provenance questions have been identified.

7.2 Concept Definitions

The following concepts will be used, and illustrated with the case study, in the steps described below.

DEFINITION 7 ACTOR. *An* actor *is a component within an application that performs actions and which interacts with other actors.*

DEFINITION 8 INTERACTION. *An* interaction *is the sending and receiving of one message between actors.*

DEFINITION 9 MESSAGE. *A* message *is a delimited block of data which is sent between actors in an interaction.*

DEFINITION 10 CAUSAL RELATIONSHIP. *A* causal relationship *between data items means that one data item (the effect) would not have been as it is if the other data items (the causes) had not been as they were, i.e. the effect is* derived *from the causes.*[1]

DEFINITION 11 PROCESS. *A* process *is a succession of actions, here taken as the actions of sending and receiving data.*

DEFINITION 12 KNOWLEDGEABLE ACTOR. *For a given data item, a* knowledgeable actor *is an actor which has access to that data item (so may be able to record a copy of it in a provenance store or pass it to another actor in a message).*

DEFINITION 13 INACCESSIBLE DATA ITEM. *An* inaccesible data item *in a model of the application, is a data item which is neither passed within a message in that model (and so will not be recorded to a provenance store) nor derivable from other data items.*

7.3 Step 2.1: Identifying Actors

Aim: Identify the actors that comprise the application at a high level of granularity.

Actions:

[1] Unambiguously defining causality is an unresolved philosophical issue, so this identifying causation is partly subjective for the designer, but we derive our definition from the commonly used counter-factual definition proposed and discussed by Lewis [1973] and formalized by Pearl [2000].

(1) Identify components that are the receivers of data.

(2) Identify components that are the senders of data.

Documentation Produced:

—A list of *actors* (those modelled at a given granularity).

7.3.1 *Description.* An *actor* is simply a component within an application that performs actions, such as a Web Service, a class, a machine, a person and so on, and which interacts with other such actors. We use the term 'actor' to refer to *instances* of users or system components. This contrasts with some other technologies, such as UML [Harman and Watson 1998], where it is used to refer to *types* of user or system component. We would expect designs referring to types of actor to map directly to the same structures and processes as used in a given instance of those types. PrIMe applies to instances rather than types because it is the particular processes which provenance questions concern and it is the particular instances of actors which are adapted to record documentation of the processes they are involved in.

Often, the application being considered will be structured into major functional components that can be readily viewed as actors in a first instance. However, in order to aid the identification of actors within applications in which this is not the case, PrIMe provides simple heuristics to identify them.

(1) Identify the components that receive data. These could, for example, be a component/service in a workflow, a script command, the GUI/desktop application into which a user enters information. Each of these components is an actor in the application.

(2) Identify the components that send data. These could be, for example, a workflow engine, a script executor, a user or a sensor (such as a blood pressure monitor for example). Each of these entities is also an application actor.

While the above rules clearly identify the actors in an application, simply applying them wholesale can lead to a more detailed decomposition of the application than is required to answer the provenance questions. At this point in the methodology, the designer should merely identify a coherent set of coarse-grained actors which they can iteratively refine in later steps.

7.3.2 *Case Study.* In applying PrIMe to ACE, we first identify the actors in the main workflow, i.e. those shown receiving or sending data in Figure 2 (a): Sequence Database, Collate Sample, and Calculate Efficiency. Note that we could also include actors such as the workflow enactor and the user, but we keep the illustration brief here.

The list resulting from this step is as follows:

—Sequence Database

—Collate Sample

—Calculate Efficiency

The application of this step to the organ transplant management application is described in Section A.2.1.

7.4 Step 2.2: Actor Interactions

Aim: Identify the interactions between actors that comprise the application at a high level of granularity.

Actions:

(1) For each actor, identify to where it passes data, and from where it receives data.
(2) For each interaction, model the message type in terms of the kinds of its contents.
(3) For each data item (effect) sent by an actor, identify those other data items (causes) received by or part of the state of the actor, for which the effect data item would be changed if the cause data items were changed or did not exist.
(4) For each cause-effect relationship, identify the type of that relationship

Documentation Produced:

—An *interaction graph*, depicting the messages sent between actors and the causal relationships between data items in those messages.

7.4.1 *Description.* The provenance architecture is based around the notion of *interaction*, and all documentation about process occurs in the context of interactions. While other software methodologies, for example the Gaia methodology used in agent oriented design [Wooldridge et al. 2000], also involve the concept of interaction, they focus on identifying interactions for the purpose of outlining and specifying system functionality. In PrIMe, however, identifying interactions enables data flow to be documented. As such, PrIMe requires a representation of such interactions in the form of *message passing* to be stated explicitly along with identification of the content of such messages (i.e. data). This then provides a model of how data is being passed around between actors, and allows the provenance architecture to be used to collect information about the application's processes.

To document this, PrIMe uses a graph notation to represent the messages being passed between the identified actors. In Figure 11, one of these graphs (termed an *interaction graph*) is shown. The nodes represent individual actors and the arcs (shown as unidirectional arrows) represent messages being passed from one actor to another. Annotations over the arrows show identifiers ascribed to message types and the data items carried in messages of these types in brackets. The message identifiers are just for convenience of reference, and should merely be unique within the interaction graph. Interaction graphs make explicit the data required to answer provenance questions, and which actors have access to the data (see Section 7.5).

Although the above interaction graph clearly shows the data flow between application actors, it does not provide enough information to understand how interactions are causally related. Consider that in Figure 11, for example, it is not possible to see how message M1 is related to messages M2 and M3, and so it is not possible to see the direction of data flow. To understand this, we must also show the data flow *within* actors, i.e. we must show how the sending of messages is *causally related* to the receiving of messages. However, since an actor may subsume the functionality of several application components (for example, an actor in the main workflow of ACE comprises a sub-workflow), the relation between the data in an incoming message and the data contained in a consequent outgoing message may be complex, involving many transformations of the original data and/or the retrieval of other data.

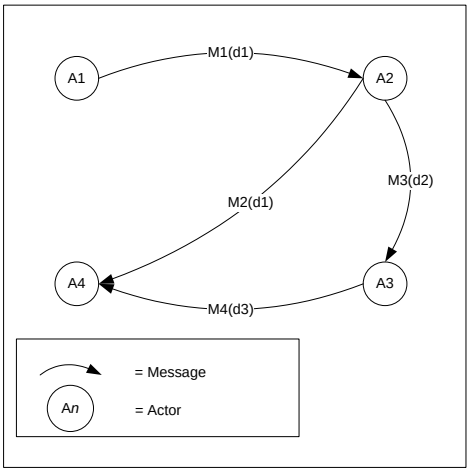


Fig. 11. An interaction graph

Fortunately, the actor-based view allows us to abstract away this detailed functionality and replace it with a simple relationship, which Figure 12 depicts. The data in the incoming message is transformed by applying to it the function F to produce the data that is sent in the outgoing message. For example, in ACE, the sample (output from actor Collate Sample) is collated from a set of sequences (input to actor Collate Sample). Actors may provide more than one service (for example, a database provides both query and update functionality).

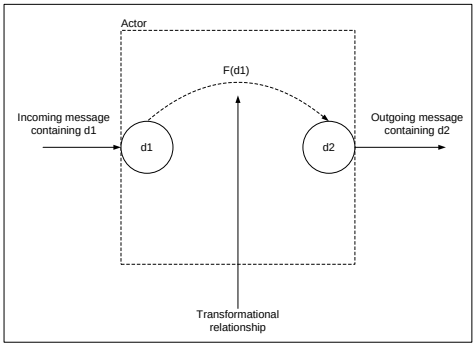


Fig. 12. The process structure assumed in PrIMe, where each actor receives inputs and transforms them into sent outputs, but is otherwise a black box.

7.4.2 *Case Study.* Taking the actors listed in the previous step, we depict an interaction graph making explicit the messages sent, data items exchanged, and causal relationships bewteen items, shown in Figure 13. As described in Section 4, the sequence database is queried to retrieve a set of sequences, so the sequences being returned are caused by the database query; then the sequences are collated into a sample, so the sample is caused by

the sequences; and the information efficiency is calculated over that collated sample, so the information efficiency is caused by the sample. Therefore, as long as all these causal connections are documented, we can trace back from the information efficiency value to the input sequences: the latter is part of the provenance of the former.

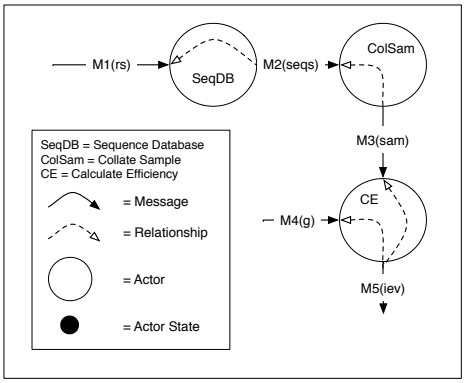


Fig. 13. An interaction graph depicting the interactions and relationships in enactment of the ACE workflow

The application of this step to the organ transplant management application is described in Section A.2.2.

7.5 Step 2.3: Decomposition to Knowledgeable Actors

Aim: For each data item required to answer a provenance question, identify how to expose it in the interaction model, so that it can be captured.

Actions:

(1) For each data item, determine whether it is communicated in one of the messages sent/received in the current interaction model.
(2) For each data item not exposed in the manner above:
   (a) Determine whether it is created and destroyed within the process of an actor in the current interaction model.
   (b) If so, apply this whole phase again to a decomposition of the actor in question.
(3) For each data item not exposed in either manner above:
   (a) Determine whether it could be computed from other data items which may be easier to capture.
   (b) Apply this step to those data items.
(4) For each data item not exposed in any manner above, document that it is not currently accessible within the system.

Documentation Produced:

—A new interaction model, in which all data items apart from those deemed inaccessible, are contained in messages between actors or derivable from data items in those messages.

—A set of derivation functions, each specifying how one data item is derived from others.

—A list of currently inaccessible data items.

7.5.1 *Description.* In this section we examine what can be done when it is discovered that the current level of decomposition of an application does not allow a provenance question to be answered. This involves the search for *knowledgeable actors*, where a knowledgeable actor for a data item is an actor that can be identified as the creator or a recipient of the data item during an application process.

Answering a provenance question requires discovering particular data within an item's provenance, e.g. the sequences used to generate an information efficiency result in the ACE case study. Such a past data item may not be present in the interaction model produced in the previous step either because *(i)* it is created, used and discarded solely within the processing of one actor treated as one black box in the interaction model or *(ii)* it is never present explicitly in the application at all but may be derivable from data which is present.

In the former case, the granularity of the interaction model is inadequate and this must be corrected by decomposing the actor manipulating the data item. This means re-performing step 2.1 and 2.2 to expose the processes (interactions and relationships) within the actor and include its components in the list of actors modelled [2].

In the latter case, the data item is obtained by a derivation function performed over the results of provenance store queries, and the function must be made explicit.

Even if a data item is present in the interaction model, copies of it may not be recorded into provenance stores due to the nature of the actors creating/receiving it, e.g. some legacy components may be difficult to adapt to record such copies. In such a case, we say the data item is *inaccessible*. Adaptations to the application to try to record the data item are considered in the next phase, Section 8.

7.5.2 *Case Study.* Consider the provenance questions examined in Section 6.4.2 and the interaction graph produced in the previous step, Figure 13. The first question ("What were the sequences used in the production of a particular information efficiency value?") refers to two data items: the information efficiency value and the sequences from which it is derived. Both of these data items are explicit in the interaction graph (denoted 'iev' and 'seqs' respectively), and have at least one knowledgeable actor: Calculate Efficiency for the efficiency value, Sequence Database and Collate Sample for the sequences.

However, the second question, "From what sequences was this recoded sample derived?" refers to another data item, the recoded sample, which is not present in the interaction graph. This is because the sample is created and used solely to calculate the information efficency, i.e. within the Calculate Efficiency actor.

We must, therefore, apply steps 2.1 and 2.2 again, to refine the interaction graph, by including the process inside Calculate Efficiency. The Calculate Efficiency service is a sub-workflow (a program), calling other services and using their outputs. Its operation is shown in Figure 2: the collated sample is input into an Encode service which also takes as input (through its configuration), the group encoding. The output of this service, the

[2]The approach of gradual decomposition of a system, expressing each level using the same abstraction, is directly comparable to *holonic* approaches [Marik et al. 2005]. A holon is an autonomous and cooperative building block, consisting of an information processing part and, often, a physical processing part [Giret et al. 2005]. Importantly, a holon can be part of another holon, so a system can be described as a hierarchy of holons gradually decomposed to reveal the comprising functionality.

encoded sample, is then passed to a Compress service, to compress the encoded sample, and a Compute Entropy service, to compute the sequence's entropy. Using the size of the encoded sequence when uncompressed and compressed, and adjusting for the Shannon entropy, the information efficiency of the sample is calculated.

The revised interaction graph is depicted in Figure 14, with the added sub-process highlighted in grey, and now includes the information we are referring to in the provenance question: the recoded sample. There is an indirect connection from the recoded sample and the input sequences, so the latter are part of the former's provenance.

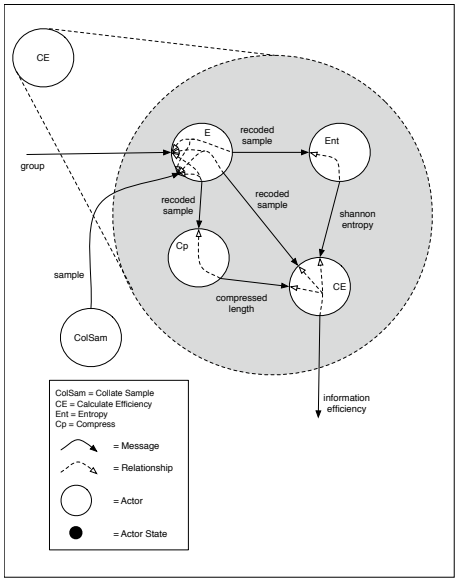


Fig. 14. An interaction graph depicting the interactions and relationships in enactment of the ACE workflow

As shown in Table III, the third question ("How long does it take to produce an information efficiency value from a particular collated sample?") refers to a few data items. Two of these, the information efficiency value and the collated sequence are already explicit in the interaction graph. The final result, the time to produce the efficiency value, is not explicit but derived from other data items: time stamps at the start and end of the process. The time stamps themselves are not created, sent or received as part of the original application (as they are not part of its function). That is, no matter how many actors we decompose in the application design, we will not make explicit these data items: they are inaccessible, and so require adaptation of the application in the next phase.

The result of applying this step is the refined interaction graph shown in Figure 14[3], a derivation function (the time period in answer to question 3 is the difference between the

[3]If the original application design was itself service-oriented, the interaction graph can be seen as annotations to that original design (expressing causal relationships between application data).

time stamps), and two inaccessible data items (the time stamps at the times of a sample being collated and of the informatation efficiency value being produced).

The application of this step to the organ transplant management application is described in Section A.2.3.

7.6 Step 2.4: Identify Hidden Actors

Aim: Identify which actors can feasibly record the contents of their interactions.

Actions:

(1) For each actor in the interaction model, assess whether it could feasibly be adapted to document its interactions.
(2) For each data item for which a hidden actor is the sole knowledgeable actor, mark the data item as inaccessible.

Documentation Produced:

—A list of actors which cannot be adapted to document their interactions, called hidden actors.
—A new list of inaccessible data items, including those only known to hidden actors.

7.6.1 *Description.* There are three reasons why it may not be possible to obtain the desired provenance of a data item. As discussed earlier, a data item may not be part of the application model to be adapted for recording (the interaction graph). In this case, as part of the previous phase, we decompose actors to make the item explicit in the model. Next, a part of the desired provenance may not be a part of the adaptable application. For instance, information of importance to answering a provenance question may be held by a person and never enter the computer system in the normal course of the application. These data items were identified as inaccessible in Step 2.3. Finally, actors who have access to a data item may not be able to record documentation to a provenance store, e.g. due to the difficulty of modifying a legacy service to record the actions of one of its components. Such actors are called *hidden*, because their processing is not apparent from the records in provenance stores.

We first identify the list of hidden actors from those identified in Phase 2. If the knowledgeable actors for any of the data items identified as part of the provenance queries in Phase 1 are all hidden, then the data item is inaccessible, despite being part of the process depicted in the interaction graph.

7.6.2 *Case Study.* The Sequence Database actor in the ACE experiment is a hidden actor. This is because it is not owned by the designers of the application themselves, it is a public database with online access provided by an external organisation. It cannot, therefore, be directly adapted to record documentation of the sequences which are downloaded from it by the designers of the ACE application.

However, no new data items become inaccessible because of this. Two data items are sent/received by the Sequence Database: the request for sequences and the sequences themselves. The former is not relevant to any provenance question, while the latter have another knowledgeable actor: Collate Sample.

The application of this step to the organ transplant management application is described in Section A.2.4.

8. PRIME PHASE 3: ADAPTING THE APPLICATION

Aim: The aim of this phase is to adapt the application design so as to document copies of data items and, as context, the process in which they were produced. This then allows the queries used to answer the provenance questions to be executed on that documentation.

8.1 Description

In the third phase, adaptations to the application design are specified. These adaptations perform two functions. First, they may cause previously inaccessible data items to be included in the recordable processes of the application. Second, they introduce the functionality to record copies of data items and their causal relationships to provenance stores as the application processes execute. The steps in which these adaptations are applied are specified in the subsections below.

8.2 Concept Definitions

The following concepts will be used, and illustrated with the case study, in the steps described below.

DEFINITION 14 HIDDEN ACTOR. *A* hidden actor *is an actor which, by its nature, cannot be adapted to record documentation about the data it sends and receives.*

DEFINITION 15 PROVENANCE WRAPPER. *A* provenance wrapper *is a component added to an application design which records copies of data items received and sent, and the causal relationships between them, on behalf of an actor.*

8.3 Step 3.1: Expose Inaccessible Data Items

Aim: Identify where actors must be modified to expose inaccessible data items.

Actions:

(1) For each inaccessible data item, determine where it exists in the world.
(2) Determine which actor could receive that data item with a suitable modification to the application.
(3) Specify the modification required to expose the data item.

Documentation Produced:

—A refinement of the interaction model, with actors receiving previously inaccessible data items as part of the same process as the start item.
—A set of actor modifications.

8.3.1 *Description.* For each inaccessible data item identified in Steps 2.3 and 2.4, we now determine how we can adapt the application design to make it accessible. Adaptations take the form of *(i)* introducing actors into the design, *(ii)* introducing interactions between actors, *(iii)* adapting interactions to carry new data items.

Not all actors which perform functions relevant to answering provenance questions may be explicitly part of the application processes. For example, a provenance question may ask about the owner of a resource used in the application, but the records of the owner's identity are not part of the application's function.

The designer, in this step, aims to ensure that knowledgeable, non-hidden actors are part of the model for each data item required for a provenance question. They must additionally ensure that the data items are included in the processes modelled, so that each data item becomes part of the provenance trace and provenance store queries can be written to obtain the item. The latter requires that newly modelled knowledgeable actors pass the data items to already modelled actors *as part of* the existing application processes, i.e. the processes are adapted to include the inaccessible data items.

8.3.2 *Case Study.* In the ACE experiment, we identified two inaccessible data items: the time stamps at the times of collating the sample and producing the information efficiency value. The actors knowledgeable about these items are the *clocks* local to the Collate Sample and Calculate Efficiency actors respectively. We therefore introduce these two actors: Collate Sample Clock and Calculate Efficiency Clock.

We then need to include the time stamps into the application process. First, two interactions are introduced for each new clock, whereby the application actor sends a request to its clock, and the clock returns a time stamp to the actor. For example, Collate Sample sends a request to Collate Sample Clock for the time, then Collate Sample Clock sends a time stamp in response to Collate Sample. The two interactions are causally related: the response occurs because of the request.

To ensure they are time stamps of the correct instants, and to include these interactions in the provenance of the start item (information efficiency value), we must insert these interactions into the logic of the application process. That is, Collate Sample's request to its clock will be triggered by the sample having been collated, and the response (time stamp) will trigger the collated sample being Calculate Efficiency actor. Similarly, Calculate Efficiency's request to its clock will be triggered by the information efficiency having been calculated, and the response (time stamp) will trigger the efficiency value being returned to the user at the end of the process. Causal relationships are added to the model to reflect these new sub-processes. The above new interactions and triggers form a set of adaptations to the design of the actors in the process.

8.4 Step 3.2: Wrapping Actors

Aim: Specify which actors will be wrapped with process documentation recording functionality so as to record copies of data items in the context of processes.

Actions:

(1) For each non-hidden actor in the interaction model, specify a *wrapper* around that actor.

Documentation Produced:

—Changes to the original application design documentation so as to deploy and connect wrappers for recording process documentation.

8.4.1 *Description.* The final step of PrIMe is to supply provenance recording functionality to the actors identified for the given provenance questions, enabling them to record process documentation to a provenance store. Each actor only records p-assertions that directly relate to its own interactions and their relationships and state, i.e. an actor cannot make assertions about other actors' interactions and states. For example an actor cannot

record a p-assertion about the interactions another actor may be involved in. This rule ensures that actors cannot make speculative p-assertions and only record what they directly know, which can help to ensure that recorded process documentation can be depended on to answer provenance questions (what an agent *knows* here concerns the designers understanding of the distribution of information in the application: more formal representations could be used to encode the distribution of knowledge, but are beyond the scope of this paper). In order to provide such process documentation recording functionality, a *provenance wrapper* is implemented by a *provenance client side library*, which provides a collection of functions that allows designers to enable actors to interact with a provenance store.

When it is clear that an actor is involved in the process underlying a data item whose provenance we are seeking, because it is part of the interaction graph as modelled in the preceding methodology, functionality must be provided for the actor to record documentation about its role in the process. To do this, PrIMe recommends using a *provenance wrapper*. The wrapper must be given access to information relating to its role in the process, e.g. incoming and outgoing messages, relationships between these messages and possibly some state of the actor as it relates to the interactions, which it can then document and send to a provenance store.

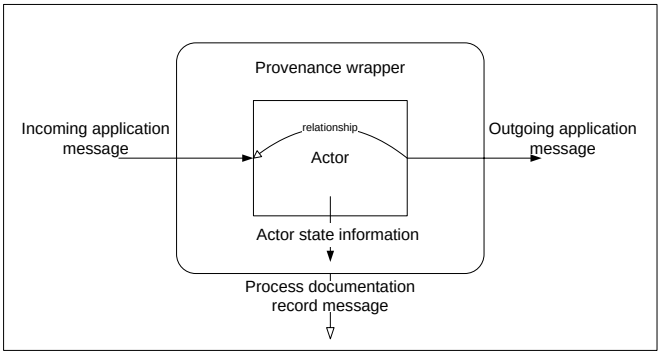


Fig. 15. The provenance wrapper

Figure 15 shows a provenance wrapper diagrammatically. Incoming and outgoing messages and their relations are intercepted by the wrapper, which also has access to relevant aspects of the actor's state. The wrapper then documents these pieces of information and sends a record message containing the documentation to a provenance store.

Having applied PrIMe and ensured all the correct actors have been provided with documentation functionality, the application can be run and documentation of its processes will be recorded into a provenance store. This documentation produces another view of the application, specifically a view of its execution. This view forms a Directed Acyclic Graph (DAG) of the causal dependencies of the application's data and allows tracebacks through the execution to discover answers to provenance questions.

8.4.2 *Case Study.* Each of the following actors are identified in the preceding steps as being involved in the processes relevant to the provenance questions asked: Sequence Database, Collate Sample, Calculate Efficiency, Encode, Compress, Compute Entropy,

Collate Sample Clock, and Calculate Efficiency Clock. As was established, Sequence Database is hidden due to its being outside the application's design, but each of the other actors is wrapped, so as to record documentation of its interactions and relationships between specific data items it handles.

9. COMPARISON WITH EXISTING APPROACH

To examine and illustrate the benefit gained by applying PrIMe, we took a comparable existing technique and applied it to the same problem: determining answers to provenance questions in the ACE experiment. We aimed to use as appropriate a technique as possible, to have a fair comparison. Given that determining provenance is a separate use of a system from its primary function, but that it concerns processes and data throughout the application, an aspect-oriented approach appeared to be particularly appropriate. Further, provenance questions are a user-level concern, and answering each kind of provenance question can be seen as a use case, rather than a change to the design for the developers' benefit, and so the technique chosen should similarly consider the provenance questions as use cases.

For these reasons, we have chosen to apply a technique described by Jacobson and Ng [2004]. The technique takes an aspect-oriented approach to the design (independently from whether aspect-oriented programming is used in implementation, but with the assumption that it will be). It divides the analysis and design into *use case modules*, so that the design for each use case can be developed largely independently. By assuming an aspect-oriented approach, they show how this independence is possible to maintain throughout the design. Note that we cannot, here, document the full application of the technique without greatly extending the length of this paper with largely unimportant detail. We concentrate on documenting the early stages of applying the technique, by which point the differences with PrIMe are already apparent.

9.1 Use Case Model

In the aspect-oriented methodology considered, each use case is separately identified, and the totality of use cases forms the *use case model*. Each use case identifies an aspect, and its concerns are kept as separate from other use cases as possible. We assume a single user in this system (the scientist enacting the experiment). The three provenance questions considered in ACE, can be re-written as use case summaries without the assumption of an underlying provenance system:

Sequences Producing Efficiency. The user determines the sequences used in the production of a particular information efficiency value.

Sequences Producing Recoded. The user determines from what sequences a particular recoded sample is derived.

Experiment Duration. The user determines how long it takes to produce an information efficiency value from a particular collated sequence.

9.1.1 *Use Case Flows.* In the aspect-oriented technique, each use case is first described as a flow of actions performed by the user and the system. Each of the use cases takes the same form, illustrated for the first use case below.

(1) The user asks the system to determine the sequences used in the production of a particular information efficiency value.

(2) The system returns the sequences with the property requested.

Provenance questions are relatively simple use cases compared to the possibilities considered in the aspect-oriented approach, which allow for multiple *use case flows* describing different possible action sequences.

9.1.2 *Use Case Relationships.* Another quality of provenance question use cases is that the actions required to complete the use cases are dealt with by the system and not the user, so one provenance question use case is not related by extension or aggregation to other use cases.

9.1.3 *Use Case Generalisation.* Provenance question use cases can all be considered instances of a generalised case: *Answer Provenance Question*. This is a reasonable generalisation as the use case flows are directly equivalent for each question, and in each case a query will need to be processed.

9.2 Realising the Use Cases

Keeping each use case separate, the aspect-oriented technique requires us to first identify classes that participate in their realisation. Each use case has a *handler class*, to coordinate the completion of the use case. Following the technique as described, most other classes initially identified will be those representing data/objects referred to in the use case flows.

Sequences Producing Efficiency. The first use case uses handler class SEQUENCESPRODUCINGEFFICIENCYHANDLER, and data classes SEQUENCE and INFORMATIONEFFICIENCY.

Sequences Producing Recoded. The second use case uses handler class SEQUENCESPRODUCINGRECODEDHANDLER, and data classes SEQUENCE and RECODEDSAMPLE.

Experiment Duration. The third use case uses handler class EXPERIMENTDURATIONHANDLER, and data class SEQUENCE and INFORMATIONEFFICIENCY.

If we took the most basic approach to applying the aspect-oriented technique, each handler, e.g. SEQUENCESPRODUCINGEFFICIENCYHANDLER, would ask the data class representing the data item provided by the user, e.g. INFORMATIONEFFICIENCY, for the data representing the answer to the provenance question, e.g. instances of SEQUENCE. However, this assumes a trivial connection between data items, e.g. that the sequences are a field of the information efficiency value. Taking this approach to its logical conclusion, every data class would include references to every data item in its provenance (possibly a vast amount), and provide means to query for each. It would also mean a large overlap between fields of different data items, e.g. the INFORMATIONEFFICIENCY and RECODEDSAMPLE classes would both provide means to access every data item preceding encoding in the ACE workflow (and every data item used to create the inputs to the workflow). Moreover, this analysis does not give any useful indication of how the data, e.g. sequences, are stored such that they can be retrieved.

9.3 Architecture

The aspect-oriented technique allows for a more structured approach than the simple analysis above, by defining an *architecture* for answering multiple use cases. The architecture should still separate concerns between use cases and separate platform specifics from platform-independent matters.

The generalised use case identified earlier, Answer Provenance Question, provides an *application-generic* layer in the architecture, on top of which the three specific use cases provide *application-specific* packages.

We are here getting closer to the system already assumed by PrIMe, where some system answers all provenance questions, but the same design problems apply to the application-generic layer as applied to specific use cases: how can one data item be found given another, by what means is the relationship between data items maintained such that such a search is possible, how do we handle the quantity of questions that may be asked of each data item. If, through standard analysis, we answer this set of questions with regards to each data item referred to in the use cases, we will be replicating the same generic machinery assumed by PrIMe, and will need to perform the same analysis of knowledgeable actors and granularity of recording built into PrIMe.

9.4 Comparison of Approaches

From the above example, we can see how the assumptions made by PrIMe act to ease and improve the development process. The assumptions are possible because PrIMe is applied to a particular kind of use case, provenance questions, and that technologies exist to aid satisfying those use cases if the design takes a particular form. This does not reflect negatively on the aspect-oriented technique in general.

As a developer attempts to change the design so as to allow a provenance question to be answered, with the aspect-oriented approach above, a design question that will always occur is: how can the data item that answers the question be found? As it is a provenance question there will be some start item identified, so this question can be divided into two parts. First, we need to know that the answer data item will be in the system so that it can be found when the use case occurs, i.e. how can we ensure it is recorded. Second, we need to know that the answer data item is connected to the start item in some way, so that the start item can be used to determine the answer.

PrIMe makes assumptions that benefit the developer in answering both of these. For the first question, it is simply that PrIMe provides conventional ways to adapt steps of processes in the existing design to record data of what occurs (and then maps this to technology able to implement the adaptation).

For the second question, the benefit is even more substantial. The particularly important assumption made by PrIMe is that steps in the application processes can be individually adapted to connect one data item, the answer to a provenance question, to another data item by causal relationships. Without such an assumption, the developer must determine how to connect start and answer items each time. As can be seen above, different choices of how to provide this connection can lead to different qualities of outcome. In particular, without using PrIMe there is no guarantee that developers can re-use the effort put into addressing one provenance question for answering another provenance question.

PrIMe, by first ensuring that the right granularity of process is found for adaption, then adapting individual steps rather than directly connecting the steps containing answer and start items, ensures that the same adaptations can be used for answering multiple provenance questions. As for the first problem above, PrIMe also ensures that the adaptations to record causal relationships have mappings to existing implementations. Although PrIMe resembles what one might get by knowledgeable application of existing techniques, it crystallizes issues that will arise in any project to build a provenance-aware system, and so helps designers avoid unnecessary duplication of work.

10. ASSESSING PRIME

Assessing the value of methodologies is difficult and often imprecise. Only through many repetitions of use and subsequent user evaluation can key properties of the methodology be tested, evaluated and comparisons made with other existing approaches. However, there are many criteria by which we can judge PrIMe at this point. In this section, we will consider evidence for the qualities of PrIMe from the perspectives of usability, applicability, traceability, quantitative metrics regarding the designs and implementations produced, and future proofing.

10.1 Usability and Applicability

In terms of usability, we have attempted to ensure that PrIMe follows an approach similar to many other successful software engineering methodologies (see Section 11 for a discussion on the similarity between PrIMe and other methodologies). This allows users of PrIMe to readily recognise the concepts it uses such as the use case analysis and the decomposition into actors that are similar to UML and agent based approaches respectively. This means that PrIMe can easily be absorbed into the development process, since users should be familiar with many of its techniques.

As evidence for the wide applicability of PrIMe, we cite the varying different application domains to which it has successfully been applied. In this paper, we have given examples from two case studies in bioinformatics (ACE) and medicine (OTM). In addition, we and our collaborators have applied PrIMe to an aerospace application that allows engineers to conduct extensive and detailed simulations on the design of new fighter aircraft [Kloss and Schreiber 2006]. Other examples where PrIMe has been applied is in agent-based electronic healthcare records [Kifor et al. 2006], scientific workflows [Rajbhandari et al. 2006], an ecological simulation system [Wootten et al. 2006] and a Brain Atlas Imaging application [Moreau et al. 2007]. These widely different applications attest to the wide applicability of PrIMe. The work referred to above used earlier (more ad-hoc) versions of PrIMe, and experience from them evolved the technique to that presented in this paper.

10.2 Traceability

Procedure 1 represents the *iterative* phase of PrIMe (Phase 2) responsible for decomposing the application to the correct level of granularity required for exposing the desired data items. It can be seen as a summary of the steps taken within Phase 2 of PrIMe. The decomposition part of PrIMe, in itself, will always terminate because the data referred to in each provenance question is part of the application being decomposed, and the iteration must therefore eventually reach a point where that data is separate from the rest of the application processes, i.e. it is isolated and so can be independently recorded and later retrieved by the provenance system. The procedure shows how, by following simple steps, the decomposition is achieved. However, the procedure will not be finite if the design processes it depends on are not themselves finite.

This procedure progressively decomposes an application into actors at finer and finer levels of granularity until the data item currently being considered is found within the interactions of those actors. Inputs to the procedure consist of provenance questions, $ProvQuestions$, which are considered to contain data items d, and the application $\mathcal{A}$, which is considered to be composed of actors. For each data item in a provenance question, the procedure decomposes the application into a set of $Actors$. As mentioned earlier,

Procedure 1 Decompose Application

Inputs: $ProvQuestions$, $\mathcal{A}$
Output: *result*
Procedure:

1: *result* $= \emptyset$
2: **for all** $d \in ProvQuestions$ **do**
3: $Actors = decompose(\mathcal{A})$
4: $\mathcal{I} = getInteractions(Actors)$
5: $\mathcal{R} = getInteractionsContaining(\mathcal{I}, d)$
6: **while** $\mathcal{R} = \emptyset$ **do**
7: $\mathcal{A}' = nonDeterministicallyChooseRelevantActor(d, Actors)$
8: $Actors = decompose(\mathcal{A}')$
9: $\mathcal{I} = getInteractions(Actors)$
10: $\mathcal{R} = getInteractionsContaining(\mathcal{I}, d)$
11: **end while**
12: $result = result \cup \{R\}$
13: **end for**
14: **return** $result$

this initial decomposition will typically follow the major functional components of the application. Next, we obtain all the interactions between this initial set of actors (where an interaction is defined as containing two actors, the sender and receiver, and the data item being communicated), and then all those interactions which contain the data item identified in the provenance question. If the data item is matched to a data item found within any of the interactions, these interactions, $\mathcal{R}$, are placed in the set $result$. If the data item cannot be matched to any interactions, we *non deterministically choose* the actor most relevant to the data item. Having chosen the relevant actor we again decompose this into a set of sub-actors, obtain their interactions and see if any contain the provenance question data item.

Traceability refers to the notion that, after applying PrIMe, it should be possible to show how any decision made can be traced back to one or more of the original use case questions identified in Step 1.1. This primarily affects the decomposition work done in Phase 2 of PrIMe, but also touches upon Phase 3, i.e. making adaptations to the application and providing provenance recording functionality. For example, we can show that any component in an application that has been made provenance aware by providing it with provenance functionality via a provenance wrapper can be traced directly back to a use case requirement. Thus, for example, in the ACE example, the fact that the Collate Sample actor has been made provenance aware can be directly traced back to a need to record information to a provenance store that will be used to answer provenance questions. From the abstract specification of the methodology in Procedure 1 or the methodology description earlier in this paper, we can see that all design decisions have a direct rationale that can be traced back to user requirements as specified by the original provenance questions. Traceability is important for *accountability*, in that it allows those examining a design to determine why and how design decisions were made.

10.3 Evaluation Metrics

Only through repeated use and evaluation in multiple applications can the value of a methodology be determined. Here we outline a set of metrics that can aid in the evaluation of PrIMe in the context of an application. Our aim is to enable the sort of comprehensive evaluation where a methodology is repeatedly applied by end users as done by Jeyaraj and Sauter [2007]. Software metrics can be determined from both implementation and design. Metrics derived from design artifacts have been is successfully used to measure software cohesion [Bieman and Kang 1998].

The metrics we describe are based on the following artifacts generated during the software life-cycle:

(1) The design of the application.
(2) The design of the application after applying PrIMe (PrIMe Design).
(3) The application itself.
(4) The provenance-aware version of the application.

The metrics we propose are in terms of these artifacts. Thus, an evaluator can vary the amount of work necessary for their evaluation. For example, they may choose to apply PrIMe but not implement the corresponding provenance-aware application. Thus, they can use the design related metrics and not those related to implementation.

The first step in determining these metrics is the application of PrIMe to the given application design. Once the evaluator has both the original design and provenance-aware design the following design-based measures can be determined. When introducing a metric we give an example of its use in the ACE application.

10.3.1 *Design-based Metrics.* Design-based metrics reflect the additional complexity introduced in an application as a consequence of following PrIMe. In an environment where PrIMe could be repeatably applied, one could then determine the average complexity introduced by PrIMe across a set of application designs. The design metrics we propose are based on the application design and PrIMe Design artifacts and are as follows:

The number of actors in the PrIMe Design versus the number of components in the original design. This metric provides a measure of what additional coding would be required by a PrIMe based application. With respect to ACE there were 7 components in the original design and 9 in the PrIMe design. Two actors were introduced to enable provenance specifically to capture time.

The number of interactions introduced in the PrIMe Design versus the total number of interaction in the original design. This provides a measure of the modifications necessary to the existing code base. For ACE, there are eight interactions in the original design and 15 in the PrIMe Design. The extra seven interaction comes from the necessity for PrIMe to model the inputs and outputs of the workflow itself as well as for tracking time.

The number of data items required by PrIMe that already exist in the application versus the total number of data items required by the PrIMe Design. This provides a measure of how compatible PrIMe is with the original design. There are 10 data items identified in both designs for the ACE application. Thus, the application has already exposed the most pertinent data items.

The number of interactions modified by the PrIMe Design versus the total number of interactions in the original design. This is another measure of the additional coding that may be required. In the ACE application, of the original 8 interactions, none were modified.

Are there any actors introduced by PrIMe Design that are essential (i.e. they affect or are affected by all other actors). The introduction of such an actor in a design can be detrimental in that it can introduce a central point of failure: something that the methodology should not lead to if possible. In ACE, there are no essential actors that are introduced by the use of PrIMe.

10.3.2 *Implementation-based Metrics.* These metrics assess the impact of applying PrIMe on application performance. While to some extent, this can be seen as measuring the performance of the provenance middleware, it provides a reasonable proxy for the impact of using PrIMe to introduce provenance functionality because a key underlying assumption is the existence of an implementation of the provenance architecture. In a rigorous evaluation setting, the evaluation could be conducted using different implementations, to factor out the middleware performance. There are two key measures.

The percentage difference between provenance-aware and regular application execution performance. The overhead of provenance-recording on ACE is 15% of application execution time [Groth et al. 2005]. Given that the particular provenance questions could not be answered prior to the application of PrIMe, this cost seems reasonable.

The percentage gain in data storage overhead between both versions of the application. In many cases, the provenance of a data item can require much larger data storage than the data item itself. This provides a measure that enables evaluators to see this data storage cost. Note, this can vary by the underlying implementation of the provenance architecture used and how it is configured. For example, some implementations allow for the use of references to data, which may decrease the data storage overhead. In the implementation of ACE we measured, for the production a single information efficiency value roughly 43K of process documentation was stored. Each information value requires roughly 1K of disk space. Hence, there is a 4300% overhead in disk space usage. This is an artifact of the size of the relative size of the information efficiency value, which is essentially a number and the only data stored in the original application, compared with the process documentation required to answer provenance questions, which contains the name of each sequence used, the grouping used, a reference to the procedure used for compression and so on. Clearly, intelligent compression of the documentation can lead to substantial reductions in storage overhead, though possibly at some performance cost [Chapman et al. 2008].

In this section, we have identified clear metrics that application developers can use to measure the impact of PrIMe on their application both in terms of design complexity and performance impact. By applying these metrics to ACE, we provide a a guide to those considering the application of PrIMe. Furthermore, these metrics provide a basis for conducting consistent systematic evaluations of PrIMe across a range of applications.

10.4 Future Proofing

In Section 6.3, we mentioned that PrIMe altered an application design so as to answer both currently known provenance questions, and some of those likely to be posed in the future. It is clear from the methodology description how known use cases affect the designer, i.e. the known use cases are used to identify which data items need to be recorded as part of process documentation and processed, and in which interactions by which actors those items are evident.

The influence of future provenance questions on the design process is more subtle. Future use cases are important because provenance questions make reference to the past.

Therefore, answering a provenance question may require data which existed transiently before the question was thought of. On the other hand, we usually cannot record everything: the amount of transient data may be vast, and events occur at every level from the program counters in PCs on which the application is hosted, up to telephone calls relating to the application between users in independent organisations. Given this uncertainty, the best we can aim to do is predict what information is *most likely* to be required to achieve as yet unknown provenance-related use cases, and then ensure that this information is available for provenance queries in advance.

PrIMe is constructed so as to use the known provenance use cases and available design information as heuristics for prediction of what data may be required for future use cases. Here, the crucial observation is that if a known provenance use case asks questions about part of a process, described at a particular granularity of detail, then this is an indication of the users' interest in that process and the data exchanged within it.

When it is known that the connection between two data items in a process may be required (i.e. where one is part of the provenance of the other), instead of simply recording those data items, PrIMe requires the designer to make explicit and to record all the interactions between actors in the process. Furthermore, when it is observed that the level of granularity in the interaction graphs is inadequate, i.e. the data of interest is part of a process within a currently black box actor, it is the whole process within the actor which is made explicit rather than just the data item of interest.

By these means, we aim to ensure that relevant process documentation is recorded to answer future use cases. With no further adaptation, a provenance query can be answered as long as it is related to the processes already documented. Of course, there will always be questions that are not answerable without further adaptation, but, as far as we can allow for it, future-proofing is an important part of making an application provenance-aware.

11. RELATED WORK

The issue of determining the provenance of results is important to many applications, and so has been considered in many branches of computer science. In particular, a great deal of effort has recently been put into determining provenance of data in scientific databases and produced by scientific workflows [Bose and Frew 2005; Simmhan et al. 2005]. Work has primarily focused on technology-specific models for provenance, inference of provenance from analysis of database operations, and adaptations of specific software to capture documentation of processes. We survey this work below, and show how it relates to PrIMe and the provenance model it employs.

11.1 Theoretical Models of Provenance

Models of provenance vary across approaches, but the majority share something akin to the causal relationship of our own approach, though variously described by terms such as dependence, derivation, lineage and others. Tan [2007] distinguishes *workflow provenance* from *data provenance*. Workflow provenance, intuitively the record of what procedures have been executed in a program or workflow and what resources were used by them, is characterised as "coarse-grained". Data provenance, intuitively the influence of one piece of data on the production of another, often in the context of a database system, is then "fine-grained". We examine related work on provenance in the context of workflows and databases further below.

Within data provenance approaches, Tan distinguishes annotation and non-annotation

approaches. Annotation approaches are ones in which some information about what occurs during the execution of a database query or update is recorded. The recorded documentation can then be used to determine provenance. In non-annotation approaches, the provenance of a data item is inferred from the database contents and the query/update expression.

Our own approach can be loosely categorised as a workflow provenance approach, and we primarily record documentation rather than infer it, though inference from existing documentation is not excluded. Others note that provenance is used to refer to notions such as authorship (a work would not have existed without its author) and applies to physical objects as well as data [Ockerbloom 2007].

11.2 Provenance-Awareness and Methodologies

While provenance models and software have received wide attention, the methods by which applications are made provenance aware is relatively under-examined. In the case of provenance-enabled workflow systems, the assumption is that the application is specified as a workflow in the appropriate language, and so is automatically provenance-aware. While this is adequate for a subset of applications, it is not a general solution.

A recurrent problem mentioned in the literature, is that it is hard to determine in advance what provenance questions users may wish to ask, and this has a direct influence on how an application is adapted. Even when, particularly in the case of database approaches such as those discussed below, inference after the fact is applied to determine what has occurred, obtaining the salient data is not always possible: Bose and Frew [2005] call this the Problem of Irretrievable Lineage. In PrIMe, we tackle this as far as is reasonably possible, by the inclusion of future use cases.

Recent work by Chapman and Jagadish [2007] has attempted to address the lack of guidance to engineers wishing to make systems provenance-aware, by examining how applications should ideally be adapted to record documentation and, more generally, identify various desiderata that need to be considered in making an application provenance-aware. These include recording a mixture of fine-grained (low-level data manipulation) and coarse-grained (user-level effects) activities, keeping track of the identity of incoming data so that it is clear where two data items are instances of the same input, planning for adequate process documentation storage capacity, recording how data is manipulated not just that it existed, recording the connections between processes in separate systems etc.

11.3 Commonalities with Existing Methodologies

The design of PrIMe was inspired by elements of other methodologies. The idea of using wrappers to legacy code is not new in software engineering. For instance, Thiran et al. [2006] use wrappers of legacy data sources to implement implicit constraints that legacy code enforce on data models, so that new functionality can be developed while still preserving data invariants. The wrapping approach we advocate here is one that preserves the interface and behaviour of software components, but records the necessary process documentation into a provenance store.

We share a common goal with De Lucia et al. [2007] since we are interested in making explicit traceability links and recording them. (We refer to them as causal dependencies.) Our respective working hypotheses differ however: we work in the context of processes fully implemented in computer systems, whereas they consider activities (software artifact management) involving humans. Hence, our approach is to design systems so that they

capture causal links explicitly, whereas theirs is to use information retrieval to infer such links a posteriori.

Similarly to Sinha and Smidts's HOTTest technique [Sinha and Smidts 2006], our methodology consists of extending an existing design by adding new requirements. In our case, the novel requirements are related to exposing the origin of data, while theirs are to make explicit constraints that will help generate test sequences.

In the spirit of object-oriented design, PrIMe introduces provenance-specific patterns, detailed by Groth [2007]. Such patterns are concerned with communications between recording actors and provenance store(s). They mandate an actor sending a message to create a unique identifier for this message, insert it into the message header, so that the unique identifier can be extracted by the message receiver. Both sender and receiver can then share a common identifier that they must use to document their actions with respect to the interaction and record this documentation into the provenance store.

11.4 Software Models and Programming

Software engineering is, of course, a process itself, and the notion of *traceability* has close parallels to that of provenance. Recording dependencies between stages of the engineering process in an electronic form allows tool-supported queries to be made about the provenance of parts of a design or implementation. Jahnke et al. [2002] examined this issue with regards to changes to class models. By recording process documentation regarding the changes in the models, a software engineer could later determine the provenance of the design, and therefore how classes within the current design related to earlier ones. They extended this idea to view re-design as a workflow-like process [2002], suggesting that the approaches to workflow provenance proposed by others may also apply to the software engineering process.

In a related context, Cheney [2007; 2007] discusses why program slicing is effectively the inference of process documentation (in our terms) in the context of programs, before applying this to database operations. Program slicing uses the semantics of source code to determine where statements are related, such as by a variable being used in both, and extract only that part of the program relevant to understanding a given issue: the value of a variable, cause of a fault etc. The relations extracted are relevant because the issue does, or may, depend on them.

11.5 Issues Comparable to Determining Provenance

Activities such as auditing, logging and version control, while not provenance in name, are immediately comparable. Here, we consider examples of engineering systems for each of these activities.

Several methodologies have been applied to handle regulatory compliance requirements (e.g., HIPAA in the US, Data Protection Act in the UK, MIFID in Europe). There is a comparability with our work since several regulations require past processes to be audited and checked to satisfy rules. Gandhi and Lee [2008] propose a methodology aimed at assessing risk, helping identify and understand the risk, and providing a chain of evidence to produce meaningful insights for risk assessment. Their approach is complementary to ours, since it would help bring a risk-based approach to identify provenance use cases. Baumer et al. [2000] discuss the implications of HIPAA on IT infrastructures, and specifically discuss recommendations that can be regarded as provenance use cases. As an illustration, the goal of "holding those who use individually identifiable health information accountable

for their handling of this information" requires handling of information and individually identifiable records to be securely logged and auditable, to enforce accountability.

Recording of information regarding a system's history often suggests a comparability with issues long addressed in systems development: logging and version control. While being affected by some the same issues, we argue that there is a difference of substance and emphasis with provenance. Logging architectures typically identify a message structure, logging methods, filtering, storage and rendering capabilities. They are usually agnostic about the contents of messages. Since they do not provide any means of "connecting" messages, they do not support explicit representation of causal dependencies. They generally offer various levels of granularity distinguishing warning, debugging information or severe errors for instance. Logging architectures are being applied to many facets of computer systems, including communication networks (PIX Logging Architecture [Philipsen 2008]), programming languages (log4j [The Apache Logging Services Project 2007] for Java), operating system (Linux logs [Walden 2003]), and Grid computing (R-GMA [Byrom et al. 2003]). A major concern investigated in such logging architectures, which we share and have investigated with regards to provenance [Groth et al. 2005], is the impact of the logging activities on application performance.

The Provenance-Aware Storage System (PASS) [Muniswamy-Reddy et al. 2006] extends logging techniques to document processes and determine provenance in the context of UNIX command execution. In PASS, the local operating environment is augmented to log details of command executions output, and the context in which it was produced, and records the connections made where a file is produced as both output from one command and input to another. The result, therefore, is a causal graph of a similar form to our own data model, albeit within a specific execution context.

Version control systems [Estublier et al. 2005] help manage the multiple revisions of the same unit of information (e.g. source code). They may capture *snapshots* of states of a data item or *change sets* specifying the difference from an earlier state, and so allow previous states to be retrieved. The provenance of an artifact would typically make reference to all its previous versions so similarly connecting versions. However, version control systems do capture a description of the process that led to the multiple revisions (only the changes due to those processes), while our notion of provenance does not by itself allow the differences between versions.

11.6 Workflows

Workflows, being explicit, user-level specifications of process, are often extended to capture process documentation. This may be either by adapting the workflow script to include recording actions, or by augmenting the workflow enactment engine to transparently record the dependencies between data items produced during execution. Taverna [Zhao et al. 2007], for example, uses the latter model, where recording is done by the workflow enactment engine or the operating system on behalf of the services used within the workflow. The provenance forms a graph, expressed in RDF. In the COMAD approach used by the Kepler workflow engine [Bowers et al. 2006; McPhillips et al. 2006], the data manipulated by a workflow is organised hierarchically and the dependencies between data items caused by executing the workflow are, therefore, between sub-trees of this hierarchy.

Davidson et al. [2007] generalise from several particular models of workflow provenance to describe a workflow as a set of modules, with data flowing between them. The relevant occurrences to be documented are, then, the data items as input to a module, the

execution of a module, and the data items as output from a module. Workflow provenance should cover the dependencies between instances of these, in particular linking the execution of one module to those that have control and/or data dependencies with it.

On a related topic, the provenance of workflow structures themselves, rather than their executions, has been studied. Freire et al. [2006] has developed a system which tracks the evolution of a workflow as it is edited by its users. Miles et al. [2008] have examined the same problem but in the automatic manipulation of workflows by a Grid-based workflow compiler.

11.7 Databases

The other context in which much research on provenance has taken place is databases, applied to both tracking changes due to update operations, and determining what data influenced the result of a database query. Cheney [2007] defines provenance as "any information that explains how... results were obtained from the underlying database".

In work developed by Buneman et al. [2001], through examining a database query and the contents of the database, they determine something of the provenance of the query's results, i.e. a non-annotation data provenance approach in Tan's categorisation. They distinguish two types of provenance, effectively the result of two provenance queries with the same start item and two different scopes in our terms. *Where-provenance*, "Where in the input database did this value come from?", regards the source relationships and records from which the data items within the query results were extracted. On the other hand, *why-provenance*, "Why was this record part of the result?", regards the values in the database that had an influence over those particular data items existing. Green et al. [2007] have added to these the concept of *how-provenance*, "How did this record contribute to the result", effectively a provenance query with (in our terms) the same scope as why-provenance but including the relationship types between occurrences in the result. All are examples of provenance queries with some hard-coded assumptions about the structure of the process that has occurred.

With regards to tracking database updates, Vansummeren and Cheney [2007] suggest augmenting database operations with further actions which cause the process documentation to be recorded at the same time. Buneman et al. [2008] take relational databases described as trees of values, where the tree expresses the relations between values and examine how any sub-tree of the database is connected to those in an updated version of the database, or the results of querying the database. They distinguish between where operations cause sub-trees to be wholly copied, and where there is a kind-preserving relationship. In a kind-preserving relationship, parts of the sub-tree have changed but the sub-tree itself still represents the same (kind of) data.

Chiticariu and Tan [2006], applied provenance to the changes in database schemas. The recorded mappings between each schema and the next are used to form a route from elements of a current schema to elements of its predecessors. This allows database schemas, when integrated into a federation, to be debugged. In our approach, we do not make any absolute distinction between the occurrences of a data structure in given states and the occurrence of a data value within the structure.

12. CONCLUSION

This document describes the PrIMe methodology. It specifies the necessary steps to take in making an application provenance-aware, where these steps are split into three distinct

phases comprising:

—Identification of provenance use cases and the data necessary for them to be answered.
—Decomposition of applications into actors and the mapping of the data flow of applications into message passing between the identified actors. This phase enables those actors that have access to the previously identified information items to be discovered.
—Adaptations to applications to enable the discovery or creation of actors that have access to information items, and adaptations to provide the necessary provenance functionality required to enable the recording of process documentation.

The PrIMe methodology provides a step-by-step guide to making applications provenance-aware. Application developers and users will only consider making their applications provenance-aware if they can see a clear and easy way to modify their applications to provide this functionality. Any development is a trade off between the effort and resources required to effect the development and the gains to be made by doing so. As shown in our comparison with an aspect-oriented methodology, PrIMe crystallizes issues that will arise in any project to build a provenance-aware system. These are manifest as assumptions built into the methodology, allowing designers to avoid common pitfalls, such as not designing the system to record adequate documentation to answer future provenance questions (see Section 10.4), or requiring multiple pieces of provenance infrastructure for answering multiple provenance questions when one, using a more general model of causation, would be adequate (see Section 9.4).

We hope to extend and improve upon PrIMe in several directions. In particular, we noted in Section 9, the comparability of our approach with aspect-oriented ones, and so we are intending to include features of aspect-oriented programming [Filman et al. 2004] to address the cross-cutting concerns that are inherent to the recording of process documentation, i.e. recording process documentation cuts across the different functionality of an application. We are also in the process of developing a number of tools for designers that will aid in the development of provenance-aware applications. These tools will help the designer to visualise the causal graphs derived from process documentation and will help to detect that the view of the application produced by applying the actor-based model accurately captures the information necessary to answer provenance questions. Ideally, these tools will be integrated with existing development tools, so that provenance can be addressed alongside other concerns.

13. ACKNOWLEDGEMENTS

We would like to thank Klaus-Peter Zauner who, as a creator and user of the Amino Acid Compressibility Experiment, allowed us to consult him for requirements and guidance on the biology as we applied PrIMe to the experiment. We are also grateful to Javier Vazquez-Salceda for work on the Organ Transplant Management case study, which, as part of the PROVENANCE European project, was of particular use in developing the technique presented here.

REFERENCES

ALVAREZ, S., VAZQUEZ-SALCEDA, J., KIFOR, T., VARGA, L. Z., AND WILLMOTT, S. 2006. Applying provenance in distributed organ transplant management. In *Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer, Chicago, Illinois, USA, 28–36.

BAUMER, D., EARP, J. B., AND PAYTON, F. C. 2000. Privacy of medical record: It implications of hipaa. *ACM Computers and Society 30,* 4, 40–47.

BIEMAN, J. M. AND KANG, B. 1998. Measuring design-level cohesion. *IEEE Transactions on Software Engineering 24,* 2 (Feb.), 111–124.

BOOTH, D., HAAS, H., MCCABE, F., NEWCOMER, E., CHAMPION, M., FERRIS, C., AND ORCHARD, D. 2004. Web services architecture. W3c working group note, World Wide Web Committee. Feb.

BOSE, R. AND FREW, J. 2005. Lineage retrieval for scientific data processing: A survey. *ACM Computing Surveys 37,* 1 (Mar.), 1–28.

BOWERS, S., MCPHILLIPS, T., LUDAESCHER, B., COHEN, S., AND DAVIDSON, S. B. 2006. A model for user-oriented data provenance in pipelined scientific workflows. In *Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer, Chicago, Illinois, USA, 133–147.

BRANCO, M. AND MOREAU, L. 2006. Enabling provenance on large scale e-science applications. In *Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer, Chicago, Illinois, USA, 55–63.

BUNEMAN, P., CHENEY, J., AND VANSUMMEREN, S. 2008. On the expressiveness of implicit provenance in query and update languages. *ACM Transactions on Database Systems 33,* 4, 1–47.

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, London, UK, 316–330.

BYROM, R., COGHLAN, B., COOKE, A. W., CORDENONSI, R., CORNWALL, L., DJAOUI, A., FIELD, L., FISHER, S., HICKS, S., KENNY, S., LEAKE, J., MAGOWAN, J., NUTT, W., O'CALLAGHAN, D., PODHORSZKI, N., RYAN, J., SONI, M., TAYLOR, P., AND WILSON, A. J. 2003. Relational grid monitoring architecture (r-gma). In *UK e-Science All-Hands meeting*. EPSRC, Nottingham, UK.

CHAPMAN, A. AND JAGADISH, H. V. 2007. Issues in building practical provenance systems. *Bulletin of the Technical Committee on Data Engineering 32,* 4 (Dec.), 38–43.

CHAPMAN, A., JAGADISH, H. V., AND RAMANAN, P. 2008. Efficient provenance storage. In *SIGMOD 2008*, J. Wang, Ed. Association for Computing Machinery, Vancouver, Canada, 993–1006.

CHENEY, J. 2007. Program slicing and data provenance. *Bulletin of the Technical Committee on Data Engineering 30,* 4 (Dec.), 22–28.

CHENEY, J., AHMED, A., AND ACAR, U. A. 2007. Provenance as dependency analysis. In *Symposium on Database Programming Languages (DBLP'07)*. Lecture Notes in Computer Science, vol. 4797. Springer, Vienna, Austria, 138–152.

CHITICARIU, L. AND TAN, W.-C. 2006. Debugging schema mappings with routes. In *Proceedings of Very Large Databases (VLDB'06)*. VLDB Endowment, Seoul, Korea, 79–90.

DAVIDSON, S., COHEN-BOULAKIA, S., EYAL, A., LUDAESCHER, B., MCPHILLIPS, T., BOWERS, S., ANAND, M. K., AND FREIRE, J. 2007. Provenance in scientific workflow systems. *Bulletin of the Technical Committee on Data Engineering 32,* 4 (Dec.), 44–50.

ESTUBLIER, J., LEBLANG, D., VAN DER HOEK, A., CONRADI, R., CLEMM, G., TICHY, W., AND WIBORG-WEBER, D. 2005. Impact of software engineering research on the practice of software configuration management. *ACM Transactions on Software Engineering and Methodology 14,* 4 (Oct.), 383–430.

FILMAN, R. E., ELRAD., T., CLARKE, S., AND AKSIT, M. 2004. *Aspect-Oriented Software Development*. Addison Wesley, Boston.

FREIRE, J., SILVA, C. T., CALLAHAN, S. P., SANTOS, E., SCHEIDEGGER, C. E., AND VO, H. T. 2006. Managing rapidly-evolving scientific workflows. In *Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer, Chicago, Illinois, USA, 55–63.

GANDHI, R. A. AND LEE, S.-W. 2008. Regulatory requirements-driven risk assessment. Tech. rep., University of North Carolina.

GIRET, A., BOTTI, V., AND VALERO, S. 2005. MAS methodology for HMS. In *Proceedings of the Second International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS 2005*. Lecture Notes in Computer Science, vol. 3593. Springer, Copenhagen, Denmark, 39–49.

GREEN, T. J., KARVOUNARAKIS, G., AND TANNEN, V. 2007. Provenance semirings. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS'07)*. ACM, Beijing, China, 31–40.

GROTH, P., JIANG, S., MILES, S., MUNROE, S., TAN, V., TSASAKOU, S., AND MOREAU, L. 2006. D3.1.1: An Architecture for Provenance Systems. Tech. rep., University of Southampton. Nov.

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)*. IEEE Computer Society, Research Triangle Park, North Carolina, 201–208.

GROTH, P., MILES, S., AND MOREAU, L. 2008. A Model of Process Documentation to Determine Provenance in Mash-ups. *Transactions on Internet Technology (TOIT) In publication*.

GROTH, P. AND MOREAU, L. 2008. Recording process documentation for provenance. *IEEE Transactions on Parallel and Distributed Systems In publication*.

GROTH, P. T. 2007. The origin of data: Enabling the determination of provenance in multi-institutional scientific systems through the documentation of processes. Ph.D. thesis, Electronics and Computer Science, University of Southampton.

HARMAN, P. AND WATSON, M. 1998. *Understanding UML, The Developers Guide*. Morgan Kaufmann Publishers Inc., San Francisco, California, USA.

JACOBSON, I. AND NG, P.-W. 2004. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley, Boston.

JAHNKE, J. H., SCHÄFER, W., WADSACK, J. P., AND ZÜNDORF, A. 2002. Supporting iterations in exploratory database reengineering processes. *Science of Computer Programming 45,* 2-3, 99–136.

JAHNKE, J. H., WADSACK, J. P., AND ZÜNDORF, A. 2002. A history concept for design recovery tools. In *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering (CSMR'02)*. IEEE Computer Society, Washington, DC, USA, 37.

JEYARAJ, A. AND SAUTER, V. L. 2007. An empirical investigation of the effectiveness of systems modeling and verification tools. *Communications of the ACM 50,* 6, 62–67.

KIFOR, T., VARGA, L. Z., VAZQUEZ-SALCEDA, J., ALVAREZ, S., WILLMOTT, S., MILES, S., AND MOREAU, L. 2006. Provenance in agent-mediated healthcare systems. *IEEE Intelligent Systems 21,* 6 (November/December), 38–46.

KLOSS, G. K. AND SCHREIBER, A. 2006. Provenance implementation in a scientific simulation environment. In *Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer, Chicago, Illinois, USA, 37–45.

LEWIS, D. 1973. *Counterfactuals*. Blackwell.

LUCIA, A. D., FASANO, F., OLIVETO, R., AND TORTORA, G. 2007. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodoloy 16,* 4, 13.

MACKENZIE, C. M., LASKEY, K., MCCABE, F., BROWN, P. F., AND METZ, R. 2006. Reference model for service oriented architecture 1.0. Oasis standard, OASIS. Oct.

MARIK, V., BRENNAN, R. W., AND PECHOUCEK, M., Eds. 2005. *Holonic and Multi-Agent Systems for Manufacturing: Second International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS 2005*. Lecture Notes in Computer Science, vol. 3593. Springer, Copenhagen, Denmark.

MCPHILLIPS, T. M., BOWERS, S., AND LUDAESCHER, B. 2006. Collection-oriented scientific workflows for integrating and analyzing biological data. In *3rd International Workshop on Data Integration in the Life Sciences (DILS'06)*. Lecture Notes in Computer Science, vol. 4075. Springer, Berlin, 248–263.

MILES, S. 2006. Electronically querying for the provenance of entities. In *Proceedings of the International Provenance and Annotation Workshop 2006 (IPAW 2006)*, L. Moreau and I. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer, Chicago, Illinois, US, 184–192.

MILES, S., GROTH, P., BRANCO, M., AND MOREAU, L. 2007. The requirements of using provenance in e-science experiments. *Journal of Grid Computing 5*, 1–25.

MILES, S., GROTH, P., DEELMAN, E., VAHI, K., MEHTA, G., AND MOREAU, L. 2008. Provenance: The bridge between experiments and data. *Computing in Science and Engineering 10,* 3 (May/June), 38–46.

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 20,* 5 (Apr.), 577–586.

MILES, S., MUNROE, S., GROTH, P., JIANG, S., TAN, V., IBBOTSON, J., AND MOREAU, L. 2006. D3.2.1: The Open Provenance Specification. Tech. rep., University of Southampton. Nov.

MOREAU, L., GROTH, P., MILES, S., VAZQUEZ, J., IBBOTSON, J., JIANG, S., MUNROE, S., RANA, O., SCHREIBER, A., TAN, V., AND VARGA, L. 2008. The Provenance of Electronic Data. *Communications of the ACM 51,* 4 (Apr.), 52–58.

MOREAU, L., LUDAESCHER, B., ALTINTAS, I., BARGA, R. S., BOWERS, S., CHIN, G., COHEN, S., COHEN-BOULAKIA, S., CLIFFORD, B., DAVIDSON, S., DEELMAN, E., DIGIAMPIETRI, L., FOSTER, I., FREIRE, J., FREW, J., FUTRELLE, J., GIBSON, T., GIL, Y., GOBLE, C., GOLBECK, J., GROTH, P., HOLLAND, D. A., JIANG, S., KIM, J., KRENEK, A., MCPHILLIPS, T., MEHTA, G., MILES, S., METZGER, D., MUNROE, S., MYERS, J., PLALE, B., PODHORSZKI, N., RATNAKAR, V., SCHUCHARDT, K., SELTZER, M., SIMMHAN, Y. L., SLAUGHTER, P., STEPHAN, E., STEVENS, R., TURI, D., WILDE, M., ZHAO, J., AND ZHAO, Y. 2007. The first provenance challenge. *Concurrency and Computation: Practice and Experience 20,* 5, 400–418.

MUNISWAMY-REDDY, K.-K., HOLLAND, D., BRAUN, U., AND SELTZER, M. 2006. Provenance-aware storage systems. In *Proceedings of the 2006 USENIX Annual Technical Conference*. Boston, MA.

MUNROE, S., MILES, S., MOREAU, L., AND VALQUEZ-SALCEDA, J. 2006. Prime: A software engineering methodology for developing provenance-aware applications. In *Proceedings of the Software Engineering and Middleware Workshop (SEM'06)*. ACM, Portland, Oregon, 39–46.

OCKERBLOOM, J. M. 2007. Copyright and provenance: Some practical problems. *Bulletin of the Technical Committee on Data Engineering 32,* 4 (Dec.), 51–57.

PEARL, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

PHILIPSEN, K. 2008. Pix logging architecture. http://www.logging-architecture.net.

RAJBHANDARI, S., WOOTTEN, I., ALI, A. S., AND RANA, O. F. 2006. Evaluating Provenance-based Trust for Scientific Workflows. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006)*. IEEE Computer Society Press, Singapore, 365–372.

SIMMHAN, Y., PLALE, B., AND GANNON, D. 2005. A survey of data provenance in e-science. *SIGMOD Record 34,* 3, 31–36.

SINHA, A. AND SMIDTS, C. 2006. Hottest: A model-based test design technique for enhanced testing of domain-specific applications. *ACM Trans. Softw. Eng. Methodol. 15,* 3, 242–278.

TAN, W.-C. 2007. Provenance in databases: Past, current, and future. *Bulletin of the Technical Committee on Data Engineering 32,* 4 (Dec.), 3–12.

THE APACHE LOGGING SERVICES PROJECT. 2007. Log4j. http://logging.apache.org/log4j.

THIRAN, P., LUC HAINAUTAND GEERT-JAN HOUBEN, J., AND BENSLIMANE, D. 2006. Wrapper-based evolution of legacy information systems. *ACM Transactions on Software Engineering and Methodology 15,* 4, 329–359.

VANSUMMEREN, S. AND CHENEY, J. 2007. Recording provenance for sql queries and updates. *Bulletin of the Technical Committee on Data Engineering 32,* 4 (Dec.), 29–37.

WALDEN, C. 2003. Windows-to-Linux roadmap: Part 5. Linux logging. Working with logs. Developerworks library, IBM. Nov.

WONG, S. C., MILES, S., FANG, W., GROTH, P., AND MOREAU, L. 2005. Provenance-based Validation of E-Science Experiments. In *Proceedings of 4th Internation Semantic Web Conference (ISWC'05)*. Lecture Notes in Computer Science, vol. 3729. Springer, Galway, Ireland, 801–815.

WOOLDRIDGE, M., JENNINGS, N. R., AND KINNY, D. 2000. The GAIA methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems 3,* 3, 285–312.

WOOTTEN, I., RAJBHANDARI, S., AND RANA, O. 2006. Provenance Use Case: BioDiversity Informatics. Tech. Rep. http://www.gridprovenance.org/publications/CardiffUseCase.pdf, University of Cardiff.

ZHAO, J., GOBLE, C., STEVENS, R., AND TURI, D. 2007. Mining Taverna's semantic web of provenance. *Concurrency and Computation: Practice and Experience 20,* 5, 400–418.

A. THE ORGAN TRANSPLANT MANAGEMENT EXAMPLE CASE STUDY

In this section, we provide examples from another case study, the the Organ Transplant Management (OTM) application, in which organ donors and patients waiting for organs must be matched up according to various criteria, such as blood type, immunology tests and so on. This application is taken from work conducted in collaboration with Technical University of Catalonia [Alvarez et al. 2006]. Part of the workflow of the OTM application is extracted to illustrate how PrIMe was applied.

The example involves conducting blood tests on donor organs in order to screen them for a variety of pathologies. Here, a donor's organs undergo a series of tests to enable a decision to be made about whether they are suitable candidates for transplantation. The high level view of this process contains three entities: the *hospital*, the *electronic healthcare records system* (EHCRS) and the *testing laboratory*. The hospital is where the donor organ is recovered from a recently deceased donor, and where the doctor who initiates the screening process resides. The EHCRS is the place where all the records for the donor are kept. Finally, the testing laboratory is where the organ blood tests are performed.

In terms of workflow, the hospital where the doctor resides must communicate first with the EHCRS to obtain the donor's records, after which the hospital can request a blood test to the testing laboratory, passing along the necessary donor data obtained from the EHCRS. Figure 16 shows the above workflow graphically.

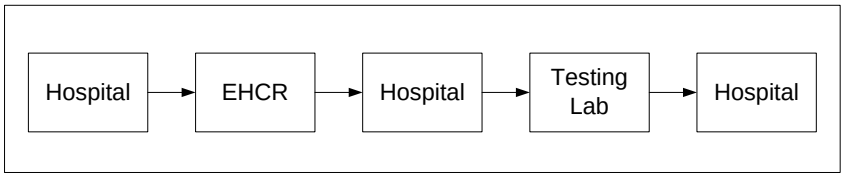


Fig. 16. The OTM workflow

A.1 PrIMe Phase 1: Provenance Question Capture and Analysis

In the following sections, we will provide the results of applying Phase 1 and 2 of PrIMe to the OTM case study. We exclude Phase 3 for brevity, but note here that all actors are wrapped to record documentation as the application executes. We will not give extensive details of the methodology itself, as these are covered in the main paper.

A.1.1 *Step 1.1: Provenance Questions.* Provenance questions which were posed by OTM users include the following.

(1) What is the medical analysis tree for diagnosis X?
(2) Did any deviations take place from the standard workflow for the diagnosis decision given for organ X? (i.e. in Figure 16, a deviation might be if the EHCRS sends a patient's records directly to the testing lab instead of back to the hospital, deviating from the workflow and potentially breaking regulatory rules).
(3) Given a diagnosis decision, X, for a given organ, Z, at what time did the doctor submit his request for test Y that was used to support X?

A.1.2 *Step 1.2: Analysing Provenance Questions.* The questions were analysed to determine the start items, scopes and further processing of the provenance trace. The results of this analysis for two questions are given in Tables IV and V

Table IV. Provenance question 1

Provenance question	What is the medical analysis tree for diagnosis X?
Start item	Diagnosis X
Scope	Medical analysis tree
Processing step	None needed

Table V. Provenance question 2

Provenance question	Did any deviations take place from the standard workflow for the diagnosis decision given for organ X?
Start item	Diagnosis decision for organ X
Scope	Workflow underpinning diagnosis decision for organ X
Processing step	Compare the workflow obtained by the query with the standard workflow and highlight differences

A.2 PrIMe Phase 2: Actor Based Decomposition

A.2.1 *Step 2.1: Identifying Actors.* The initial actors identified in OTM were those described above: Hospital, Electronic Healthcare Records System and Testing Laboratory.

A.2.2 *Step 2.2: Actor Interactions.* In this step we analyse the interactions of our identified actors. As shown in the interaction graph in Figure 17, we see that when a donor organ is to be screened, the hospital (H) communicates with the EHCRS to obtain the *donor ID*. Then, the hospital communicates with the testing laboratory (TL) to obtain the *test results* for the patient identified by the donor ID. In this simple example, there are four messages being passed between the three actors, where each of these messages contain the following data items: the *query* (q1), the *donor ID* (DId), the *request* (rq1) to perform a blood test on the organ, and the test *results* (r1) from the laboratory. The relationships between pairs of messages are shown as dashed lines. Finally, the query itself is shown in the figure as a dark circle that indicates the sending of message M1. The arrows depicting the relationships between messages point towards the *source* of causality, i.e. that the message (M2) containing the donor id was caused by the message (M1) containing the request for the donor id.

A.2.3 *Step 2.3: Decomposition to Knowledgeable Actors.* The model produced in the previous section cannot answer the question "Given a diagnosis decision, X, for a given organ, Z, at what time did the doctor submit his request for test Y that was used to support X?" This is because there is no information about the time at which requests are put into the system within the interactions between any of the identified actors.

Since requests for tests are first issued at the hospital, the first task is to identify the actor within the hospital that has access to the timing information. After examining the hospital, it is discovered that doctors access a user interface (UI) to issue diagnosis requests, and that this user interface first obtains a timestamp from a timing component that it associates

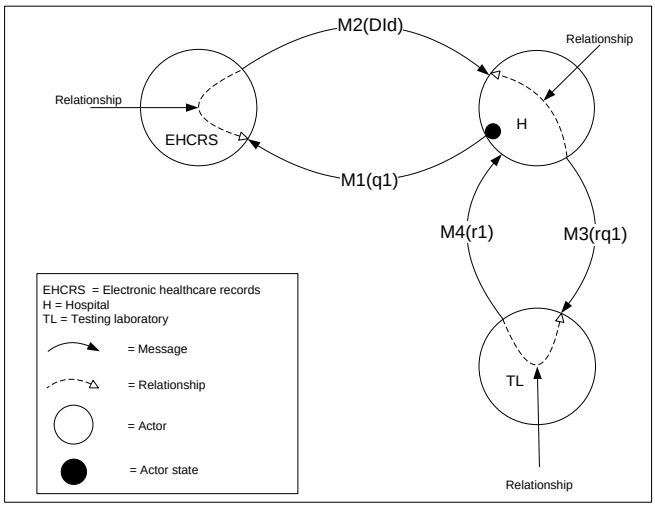


Fig. 17. The interaction graph of the OTM example

with the request. Only after a timestamp has been associated does the UI send the request to a component in the hospital called the *donor data collector* (DDC). It is the DDC that then sends the request for a patient ID to the EHCRS and then passes this on, along with the doctor's diagnosis request to the testing laboratory. The test results are then passed back to the DDC which then sends it back to the UI where the doctor retrieves it, makes a diagnosis decision, sends this to the UI which sends it to the EHCRS to update the patient's file.

This more detailed model of the hospital is represented in Figure 18, which illustrates the decomposition of the hospital into four new actors (contained in the shaded, dashed circle): the Doctor (D), User Interface (UI), Timer (T) and the Donor Data Collector (DDC).

A.2.4 *Step 2.4: Identifying Hidden Actors.* There are no hidden actors in this case study.

A.3 Illustrative Usage Scenario

In this section, we provide an illustrative scenario in which a provenance question is asked.

As a hospitalized patient's health declines, and in anticipation of a potential organ donation, one of the attending doctors requests the full health record for the patient and sends a blood sample for analysis. Through the user interface (UI), these requests are made by the attending doctor and passed to the Donor Data Collector (DDC) responsible for collecting all the expected results. After brain death is observed and logged into the system, if all requested data and analysis results have been obtained, a doctor is asked to make a decision about the donation of an organ. The decision, i.e., the outcome of the doctor's medical judgment based on the collected data, is explained in a report that is submitted as the decision's justification. Figure 18 displays the interaction graph for this scenario (we omit relationships and actor states for clarity). The UI sends requests (I1, I2, I3) to the DDC service, which gets data from the patient records database (I4, I5) and analysis results from the laboratory (I6, I7), and finally requests a decision (I8, I9). Our provenance-aware

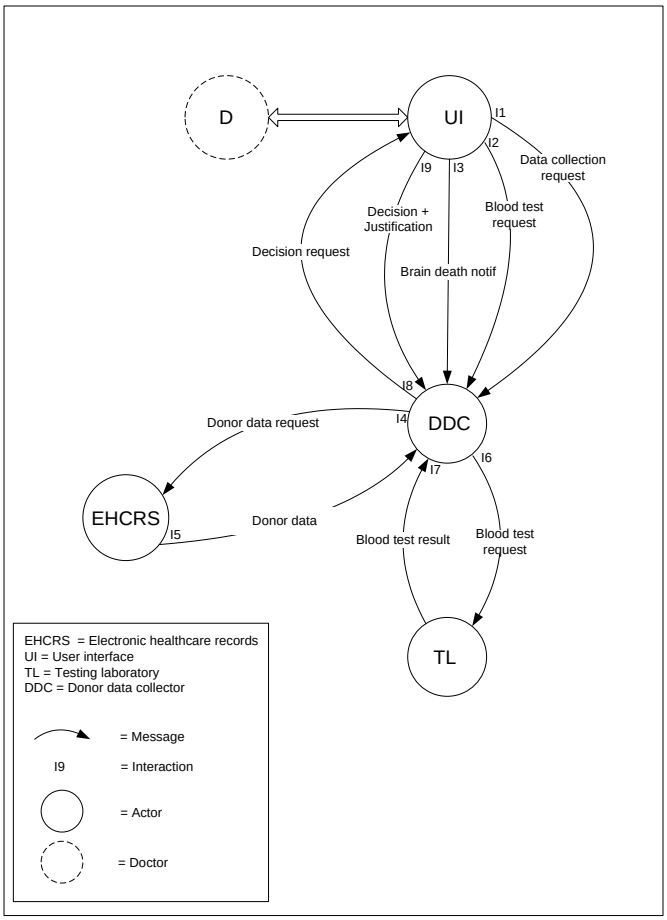


Fig. 18. The interaction graph of the organ donation scenario

application now has the capability to produce an explicit representation of the process actually taking place. This includes p-assertions for all interactions (I1 to I9), relationship p-assertions capturing dependencies between data, and actor state p-assertions. In Figure 19, we find the DAG that represents the provenance of a donation decision, made of relationship p-assertions produced by the provenance-aware OTM application. DAG nodes denote data items, whereas DAG edges represent relationships such as data dependencies (is based on, is justified by) or causal relationships (in response to, is caused by). Each data item is annotated by the interaction in which it occurs. Furthermore, the UI asserts an actor state p-assertion, for each of its interactions, about the user who is logged into the system. To locate the donation decision in the collection of process documentation, the user must specify its *data handle* — a unique, intensional description for the data item whose provenance is being sought (see Section 6.4 for a discussion about *data handles*). Over such documentation, we can issue provenance queries that navigate the provenance graph and prune it according to the querier's needs [Miles et al. 2007]. For instance, from

the graph, we can derive that users X and Y are both causing a donation decision to be reached. Figure 19 is small, but in real life examples with vast amount of documentation, users benefit from a powerful and accurate provenance query facility.

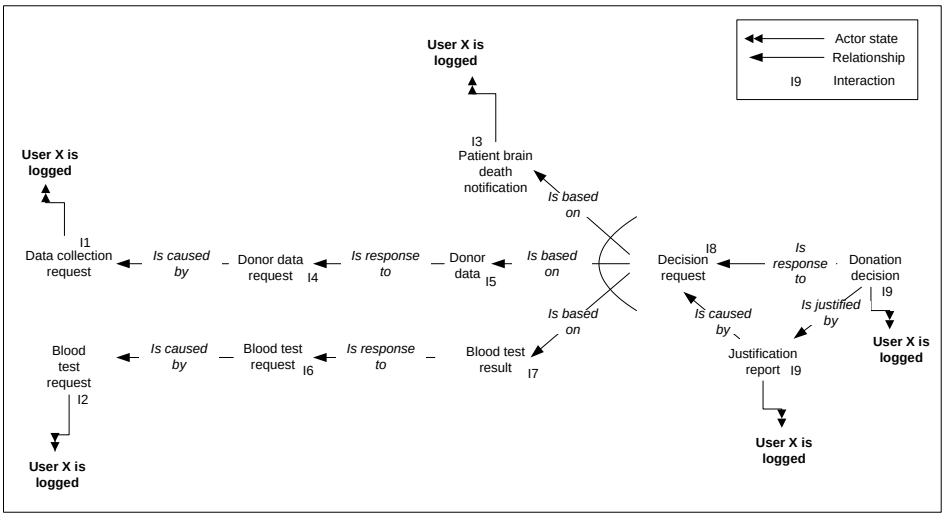


Fig. 19. A Provenance trace through the organ donation scenario