PrIMe: A Software Engineering Methodology for Developing Provenance-Aware Applications

Steve Munroe, Simon Miles, Luc Moreau
Electronics and Computer Science,
University of Southampton,
Southampton, UK
{sjm,sm, l.moreau}@ecs.soton.ac.uk

Javier Vázquez-Salceda
Knowledge Engineering and Machine Learning Group,
Universitat Politècnica de Catalunya,
Barcelona, Spain
jvazquez@lsi.upc.edu

Provenance is a concept often used in the Art world to refer to the documented history of an artifact, providing information about the artifact's lineage and authenticity. Provenance-aware applications similarly allow their users to have confidence about the data they produce, and can enable users to make judgements relating to notions of trust, accountability, validation, replication and compliance of their data. PrIMe is a software engineering methodology for adapting applications to enable them to interact with a provenance middleware layer, thereby making them provenance-aware. Such applications allow users to answer questions about *provenance use cases*, which are descriptions of scenarios in which a user interacts with a system by performing particular functions on that system. In order to illustrate how PrIMe can make applications provenance-aware, an Organ Transplant Management example application is used.

1. INTRODUCTION

Provenance is already well understood in the study of fine art where it refers to the trusted, documented history of some art object. Given that documented history, the object attains an authority that allows scholars to understand and appreciate its importance and context relative to other works. Art objects that do not have a trusted, proven history may be treated with some scepticism by those that study and view them. This same concept of provenance may also be applied to data and information generated within computer applications.

In general, computer applications produce data, and making an application provenance-aware allows its users to understand the provenance of their data, understood as *the process that led to that data* [7]. To be able to determine the provenance of data, it must be possible to *document* an application's execution and to then perform *queries* over that documentation. Such documentation is called *process documentation* and is comprised of multiple, individual pieces of information, called *p-assertions* [7], which are recorded during execution and then stored and maintained in a repository of such information called a *provenance store* [7]. One difficulty that remains, however, is to ensure that necessary and sufficient forms of process documentation is captured so that queries can return a satisfactory account of a given data item's provenance. It is the role of such software engineering tools as PrIMe to ensure this is achieved.

The concept of provenance is already used within databases (e.g. [14]) and is becoming increasingly important in computer applications in general (e.g. [3, 4, 16, 11]). In [7] a provenance architecture is described that provides a middleware layer that includes a provenance store for maintaining and storing process documentation and interfaces for recording, querying and viewing process documentation. However, in order for developers of applications to fully exploit such an architecture, software engineering practices for developing provenance-aware systems are needed. To meet this need, we have developed a *Provenance Incorporating Methodology* (PrIMe) that enables developers to analyse and make adjustments to applications in order that the functionality provided by a middleware-based provenance architecture can be fully exploited in the making of provenance-aware applications.

PrIMe is a tool to be used by *system developers* who make modifications to applications by applying the steps of PrIMe after querying the application's *users* for the kinds of information they require from their application. In the remainder of this document, PrIMe and how it can be applied to computational systems is described in detail, and the following structure is followed. In the next section, PrIMe is introduced and its overall structure is presented. In Section 3, an example application from the medical domain is introduced in order to ground the subsequent discussion and explanation. In Section 4, we describe each step of PrIMe in detail and apply them to the (example) application, and in Section 5 we discuss the security implications of such provenance-aware applications. Section 6 discusses some related work, Section 7 outlines future work and, finally, Section 8 offers some concluding remarks.

2. 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: *(i)* the identification of provenance use cases and the pieces of information that will be used to answer them, *(ii)* the decomposition of the application into a set of actors and their interactions

Sixth International Workshop on Software Engineering and Middleware '06
Portland, Oregon USA

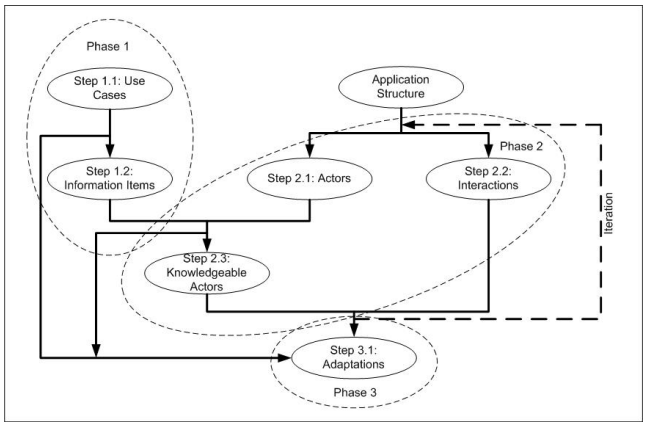

Figure 1: Overall structure of PrIMe

(this phase is iterative for reasons described in Section 4.2.4) and, *(iii)* applying a set of principled adaptations to the application in order to ensure the required information items are available for documentation.

Traversing this process, PrIMe starts from the application itself, and it is assumed that the structure and purpose of the application is known beforehand. This does not mean that the application must already exist, but that the overall functionality of the application has been identified and the general structure has been determined. Given this assumption, the steps through PrIMe are as follows.

- Phase 1
  - Step 1.1: Provenance use case analysis.
  - Step 1.2: Identify use case information items.
- Phase 2
  - Step 2.1: Identify application actors.
  - Step 2.2: Map out actor interactions.
  - Step 2.3: Identify knowledgeable actors.
- Phase 3
  - Step 3.1: Introduce application adaptations.

The rest of this paper discusses each step in turn.

3. THE ORGAN TRANSPLANT MANAGEMENT EXAMPLE APPLICATION

The example application used in this paper is an 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 done by the Technical University of Catalonia [2].

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. Figure 2, shows these entities and the communication links between them (shown by the arrows). 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.

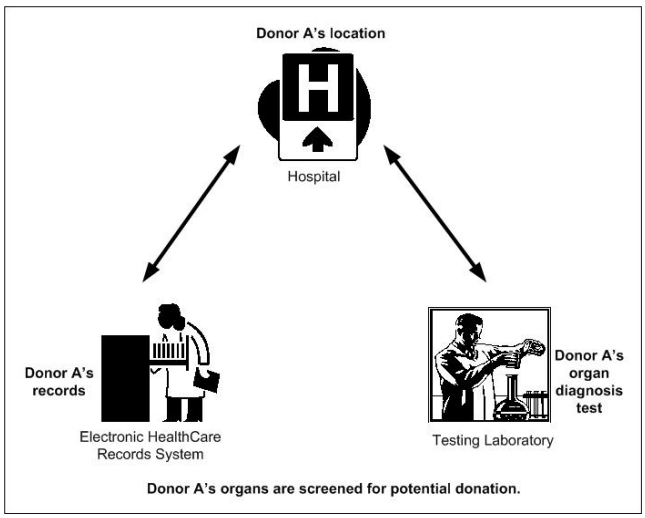

Figure 2: The components in the OTM application involved in the testing of blood

4. STEPPING THROUGH PrIMe

PrIMe provides designers with the tools to enable applications to use the Provenance Architecture described in [7]. This architecture defines different forms of process documentation or *p-assertions* that enable a complete account of an execution to be documented. Specifically, three types of p-assertions are defined as follows. *(i) Interaction* p-assertions; these are used to record the interactions between application components and provide a complete model of information flow between components. *(ii) Relationship* p-assertions; these record the relationships between a component's incoming and outgoing messages and provide a model of information flow within a component. *(iii) Actor state* p-assertions; these record state information of application components. Note that the combination of interaction and relationship p-assertions provide a complete account of an application's data flow. The following steps of PrIMe describe how to identify what kinds of information within an application should be the focus of such p-assertions and how they can be captured.

4.1 PrIMe Phase 1: Use Case Capture and Identifying Use Case Answers

In Phase 1 of PrIMe, the kinds of provenance related questions to be answered about the application must be identified. These *provenance use case questions* determine how PrIMe will be applied by highlighting which parts of execution needs to be documented and subsequently which parts of the application must be made provenance aware. Use cases in this sense are similar to those found in UML, i.e.

descriptions of scenarios in which users interact with an application [8]. They drive the process of making an application provenance-aware by informing application developers of the granularity of the processes to be considered and the critical information to expose.

Obtaining use case questions involves several steps. First, the use cases must be identified, then an analysis must be undertaken in order to discover the information within the application that constitutes the answers to the use cases. In the next few sections, descriptions of each of these steps is given in detail.

4.1.1 Step 1.1: Provenance Use Case Analysis

It is not always obvious to users what provenance use cases 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. [12, 5]) to help developers collect the core provenance use cases from users. The elicitation process comprises two steps. *(i)* Provide an explanation and definition of provenance in computational systems. *(ii)* Give examples of general questions that can be answered by a provenance-aware application.

The definition of provenance that PrIMe uses is taken from [7]: *"The provenance of a piece of data is the process that led to that piece of data"*.

Some generalised use case questions may take the following form.

- What are the details of 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 because of 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?
- What data was used as input to a process.

After these two steps have been executed, the users of the application are invited to generate their own use case questions. In the case of the OTM example, two such use cases might be as follows. *(i)* Donor A's organs are screened for potential donation. What is the provenance of the donor's organ diagnosis? *(ii)* A donor organ is considered for diagnosis. How many doctors have been involved in the organ's screening case?

4.1.2 Step 1.2: Information items

When considering how to answer a use case, it is necessary to identify the information items that would provide answers; there may be many such items, e.g., a given result, or a sequence of decisions. In the OTM example, the first use case question refers to the provenance of an organ diagnosis decision. To obtain this, it is necessary to obtain the the donor record from the EHCRS, the organ test results and the diagnosis decision. It is also necessary to obtain the relationships between these data items, since otherwise we would have no way of linking a given test result to a given organ and a given donor. Some examples of the different kinds of information items that can be used to answer provenance use cases are described below.

Data Items Data items represent specific pieces of data such as a result of a computation, the outcome of a decision or the state of a given component.

Processes Interactions between components represents an application's processes.

Relationships When seeking the provenance of some data, it is often necessary to identify the relationships between the different components involved in its generation.

In order to obtain such pieces of information, we must move to the next phase of PrIMe.

4.2 PrIMe Phase 2: Actor Based Decomposition

In this phase of the methodology, the aim is to associate every information item necessary to answer a use case question as identified above with a particular component within the application. To achieve this, PrIMe *decomposes* the application into a set of *actors* and performs an analysis of their *interactions*. This approach is similar in nature to object oriented approaches to modelling systems (e.g. [12]), which decompose applications into classes and objects.

Decomposing an application into actors follows an *iterative* approach, comprising the following three steps. Step 2.1: Identify an initial set of application actors. Step 2.2: Map out the interactions between these actors. Step 2.3: Identify those actors that have access to the identified information items. These steps may need to be repeated if it is discovered that no actor can be identified at the current level of granularity that has access to a use case related information item (see Section 4.3).

4.2.1 Step 2.1: Identifying Actors

An *actor* is an entity within an application that performs actions, such as a Web Service, software component, machine, a person and so on, and which interacts with other actors. The identification of suitable actors in an application is a key part of making the application provenance-aware. In order to aid this process, PrIMe provides two simple *actor identification heuristics*: *(i)* identify the components that are the receivers of information (these may be components/services in a workflow, a script command, or the GUI/desktop application into which a user enters information) and, *(ii)* identify the components that send information in an interaction (these may be a workflow engine, a script executor, a user or a sensor).

In the OTM example, the hospital, the EHCRS and the testing laboratory all communicate with each other, so an obvious place to begin is to classify each of these components into high-level actors.

4.2.2 Step 2.2: Actor Interactions

The next step in PrIMe is to map out the interactions between the identified actors. Other methodologies, for example the GAIA methodology used in agent oriented design [15], also provide methods to identify interactions between application components. In PrIMe, a similar approach is taken. Specifically, the way that information flows between actors is modelled as message passing and PrIMe requires a representation of such messages to be stated explicitly along with identification of the content of such messages. This makes it possible to discover which actors have access to information items necessary for answering provenance use case questions. Figure 3 shows the notation by which we define application interactions as a graph. The nodes represent individual actors and the arcs (shown as uni-directional arrows) represent messages being passed from one actor to

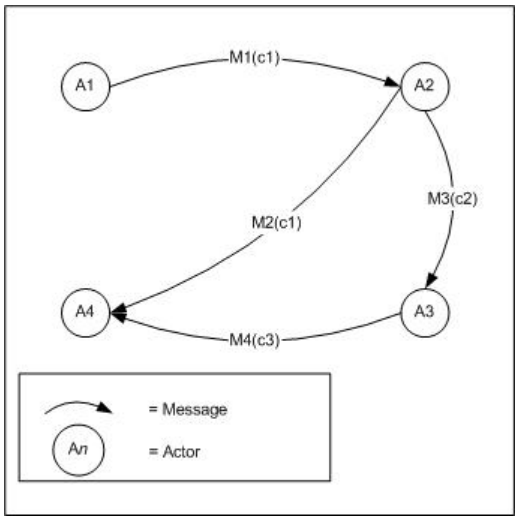

Figure 3: An abstract data graph

another. Annotations over the arrows show messages and content identifiers. The use of such graphs makes explicit which actors have access to which information items, and thus enables application developers to identify *knowledgeable actors* (see Section 4.2.4). We do not show the full data contents of messages here for reasons of space.

4.2.3 Interactions in the OTM Example

Examining the OTM example, the developers determine that when a donor organ is to to be screened, the hospital (H) must send a *query* to the EHCRS to obtain the *donor ID*. Then, the hospital sends a *request* to perform the necessary tests to the testing laboratory (TL) along with the 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*, the *donor ID*, the *request* to perform a blood test on the organ, and the test *results* from the laboratory.

Using the information gained from this step, a data graph of the system is constructed to clearly show the interactions between each actor (shown in Figure 4).

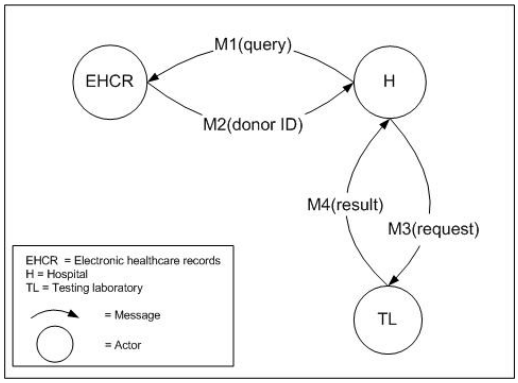

Figure 4: The data graph of the OTM example

4.2.4 Step 2.3: Knowledgeable Actors

Any actor that has access to an information item is known as a *knowledgeable actor* and, as stated above, the aim is to associate every information item necessary to answer a use case question with such an actor. If this can be achieved, then the developer can add the necessary provenance functionality (see Section 4.3.3). However, there are two reasons where such an association between actors and information items may not be possible. Either: *(i)* an information item is located within an actor and has not been exposed in the interactions between actors through message exchanges or, *(ii)* no more decomposition is possible and there are still information items to be located and associated to an actor.

For *(i)*, it is necessary to iteratively apply steps 2.1, 2.2 and 2.3 of PrIMe until a level of granularity is reached where information items are revealed in the interactions between actors. This is discussed in the next few sections, taking the OTM application as an example. For *(ii)*, a number of adaptations must be performed to change the application in structured ways in order to be able to make the required information items explicit and associable to an actor. Section 4.3 discusses this in detail.

4.2.5 Knowledgeable Actors in the OTM Example

The structure of the OTM application has so far been decomposed into three actors: *(i)* the hospital, *(ii)* the EHCRS and, *(iii)* the testing laboratory. Imagine however, that we try to answer the second use case question listed in Section 4.1.1, i.e. *How many doctors have been involved in an organ screening case?*. To answer this, the information item containing the relevant information must be identified, i.e. a record of all doctors involved in a given case. If the actor-based decomposition of the OTM application that was performed earlier is examined (Figure 4), this question cannot be answered because the information about how many doctors are involved in a given screening case is not available within the interactions between any of the actors. What must be done in this situation is to perform a further decomposition of the hospital, exposing its subactors and revealing, through their *state* and *interactions*, the information items needed to answer the use case question.

The first task is to identify the actor within the hospital that has access to the information needed. After examining the hospital actor, it is discovered that doctors access a user interface (UI) to issue screening requests, and it is this component that sends the request to a component in the hospital called the *donor data collector* (DDC). The DDC then sends the request for a donor ID to the EHCRS and then passes this on, along with the doctor's screening request, to the testing laboratory.

Having performed another, deeper level of analysis, steps 2.2 and 2.3 of PrIMe must be executed again to map out the interactions between these new actors and identify which are knowledgeable about the information item for the use case. In Figure 5, these steps have been completed. Actor H in Figure 4 has been decomposed further into three actors: the *doctor*, logging onto a *user interface* (UI) and the *Donor Data Collector*. Three new interactions have been also identified to show how the screening request (originating from a doctor logging onto the UI) goes from the User Interface to the Donor Data Collector, and how the test result is sent back to the User Interface from the the Donor Data Collector.

4.3 PrIMe Phase 3: Adapting the Application

In this section, we describe several application adaptations used to reveal information items that are currently

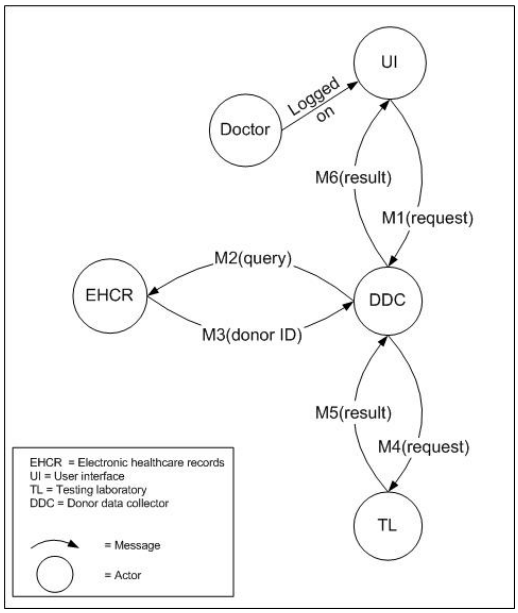

Figure 5: The modified OTM example

inaccessible, and to provide modifications to actors to enable them to record process documentation. We begin with the former kind of adaptations.

There are three reasons why it may not be possible to associate an information item to an application actor. Either more decomposition is required until an actor can be found that has access to an information item (in which case Phase 2 of PrIMe must be re-applied as described above); the information item has not been made explicit in the interactions between the actors of the application (such as when information required to *produce* the necessary information item is distributed throughout the application), or actors who have access to the information cannot record process documentation (for instance, where a human owns the information item and thus requires a way to introduce the information into the application). To overcome the latter two problems, a number of modifications to the application can be made in order to expose these information items or to move information items to actors who can record process documentation. PrIMe identifies two possible modifications that can be made, which we detail below.

4.3.1 Modifying the Set of Actors

In those cases where an actor cannot be found that has access to an information item needed to answer a use case, certain modifications can be made to the set of actors in the application. An actor that does not have access to a given information item is termed a *non-knowledgeable actor* for that item. Note that an actor may be non-knowledgeable for one information item and knowledgeable for another. A non-knowledgeable actor may be modified so that it gains access to an information item that is local to it, i.e. that does not need to be sent to it by another actor (since if this were so, then the sending actor would be the knowledgeable actor). Such cases may include situations in which information outside of the system is important for answering provenance use cases — such as a data sensor taking measurements of the external environment — or the information is held by a human. Here, an actor may be modified by adding sensing functionality to it so that it can read the information from the data sensor or be given an interface to record information from a human user. Thus, the actor can gain access to previously hidden information and can subsequently document it.

It is also possible to introduce actors into an application to help in the answering of provenance use cases. In the OTM example shown in Figure 5, we can imagine a use case question asking how long it takes for a request entered into the user interface by a doctor to reach the EHCRS. It is clear that this question cannot be answered with the actors so far identified in the application, since none of the actors currently have access to the necessary time information (according to our decomposition). To overcome this, another actor can be added that receives first a message (M7) from the UI stating when a screening request is received from a doctor (marked as *timer_1*), and second, a message (M8) from the EHCRS stating when it received a donor ID request from the DDC (marked as *timer_2*). This is shown in Figure 6, which makes an addition to the graph by adding the node labelled 'T' representing a new Timer actor to record this information.

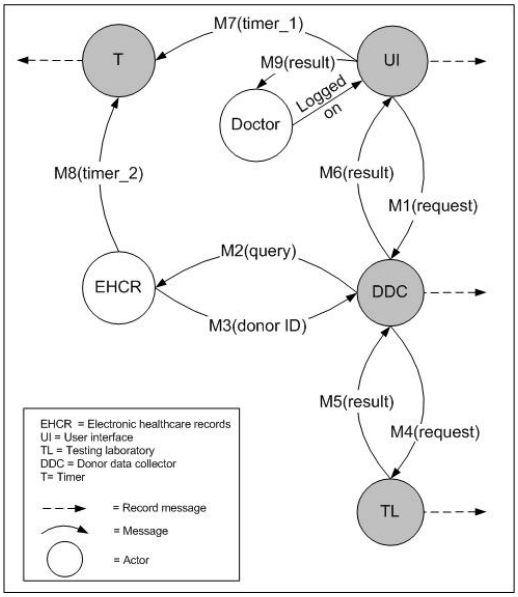

Figure 6: Modifications to the OTM Example

4.3.2 Modifying the Set of Interactions

The second method for locating information items necessary to answer use case questions is by modifying the set of *interactions* between the actors in an application. For example, interactions can be modified to exchange more information between a knowledgeable actor and a non-knowledgeable actor, allowing the latter to use this information to make an information item explicit. Similarly, in situations where a knowledgeable actor cannot record process documentation but the non-knowledgeable actor can, a message can be sent from the former to the latter containing the information to be recorded. (Examples of knowledgeable actors that cannot record process documentation include a human actor, or a component that, for security or legacy reasons, cannot be

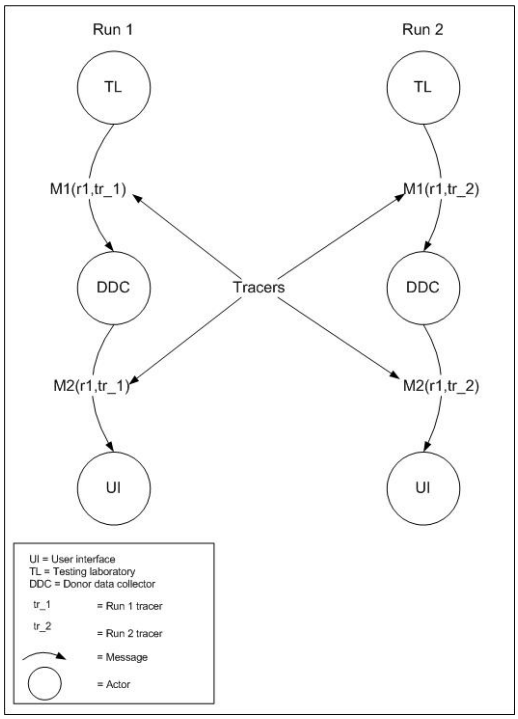

Figure 7: Demarcating process using tracers

modified to record process documentation.)

One special form of interaction modification involves the use of message *tracers*. Many provenance use cases require that certain application processes be identified. A problem here is that if the application makes several distinct runs of the same process, and process documentation is recorded for each, then how are the p-assertions about each process to be distinguished? If the actors involved in these different instances record process documentation, then how is it possible to ensure that documentation referring to one process does not become associated with the other process? One simple way to ensure this cannot happen is to modify the messages sent by each actor recording process documentation so that whenever they send messages to other actors about a given process they may be involved in, they include a unique *tracer* in that message.

Figure 7 illustrates this graphically for the OTM example. The boxes represent the actors involved, and the left side of the figure represents (a part of) one run of the donor screening process, while the right side of the figure represents another run. The marker tr_n represents tracers added to the messages being passed between the actors involved in the process, where n is replaced by a number representing the identifier of the tracer. The tracers added to the process in the left hand side run can be distinguished from the messages from the right hand side run. At a later time, a querier can now retrieve all p-assertions relating to one particular run of the process by indicating which tracer it is interested in.

4.3.3 Adding Provenance Functionality

The final adaptation that PrIMe discusses is how to provide the necessary functionality to actors so that they can record process documentation.

When it is clear than an actor has access to an information item necessary for answering use cases, functionality must be provided for the actor to record documentation about it. To do this, PrIMe recommends using a *provenance wrapper*. The wrapper must be given access to those information items that must be recorded, where these might be evident in the incoming and outgoing *interactions* of the actor, the internal *states* of the actor and also the *relationships* between incoming and outgoing messages. To capture information items in these circumstances the provenance wrapper allows three kinds of p-assertions to be recorded: *(i)* interaction p-assertions, *(ii)* relationship p-assertions and, *(iii)* actor state p-assertions as described in Section 4. Once p-assertions that contain the necessary information have been captured by the wrapper, they can then sent to a provenance store for later retrieval and querying.

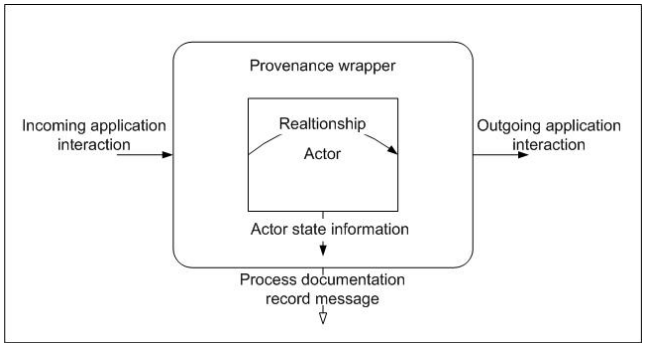

Figure 8: The provenance wrapper

Figure 8 shows a provenance wrapper diagrammatically. Incoming and outgoing interactions are intercepted by the wrapper, which also has access to relevant aspects of the actor's state and has access to the relationships between incoming and outgoing messages. The wrapper then documents these pieces of information using the provenance client side library (see below) and sends a record message containing the documentation to a provenance store.

4.3.4 Embedding The Provenance Client Side Library

The Provenance client side library [9] is a collection of functions that allows developers to provide the functionality of a Provenance wrapper. It comprises a number of high level client classes that provide the necessary recording and querying interfaces that enable developers to implement the necessary provenance functionality.

In order to determine which actors require provenance functionality, two simple heuristics should be observed.

- For any important piece of data there should be a knowledgeable actor capable of recording it, and
- for any interaction message there should be at least one actor recording its view of such an interaction

Once the necessary functionality has been provided to the actors in an application, they can then record the necessary process documentation. In the OTM example, several actors to record process documentation are identified. This is shown in Figure 6, where the shaded nodes represent those actors that record process documentation.

4.3.5 The Modified OTM Example

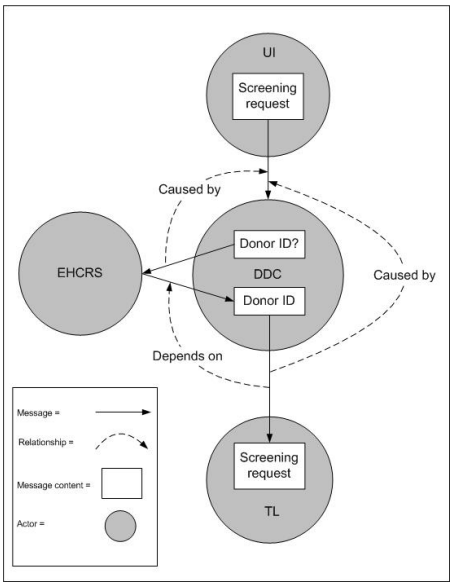

Figure 9: Relationships between incoming and outgoing messages for the DDC in the OTM example

Figure 6 shows how the OTM example looks once the appropriate modifications have been made. The figure shows that the DDC receives a message (M1) from a doctor's UI, which contains a *request* for an organ screening. Receiving this message, the DDC passes it to the EHCRS (M2). Both the UI and the DDC document their roles in this process, with the UI sending an *interaction* p-assertion to a provenance store (not shown) stating that it has sent a message (M1) to the DDC containing a request for an organ screening. The UI also records information about the doctor that is currently logged onto the interface as an *actor state* p-assertion to be able to identify who created the request. Upon receiving M1 from the UI, the DDC records an *interaction* p-assertion stating it has received a message from the UI about a donor organ. The DDC may then record a *relationship* p-assertion stating that the message it sends to the EHCRS requesting a donor's identifier was *caused* by the receipt of the message from the UI. This is shown in Figure 9, which shows the relationships that exist between the DDC's incoming and outgoing messages.

Upon receiving the reply message from the EHCRS (M3), in Figure 6, containing the donor identifier, the DDC then sends a *request* (M4) to the TL to perform an analysis of the blood of the donor's organ identified by the donor ID. This in turn can be documented as a relationship p-assertion, such that the sending of this message, although ultimately caused by the receipt of M1, *depended* upon the receipt of M3 containing the donor ID (see Figure 9). Once this is done, the TL may record interaction p-assertions and relationship p-assertions, which state respectively that it had received a message from the DDC to perform an organ test and that the message it sends back in response containing the *result* was caused by this message.

By storing the above information, it is now possible to answer the use case questions listed above. The provenance of a given organ diagnosis decision is given by understanding how the individual components of the system were involved in the process. We know that the provenance of the decision depended upon the interactions between the doctor, the UI the DDC the EHCRS and the TL. The test performed on the organ supplied by the identified donor resulted in the doctor being able to make a decision based upon the result of that test.

We can also understand how many doctors were involved in a given case by examining the process documentation of the UI that recorded who first initiated the request for an organ screening test. Other components may also record similar information if they have the requisite functionality.

5. DOCUMENTATION STYLES

When recording documentation about processes, certain pieces of application data will become associated to it that will require access control. For example, the Donor ID found in the OTM example constitutes private data and must be kept so [2]. In such cases, action must be taken to ensure that the information is adequately protected from a security standpoint. To effect this, developers can utilise a set of *transformations* that can be made to process documentation that can either restrict access to, or obfuscate the associated data contained within process documentation. These transformations or *documentation styles* [7] are used to transform certain parts of individual p-assertions to achieve a number of different aims. Those that are relevant for security are as follows:

- *Reference-Digest documentation style* transforms part of a p-assertion creating a reference to the original data stored elsewhere along with a digest. Secure access to the original data is controlled by access mechanisms and policies of the data container in a way that only authorized users can access the original data. A querying actor (with authorization to access the original data) can compute and compare the digest in the p-assertion with that created with the original data to ensure the data's integrity.

- In the OTM example used above, donor identifiers are private information, and so any process documentation that contains these identifiers must be transformed to protect the privacy of donors. In such cases as this, the *anonymised documentation style* can be used. This style replaces restricted information with an anonymised identifier.

- *Security signing documentation style* allows for parts of a p-assertion to be signed by a recording actor in order that it can be associated to a specific actor and thus ensures accountability.

- *Security encryption documentation* style transforms a message by encrypting some of its content. This enables the p-assertion to be accessible to only those actors that have explicit rights to decrypt the encrypted data.

6. RELATED WORK

PrIMe is the first methodology to specifically target the development of provenance-aware applications. As such, there is nothing else to compare it to directly. However, there are many similarities that PrIMe shares with object-oriented (OO) (e.g. [12]) and agent-oriented (AO) methodologies (e.g. [15]). Like both OO and AO approaches, PrIMe

is a top down decomposition approach in which the developer must identify those actors (objects in OO, agents and roles in AO) required to record process documentation to answer provenance queries. However, whereas OO approaches go on to identify the functionality of the objects they identify and AO approaches identify the roles and tasks that each agent must execute, PrIMe does not need to go this far, since it is assumed that the application has already been specified.

Similar to AO approaches such as GAIA [15], PrIMe also maps out interactions between identified actors, however, unlike GAIA, the emphasis is on where the messages are going and what their contents are so that it becomes possible to locate information items within the application.

PrIMe also borrows the concept of use cases from the UML [8] modelling language, but specifies these to be specifically provenance-based use cases.

7. FUTURE WORK

We hope to extend and improve upon PrIMe in several directions. First, we are intending to include features of Aspect Oriented Programming [6] to enable annotations to be made regarding the forms of information that need to be captured in process documentation. Secondly, we are examining ways to provide evaluative metrics that will enable us to provide examples of the benefit and cost trade-offs in using PrIMe in the application design process. This is necessary, since although we have applied PrIMe to several, disparate application domains, such as bioinformatics [13], aerospace engineering [10], health care [2] and a variety of other applications within the Provenance Challenge [1], all of which show the general applicability of the methodology, we still require more quantitative evidence of its effectiveness as a tool.

8. CONCLUSION

This document describes the PrIMe methodology. It describes in detail the necessary steps to take in making an application provenance-aware and thus be able to exploit the provenance-based middleware architecture presented in [7]. The steps described are split into three distinct phases comprising: *(i)* identification of provenance use cases and the kinds of information necessary for them to be answered; *(ii)* 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, or makes it evident where access to information items is not yet possible and, *(iii)* adaptations to applications to enable the discovery or creation of actors that have access to information items and can record them to provenance stores.

The PrIMe methodology provides a step-by-step guide to making applications provenance-aware, and is vital to the development of provenance-aware applications. 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. The availability of PrIMe for developers and users of applications helps to ensure that the effort required to make applications provenance-aware is minimised.

9. REFERENCES

[1] The Provenance Challenge. http://twiki.ipaw.info/bin/view/Challenge/WebHome, 2006.

[2] S. Álvarez, J. Vázquez-Salceda, T. Kifor, L. Varga, and S. Willmott. Applying provenance in distributed organ transplant management. In L.Moreau and I.Foster, editors, *LNCS: Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, volume 4145. Springer-Verlag, 2006.

[3] P. Buneman, S. Khanna, and W. Tan. Data provenance: Some basic issues. In *Foundations of Software Technology and Theoretical Computer Science*, 2000.

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

[5] C.Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language*. Oxford University Press, 1977.

[6] S. C. Filman, R.E.; T. Elrad. and M. Aksit. *Aspect-Oriented Software Development*. Addison Wesley, 2004.

[7] P. Groth, S. Jiang, , S. Miles, S. Munroe, V. Tan, S. Tsasakou, and L. Moreau. An Architecture for Provenance Systems. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[8] P. Harman and M. Watson. *Understanding UML, The Developers Guide*. Morgan Kaufmann, 1998.

[9] S. Jiang, P. Groth, S. Miles, V. Tan, S. Munroe, S. Tsasakou, and L. Moreau. Client side library design and implementation. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[10] G. Kloss and A. Schreiber. Provenance implementation in a scientific simulation environment. In L.Moreau and I.Foster, editors, *LNCS: Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, volume 4145. Springer-Verlag, 2006.

[11] L.Moreau and I.Foster, editors. *International Provenance and Annotation Workshop (IPAW'06)*, volume 4145. Springer Verlag, 2006.

[12] J. Rambaugh. *Object oriented Modeling and Design*. Prentice Hall, 1991.

[13] W. F. P. G. Sylvia C. Wong, Simon Miles and L. Moreau. Provenance-based validation of e-science experiments. In *In Proceedings of 4th Internation Semantic Web Conference (ISWC'05)*, november 2005.

[14] A. G. Woodruff. *Data Lineage and Information Density in Database Visualization*. PhD thesis, University of California at Berkeley, 1998.

[15] M. Wooldridge, N. R. Jennings, and D. Kinny. The GAIA methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

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