PrIMe: A Methodology for Developing Provenance-Aware Applications

Authors:	Steve Munroe
	Simon Miles
Reviewers:	Paul Groth
	Sheng Jiang
	Victor Tan
	Luc Moreau
	John Ibbotson
	Javier Vazquez
Identifier:	D3.2.2
Type:	Discussion
Version:	0.2.2
Version:	August 2, 2006
Status:	internal

PrIMe is a methodology for adapting applications to make them *provenance-aware*, that is to enable them to document their execution in order to answer provenance questions. A provenance-aware application can satisfy *provenance use cases*, where a use case is a description of a scenario in which a user interacts with a system by performing particular functions on that system, and a provenance use case requires documentation of past processes in order to achieve the functions. In this report the PrIMe is described. In order to illustrate the steps necessary to make an application provenance aware, an Organ Transplant Management example application is used.

The PROVENANCE project receives research funding from the European Commission's Sixth Framework Programme

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 applications provenance aware allows its users to understand the provenance of their data, where this is defined as *the process that led to that data*. To be able to determine the provenance of data, documentation of an application's execution must be available, and so provenance-aware applications must document their own execution. Such documentation is called *process documentation* and is comprised of multiple individual pieces of information, called *p-assertions*, which are recorded during execution and then stored and maintained in a repository of such information called a *provenance store*. This ensures that necessary and sufficient forms of process documentation can be captured to give a complete account of any data item's provenance. For example, p-assertions allow application developers to document various aspects of execution, and thus provide descriptions of those parts of an execution that relate to, or impact upon, a given data item. This allows the determination of a data item's relationships to other data items and processes, such as its dependencies or causal effects and, at the same time, provides a description of the data flow through an application.

In order to facilitate the development of provenance-aware applications it is essential that developers have at their disposal a structured approach to integrating provenance functionality into their own applications.

To enable such an approach the PrIMe is presented, which stands for a *PRovenance Incorporating Methodology*. In the remainder of this document, PrIMe and how it can be applied to computational systems, or applications composed of computational entities, is described in detail. Though the effects of physical entities on such applications (such as humans or machines) and how PrIMe can deal with them are dealt with briefly (see Section 6.1.2), PrIMe focuses mainly on computational entities.

The format of this document is as follows. In the next section, PrIMe is introduced, and some assumptions adopted by PrIMe as well as 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, the process for identifying provenance use cases is described as well as how to identify the kinds of information required for answering such use cases. Section 5 then describes how to decompose applications and how to map out the flow of information within them. Section 6 then goes on to describe different kinds of adaptations that can be made to applications in order to facilitate the recording of process documentation. Finally, Section 10 offers some concluding remarks.

2 Introducing PrIMe

PrIMe is a guided approach for making applications *provenance aware*. This is achieved by exposing application information that can then be documented through a series of analysis steps and well-specified *adaptations*, where an adaptation is a modification to the application design, each of which is supported in implementation by elements of the *provenance architecture* [**?**].

In the development of PrIMe, the following desirable criteria were used as guidelines.

Usability PrIMe should be easy to apply.

Traceability All design decisions made using PrIMe should be traceable back to one or more use case requirements.

Applicability It should be possible to successfully apply PrIMe to a wide range of applications.

2.1 Initial Assumptions

PrIMe adopts a particular view on the way that applications are constructed, in particular, one in which applications are composed of *actors* that *interact* with one another through the exchange of *messages*. An actor is defined as a component of an application that performs some functionality and interacts with other actors. Adopting this view, however, does not limit the range of applications to which the provenance architecture is applicable since it is argued that all applications can be mapped onto this view with more or less effort (for example, the EU Provenance project [**?**] is using PrIMe to develop provenance-aware systems in two very different exemplar application: organ transplant management [AVSK+06], and aerospace engineering [KS06]). The key aspects of this actor-based view are described below.

- All changes in an application 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 another application actor.
- All assertions in a provenance store have been submitted by application actors that have direct access to the information being asserted, i.e.
  - If an actor receives or sends data, it can record that data.
  - At the time of receiving/sending, an actor has a given state and so can record details about that state.

  - An actor can know, and so record, how the information in two messages it has sent/received are related, e.g. one is in response to another.

2.2 Structuring PrIMe

The overall structure of PrIMe is shown in Figure 1. Each oval in the diagram corresponds to a set of entities identified within the application and each has a numbered step showing the order that each set of entities is to be identified. The lines between sets of entities indicate how identifying one set influences how the subsequent sets are identified. The dashed ovals delimit different phases of the methodology. Phase 1 involves the identification of provenance use cases and the pieces of information that will be used to answer the use cases. Phase 2 involves the decomposition of the application into a set of actors, and their interactions. Phase 3 involves making principled adaptations to the application in order to ensure the required information items are available for documentation. Traversing these steps, PrIMe starts from the application itself. PrIMe assumes 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 steps are taken as follows. First, an analysis is performed to identify the set of *provenance use cases* that are to be answered by the provenance architecture (Step 1.1), then the *information items* (pieces of information) that are required in order to answer these use case questions are identified (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 (Step 2.2), thus revealing the information flow through the application. Once this is done, it is then possible to determine which application actors have data representing the information items necessary to answer the use cases as the application is run: 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, i.e. it is still not possible to identify an actor that has access to an information item. In this case, the process of identifying actors and interactions is repeated until an actor can be located that knows about the information item in question. Finally, *adaptations* are introduced into the application in order to expose information items and add provenance functionality (Step 3.1). 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 querying actors to perform queries on the documentation in order to answer provenance use cases.

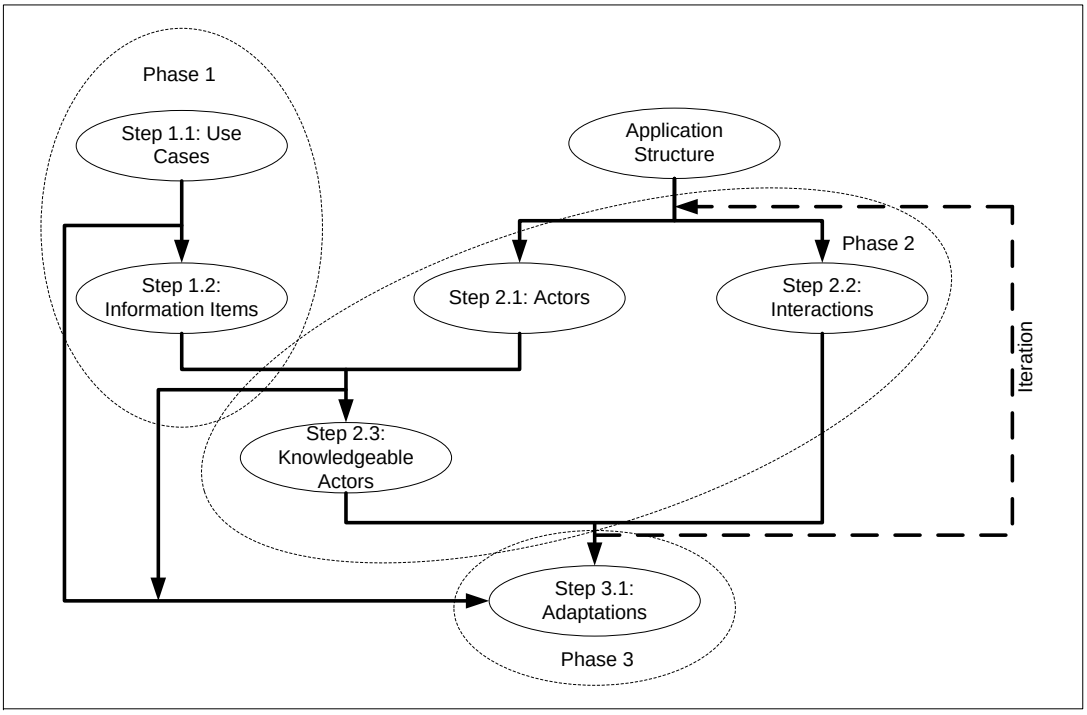

Figure 1: Overall structure of PrIMe

3 The Organ Transplant Management Example Application

The example application used in this document is the Organ Transplant Management (OTM) application used as one of the exemplar applications in the EU Provenance project. In the application, organ donors and patients waiting for organs must be matched up according to various criteria, such as blood, immunology tests and so on. For this document, part of the workflow of the OTM application is extracted to facilitate our explanation of PrIMe.

3.1 Patient Organ Testing

The example used in this document involves conducting blood tests on patients. Here, potential donor recipients undergo a series of tests to enable a decision to be made about whether they are suitable candidates for organ transplants. 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 patient is being taken care of, and where the doctor who initiates the testing process resides. The EHCRS is the place where all the records for every patient are kept. Finally, the testing laboratory is where the 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 patient's records, after which the hospital can request a blood test to the testing laboratory, passing along the necessary patient data obtained from the EHCRS.

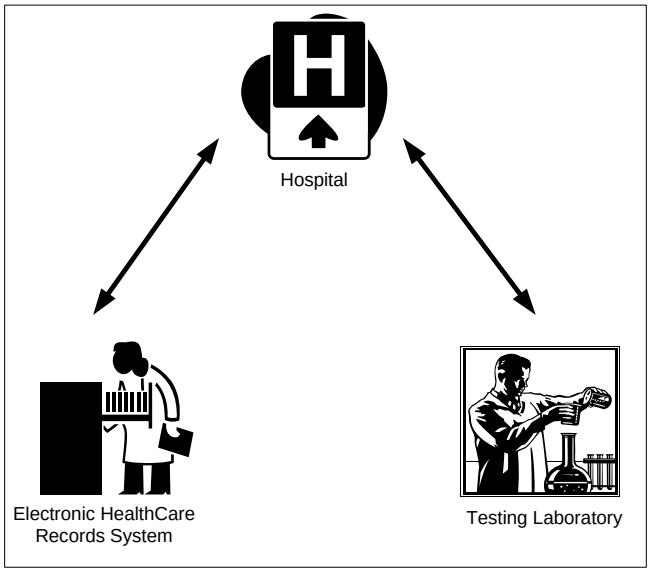

Figure 2: The OTM example application

Figure 3 shows the above workflow graphically.

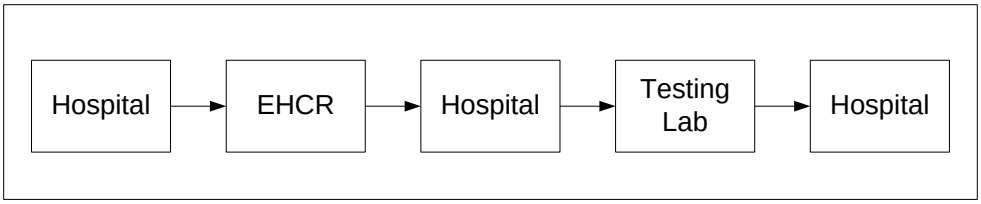

Figure 3: The OTM workflow

4 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. This process involves several steps, each of which plays a specific role in obtaining use cases. Once the use cases are identified, a process of analysis must then be undertaken in order to discover the pieces of information within the application that constitute the answers to the use cases. In the next few sections, descriptions of each of these steps is given in detail.

4.1 Step 1.1: Provenance Use Cases

The starting point for making an application provenance aware is to identify the set of provenance use cases that are to be answered, and PrIMe distinguishes between the two following types.

Core provenance use cases are use cases known when PrIMe is applied.

Future provenance use cases are use cases not known about until after the application has been made provenance aware.

Core provenance use cases drive the process of making an application provenance aware by helping to inform the developers of the granularity of the processes to be considered and the critical information to expose. Future provenance use cases, however, are not known by the developer at the time the application is being made provenance aware, but by ensuring that the application is designed to capture as much potentially useful process documentation as possible, they can be anticipated and thus increase the chances that the application is future-proof.

4.1.1 Eliciting Use Cases

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 to help designers collect the core provenance use cases. This process works as an *intuition pump* to help the user's identify provenance use case question that will be helpful for them to understand their application better.

The elicitation process comprises three simple steps.

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

4.1.2 Defining provenance

According to the Oxford English Dictionary, provenance is defined as *(i) the fact of coming from some particular source or quarter; origin, derivation. (ii) the history or pedigree of a work of art, manuscript, rare book, etc.; concr., a record of the ultimate derivation and passage of an item through its various owners.*

For computational systems, a definition of provenance that relates specifically to data is required. PrIMe adopts the definition supplied in [**?**].

Definition 1 (Provenance of a piece of data) *The provenance of a piece of data is the process that led to that piece of data.* □

By explicitly stating the definition of provenance, the user's obtain the right understanding of what provenance means in the context of computational applications and data.

4.1.3 Examples of provenance use case questions

In order to provide examples of the kinds of provenance use cases a provenance-aware application can support, several generalised, non-application specific use case 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 and its use and transformation.

- 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?
- Did the process that produced this data follow the original plan?
- Did the process that produced this data meet regulatory rules?
- What data was used as input to a process.
- What was data was used at point X in a process.
- What operations were performed on a given piece of data.

In the context of the OTM example application, more concrete example use case questions can be given. Below are just a few instances.

- Retrieve data linked to all actions / events associated with a patient (recipient or donor)
- What decisions were made for a particular case?
- What is the medical analysis tree for a given organ?
- Determine if any deviation took place from the standard workflow for a given organ (i.e. in Figure 3, a deviation might be if the EHCR 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).

In many instances, use case question will be focused on a few important objects within the application. For example, in the OTM application, provenance use case questions relate primarily to four key objects.

- Organ Donors.
- Recipients of organs.
- Organs.
- Decisions.

4.1.4 Expressing Provenance Use Cases

In order to collect the information made explicit in the use case elicitation process, PrIMe provides a simple form in which use cases can be expressed and recorded. The format aims to be explicit, and mirrors the information content of UML use case diagrams [HW98], and is the basis for refining information in the future. The form should capture the following information.

- Statements describing something that already happens in the application.
- Statements describing a specific provenance-related use case question.
- A list of the application components involved in carrying out the activity identified above.
- A list of the information item(s) necessary to answer the identified use case question.

Below is a more specific example use case in the OTM application.

Donor A's organs are screened for potential donation. What is the provenance of the donor's organ diagnosis?

In this example, a description of application functionality is given in the first sentence, and the question to answer is given in the second sentence. In order to extract the

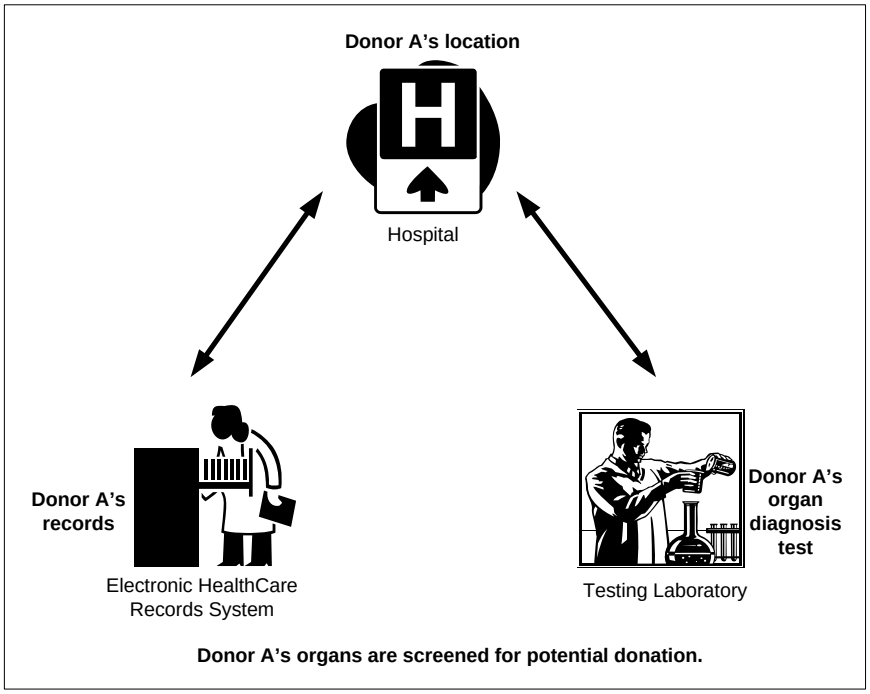

Figure 4: The components in the OTM application involved in the testing of blood

Table 1: Expressing provenance use cases

Application description	Donor A's organs are screened for potential donation.
Use case question	What is the provenance of the donor's organ diagnosis?
Involved components	Hospital, EHCRS, Testing laboratory.
Relevant information items	?

right kind of information to answer the question, the main entities within the application must be examined and the identity of those involved in carrying out the described functionality must be established. Looking at Figure 4, it is possible to see that the three main components in the application involved in the process of determining a patient's organ diagnosis are the hospital, the EHCRS and the testing laboratory.

Going through this process for each use case enables the information to be captured in the manner shown in Table 1. Note that, as yet, the information items have not been identified. How this is achieved is discussed in the next section.

4.2 Step 1.2: Information items

When considering how to answer a use case it is necessary to identify what *information items* are relevant, and there may be many such information items, e.g., a given result,

or a sequence of decisions. Therefore, for each use case the information items required for its satisfaction must be identified.

It is also important at this point to consider future use cases, and what forms of information might be useful to capture in order to answer them. Of course, this cannot be an exact activity, but by identifying potentially useful information items, the task of answering use cases that come to light in the future can be facilitated.

Answering a use case may involve identifying a part of, or a whole *process* within an execution, understanding the *relationships* between the different data items of the application, knowing particular *states* of components during execution (which may represent particular data items), or some combination of these. It is, therefore, necessary for a process of analysis to take place, in which the use case is examined and the appropriate information items necessary to answer it are identified. 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 information such as a result of a computation, the outcome of a decision or the state of a given component, and such items are found in the interactions between different components, and also in the states of the various components.

Processes Interactions between components and the relationships between data represents an application's processes. Consequently, in order to identify a process, either in part or in whole, it is necessary to identify interactions and data relationships.

Relationships When seeking the provenance of some entity, it is often necessary to identify the relationships between it and other entities. For example, a data item found in an interaction may be related to other data items in other interactions and these relationships must be identified and named.

Examining the OTM example, the information items involved in answering the use case question are the patient record (or the patient identifier, drawn from the records), the test results and the diagnosis decision. These can now be listed as shown in Table 2.

Once this process has been completed for every use case question, Phase 2 of PrIMe can begin in order to decompose the application into actors. This phase establishes which actors in the application have access to the identified information items, and thus are the entities responsible for recording process documentation.

5 PrIMe Phase 2: Actor Based Decomposition

When analysing a system in order to apply PrIMe, it is necessary to follow PrIMe's approach to modelling computation systems, in which applications can be mapped onto

Table 2: Expressing provenance use cases

Application description	Donor A's organs are screened for potential donation
Use case question	What is the provenance of the donor's organ diagnosis?
Involved components	Hospital, EHCRS, Testing laboratory
Relevant information items	**Patient ID (data item), test results (data item), diagnosis decision (data item), links between the patient ID, test results and diagnosis decision (relationships)**

a model of *actors* and *interactions*. This approach is similar in nature to object oriented approaches to modelling systems (e.g. [Ram91]) and identifying classes, except that in PrIMe, considerations of provenance use cases are required. As can be seen below, use cases determine the level of granularity, or the depth of the decomposition of a system required when making an application provenance-aware. However, other considerations must be taken into account too; a fine-grained decomposition has the advantage that a clearer picture can be formed of which actors have access to pieces of information leading to more detailed process documentation being obtained. This then leads to more accurate traceability and a concomitant increase in an ability to check and modify the system. The advantage of a more coarse-grained decomposition is that less actors need to be modelled in total and, in particular, less actors are modelled that are irrelevant to satisfying provenance use cases. Ultimately a balance must be found, though PrIMe encourages a bias towards finer-grained decompositions, since the robustness of designs produced by PrIMe relies on adequate traceability (which is one of the criteria given above for judging the methodology to be effective), and the availability of process documentation. An important concern is to include any application actors referred to in provenance use cases, because information about them is primarily known or knowable by them.

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

- First: identify obvious actors (Step 2.1).
- Second: map out the interactions between actors (Step 2.2).
- Third: identify actors that have access to information items (Step 2.3).

These three steps may need to be repeated if it is discovered that no actor is identified at this level of granularity that has access to a use case related information item.

5.1 Step 2.1: Identifying Actors

An *actor* is an entity within an application that performs actions, such as Web Services, software components, machines, people 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, and many issues must be considered. For example, at what level of granularity should the analysis of application actors be conducted? In Web Service based applications, should the granularity be at the service level, i.e. should each service be an actor, or is there functionality within a service for which documentation is required, which would entail the need to decompose the service into a set of actors. In order to aid the identification of actors within an application, PrIMe provides *actor identification heuristics*.

5.1.1 Actor Identification Heuristics

Actor identification heuristics aid the designer by providing rules of thumb in identifying which components of an application should be classified as actors. If the application is considered as being comprised of components that send information to each other in order to perform an execution, then the first step is to apply the following two simple rules.

1. Identify the components that are the receivers of information. 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 application actor.

2. Identify the components that send the information in each interaction. 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 too fine grained decomposition of the application. Indeed, following these rules slavishly leads to the classification of every operation as an actor, and is clearly overkill. What is important when applying the rules is to take into account the provenance use cases that need to be answered, and the information items required to achieve this. By basing the process of identifying actors on the need to identify the relevant information items, appropriate levels of decomposition can be achieved. When considering whether or not a component is an actor, it is important to determine if the component has access to information items, in which case it can be said that the component is a *knowledgeable actor*. When no further decomposition is required or possible, the actors identified are termed *primitive actors*, in the sense that there are no details available about their internal processes. If further decomposition is possible to carry out, then such internal processes can, in turn, be mapped onto actors

(or subactors of the higher level actor) and the interactions between these subactors can also be mapped.

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 do classify each of these components into high level, primitive actors.

5.2 Step 2.2: Actor Interactions

The next step in PrIMe is to map out the interactions between the previously identified actors. Other methodologies, for example the GAIA methodology used in agent oriented design [WJK00], also provide methods to identify interactions between application components. In PrIMe, a similar approach is necessary. 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. To do this, it is necessary to identify to which actors an actor is sending messages and from which it is receiving messages. This is important since, by identifying messages and their contents, it is possible to produce a complete model of data flow through the application, and thus discover which actors have access to information items necessary for answering provenance use case questions. The approach in PrIMe is to identify the information flow through applications, and to construct graphs of the different actors and the messages they send to each other. In Figure 5, one of these graphs (which are termed abstract data graphs) is shown. The nodes represent individual actors and the arcs (shown as uni-directional arrows) represent messages being passed from one actor to another. Annotations over the arrows show message and content identifiers. It is important to note that, at this level, it is not important to identify whether the content c1 in message M2 has exactly the same format as in message M1 or a transformation has been made; this will be identified in later steps of the methodology. Also it is important to note that, if numbers are used in message identifiers, these may or may not correspond to the order such messages are produced, as in some scenarios there may be several messages being created in parallel. The use of such graphs makes explicit which actors have access to which information items, and thus enables developers to identify knowledgeable actors (see Section 5.3).

Initially the process is to map out the *existing* interactions between the actors of a system — without regard to answering provenance use cases. Once the application is mapped onto this actor/interaction-based model, the developer can then begin to consider the previously identified use cases and try to identify which actors, given this current model of the application as actors and interactions, have access to the information items necessary to answer the use case questions.

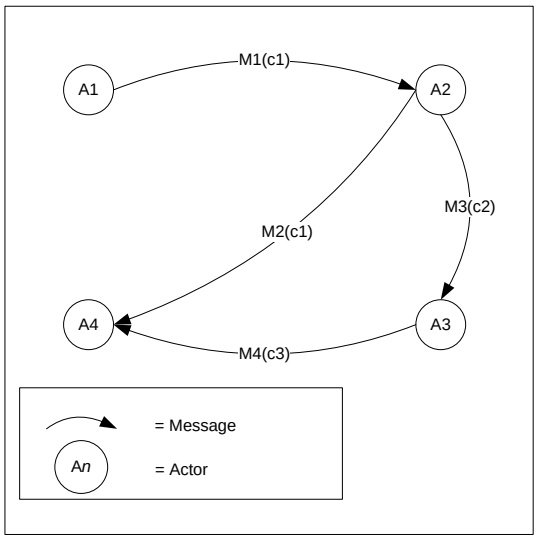

Figure 5: An abstract data graph

5.2.1 Interactions in the OTM Example

In this section the guidelines provided by PrIMe to the OTM example are used in order to identify the actors in the application and the interactions between those actors.

Examining the OTM example, it can be seen that when a patient is to have organ diagnosis performed, the hospital must send a query to the EHCRS to obtain the patient ID. Then, the hospital sends this ID to the testing laboratory with the request to perform the necessary tests. In this simple example, there are four messages being passed between the three high level primitive actors, where each of these messages contain certain data items: the query (q1) to the EHCR, the patient ID (pid), the request to perform a blood test to the patient (with the same pid), and the test results from the laboratory (r1). Figure 6 shows the message exchanges (along with their contents) between the identified actors in the system.

Having gone through this process, it is then possible to tabulate the interactions of each actor along with all the information items it has access to. For example, in Figure 7 the hospital's interactions are recorded in the sending and receiving tables, along with the recipient or sender actors respectively and the information items contained in the contents of the messages.

5.2.2 Constructing Abstract Data Graphs of Application Interactions

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 8). The actors and the messages correspond to those identified in Figures 6 and 7.

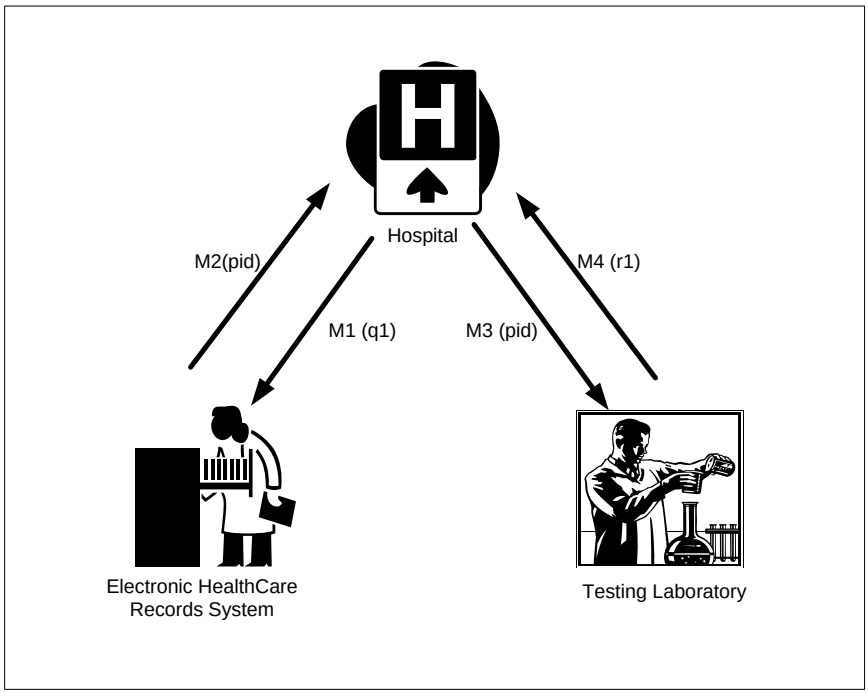

Figure 6: Interactions between actors in the OTM example

5.3 Step 2.3: Knowledgeable Actors

At this stage in the application of PrIMe, provenance use cases and the information items necessary to answer the use cases have both been identified, a decomposition of the application into actors has been conducted and, lastly, their interactions have been identified and captured. Next, it is necessary to examine the information items identified previously and attempt to associate them with actors identified in the application. Any actor that has access to an information item is known as a *knowledgeable actor* and the aim is to associate every information item with such an actor. If this can be achieved, then the developer can move to Step 3.1 of PrIMe to introduce the necessary provenance functionality (see Section 6.3). However, there are two reasons where such an association between actors and information items may not be possible. Either:

- a) an information item is located within an actor and has not been exposed in the interactions between actors through message exchanges or,
- b) no more decomposition is possible and there are still no actors with which an information item can be associated. This can occur when an information item is required that can only be derived from existing information items in the application. In other words, the information is implicit and must be made explicit by combining existing information items.

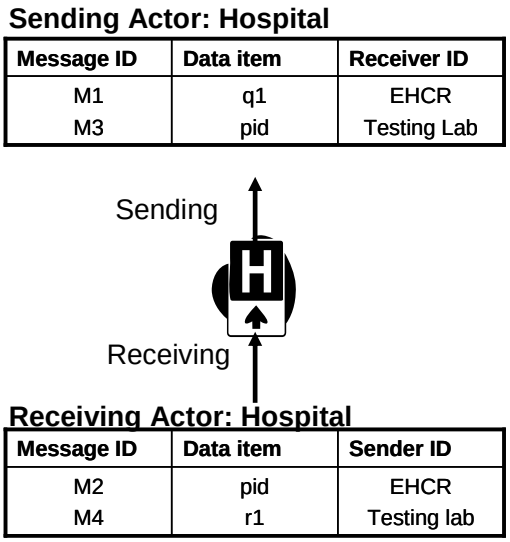

Figure 7: Messages sent and received by the hospital

In the case of a), it is necessary to iteratively apply steps 3, 4 and 5 of PrIMe until a level of granularity is reached where actors are identified that have knowledge of the information item in question. This is discussed in the next few sections. In the case of b), a number of adaptations must be performed to change the application in structured ways in order to be able to make the required information item explicit. Section 6 discusses this in detail.

5.3.1 Step 2.3: Identifying Knowledgeable Actors

A *knowledgeable actor* is an actor that has access to an information item. The *primary knowledgeable actor* for an information item is the primitive actor who first becomes aware of that information, for one of the following reasons.

- The actor creates the information item.
- The actor receives or observes the information item from outside the application.

In the first iteration of the methodology, the developer should identify the primary knowledgeable actor for each information item. Thus, for each use case, the designer must identify the relevant information items and, for each of these, the associated knowledgeable actors. However, given the level of decomposition of an application in actors in the first application of Phase 2 of PrIMe, this may not be possible, since the correct level of granularity in terms of actor decomposition may not have been reached and, thus, the looked for information items may not have yet been exposed. In such instances, more decomposition must be performed, and the next section makes use of the OTM example to make this point clear.

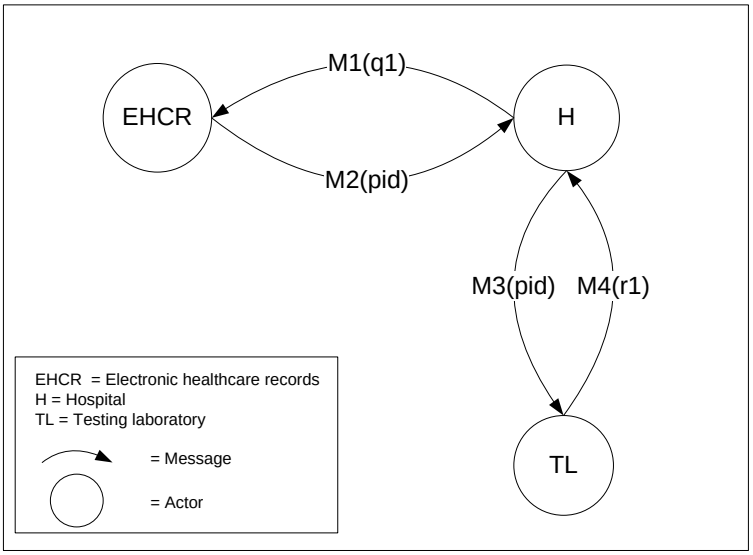

Figure 8: The data graph of the OTM example

5.3.2 Knowledgeable Actors in the OTM Example

The structure of the OTM donor diagnosis example application has been modelled through the application of PrIMe into three actors:

1. The hospital.
2. The EHCRS.
3. The testing laboratory.

Imagine, however, that another use case question asks: *How many doctors have been involved in a donor's diagnosis case?*. To answer this use case question, the information item that contains 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 6), it can be seen that this question cannot be answered, because the information about how many doctors are involved in a given diagnosis case is not available within the interactions between any of the actors. Thus, the required information item (the number of doctors in a given case) cannot be associated with any of the actors so far identified. What must be done in this situation is to examine the hospital and perform a decomposition to a lower level of granularity, thus 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 have access to the information needed. After examining the hospital, it is discovered that doctors access a user interface (UI) to issue diagnosis requests, and that this user interface sends the

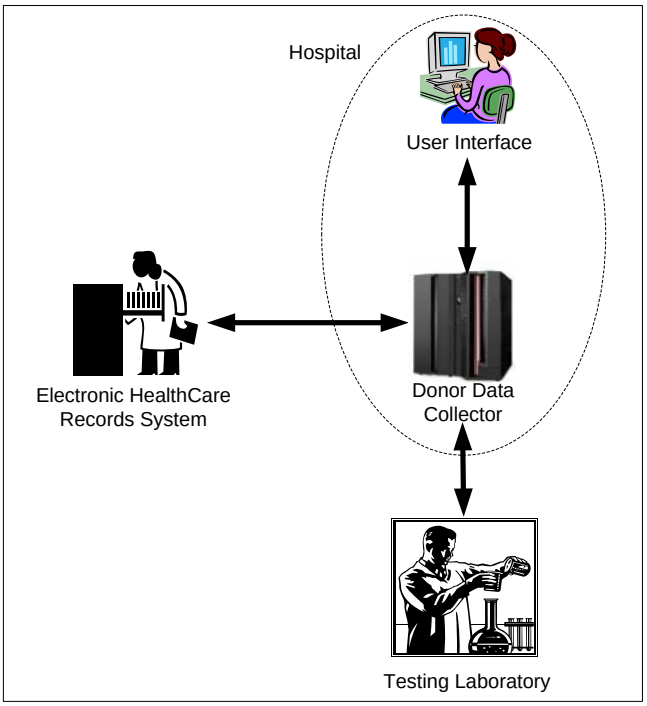

Figure 9: Decomposition of the Hospital into a UI and the Donor Data Collector

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. This more detailed model of the hospital is represented in Figure 9, which illustrates the decomposition of the hospital into two new primitive actors (the UI and the DDC within the dashed ellipse denoting the hospital). It is important to note here that we have not introduced users as actors in the system, as in those cases where all interaction between the system and the user is done through a user interface, such interfaces can be seen as acting on behalf of the user, translating the user inputs into inputs for the system and translating the system outputs into a format the user can comprehend.

Having performed another, deeper level of analysis, steps 4 and 5 of PrIMe must be executed to map out the interactions between these new actors and identify which are knowledgeable about the information item for the new use case. In Figure 10, these steps have been completed and an abstract data graph for the new, more detailed model of the OTM application is produced. Actor H in figure 8 has been substituted with actors UI and DDC, representing the User Interface and the Donor Data Collector, respectively. Two new interactions have been also introduced to show how the blood request (q1) goes from the User Interface to the Donor Data Collector, and how the test result (r1) is sent back to the User Interface.

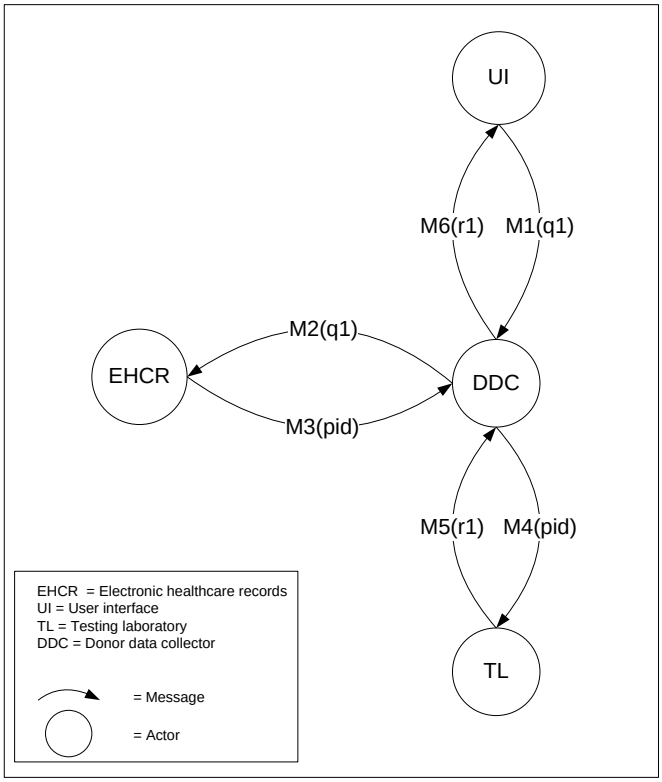

Figure 10: The modified OTM example

Once knowledgeable actors have been found for every information item required to answer all the use cases, decomposition can stop and Phase 3 of PrIMe can begin.

6 PrIMe Phase 3: Adapting the Application

Within Phase 3 of PrIMe, there are two main approaches for adapting an application. The first of these involves modifying the application in order to help make explicit those information items that are currently implicit. This approach may or may not be necessary, depending on whether the earlier phases of PrIMe have successfully identified a knowledgeable actor for each information item. The second form of adaptation, unlike the first, is compulsory, since it involves giving the application the necessary functionality to record process documentation. In this section, both forms of adaptation are described, beginning with the former.

6.1 Application Adaptations to Expose Information Items

Section 5.3 listed two reasons why it may not be possible to associate an actor with an information item: either because more decomposition is required or the information item has not been made explicit in the interactions between the actors of the application. To overcome the latter problem, a number of modifications to the application must be made in order to expose these information items. PrIMe identifies two possible modifications that can be made:

- modifications to actors and,
- modifications to actor interactions.

The first type of adaptation involves modifying the set of already identified actors, or adding actors to this set. The second involves altering the interactions between application actors and, thus, altering the data flow through the application. The first of these is described in the following sections.

6.1.1 Actor Modification

An actor that does not have access to a given information item is termed a *non-knowledgeable actor*. Note, that an actor may be non-knowledgeable actor for one information item and knowledgeable for another, meaning that these terms apply to actors in connection to individual information items. 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 a data sensor taking measurements of the external environment. Here, an actor may be modified by adding sensing functionality to it so that it can read the information from the data sensor, and subsequently become knowledgeable about it.

In terms of the OTM example, it is possible to add extra functionality to the UI actor to obtain more information from the doctors using it, thus making the UI actor a knowledgeable actor for this new piece of information. For instance, designing the interface to introduce a medical diagnosis in a way that doctors can select the data sources for their decisions (e.g. ticking the boxes of the tests that were considered for the diagnosis, that is, the tests that such a diagnosis is based on). In such cases, it is recommended that HCI design principles be adhered to in order that the information capture minimises intrusiveness [GSC+06].

6.1.2 Actor Introduction

A new actor can be introduced to the application to help in the answering of provenance use cases. For example, a new actor could be a new user interface by which users

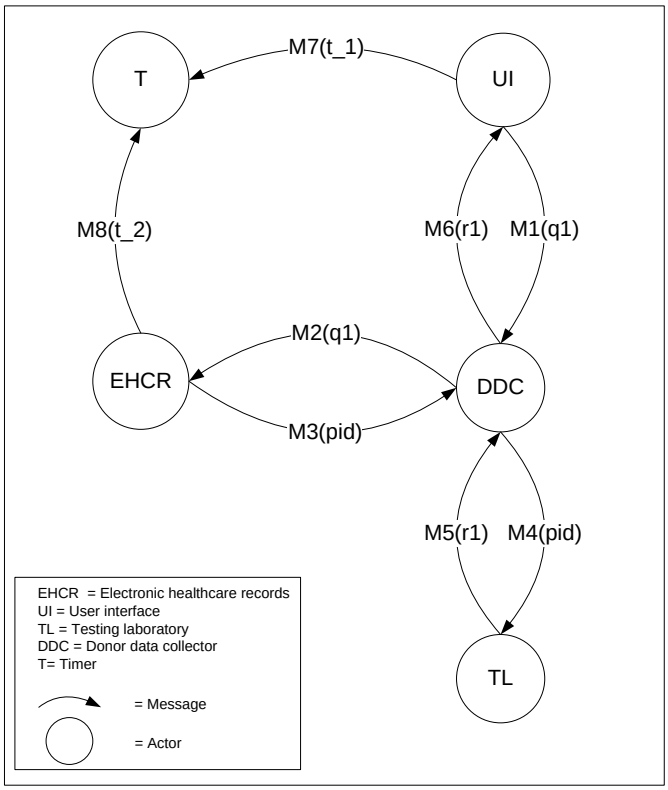

Figure 11: Actor introduction

record what occurs in a physical world process. Taking the OTM example shown in Figure 10, in which a use case asks how long it takes for a query entered into the user interface by a doctor to reach the EHCRS. It is clear that this question cannot be answered with the actors currently in the application, since none of the actors currently have access to this information (according to our decomposition). To overcome this, another actor can be added that receives a message from the UI stating when the request from the doctor for a diagnosis is received, and a message from the EHCRS that states the time it receives the request. This is shown in Figure 11, which makes an addition to the graph by adding the node labelled 'T' representing the new Timer actor. This actor receives messages from the UI and the EHCRS; it can now be given provenance functionality to record these messages as process documentation for later querying (see Section 6.3 for how provenance functionality can be added to actors).

Many applications require information to be introduced from external sources. Such information may only be known by non-application actors such as machines or humans and, in order to make this information accessible for documentation, these external, or *hidden actors* must somehow be represented. In many instances, this can be achieved for human actors by introducing a user interface actor into the application, which can be used by the human to record the information that is available. In

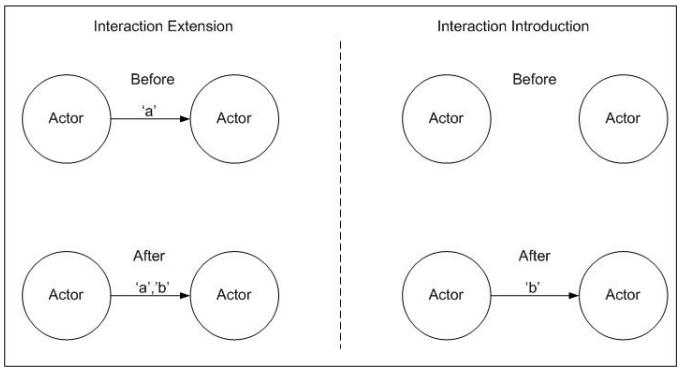

Figure 12: Interaction modification

such cases, it is necessary to conduct an analysis on the information the human actor generates and has access to in order to summarise in an appropriate manner the derivation of such information (which may come from many disparate sources, such as other humans, decision processes carried out by the human or physical observation).

6.2 Interaction Modifications

Interactions between actors constitute the data flow in an application, and so can be used to reveal processes. Once actors have been defined, it is possible to trace their interactions to reveal execution that can then be documented. In some cases, different adaptations may be needed to reveal more fully the processes within an application and here they are outlined.

6.2.1 Interaction Extension and Introduction

An interaction in an application 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 and thus making the actor knowledgeable. This is shown in Figure 12 (left) where the existing interaction between the actors involves the passing of information item ‘a’. After the interaction extension has been introduced, the interaction now involves the passing of a new piece of information, ‘b’ along with the old piece of information. This kind of adaptation changes the flow of information through the application, thus moving information to different actors and can aid in making implicit information items that have no representation as yet in an application, explicit.

Similarly, new interaction between actors can be introduced into the application in which a knowledgeable actor sends the information item to another actor, which is then knowledgeable. Figure 12 (right) illustrates this. Here there is initially no interaction between the two actors, but by introducing a new interaction between these

two actors the receiving actor can now become knowledgeable about information item 'b', and may be able to use this information item in order to make explicit a required information item.

6.2.2 Demarcating Processes

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 run, then how are the p-assertions about each process to be distinguished. For example, there may be more than one instance of a given process. 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. This tracer can then be propagated throughout all the subsequent messages related to that process sent out by the various actors involved. This achieves an effective demarcation of that process from others, by virtue of the unique tracer contained in all the processes' messages.

Figure 13 illustrates this graphically for the OTM example. The boxes represent the actors involved, and the left side of the figure represents one run of the donor diagnosis 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.

6.3 Adding Provenance Functionality to Applications

The final step of PrIMe is to enable provenance functionality to the actors identified for the given provenance use cases. This enables these actors to record process documentation. Such documentation is composed of p-assertions, which come in three forms:

- Interaction p-assertions: assertions about the interactions an actor has with other actors.
- Actor state p-assertions: assertions about an actor's state.
- Relationship p-assertions: assertions about relationships between interactions.

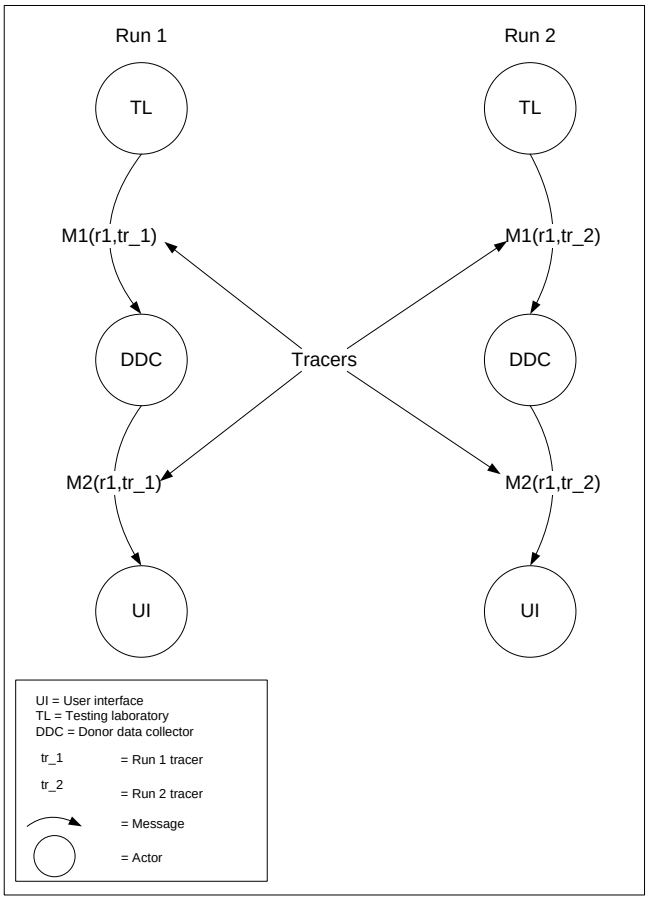

Figure 13: Demarcating process using tracers

Each actor only records p-assertions that directly relate to its own interactions and state, i.e. an actor cannot make assertions about other actor's 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 is of *high quality* [GMM06]. In order to provide such process documentation recording functionality a *provenance wrapper* is applied, which implements the *Provenance client side library*.

6.3.1 A Provenance Wrapper

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 some of the information that the actor has access to, e.g. in-

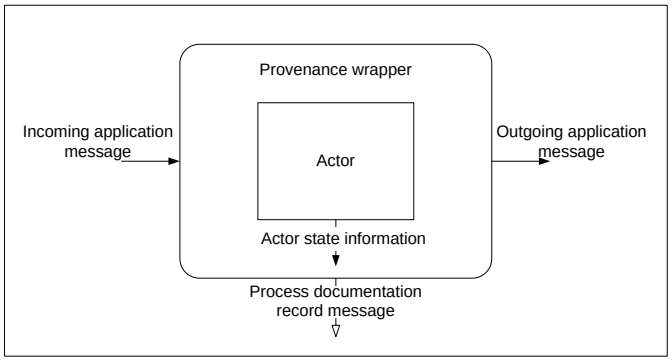

Figure 14: The provenance wrapper

coming and outgoing messages, possibly some state of the actor (if the actor must record actor state p-assertions) and also relationships between the incoming messages of an actor and outgoing messages, which it can then document and send to a provenance store.

Figure 14 shows a provenance wrapper diagrammatically. Incoming and outgoing messages 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.

6.3.2 Embedding The Provenance Client Side Library

The Provenance client side library is a collection of functions that allows designers 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 designers to implement the necessary provenance functionality. Other documents discuss this library in detail (i.e. [JGM+06]), and interested readers are directed there.

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 15, where the shaded nodes represent those actors that record process documentation.

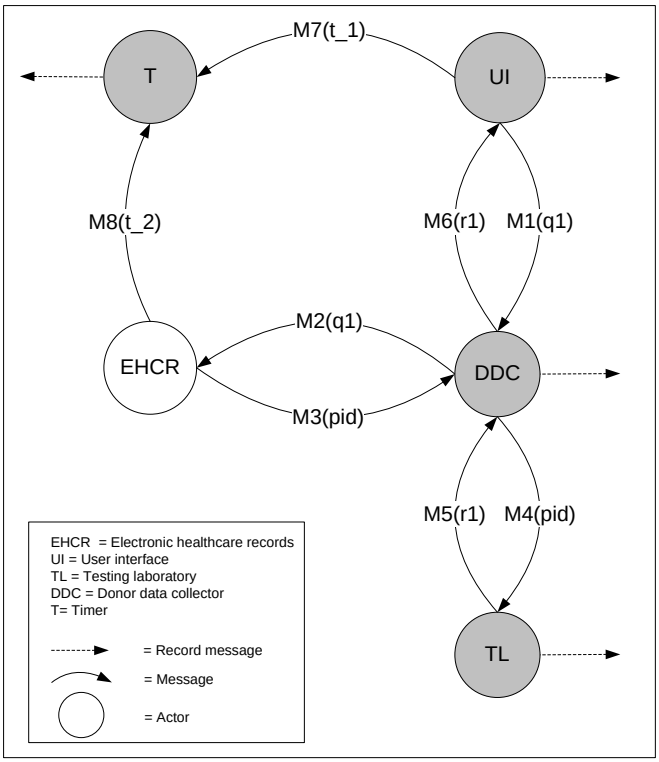

Figure 15: The actors identified for provenance functionality

Once actors have been identified and have had functionality to record process documentation added then one question remains about what to do with such documentation. As mentioned briefly earlier, process documentation is stored in provenance stores. These are repositories for process documentation that can later be queried in order to answer provenance use case questions. Certain management functionality may also be available for these stores such as indexing, deletion and so on. In the OTM example, actors may record all their process documentation into one central repository or each actor may be associated to a separate store. Such configuration considerations must be decided by the application developers to suit requirements on distribution such as the need to provide redundancy, or the need to distribute stores across different systems or subsystems of an application. We address these issues in Section 8.

Examining Figure 15, the kinds of process documentation being recorded can be explored. The figure shows that the donor data collector (DDC) receives a message (M1) from a doctor's user interface (UI), which contains a query about a patients blood (q1). Receiving this message, the DDC sends a message (M2) to the electronic health care records center (EHCR), which contains the query (q1). Both the UI and the DDC document their roles in this process. Thus, the UI sends an *interaction p-assertion* to a provenance store (not shown) stating that it has sent a message (M1) to the DDC

containing a query about a patient's blood (the identity of the patient may need to be kept private and so this part of the information contained in the p-assertion may be transformed in some manner to effect this; see Section 8.1.2 for a discussion on the different ways this might be accomplished. The UI also records information about the user that is currently logged into the interface as an *actor state p-assertion*, to be able to identify who is the user creating the request.

Upon receiving message M1 from the UI, the DDC itself records an interaction p-assertion stating it has received a message from the UI about the patient. The DDC may then record a *relationship p-assertion* stating that the message it sends to the EHCR requesting the patient's identifier was itself *caused* by the receipt of the message from the UI. Upon receiving the reply message from the EHCR (M3) containing the patient identifier (pid), the DDC then sends a request (M4) to the testing laboratory (TL) to perform an analysis of the blood of the patient identified by the patient identifier. 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 patient's identifier. Once this is done, the TL may record interaction p-assertions, relationship p-assertions and actor state p-assertions, which state respectively that it had received a message from the DDC to perform a test, that the message it sends back in response was caused by this message, and finally that the number of such tests it has performed has increased by 1. A further relationship p-assertion may state that this change in state was caused by the receipt of M4.

7 A Provenance-Aware Aerospace Simulation Application

In order to demonstrate the generalised nature of PrIMe, this section presents another application that differs significantly from the OTM example used so far throughout the document to illustrate the use of PrIMe. The application used here as an example is part of the SikMa project run at the German Aerospace Center in Germany. The particular application that forms the focus of the example is an Aerospace Engineering application (AEA), which investigates numerical simulation of various elastic properties of fighter aircraft [KS06].

The application is run as a workflow using the TENT environment to link up various parts of the application. Various simulations can be carried out to test the plane's aerodynamics, flight mechanics and aero-elasticity under numerous different parameter configurations.

Developers of the system realized that by making the application provenance-aware, they could easily re-run previous simulations with different configuration parameters, understand more clearly why a particular simulation run produced the results it did and answer several other interesting questions about the provenance of the data that the application produces. To make the application provenance-aware the develop-

ers went through a similar process described above for the OTM example, using PrIMe to identify provenance use case questions, identifying actors and their interactions and locating the required information items to answer the use case questions.

7.1 The AEA Application

As mentioned, the AEA runs in TENT — an application environment that manages and runs simulation applications. TENT comprises a GUI front end, a container holding and controlling the workflow, several factory components and data and naming servers. The core simulation components are three modules that perform simulations on airplane flight manoeuvres under different environmental and initial setup conditions. Each of the three modules perform different aspects of a simulation where these are:

- Aerodynamics
- Aeroelasticity
- Flight mechanics

To complete a simulation, very large amounts of calculations must be performed and the results of each of the specialist simulation components above feed into the calculations of the others. Many different initial setup and environmental configurations can be tested and the amount of data produced by one simulation is very large.

Making this system provenance-aware helps in the analysis of the results of the application and can inform the configuration of further simulations.

7.2 Provenance Use Case Questions

When developing the provenance-awareness of the AEA application the steps of PrIMe are followed. Initially, therefore, a set of potential provenance use-case questions were identified, as shown below.

- Given some set of simulation results, what has been the simulation case?
- Given some control parameter, in what simulation has it been used?
- What monitoring data has been recorded in a simulation with parameter == X?
- What simulations have been run using a given numeric/model configuration?
- Given a numeric/model configuration, were has it been simulated?
- Given two or more simulations with the same input, what is the result and the difference in provenance?

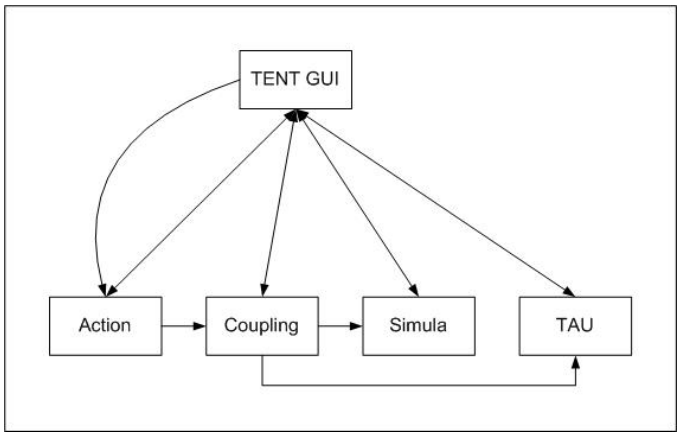

Figure 16: Actor interaction in the AEA Application

7.3 Application Decomposition

It was clear from the developers of the AEA that each of the major components of the application should be identified as actors, the full list is as follows:

- TENT GUI (the only actor accessed by the user).
- Action Component (process control actor for simulation).
- Coupling manager
- Simula (flight mechanics).
- TAU (computational fluid dynamics).

Given this decomposition, the next task is to map out the interactions between the identified actors.

7.4 Interactions between the AEA Actors

Each run of a simulation in the AEA application follows a well define workflow, this allowed the developers to easily map out the interactions between each of the application actors as shown in Figure 16.

7.5 Provenance Store Deployment

After an analysis, the developers of the AEA application determined that only one provenance store was necessary for the application that all identified actors were to record process documentation to.

8 PrIMe Extensions

The PrIMe methodology focuses mainly on the issues surrounding the setting up of applications in order to enable the recording of process documentation that can then later be used to answer provenance queries. Other, less functional issues can also be identified however. In particular, the introduction of provenance capabilities such as the recording of process documentation and querying the same to answer provenance queries poses several questions about security. Similarly, issues arise when provenance capabilities such as the above are incorporated into distributed systems. In this section, these issues are discussed, not as exhaustively as the issues surrounding the recording of process documentation was discussed, since these issues are somewhat less central but, at the same time, no less important in applications that must address these concerns.

8.1 Security

Security is a central concern in most real world applications. Adapting applications to make them provenance-aware brings a number of interesting issues that must be dealt with if security concerns are to be met. In the general case, however, securing a provenance-aware application can follow the same procedures used when securing a database. In effect, the storage of process documentation in a provenance store and their retrieval through queries can be secured using the same kind of access control mechanisms that are used in many back-end database systems.

Aside from the above general considerations, a number of security issues also arise that are unique to the storage and retrieval of process documentation. The first of these concerns controlling the ability of an actor to answer provenance queries using process documentation that it has gathered in answering other queries. The second relates to the storage of restricted data within process documentation. The next section describes the former of these issues and the one after describes the latter.

8.1.1 Reconstructing Process Documentation to Answer Restricted Provenance Queries

A querier, obtaining process documentation for the purpose of answering a provenance query on the behalf of a user, should only do this if the user has access rights to the answer. However, if the querier returns the process documentation to the user for a number of different queries, all of which the user has access to, there is the possibility that the user can recombine process documentation gathered across different queries to answer other queries that it does not have rights to access. This problem is difficult to solve but two approaches can be adopted.

The first approach uses cryptographic protocols that allow access to information within a collection of process documentation only if the querier has the right to access information related to that specific collection. The protocol prevents the querier from

accessing information within individual p-assertions making up the process documentation in order to prevent the querier from accumulating information and combining it in new, unexpected ways to get new types of information.

The second approach uses a front layer that accepts provenance queries and returns results directly rather than in the form of process documentation. Then there can be no worry about queriers accessing collections of process documentation that will allow them to have information. The disadvantage to this approach is that it limits queries over process documentation to a pre-defined, limited set.

The approach used must depend on considerations made by the security manager of the application who must decide which is appropriate given application requirements.

8.1.2 Documentation Styles

When recording documentation about processes, certain pieces of application data will become associated to it that will require access control. There may be, however, restrictions placed on the access to such data. Consequently action must be taken to ensure that information contained within process documentation 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* 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 the access mechanisms and access 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, patient identifiers are a restricted piece of information, and so any process documentation that contains these identifiers must be transformed to protect the privacy of patients. In such cases as this, the *anonymous 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.

8.2 Distributed Provenance-Aware Applications

So far, PrIMe describes a structured process for making applications provenance-aware. This process focuses on identifying how to answer provenance use case questions and enabling applications to capture relevant process documentation so that these questions can be answered. After capture, process documentation must be stored somewhere so that later queries can access this information; this is where the concept of a provenance store becomes relevant. However, in many instances an application may require several provenance stores, and this section presents guidelines for using a set of *recording patterns* that identify several different ways in which process documentation can be collected. Furthermore, it explains how process documentation stored in several different provenance stores can be linked together to aid queries.

In distributed scenarios it is important to ensure that solutions are scalable. Architectural scalability addresses how application components can be organised and used to cater for increasingly large loads in terms of such measures as computation, bandwidth and storage. The patterns presented in the next sections have been developed taking into consideration scalability issues.

8.2.1 Recording Patterns

A pattern [CIS77, Ale79, GRRJ95] describes a solution to a common design problem; the solution described must strike a balance between being concrete enough to be applicable, and abstract enough so that it can be applied to a range of similar problem situations. In the context of PrIMe, patterns provide solutions that developers can use to integrate p-assertion recording into their applications.

8.2.2 SeparateStore Pattern

This first pattern deals with situations where a recording actor cannot store itself the process documentation it generates. Reasons why this might occur are that the quantity of process documentation recorded is too high and a separate dedicated storage location is required, or that the process documentation must remain available after the lifetime of the actor. In such cases, it is necessary to introduce a dedicated provenance store that acts as a repository for process documentation. Figure 17 shows the SeparateStore pattern graphically.

Such a store should be available in a long-term manner in comparison to the application actors that submit process documentation to it. This property allows process documentation recorded by an application actor to be accessed after the application actor has become unavailable.

8.2.3 ContextPassing Pattern

Actors record process documentation about the messages they send to and receive from one another and each actor may store its process documentation in separate provenance

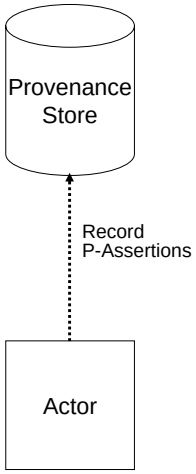

Figure 17: SeparateStore Pattern Diagram

stores. Later, queriers may try to obtain process documentation about an interaction that may be stored in separate stores. In such cases, a difficulty is in matching the documentation stored in one store with documentation stored in others. To overcome this developers can adopt the ContextPassingPattern. In this pattern, when an actor sends a message to another actor, it includes information that provides a *context* to the interaction such as an interaction identifier (to identify a single interaction) or a tracer (to identify interaction sequences, as seen in section 6.2.2). These identifiers and tracers can then be placed in all the process documentation generated by both actors that relates to these interactions. By doing this, queries can find related information across separate provenance stores.

Figure 18 shows the ContextPassing pattern graphically. In the figure are two actors, one a client and one a service, each of which are recording to separate provenance stores. When the client sends a message to the service, it also includes context information. The client records process documentation stating that it sent a message to the service and in the process documentation it includes the context information that it send to the service within its message. Upon receipt of the message the service then extracts the context information and puts this into the process documentation that it records about this information into its own separate provenance store. Later, a querier seeking the process documentation about this interaction must extract it from both provenance stores, and to do this it searches for the information within the context information such as the interaction identifier or the tracer.

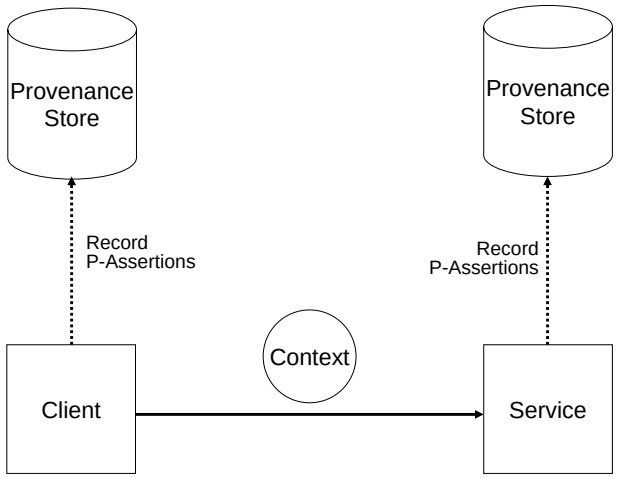

Figure 18: ContextPassing Pattern Diagram

8.2.4 SharedStore Pattern

In many cases, the need for separate provenance stores may not be great and developers should be able to allow actors to record to the same provenance store using the SharedStore pattern. This pattern alleviates the need to traverse separate stores to link up process documentation, which may prove too time consuming for some applications. More than one actor recording to the same provenance store is in fact a combination of both the SeparateStore and ContextPassing patterns. Each actor records to one provenance store as in the SeparateStore pattern and each also records context information so that queriers can link up process documentation about the same interaction recorded by both actors. This context information is still necessary even if both actors are using the same provenance store since the documentation must still be associated.

Figure 19 displays the SharedStore pattern. Each actor records to the same provenance store including in their record messages the necessary context information. SharedStore emphasises that actors can record their p-assertions in any store they choose and provenance stores may hold p-assertions from multiple actors. It does not prescribe how many stores there should be and which provenance stores should be shared. It is left to the developer applying the pattern. SharedStore allows developers to determine the distribution of provenance stores that fits their application.

8.2.5 Combining Patterns

By combining the above patterns in different ways, application developers can tailor the recording of provenance to the needs of the application. Figure 20 shows

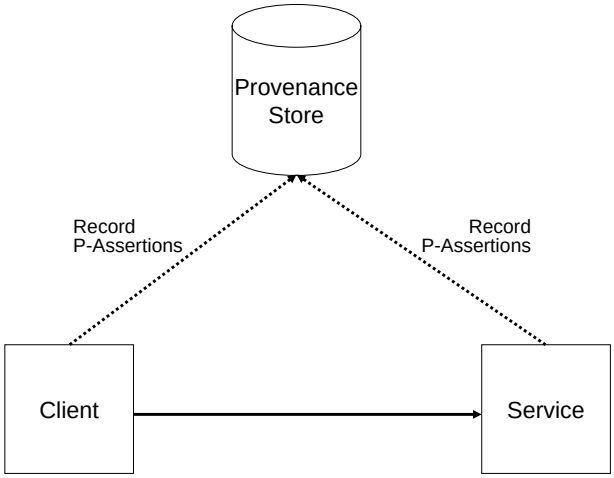

Figure 19: SharedStore Pattern Diagram

an example in which each of the above patters are used and combined to produce a provenance-aware application that can flexibly record process documentation into several provenance stores. In the figure, three actors are shown: a client, a service and a workflow enactment engine. Each of these actors records process documentation. The client and the workflow enactment use the SeparateStore pattern (provenance stores 1 and 2) and the ContextPassing pattern to record into provenance stores. The workflow enactment engine and the service also record into a provenance store using the SharedStore and ContextPassing patterns into provenance store 3. it is clear, from the figure that many other different combinations are possible and the existence of the patterns does not place any restrictions on how they might be combined, since this is dependent upon application requirements and, thus, is left to the designer to make deployment judgements.

9 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. [Ram91] and agent-oriented (AO) methodologies (e.g. [WJK00]. 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

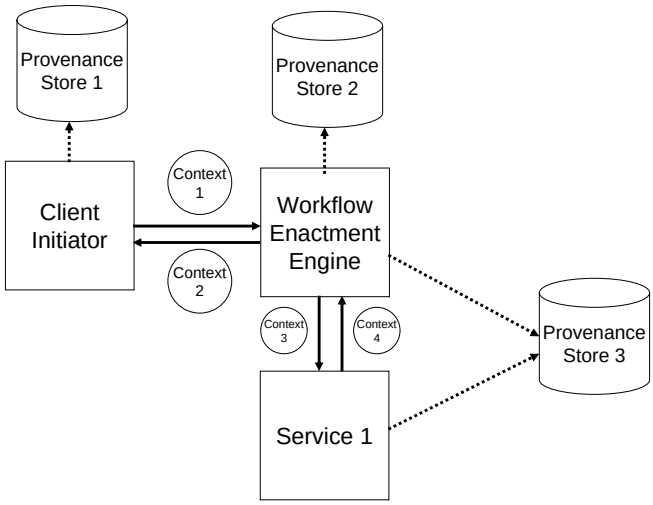

Figure 20: A system in which SeparateStore, ContextPassing and SharedStore have been applied multiple times

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 [WJK00], PrIMe also maps out interactions between identified actors, however, unlike AO approaches, 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 [HW98] modelling language, but specifies these to be specifically provenance-based use cases.

By combining ideas from these related but distinct methodologies and modelling languages, but focusing on the issues surrounding provenance, PrIMe represents a structured way for application developers to modify their applications borrowing, where appropriate, useful and widely used conventions drawn from these other methodological approaches. Such an approach has the advantage that users of the methodology can easily map what they already know from these other existing methodologies onto the process of making applications provenance-aware. This ease of use facilitates the development of provenance aware applications and increases the likelihood of adoption.

10 Conclusion

This document describes the PrIMe methodology. It describes in detail 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 kinds of information 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.

PrIMe also discusses several issues that relate to security and distribution of provenance systems, providing guidance for securing access to retrieved process documentation and ways to transform such documentation to meet specific security concerns. In terms of distribution, PrIMe offers several recording patterns that allow application developers to distribute the recording of process documentation across disparate 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 designers and users of applications helps to ensure that the effort required to make applications provenance aware is minimised.

References

[Ale79] C. Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[AVSK+06] S. Álvarez, J. Vázquez-Salceda, T. Kifor, L.Z. Varga, and S. Willmott. Applying provenance in distributed organ transplant management. In *LNCS: Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, Chicago, Illinois, May 2006. Springer-Verlag.

[CIS77] C.Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language*. Oxford University Press, 1977.

[GMM06] P. Groth, S. Miles, and S. Munroe. Principles of high quality documentation for provenance: A philosophical discussion. In *In Proceedings of Third International Provenance and Annotation Workshop*, 2006. in press.

[GRRJ95] E. Gamma, R.Helm, R.Johnson, and J.Vlissides. *Design Patterns*. Addison-Wesley Professional, 1995.

[GSC+06] A. Gibson, R. Stevens, R. Cooke, S. Brostoff, and m. c. schraefel. myTea: Connecting the web to digital science on the desktop. Edinburgh, 2006. World Wide Web Conference. submitted.

[HW98] P. Harman and M. Watson. *Understanding UML, The Developers Guide*. Morgan Kaufmann, 1998.

[JGM+06] 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.

[KS06] G.K. Kloss and A. Schreiber. Provenance implementation in a scientific simulation environment. In *LNCS: Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, Chicago, Illinois, May 2006. Springer-Verlag.

[Ram91] J. Rambaugh. *Object oriented Modeling and Design*. Prentice Hall, 1991.

[WJK00] 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.