Title:	*Pre-Prototype Design*
Author:	*Fenglian Xu, Alexis Biller, Liming Chen, Victor Tan, Paul Groth, Simon Miles, John Ibbotson and Luc Moreau*
Reviewers:	*All partners*
Identifier:	*D9.1.1*
Type:	*Deliverable*
Version:	*1.0*
Date:	*24th February 2005*
Status:	*Confidential*

Summary

This document contains the pre-prototype design using existing software components

Members of the PROVENANCE consortium:

IBM United Kingdom Limited	United Kingdom
University of Southampton	United Kingdom
University of Wales, Cardiff	United Kingdom
Deutsches Zentrum fur Luft- und Raumfahrt s.V.	Germany
Universitat Politecnica de Catalunya	Spain
Magyar Tudomanyos Akademia Szamitastechnikai es Automatizalasi Kutato Intezet	Hungary

Foreword

In the first phase of the EU provenance project, it is necessary to clearly define the concept of provenance within the context of the project, as this will provide the driving impetus for the subsequent design and implementation of a provenance architecture. Towards this end, we have chosen to design and develop a simple system, which we term the pre-prototype, that will effectively articulate the project's conception of provenance. We believe this approach presents a more effective method of clarifying the concept of provenance in the project as opposed to producing a purely theoretical, universal definition of provenance that will inevitably be abstract or vague. In addition, the pre-prototype also serves to explore some of the design and implementation issues of the provenance architecture that the project aims to develop within a service-oriented grid computing environment.

We have chosen to base the pre-prototype on a conceptualized process for baking a specific type of cake; cake baking being a widely understood and intuitive process and thus well suited for the aims of the pre-prototype. The pre-prototype, when situated in the context of a service-oriented computing paradigm, provides a simple but comprehensive scenario for identifying the key characteristics and requirements of a provenance architecture.

This document is organised as follows: Section 1 introduces the scenario for the cake baking process based on a publicly available recipe. Section 2 presents some questions that cake makers might ask in order to produce cakes that conform to some standard, for example, the quality of taste. From the discussions of these questions, we obtain a conceptual understanding of provenance. In section 3, we analyse the cake baking application in detail by framing it within a service-oriented viewpoint, in which entities or participants provide services to each other via interactions. Section 3 also defines and discusses provenance in the context of a service-oriented architecture (SOA). We describe data modelling and the service interface design for the cake baking application in sections 4 and 5. Following this, we provide the implementation details of the pre-prototype in section 6. Section 7 illustrates the use of provenance data through a suitable query mechanism. We finally conclude the document in section 8 with a discussion and initial conclusions drawn from the pre-prototype.

Table of Contents

Table of illustrations

.

1. The pre-prototype scenario

The pre-prototype scenario is based on the process of baking a cake, more specifically, the Victoria Sponge Cake. As is the case for baking a new type of cake, a common preliminary is obtaining the recipe from a reliable source. In our scenario, the source is an online bakery website [1]. For the pre-prototype, we adopt a conceptualized process of baking the cake (which we shall term as the Baking Victoria Sponge Cakes, or BVSC, process) that is based on an abstraction of this recipe. This abstract or idealized recipe can be considered to be composed of four distinct steps or activities that correlate roughly with the original recipe; these are outlined below:

Step 1: Whisk together a certain amount of butter and sugar (in proportion) until light and creamy

Step 2: Beat the required eggs for certain duration and add it to the whisked sugar and butter mixture

Step 3: Fold some flour into the beaten mixture and add the flavouring preferred (vanilla, lemon)

Step 4: Put the folded wet dough into an oven and bake it for a given time at a specified temperature

Each activity requires a number of ingredients as inputs and produces an intermediate or final output. As an example, the first activity corresponding to step 1 consumes a specified amount of butter and sugar as inputs and produces a mixture as an output after undergoing the action of whisking.

To provide a structure for interactions within this pre-prototype scenario, we introduce three additional roles that are involved in execution of the BVSC process. A baking assistant is responsible for performing the listed activities; with each activity corresponding to a single assistant. A baker oversees the entire process by sending appropriate instructions to each assistant (which would consist of the appropriate input ingredients), and receiving corresponding outputs from the respective coordinators. We assume that the baker is aware of the idealized recipe and will coordinate the overall cake baking process in accordance with it. Finally, a user initiates the process by providing the baker with the various quantities and amounts required in the idealized recipe for the Victoria Sponge cake. These would include the required amount for each of these ingredients as well as the duration for baking and beating the eggs. These amounts and durations are determined by the user and can be different for each baking process attempt that produces a cake; this in turn has consequences for the motivating questions that we discuss in the next section.

In the simplest case, a single entity can assume all three roles, but for purposes of discussion later, we maintain the distinction of these separate roles. We also assume that the user possesses some a priori knowledge about the elements in the baking process (e.g.:- the duration for beating the eggs) that will directly influence the quality of the produced cake.

2. Motivating questions

Although baking cakes in accordance with a standard recipe appears to be a routine task for many people, it is quite common that cakes produced by different users are likely to be different from each other. It is equally likely that cakes baked by a single user in accordance to the same recipe are likely to be different for each baking attempt. These differences can be reflected in a multitude of ways that pertain to the quality of the cakes in question, for example, the colour (yellow, brown or black), the shape, the taste (too salty, too sweet or tasteless), the texture and so on.

As such, it is expected that a user would require several attempts from the initial one in order to produce a cake that conforms to some idealized quality. These attempts constitute a form of iteration in which the user would try to understand how the deviation from the idealized quality in a produced cake is caused, and subsequently apply this understanding to refine the baking process in the subsequent attempt.

To facilitate our discussion, we assume that a user has initiated a BVSC process that produces a final cake that does not taste as delicious as expected or as indicated in the recipe. In order to understand

how such a situation might have transpired, the user would resort to trying to answer certain types of questions pertaining to the relevant factors in the BVSC process that the user understands to be directly related to the cake taste. A sample list of such general motivating questions are given below :

1) *Did the baker follow the user's instructions correctly?* The user supplies the various ingredient amounts and duration times required in the idealized recipe to the baker, who oversees the BVSC by relaying these parameters onwards to the respective baking assistants. It is useful to ascertain that these parameters received by the assistants from the baker correspond to those submitted by the user to the baker, in accordance to the individual activities of the idealized recipe.

2) *Were the correct ingredients used at each step of the activity?* The baker submits a set of input ingredients to the respective baking assistants, and it needs to be ascertained whether these were the actual ingredients utilized by the assistants in performing the activity concerned. Checks may also need to be performed to ensure that the right order of adding ingredients is adhered to and that the correct quantity and unit of ingredient is used. This is particularly relevant when the user does not strictly follow the amounts recommended in the idealized recipe.

3) *Was the correct sugar amount as specified in the recipe used?* The amount of sugar used in the whisking activity is one of the factors that may affect a cake's quality of taste; too little sugar could produce a tasteless cake. There will generally be a guideline on the minimum amount of sugar to be used in order to attain a minimum taste quality; this could be specified in the original recipe or be part of the a priori knowledge that the user possesses.

4) *Was the correct oven temperature as specified in the recipe used?* An excessively high temperature can result in a burnt cake, while an excessively low temperature can affect the texture of sponges; both of which would result in a cake tasting bad. However, there is usually some permissible range for an appropriate baking temperature. Deviations within this range would therefore be unlikely to affect the taste of a cake. Ovens may have different specifications and measures for temperature (ex:- Celsius/Farenheit). There are ovens that use gas marks for setting and indicating temperatures. There is therefore a need to check both the magnitude and unit of the temperature specified in the recipe.

5) *Was the correct amount of flour used for the oven baking activity at a given location?* Ovens may be located in different locations with different altitudes; the altitude in turn influences the amount of flour required to produce a baked cake that tastes good. The higher an oven's altitude, the greater amount of flour is required.

6) *Which activity has the longest duration during the BVSC process?* It is often of interest to identify the most time consuming activity in the creation of a cake. The time spent in an activity can be worked out by measuring the starting and ending time of the activity.

7) *What is the amount of ingredients used the last time a better quality cake was produced?* When a user produces a cake that is of a comparably lesser quality than a previous attempt, it might be useful to identify the difference in the quantity of ingredients used in the two attempts. It is a common practice to use successful results from the past to provide feedback on refining future attempts.

8) *Was the proportion of ingredients used right for the intended size of the cake?* If a user has made a cake that is proportional to the cake size specified in the idealized recipe, special attention should be paid to check that the ingredients submitted to the baker are used in equivalent proportional amounts as well.

To answer the above questions, a user would need to record down the various details in the BVSC process; this might include the original recipe, idealized recipe, actions performed by the baker and baking assistants as well as the specific ingredients, amounts, proportions, duration and temperatures used. In practice however, most users would not bother to record down these details during the cake baking process. Even when there are attempts to note down key information, these attempts are likely to be haphazard and error-prone, making their systematic storage and retrieval very difficult. It is usually the case that the occasion for making use of such information only arises when a cake that deviates from some measure of quality (like taste) is produced. What is then needed is the full documentation of the BVSC process involved in the production of a cake by a user. This essentially constitutes the concept of provenance in our scenario.

Given that the provenance of cake baking needs to be recorded and that no users would like to do this (in particular, manually), the central question is then how to achieve this recording functionality effectively. A possible solution would be to introduce an additional entity into the scenario who works in the background recording every detail of every action undertaken by the user, baker and baking assistants in a log book. The log book would be available to the user at different points during the baking process, and would provide the necessary information to answer the types of questions that we have just discussed. In essence, we require a provenance infrastructure that can capture and record provenance data in a user-transparent manner and reason over this data in order to answer relevant questions posed by the user.

In the sections that follow, we investigate in greater depth the issues that might arise in the conception and design of such a provenance infrastructure, as well as the technologies that are involved in the context of our BVSC process.

3. BVSC application analysis

This section analyses the BVSC process, its entities, the interactions and information flow among these entities from a service-oriented perspective. We first expand on the view of the BVSC process in a service-oriented architecture in which entities provide services to each other through interactions. Then we use a sequence diagram to delineate the interaction and information flow of the BVSC process. Following this, we introduce the concept of interaction and actor provenance in the context of BVSC and the use of provenance stores. A sequence diagram is provided to demonstrate how provenance is added and traced with the involvement of a provenance store.

3.1 A service-oriented view

A service-oriented view is a way of viewing large, complex systems in terms of services they provide. Here a service can be simply viewed as an abstract characterisation and encapsulation of some content or processing capabilities. A corollary to the service-oriented view is the service-oriented architecture (SOA), in which services are the primitive building blocks. In a SOA, problem solving processes amount to the discovery, aggregation and execution of a sequence of loosely coupled services.

In the context of the pre-prototype scenario, an activity is in essence the behaviour of a service or an operation provided by a service. Thus each of the four activities in the idealized BVSC recipe can be viewed as a service. In addition, the user and the baker are also services by the virtue of the activities they perform respectively in the BVSC process. The term actor is used to refer to a system that provides a specific function, or individuals that carry out activities with supplied inputs to produce outputs. In the SOA, an actor refers to the concrete code that conducts an activity described by a service. For our case, all services are correspondingly actors as well.

Most SOAs have a primary functional characteristic of executing workflows; a workflow being a process by which a series of services are executed in a specific sequence, including the specification of how outputs of services are routed to the services of other tasks. There is usually a component in the SOA known as the service enactment engine which undertakes the action of executing a workflow. In

the context of the pre-prototype scenario, the BVSC process corresponds to a workflow run that produces a cake, with the baker assuming the role of a service enactment engine that executes the workflow specification (idealized recipe) provided by the user.

Figure 1 depicts all the actors/services in the BVSC process.

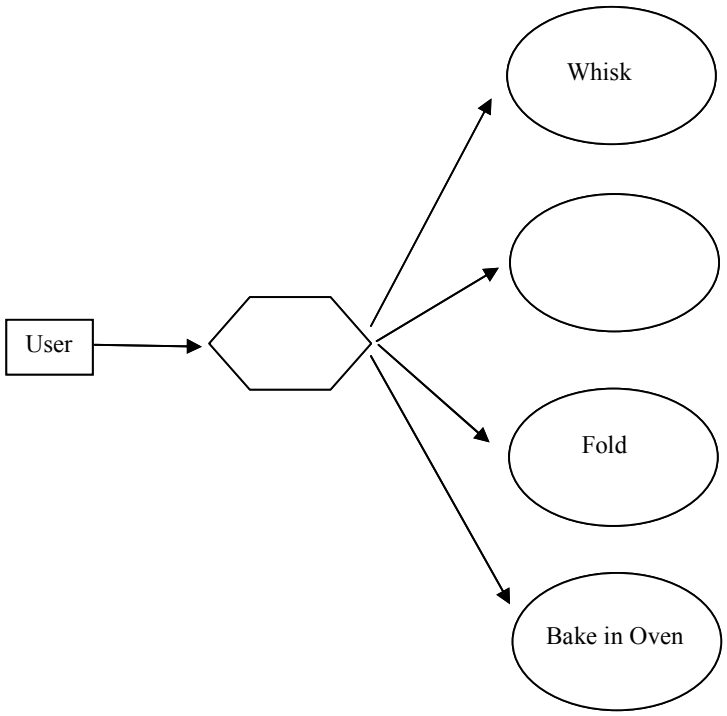

Figure 1 Services/Actors in the BVSC process

3.2 BVSC process sequence diagram

A sequence diagram provides a visual representation of a process. It depicts all services contained, all events taking place and all interactions between services in the process in a temporal order. The advantage of using a sequence diagram is that it can expose vividly all interactions and the participants of the interactions as a process unfolds. An RPC-like interaction consists of two messages: an invocation message and a result message. The invocation message is defined by an operation name and parameters carried by the operation. The result message is defined by a return name and the results returned by the service.

Figure 2 shows the sequence diagram of the BVSC process based on the service-oriented view described in the preceding section. In this diagram, services are arranged horizontally from left to right in the order of invocation. Interactions are arranged vertically from top to bottom as the process proceeds from the start to the end. The services are shown as rectangles containing their respective names. The user interface corresponds to the user role of the BVSC and is also a service as well, although it is considered as a client in context of its interaction with the baker service. The sequence of invoking messages is represented as solid arrow lines and the return messages are represented as dashed arrow lines. The input parameters and the name of the messages are shown above the lines and separated by a colon. Sugar, flour, butter, eggs, and temperature are all defined by some quantity and a unit. The duration is an elapsed time.

As can been seen from the diagram, the user interface initiates the BVSC process by invocating the Baker service with some control parameters (i.e. sugar, duration, flour and temperature). The Baker service then interacts with Whisk service, Beat service, Fold service and Oven service respectively.

During each interaction, the Baker service passes an invocation message to the invoked service and the invoked service returns a result message to the Baker service. At the end of the process, the Oven service returns a cake to the Baker service that in turn returns the cake to the User Interface. Both invocation and result messages contain concrete input/output information. For the purposes of this application, we do not seek to detail the results returned by message, which we consider to be ingredients without specifying an amount and a unit.

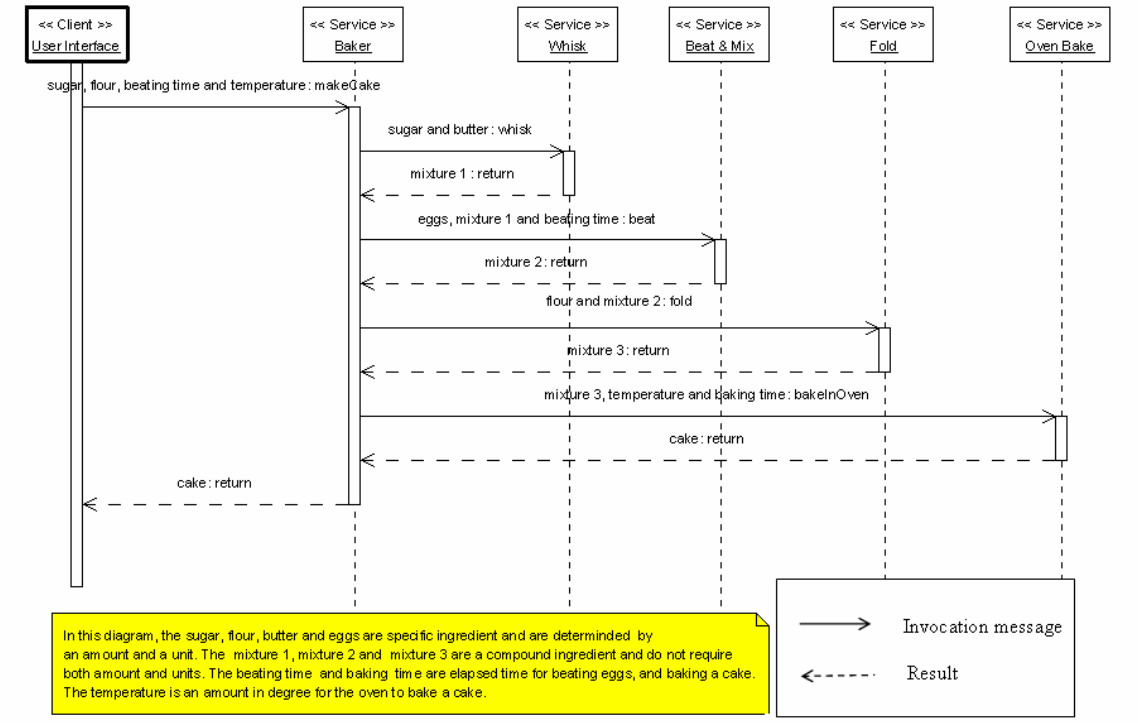

Figure 2 The BVSC process sequence diagram

3.3 Provenance, provenance recording and provenance stores

We have argued that provenance of a data item, i.e. the documentation of a process that led to a data item , is required to answer the questions detailed in section 2. We have analysed the BVSC process from a service-oriented view and discussed the service-based BVSC process in section 3.1. In this section, we seek to examine the concept of provenance within the context of an SOA.

In essence, a SOA can be broken down into two types of actors (entities): clients who invoke services and services that receive invocations and return results. Given these types of actors and their method of communication, we have identified two kinds of provenance that exist in a service-oriented architecture [4]. The first kind of provenance is interaction provenance. For some data, interaction provenance is the documentation of interactions between actors that led to the data. In a SOA, interactions are fundamentally, a client invoking a service. Therefore, interaction provenance can be obtained by recording the inputs and outputs of the various services involved in generating a result. The second type of provenance we have identified is actor provenance. For some data, actor provenance is the documentation that can only be provided by a particular actor pertaining to the process that led to the data.

In addition, an appropriate storage for the documentation accumulated in interaction and actor provenance needs to be provided. This can assume the form of an additional service within the SOA whose primary activity in the archival of provenance generated from other services. We term this service the provenance store.

The structure of the provenance store is currently designed in the way as shown in Figure 3. The URL of the host where the provenance store is located is defined as the root node of the provenance store. Under this top-level node, there will be multiple sessions. Each session refers to a workflow run with a unique identifier (ID). Each session contains at least one activity with a unique activity ID. An activity describes an interaction between a client and a service. An interaction is further split into an invocation message and a result message. Both messages are stored by both client and service sides. Apart from interaction messages, the actors involved in the interaction will also be recorded in the provenance store. While Figure 3 presents an abstract outline of the provenance data structure, there could be many different implementations.

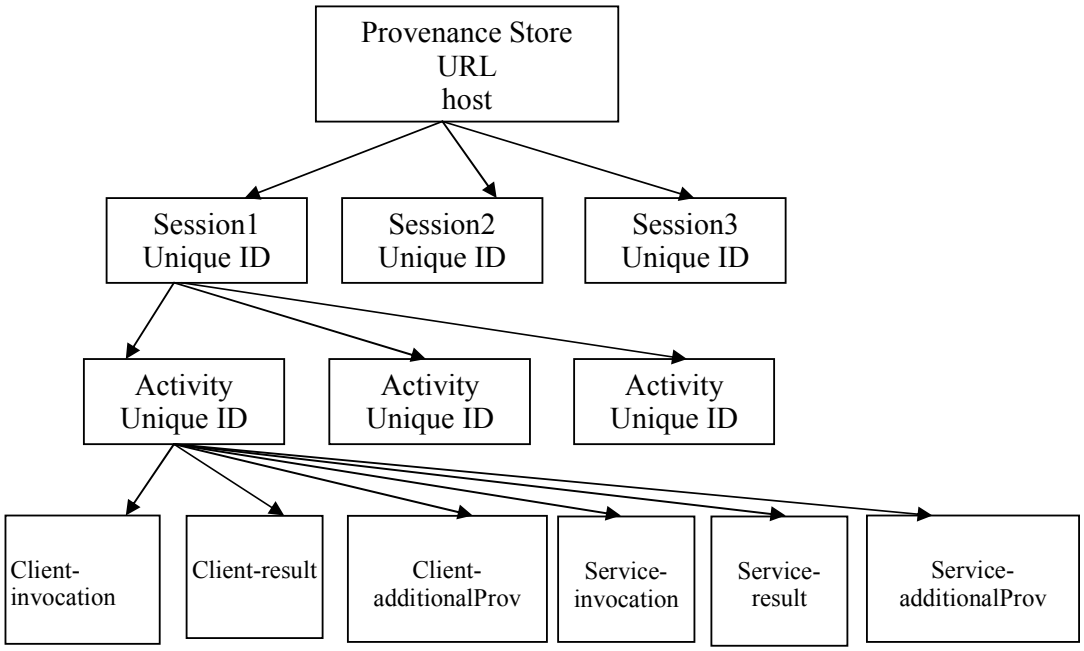

Figure 3 The structure of provenance store

Let us take a closer look at the interaction provenance in the service-based BVSC scenario. We handle interaction provenance in the following way. For each interaction between a client and service consisting of an invocation and a result, each party is required to submit their view of the interaction to a common provenance store. Even though the BVSC process considers multiple actors, the interaction between all these actors can be reduced down to a common triangular pattern of interaction, i.e. the client, service and provenance store.

The BVSC process sequence diagram in Figure 4 shows all of interaction provenance in the process and the provenance data recording mechanism. As can been seen, an extra provenance service is added to the far right end. This service is provided by the provenance store actor and is responsible for recording interaction provenance from both client and service sides. For example, a message *makeCake(sugar, flour, beating time, temperature):record* is sent to the provenance store for recording by both the User Interface and the Baker service. The return message is recorded in the same way.

We have added actor provenance into the BVSC process sequence diagram as shown in Figure 5. As discussed before, a service is provided by an actor. This means there will be a user interface, baker, whisk, beat & mix, fold, oven bake and provenance store services in the sequence diagram. An interaction is actually conducted by actors. For example, the makeCake interaction is initiated by the user interface actor and responded to in return by the baker actor.

Actor provenance is aimed at recording information for a participating actor of an interaction. It usually consists of such information such as the time of sending, receiving a request or a response message and the property of actors themselves. As an example, the oven bake actor contains an altitude to indicate its location. Other provenance information may include the brand, date of manufacture, etc.

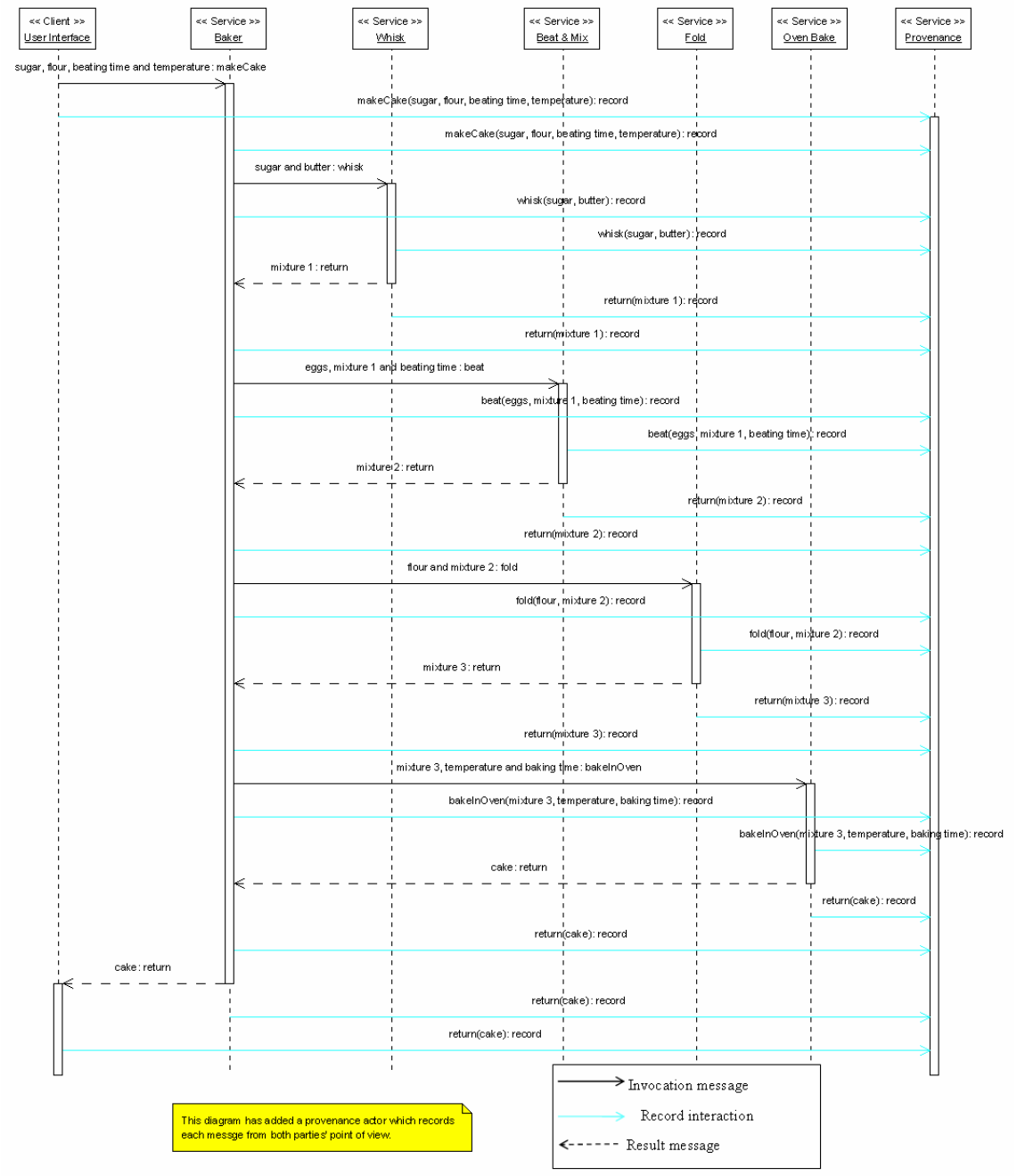

Figure 4 The BVSC process sequence diagram with interaction provenance

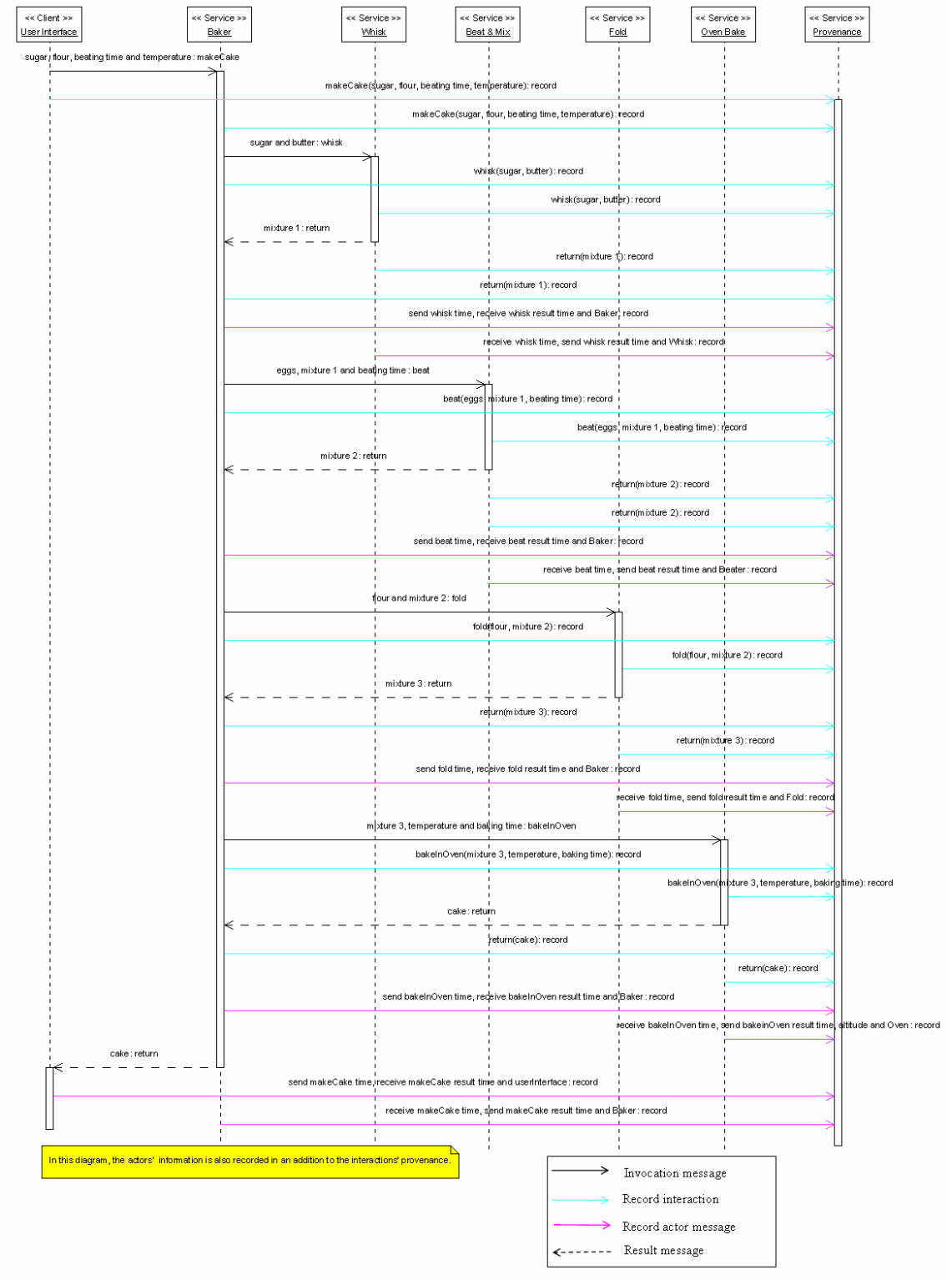

Figure 5 The BVSC process sequence diagram with interaction and actor provenance

4. Data modeling in the BVSC application

In a SOA, an interaction means an invocation between a client and a service. Interaction provenance data is, in essence, the inputs and outputs of the services involved in the interaction. From a Web Service perspective, a service's inputs and outputs are encoded in the messages to and from the service. A message represents an abstract, typed definition of the data being communicated. It consists of logical parts, each of which is associated with a definition within some type system. Therefore, data modelling is necessary to define the data types and messages used by all services in the BVSC application.

In line with the common practice in Web Service design, we have used an XML schema to model all data types and messages for the interactions and actors in the service-based BVSC application. We have identified the data types and messages used in the BVSC application through the process analysis in section 3, in particular the sequence diagrams. The XML schema for the data and message models of the BVSC application can be found at [7].

It is often easier to understand complex XML vocabulary definitions when the XML schema-represented models are expressed graphically. Tools for assisting editing XML schemas have been available, but they are generally limited to a strict hierarchical view of the vocabulary structure. More recently, complex data structures have been modeled as a combination of interrelated schemas. This component-oriented data modeling approach is more expressive and straightforward if represented graphically, for example, using UML class diagrams [3].

The Unified Modeling Language (UML) is the latest evolution of graphical notations for object-oriented analysis and design. UML includes the necessary metamodel that defines the modeling language itself. For more information on these topics, refer to the Web portal at XMLModeling.com.

4.1 Data type modeling

This section briefly discusses data type models in the BVSC application. There are several data types defined in the baking schema. **Units, NonNegativeDecimal** and **Temperature** are used for measurements. **BeatTime** and **BakeTime** are used to represent time duration, while **Cake, Ingredient** and **Mixture are** complex types that can accommodate additional attribute information. **Sugar, Butter, Flour and Egg** are all extended from the **Ingredient** type, and are used to denote a specific type of ingredient.

The relationships among data types and their attributes are depicted graphically in the UML model in Fig. 6. The stereotypes <<xs:complexType>>, <<xs:simpleType>> and <<xs:enumeration>> are standard XML schema types and represent constraints for an object or a class.

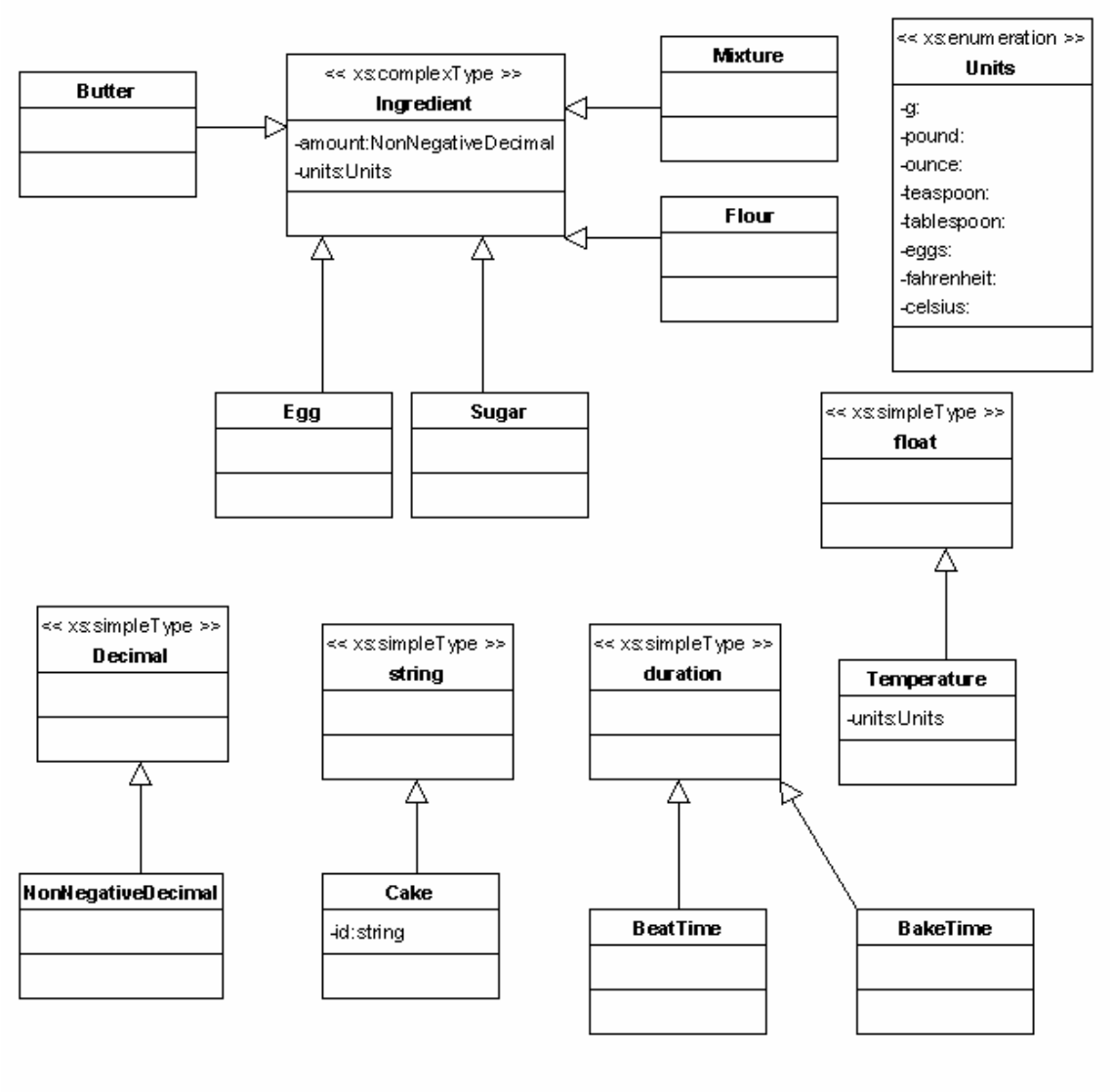

Figure 6 The UML model of the BVSC data types

4.2 Message modeling

Messages define all invocation and result data exchanged during interactions among services in the BVSC application. The message models in the BVSC application are represented in UML in Fig. 7 – 11 for the messages **MakeCake, Whisk, BeatMix, Fold** and **BakeInOven**. The stereotype <<xs:complexType>> represents a standard XML schema type.

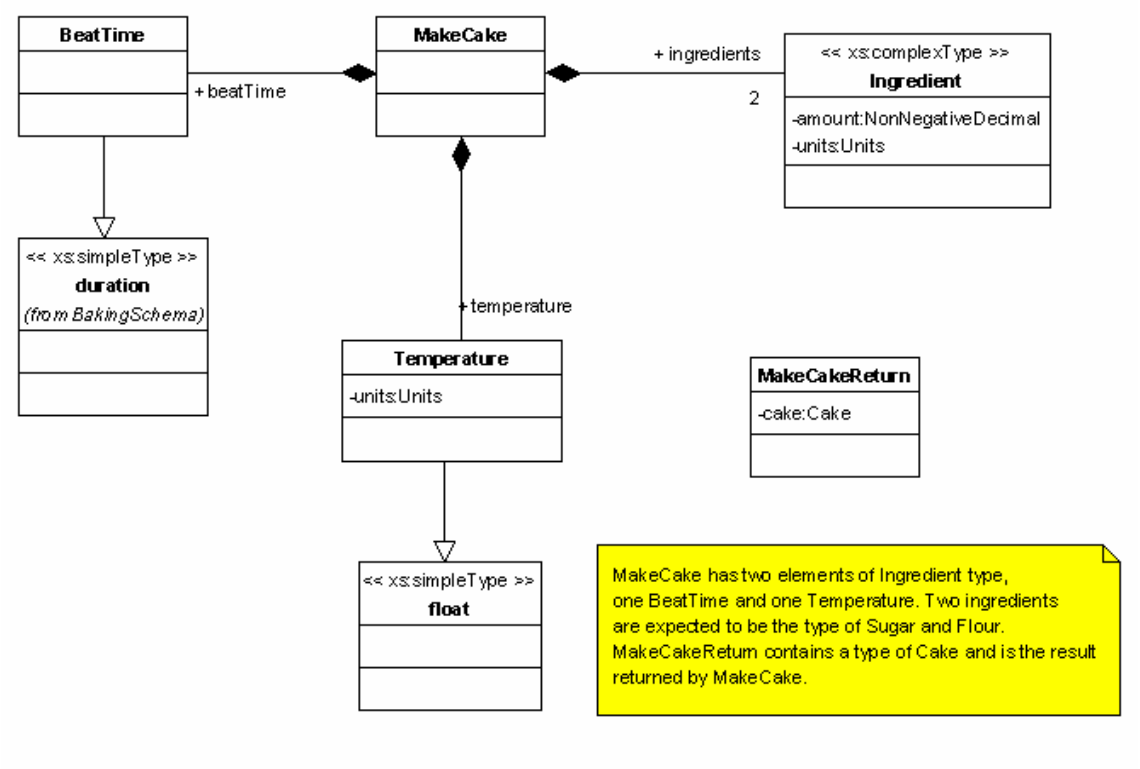

Figure 7 The UML model of the MakeCake messages

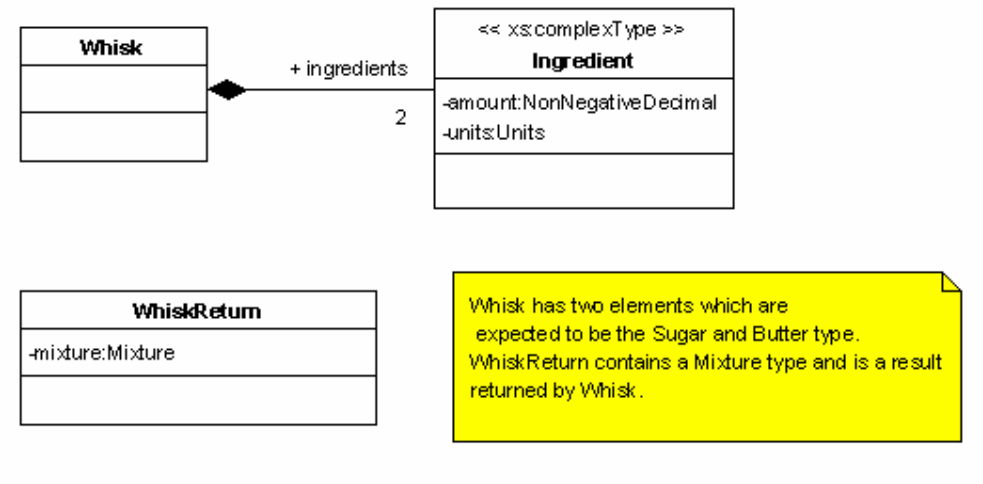

Figure 8 The UML model of the Whisk messages

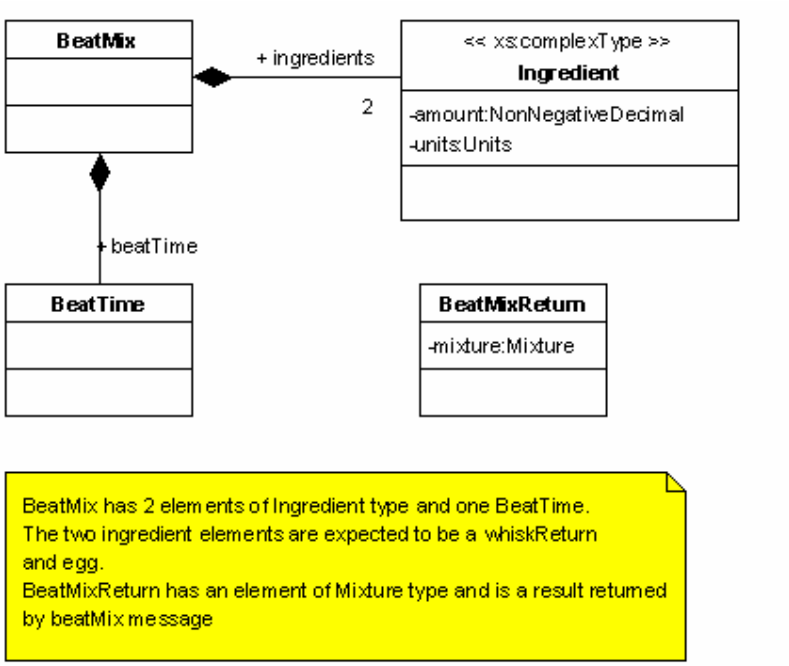

Figure 9 The UML model of the BeatMix message

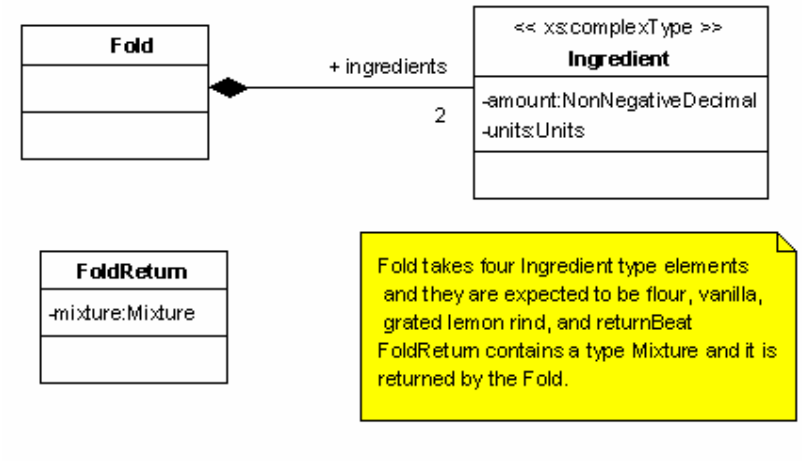

Figure 10 The UML Model of the Fold message

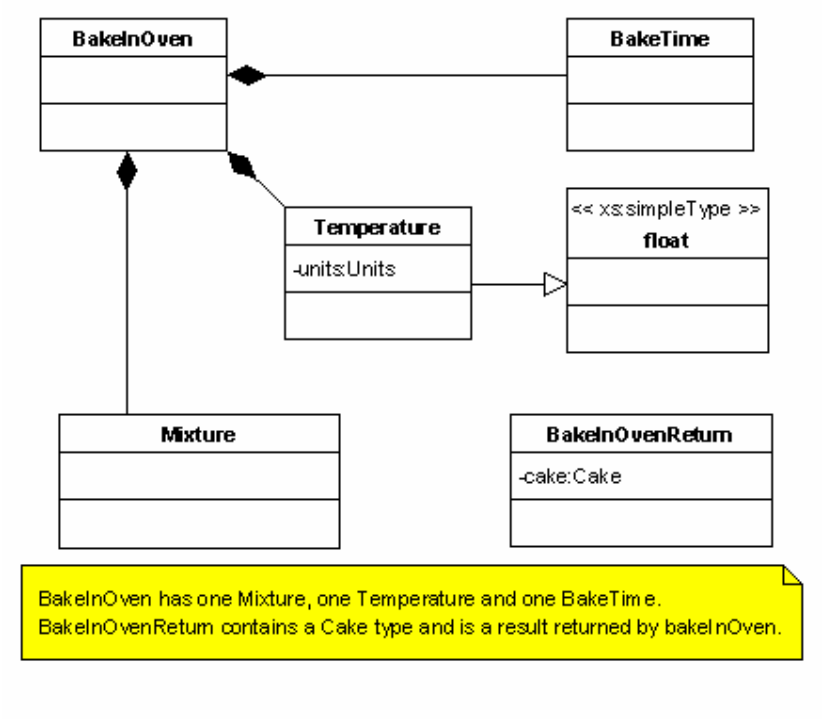

Figure 11 The UML model of the BakeInOven message

4.3 Inheritance and SubstitutionGroup in XML Schema

W3C XML Schema provides a mechanism, called substitutionGroup, that allows elements to be substituted for other elements. More specifically, elements can be assigned to a special group of elements that are specified to be substitutable for a particular named element called the head element. Both the head element and substitutable elements must be declared as global elements. To illustrate this, we declare `sugar`, `butter`, `eggs`, `flour` and `mixture` elements and assign them to a substitutionGroup whose head element is `ingredient`. With this declaration, the `sugar`, `butter`, `egg`, `flour` and `mixture` elements can then be used wherever the `ingredient` element `can be used`. Elements in a substitutionGroup must have the same type as the head element. Alternatively they can have a type that has been derived from the head element's type. To declare these derived elements, and to make them substitutable for the `ingredient` element, the following syntax is used in the schema file:

```
<element name="butter"  type="ns:Butter"  substitutionGroup="ns:ingredient"/>
<element name="flour"   type="ns:Flour"   substitutionGroup="ns:ingredient"/>
<element name="sugar"   type="ns:Sugar"   substitutionGroup="ns:ingredient"/>
<element name="egg"     type="ns:Egg"     substitutionGroup="ns:ingredient"/>
<element name="mixture" type="ns:Mixture" substitutionGroup="ns:ingredient"/>
```

With the above declarations, the ingredient in the following declaration can be substituted by the element `sugar` and `butter` in an instance of the element `whisk`.

```
<element name="whisk" type="ns:Whisk"/>
<complexType name="Whisk">
  <sequence>
    <element ref="ns:ingredient" minOccurs="2" maxOccurs="2"/>
  </sequence>
</complexType>
```

For example, an instance of the element `whisk` would look like something as follows:

```
<ns:whisk>
  <ns:sugar  amount="100" units="g"/>
  <ns:butter amount="100" units="g" salted="true"/>
</ns:whisk>
```

Note that when an instance document contains element substitutions whose types are derived from those of their head elements, it is not necessary to identify the derived types using the xsi:type construction. Further details can be found in using derived types in instance documents [4]

4.4 Actor provenance data modeling

Actor provenance data modeling seeks to define data types and data structures used in actor provenance in the BVSC application. The UML models for the actors are shown in Figure 12.

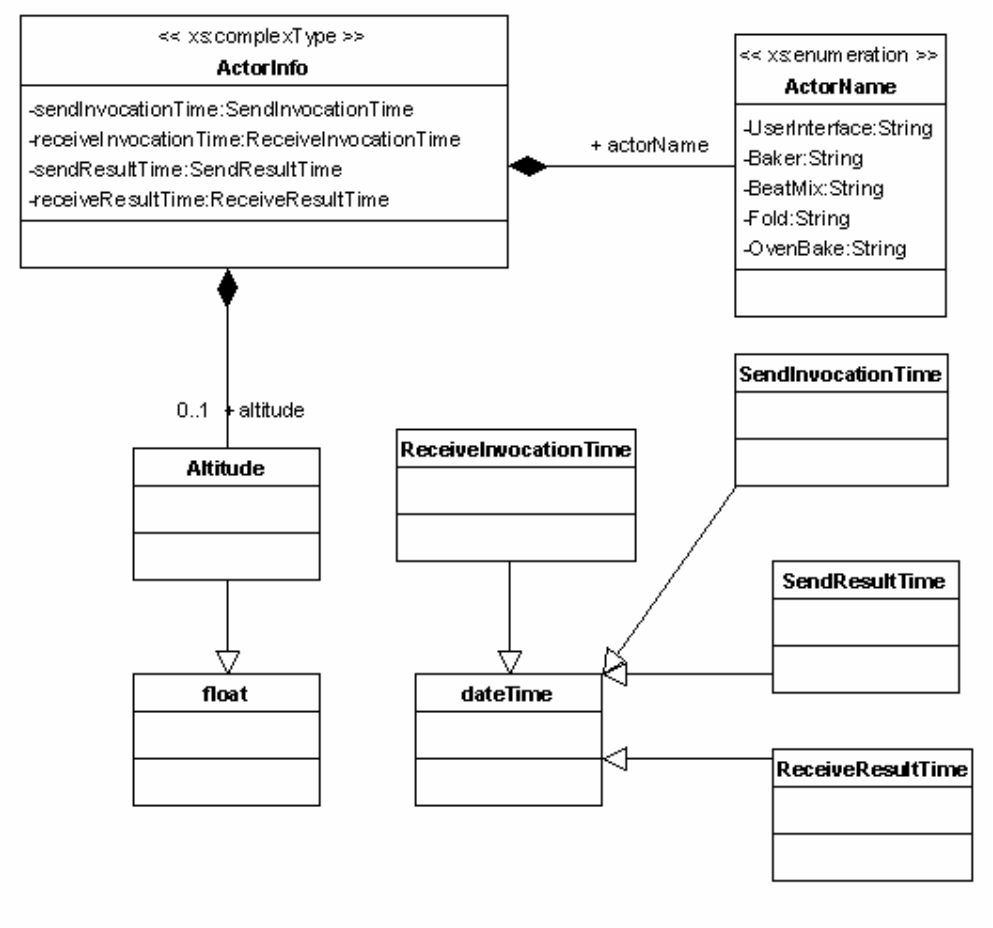

Figure 12 The UML model of actor provenance data

5. Interface design for BVSC services

The service-oriented analysis of the BVSC process in section 3.1 has identified that five services, i.e. Baker, Whisk, Beat & Mix, Fold and Oven Bake, are required in order to implement the BVSC application in a SOA. Section 4 has defined data types and messages used in all interactions among these services. This section will describe these services' interfaces using WSDL and represent them graphically using UML.

A Web Service is a software system designed to support interoperable machine-to-machine interactions over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

A WSDL document defines services as collections of network endpoints, or ports. A port is defined by associating a network address with a reusable binding. A reusable binding is composed of a concrete protocol and data format specifications for a particular port type. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. Operations can be defined either in the local WSDL file or in a separate XML schema file. Separating an operation definition from the WSDL in which it will be referenced would bring flexibility to the implementation of Web Services and operation design. We adopt this approach to define operations in this pre-prototype implementation. As all WSDLs of these services have similar structures, we only show the WSDL of the Baker service as an example [7].

The following figures show the UML models of the BVSC services. Each UML model illustrates the WSDL description of a service, which includes the supporting Web Service XML Schema (WXS) types, the service implementations and the nature of client interactions with the services. The UML model also presents information on where a data type or a message is defined. The WXS is an XML message with its format determined by a schema.

The UML model of the Baker service is shown in Figure 13.

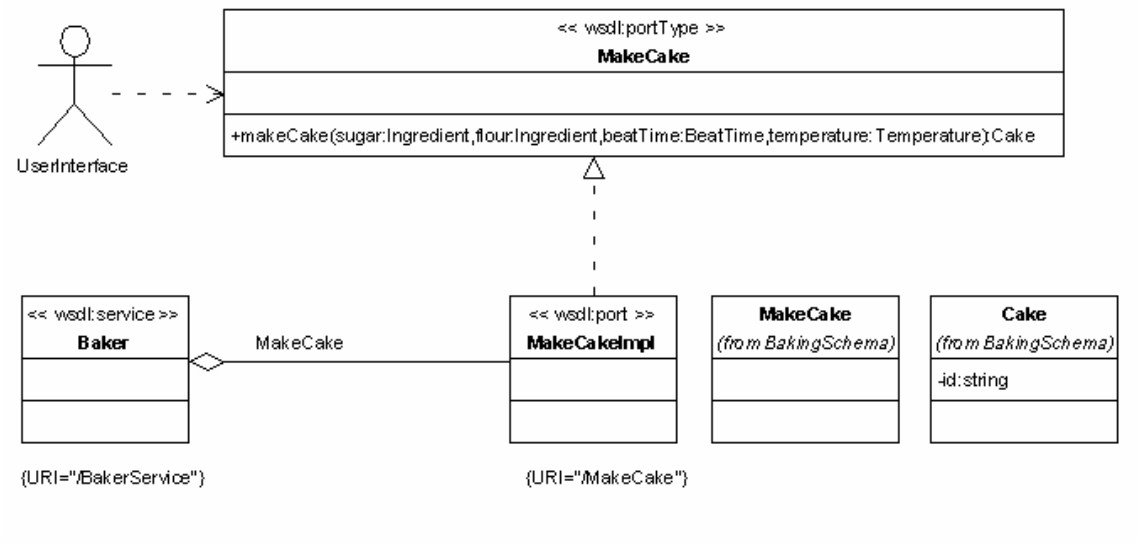

Figure 13 The UML model of the Baker service

The UML model of the Whisk Service is shown in Figure 14. As can be seen in the figure, the Baker service acts as a client. It interacts with the Whisk service through the service's portType, which is realised by the WhiskImpl port of the Whisk service.

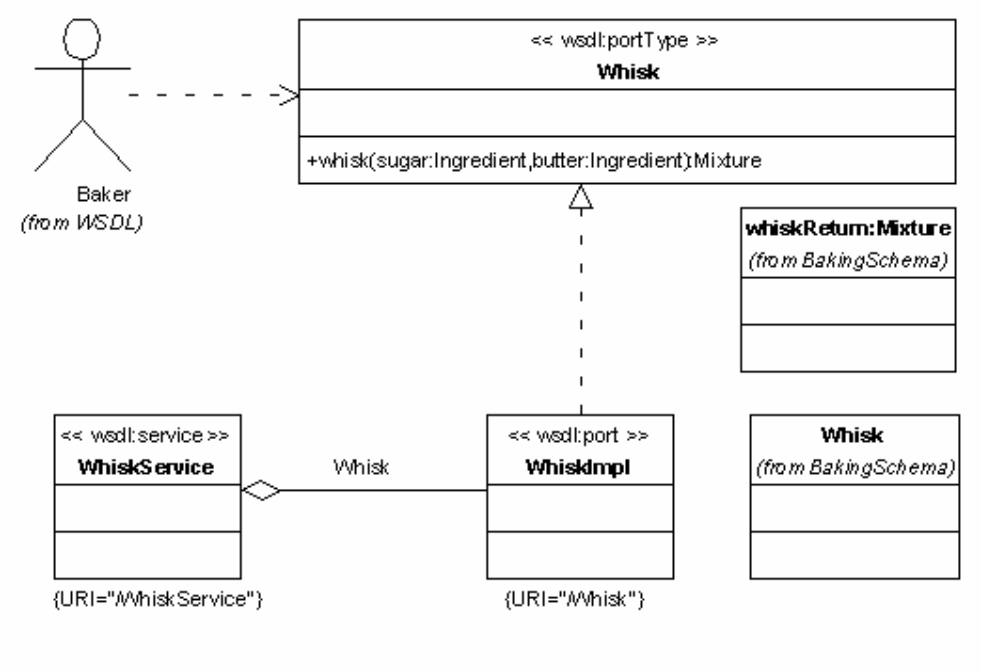

Figure 14 The UML model of the Whisk service

Similar models are applicable as well to the interfaces of the Beat & Mix, Fold and OvenBake services.

6. Provenance query design

In order to answer the questions discussed in section 2, we need to query provenance stores in order to retrieve relevant data. In this section, we first introduce a query algorithm used to find a general data item in a provenance store on the basis of a given result data identifier. Then we present three query examples we have performed in the BVSC application to demonstrate the query mechanism and, most importantly, the usage of provenance data.

6.1 Query algorithm

Our query algorithm is based on two assumptions. First, we assume that provenance data is organized in a data structure as defined in section 3.4. Second, we assume that the final result of the BVSC process (the cake) has a unique ID, which we term resultID. The input to the query algorithm is the resultID and the name of a data item (which we term searchItem), and the output is the quantity or unit associated with searchItem. The algorithm is given below:

```
foundID = false;
foundData = false;
located SessionID = false;

For all session IDs in a provenance store {
  For all activity IDs in current session ID {
    For all messages in current activity ID {
      if (resultID exists in current message)
      {
        located session ID = current session ID;
        foundID = true;
        break;
```

```
      }
    }
    if (foundID) break;
  }
  if (foundID) break;
}

if (not foundID)
  Show error message and exit;

For all activity IDs in located sessionID {
  For all messages in current activity ID {
    if (searchItem exists in current message)
    {
      foundData = true;
      return unitQuantity corresponding to searchItem;
    }
  }
}

if (not foundData)
   Show error message and exit
```

6.2 Query examples

We use three questions (Questions 4-6) from the ones we outlined in section 2 to demonstrate how provenance can be used to answer questions through queries.

Query 4: Was the correct sugar amount as specified in the recipe used? As discussed, the amount of sugar used in the whisking activity is one of the factors that may affect a cake's quality of taste; and there is generally a guideline on the minimum amount of sugar to be used in order to attain a minimum quality of taste.

The purpose of this query is to retrieve the amount of sugar used for baking a specific cake from a provenance store, and subsequently performing a comparison with the guideline on the minimum amount. Assuming that a cake with a unique ID is given, the query trail is as follows based on the following instantiation of the query algorithm just described:

- Search through all messages in the provenance store until a makeCakeReturn message is located which contains the unique cake ID;
- Locate the makeCake activity which contains the makeCakeReturn message;
- Locate the session ID that contains this makeCake activity;
- Locate the Whisk activity corresponding to this session ID;
- Extract the amount of sugar shown in the Whisk message within the Whisk activity. Both the client and service view of the message should coincide.

Once the actual sugar amount used is obtained from the above steps, we can compare it with the guideline for the minimum amount of sugar and draw an appropriate conclusion.

Query 5: Was the correct oven temperature as specified in the recipe used? As discussed, an excessively high or low temperature can affect the quality of a cake, and there is usually some permissible range for an appropriate baking temperature.

The purpose of this query is to retrieve the oven temperature used in baking a specific cake from a provenance store, and subsequently performing a comparison with the guideline on the permissible range. Assuming that a cake with a unique ID is given, the query trail is as follows using the query algorithm provided:

- Search through all messages in the provenance store until a makeCakeReturn message is located which contains the unique cake ID;
- Locate the makeCake activity which contains the makeCakeReturn message;
- Locate the session ID that contains this makeCake activity;
- Locate the OvenBake activity corresponding to this session ID;
- Extract the temperature from the BakeInOven message within the OvenBake activity.

The temperature can then be compared with the permissible range guideline.

Query 6: Was the correct amount of flour used for the oven baking activity at a given location? As discussed, ovens may be located at different altitudes; the altitude in turn influences the amount of flour required to produce a baked cake with a certain quality of taste. The higher an oven's altitude, the greater the amount of flour is needed. A table like that shown below can be used to determine the correct amount of flour to be used corresponding to a given altitude. The altitude is part of the actor provenance for the OvenBake activity for a specific cake and can be obtained in a similar query trail to the two preceding queries. The amount of flour by the activity can then be obtained from the Fold message in the Fold activity corresponding to the same session ID as the OvenBake activity (as they both belong to the same workflow/session). A comparison can be then be drawn with the given table.

Altitude(m)	***Flour(g)***
0	100
1000	150
2000	200
3000	250

7. Implementation

With the well-defined data types, messages and WSDL interfaces, the BVSC services can be implemented using many service development tools. In the pre-prototype, we have used Apache Axis 1.2 [4] to generate both client-side stubs and server-side skeletons. The client-side stubs act as proxies for services. The server-side skeletons can be fully implemented to form a concrete Web service. It also acts as the single entry point for the client.
Interaction and actor provenance recording, provenance store and query APIs have been implemented in the PReServ software package [3]. PReServ provides a provenance service that allows services to record provenance as well as query stored provenance, a client library to record and query provenance, an Axis handler to automatically record SOAP messages exchanged between Axis web services and Axis clients and a sample web service demonstrating the use of the provenance service and Axis handler. We have also implemented a simple query mechanism and performed the queries discussed in the previous section against the BVSC provenance store.

8. Summary

In this section, we discuss the key points and lessons learnt from developing the pre-prototype, draw some conclusions and point out directions for future work.

8.1 Discussion

By describing the BVSC process and the motivating questions in section 1 and 2, we have outlined the concept of provenance. Provenance has been investigated in other contexts [8, 9] using definitions such as audit trail, lineage, dataset dependence and execution trace. We have chosen here instead to refer to provenance as the documentation of a process that led to a data. This process-centered view on provenance which we adopted in the description of the BVSC process is motivated by our observation that most scientific and business activities are usually accomplished by a sequence of actions performed by multiple participants. The recently emerging service-oriented computing paradigm, in which problem solving amounts to composing services into a workflow, is a further motivating factor towards the adoption of our process-centered view on provenance. Further discussion about SOA is beyond the scope of this document However; appropriate references can be found at [10, 11].

In section 3, we analysed the BVSC application from a service-oriented view. From this analysis, we further identify that the provenance in the context of a SOA consists of two main types of provenance: interaction provenance and actor provenance. Interaction provenance focuses on the capture of an execution trace while actor provenance concentrates on the information pertaining to participating entities. We have placed special emphasis on the interaction provenance, since services are usually dynamically discovered, aggregated, executed and discontinued in a virtual organization on the Grid. In this context, information on how services are invoked, what messages are passed among them, and when they are invoked are usually required in order for a workflow result to be analysed or for a workflow to be repeated.

We have developed a provenance service to carry out provenance recording and storage for the BVSC application. The decision to employ a service-oriented implementation is made based on several considerations. Firstly, provenance can provide more added value for complex distributed applications that are increasingly adopting a service-oriented view for modelling and software engineering, as demonstrated in grid computing. Secondly, a service-oriented implementation of the provenance infrastructure simplifies its integration into a SOA, thus facilitating the adoption of the infrastructure in SOA-based applications. Finally, a service-oriented provenance infrastructure deploys easily into heterogeneous distributed environments, thus facilitating the access, sharing and reuse of provenance data.

Once provenance is recorded and stored, an important question then is how to access, explore and reuse provenance data in an optimally beneficial manner. From the end-users' point of view, the exploitation of provenance in enhancing problem solving processes (for example, speeding up or lowering its cost), is likely to be of greater consequence than the preliminary activities of provenance recording and archiving. Successful use of provenance data for problem solving will in turn motivate users to adopt system infrastructures that support provenance recording.

We have also developed a simple query API and used it to implement some sample queries. Although these queries are relatively limited in complexity, the query implementation at this stage demonstrates two important points. The first is that provenance data can be accessed through the designed algorithm. The second is the most important one, in that it demonstrates how provenance data can be used to answer questions. While there are undoubtedly many different questions in terms of application characteristics and many different ways of accessing and retrieving provenance data, the query algorithm and examples present a showcase for the viability of provenance usage.

In the development of the pre-prototype, we have outlined a concept of provenance, further elaborated this concept within the context of a SOA and implemented the BVSC services and the provenance

service. The provenance data has been stored in the provenance store against which sample queries have been performed. While the web service design and implementation as described in section 8 are not novel, the successful operation of the entire system has demonstrated the viability of the provenance concept and our implementation of it. The benefits of using provenance data for problem solving are self evident from the sample queries.

The BVSC application provides a simple but typical scenario from which key concepts, system requirements and software components have been identified as discussed above. However, the simplicity does impose some limitations on the investigation of other issues. For example, the BVSC process involves a linear sequence of services, and hence we have not considered the issue of iterative loops and/or parallel processing. The BVSC services are also implemented in a centralized fashion, and there are clearly additional issues of distribution and scalability to consider if a provenance infrastructure was implemented for a distributed Grid environment. The actor provenance in the BVSC application is currently vague and does not have explicit semantics associated with it. Actor provenance has to be modelled properly in order for it to be manipulated in a meaningful manner by end users.

Other issues that have not been studied yet include the management of large provenance data sets, management of distributed provenance stores, attachment of semantics to provenance data and extraction of knowledge from them. An appropriate security infrastructure for protecting provenance data is also an important future consideration.

In a separate document [12], we propose a strawman provenance architecture based on the findings of the pre-prototype development. The architecture will consist of a number of key components that we believe should be an integral part of the architecture, but which have not been discussed in the development of the pre-prototype.

8.2 Conclusions

In this document, we have discussed the prototype scenario, analysis, design and implementation. We have developed an implementation of a provenance architecture for recording provenance for the BVSC application, and used this recorded provenance data for answering some sample queries. In developing the pre-prototype, we have defined the provenance concept and further clarified it within the context of a SOA. We have identified the core components for provenance infrastructure (i.e. the recording, storage service and provenance store) and implemented them for the BVSC application.

The pre-prototype serves three main purposes. Firstly, it provides proof of concept for provenance and provenance infrastructure in the specific context of the BVSC application. Secondly, it provides guidelines towards the construction of a basic provenance architecture. Finally, it demonstrates a possible design and implementation pattern for provenance-enabled applications. Future work will include the following items:

1) A specification of the provenance system
2) Exploration and design of an appropriate query interface
3) Design of a comprehensive provenance architecture
4) Investigation, specification and design of a complementary security infrastructure
5) Addressing of scalability and distribution issues.

References:

[1] Victoria sponge cake recipe, http://thefoody.com/baking/victoriasponge.html

[2] For a quick, very accessible introduction to UML and its graphical notion, see: Terry Quatrani Visual Modeling with Rational Rose and UML, Addison-Wesley.

[3] The PASOA Project 2004, http://www.pasoa.org

[4] Paul Groth, Michael Luck, and Luc Moreau. A protocol for recording provenance in service-oriented grids. *In Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04)*, Grenoble, France, December 2004.

[5] The Stylusstudio http://www.stylusstudio.com/w3c/schema0/UseDerivInInstDocs.htm#UseDerivInInstDocs

[6] WebServices Axis http://ws.apache.org/axis/

[7] http://twiki.gridprovenance.org/schemas/

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

[9] P. Buneman, S. Khanna, and W.-C. Tan. Why and where: A characterization of data provenance. In International Conference on Databases Theory (ICDT), 2001.

[10] What is Service Oriented Architecture ? http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

[11] Service Oriented Architecture Definition**.** http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html

[12] Logical Architecture Strawman. Liming Chen, Paul Groth, Simon Miles, Victor Tan, Fenglian Xu, Luc Moreau