Business Processes and Workflow Coordination of Web Services

Jacky Estublier  
LSR-IMAG  
220, rue de la Chimie BP53  
38041 Grenoble Cedex 9  
France  
jacky.estublier@imag.fr

Sonia Sanlaville  
LSR-IMAG  
220, rue de la Chimie BP53  
38041 Grenoble Cedex 9  
France  
Sonia.Sanlaville@imag.fr

Abstract

Orchestration is currently presented as a way to coordinate web services in order to define business processes. In this paper we revisit these concepts, showing the current limitation of workflow coordination of web services, and we show that business process is different from web service coordination.

The paper present how we have extended the capabilities of workflow for the orchestration and coordination of services (web or not), and we show how business process and orchestration can collaborate in an application. We shortly describe our tool, Melusine, in which the different propositions have been integrated and experimented.

1. Introduction

Companies need to develop applications out of a number of components designed and developed to be used in isolation; these components can be legacy applications, code developed in-house, bought components, commercial software applications (COTS: Commercial Off-The Shelf Software), software applications owned by partners or web services. The software engineering world becomes more heterogeneous than ever. In this context, building an application is no longer integrating the different parts, where a piece of code calls another one, but rather controlling complex interactions between heterogeneous components.

Web services, while not a revolutionary technology (Remote Procedure Call over the internet is not that new), introduce a new way to build and maintain software applications because they are developed and maintained by third parties, they run on third party computers, and are accessed through the internet. Therefore their availability, the quality of service and their evolution cannot be controlled. This raises serious issues for some non functional properties, like error recovery, consistency control, efficiency, real time and maintenance.

The classical way to build software applications, using own pieces of code, is not satisfactory as soon as web services are part of the application. Provided the commercial importance of web services, a large number of works have addressed the issue of building applications out of web services. Virtually all works agree that a more flexible way to compose, control and coordinate web services is required. A solution proposed by many authors is to rely on workflow technology.

Workflow technology as worked out during the 90s [4], [10], [14], was primarily developed to address issues of high level modeling and execution of activities performed in an evolving real world. These issues include incremental instantiation of high level (often graphical) models, long transaction management, dynamic adaptation (model and instance), process capture and so on. Even if current workflow implementations only address a tiny subset of these issues, many authors realized that workflow could be a convenient solution for the modeling and control of a set of cooperating web services, with the simple equation: a web service execution = an activity.

2. The Melusine system

The following is a short and superficial description of our platform Melusine [6],[7] which is a platform for the design and execution of large applications. In Melusine, a component can be any kind of piece of code, classic code, either O.O. or following a component model, it can be legacy code, but also commercial products (like Excel, MSProject…) not designed to be integrated in a larger application. Melusine was extended recently to be able to consider a web service just as a special case of component.

2.1 Coordination: workflow and adaptors

Melusine clearly separates the coordination layer from the component layer. Between the workflow and the component lies an adaptation layer which links the activities found in the coordination model, to one or more components. Conversely, components, through the adaptor, can change the workflow instance state, its data (local and global), and its model (figure 1). This adaptation layer has similarities with the Workflakes adaptation platform [13].
The workflow languages currently proposed for web service coordination like BPML [1], BPEL4WS [8], WSCI [2], hypothesize that “1 activity = 1 web service”, the link between activity and web service has to be done manually in the workflow model itself. This approach requires effort and expertise but also to reason in terms of solution (the actual web services), not in terms of the problem to solve.

In the approach we propose, the workflow model (a graphical model), only defines activities and data flow, irrespective of the actual Web services and independently of any technology. An *adaptor* associates an activity with an arbitrary number of components, among others web services. There is no longer the strict association activity = web service and the workflow is now independent from web services (see figure 1).

### 2.2 The mediation layer

The approach proposed above still lacks flexibility; for example, the substitution of a web service by another one requires to change the adaptor. This solution still requires from developers to take care, manually, of the many technical detail involved in communicating with the component. The transparency of the heterogeneous nature of component is not achieved.

![Diagram of Coordination, mediation and components](image)

**Figure 1** Coordination, mediation and components

The mediation layer distinguishes the roles and the proxies. A role is an abstract service which defines functionalities offered (interfaces) and used (receptacles). A proxy can be a piece of code realizing the service, but most often it is a mediator which calls the real component. Therefore, there is not necessarily a direct relationship between a service provided by a role and the functionalities provided by the component.

The Mélusine development environment provides an editor allowing to model the components to be used. Web service is one of the recognized component types. The Melusine engine is in charge to fully handle the communication between the proxy and the actual component. In the case of a web service, the WSDL file (Web Services Description Language) is generated and the methods called by the proxy are transparently transformed into SOAP messages (Simple Object Access Protocol). In the case of a web service, there is no need for a wrapper.

The adaptor only knows the roles, thus we can substitute, even during execution, a component by another one, and as a special case, a web service by another one, as long as it can play the same role.

This architecture, in three layers, has many advantages over the usual workflow technology used for web services:
1. the coordination layer relies on a high level and graphical workflow model, independent from the nature of components;
2. the mediation layer adapts component capabilities to roles and
3. in the component layer, heterogeneous components are explicitly modeled and the system handles all the communication technology.

But business processes still require more.

### 3. Domains: the conceptual layer

Suppose a company which produces electric devices. To manage the production of a new product P1, the company has a BPM (Business Process Model). This process represents the know-how of the company and is therefore confidential. This process contains the concepts of device, teams and meetings. A meeting requires to organize a travel for the different participants to physically meet. To that purpose, the travel workflow presented above, can be used.

It is easy to see, on that example, that the workflows presented above are NOT the company business processes, but elementary processes, that can be provided by others or that can be shared with others, including competitors. This contrast sharply with the business processes, which are related with the company raison d’etre and are the company asset. We need to separate the description of the business logic, described by a (process) model, from implementation.

#### 3.1 The Mélusine Conceptual domain

The business layer, in Mélusine, is called the conceptual layer because it does not take into account most of the implementation details. The conceptual domain follows the OMG 4 layers [3], [5], [12] with MOF at the top, but also is split in two parts (see figure 2).

In the application part (on the right) we find in M1 a model of the application. This model defines the company concepts, like electrical device, the teams, meetings, and so on. This model is a UML model conform to the UML meta model, itself conform to the MOF.

In the process part (on the left side), at the M1 layer is found the process model. The process model describes how the entities found in the application should evolve, it drives the application.

We call a synchronizer the mechanism by which the process model drives the application. A synchronizer is based on relationships established between entities. For
example the “theDevice” entity found in the process model is related to the class “Device” in the application. Defining a synchronizer consists in defining the semantics of these relationships.

The Mélanieux development environment provides an editor and a language in which the synchronizers can be defined. For example, a synchronizer manages the relationship between “Decision” in the process and “Meeting” in the application; the synchronizer expresses what it means, for the application, to take a decision in the process. This relationship is defined at the model level, but executes at the instance level. This synchronizer is activated when a “Decision for D1 production” activity is created in a process instance. The synchronizer creates a “Meeting device D1” entity in the application instance, establishes a relationship, and sends the relevant information to the persons involved in the meeting.

The creation of activity “Decision for D1 production” involves the creation of an entity product called “D1”; the product synchronizer creates an entity “Device D1” of type “Device” in the application, and creates the relationship between them. In this way, transferring D1 from the activity “Decision for D1 production” to the activity “Device D1 production” will trigger the product synchronizer that will send the files describing the product “Device D1” resulting from the meeting (in the application), to the actual person responsible of the production activity.

It is important to emphasize that the synchronizers are NOT part of the process model, NOR part of the application model. They are an independent description, linking both domains. It means that the synchronization strategy can be changed without having to change the process nor the application; the same process model can be applied to a completely different company and a different application model; “only” the synchronizers are to be rewritten. Process models can be reused much more easily, and any application can be driven by processes without any need to rewrite it.
3.2 Adapters

The conceptual level is conceptual, there is no link toward real entities; to make the system really behave and act against the real world, there is a need to connect it to reality. For that purpose, we use the adapter facility presented in 2.1. In this case, the adapter is not attached to the process, but to the application instead; actions on an entity in the conceptual application domain trigger actions in the layer below, as presented above.

It is interesting to consider that the conceptual workflow, and the elementary workflows use the same technology. In both cases they are implementation independent, but they represent different processes, at different levels of abstraction.

In our example (section 3), the creation of an entity “Meeting device D1” in the conceptual application is captured by an adapter, which interprets it and launches the travel management workflow with the right information, in order to reserve the travels required for that meeting.

4. Conclusion

Current propositions for orchestration and coordination of web services result from a fast bottom up evolution and do not integrate the advances made in different domains like process support, workflow and interoperability. As a result, they are rather immature and low level.

The web service orchestrations standards, XLANG[11], BPML [1], and WSFL [8] are not executable; except BPML, they do not support different types of participants; none support simultaneously persistency, error recovery and monitoring, none support dynamic process evolution, dynamic substitution of web service. These workflow formalisms, and others like BPEL4WS [8], and WSCI [2] cover only the process model; they do not describe how they are connected to the actual web services; there is no help for designing, developing, debugging, monitoring and so on.

We claim that the current approaches, do not really provide support for Business processes.

We believe that our work contributes in many respects to the current state of the art.

Architecture. We propose an architecture clearly separating the different layers.

Heterogeneity. Our mediator and role approach allows to use in an homogeneous way heterogeneous components.

Communication. The component model used by Melusine is in charge to handle the communication protocol including error recovery.

Support. Melusine is a complete environment which supports design, implementation, execution and monitoring.

We have used our system to develop different applications for different client companies, and even to develop our own system Melusine. Our environment is in industrial use for years. The accumulated experiences have shown the validity of the approach, and especially the dynamic aspects and the interest of separating the business level from implementation.

References