Service-oriented Modelling for e-Business Applications Components

Giacomo Piccinelli and Mathias Sallé
Hewlett-Packard Laboratories
Bristol (UK)
{Giacomo_Piccinelli, Mathias_Salle}@HP.com

Christian Zirpins
Distributed Systems Group- VSYS
University of Hamburg, Hamburg (Germany)
zirpins@informatik.uni-hamburg.de

Abstract

The emerging trends for e-business engineering revolve around specialisation and cooperation. Successful companies focus on their core competences, and rely on a network of business partners for the support services required to compose a comprehensive offer for their customers. Modularity is crucial for a flexible e-business infrastructure, but related requirements seldom reflect on the design and operational models of business information systems.

Software components are widely used for the implementation of e-business applications, with proved benefits in terms of system development and maintenance. We propose a service-oriented componentisation of e-business systems as a way to close the gap with the business models they support. Blurring the distinction between external services and internal capabilities, we propose a homogeneous model for the definition of e-business applications components. After a brief discussion on the foundational aspects of the approach, we present the process-based technique we adopted for component modelling. We then present an infrastructure compliant with the model proposed that we built on top of an EJB (Enterprise Java Beans) platform.

1. Introduction

E-business has certainly attracted a lot of attention from software vendors, system integrators, solution providers, and ultimately from businesses. The traditional idea of e-business revolves around offering to customers, suppliers, and business partners the capability to automate their interaction with the sales or procurement department of a company. The Internet acts as an additional channel, offering unprecedented possibilities to businesses in terms of speed and automation for interaction processes. The e-service model [4] builds on the power of existing e-business capabilities, and extends it with the aim of making the Internet a pervasive reality into businesses.

In the e-service model, any type of asset can be engineered and presented as a service to potential users inside and outside the boundaries of a company. The encapsulation of specific sets of business capabilities into well-defined service modules improves internal management and execution. Modularity helps localise points of weakness, over sizing, under sizing, and integration problems with other parts of the business infrastructure [12]. Modularity enables the outsourcing of specific business activities, as well as the external offer of excess capacity. The combination of the e-services model and enabling business infrastructures like electronic marketplaces gives a dynamic angle to internalisation and externalisation of service components. The focus shifts from the connection to a specific business partner, to the definition of a specific business need. The link with the business partner offering the best conditions for a service, at every point in time can be built exploiting the aggregation power of open electronic marketplaces [1].

After a brief overview on the e-service vision, we present a component model for e-business applications based on the concept of service modules. We first describe the process-oriented approach we took to service specification. We then present the EJB-based (Enterprise Java Beans) prototype for an application platform based on service modules.

2. E-Services Vision

Until recently, the Internet was about the creation of e-business and e-commerce systems, and it was dominated by web sites and storefronts. We have now entered the next Internet evolution: the proliferation of e-services. E-services are modular, nimble, electronic services that perform work, achieve tasks, or complete transactions [4]. Almost any asset can be turned into an e-service and offered efficiently via the Internet to drive new revenue streams. Chapter 1 of the Internet was about businesses getting wired to their employees, customers and partners; key business processes getting linked to the Internet, and a critical mass of consumers coming online.

Chapter 1 was about the creation of e-business and e-commerce systems that form a critical foundation. Businesses were learning how to use what looked like a promising new tool. Now, the Internet is ready for its next evolution. It won't be about businesses looking at the web as a technology. Internet has been absorbed into the core business infrastructure, and businesses are ready to

0-7695-1269-0/01 /$10.00 © 2001 IEEE

capitalise on this new asset. Chapter 2 of the Internet will be about the mass proliferation of e-services.

These services will be modular units that combine and recombine to solve problems, complete transactions, and make life easier. Some will be available on web sites, but others will be delivered via TV, phone, pager, car, email in-box, or virtually anything with a microchip in it. Some will even operate behind the scenes, automatically working on behalf of consumers and providers.

A definition: *an e-service is any asset that is made available via the Internet to drive new revenue streams or create new efficiencies.*

In Chapter 2, successful companies will be those that determine how to turn their assets into services delivered via the Internet. Successful companies will adopt an entrepreneurial approach to looking at their assets figuring out how to best leverage not only their core business offerings, but also their proprietary processes, data, relationships, knowledge, experience. In Chapter 2, we will see more companies turn these assets into services and offer them via the Internet.

3. Use context for service components

The first step to turn an existing asset or service into an e-service revolves around accessibility. The electronic virtualisation of the service has to provide communication channels that support automated conversational capabilities. Automation is fundamental at each step of the service delivery chain. Beyond the basic capability to exchange electronic messages using standard protocols on top of an XML transport, the business logic behind the service provision and partner interaction has to be enforced. For example, the service offer has to be presented in a way that allows automated discovery to take place. The service description should enable advanced offer-request matching (beyond the basic pricing), as well as automated negotiation on contractual terms and parameters. The role of advanced directory services (e.g. UDDI), and in particular of electronic marketplaces is fundamental. An e-service is not a standalone entity; rather it is a first-class citizen of a highly dynamic ecosystem enabled by e-marketplaces.

The second step towards the realisation of the full potential for the e-service vision focuses on composition and interaction orchestration. Beyond business conversations for point interactions [3], an e-service has to expose all the interaction processes involved in the service delivery. Far from saying that a company should expose its core competences, the requirement is to handle the internal and external business networks dynamically created by each and every instance of service delivery [10, 11]. A service delivery may no longer be a one-to-one (buyer-to-seller) relationship. As an example, let us assume that the company iBuild has selected the company iMove for a shipment contract. The final product of iBuild may be packaged by a company iPack, and iBuild may want iMove to interact with iPack for arranging the logistics behind collecting the goods. Similarly, iMove operational structure may be such that it focuses on hub-to-hub transport using lorries, and it relies on partners for the hub-to-customer transport. In the case of the service sold to iBuild, iMove may select (directly or using an e-marketplace) a company iVan to do the first leg of transport. As a consequence, iVan has to synchronise with iBuild and iPack. The end customer will still be iBuild in the same way as the overall responsibility for the end-to-end transport will still be on iMove, as far as both iBuild and iMove are concerned. The thing to observe is how in the scope of a specific instance of service delivery, multiple parties are dynamically pulled together. Some of them know some of the others, but in some cases (e.g. iVan) the service providers might not have had previous relationships. From an operational point of view, an e-service should be able to cooperate with a dynamically selected mix of other e-services. This imply the capability to automatically verify the behavioural compatibility of the various execution processes, as well as the capability to adapt them (within feasibility boundaries) in order to make cooperation possible.

From a technology perspective, there is a proliferation of initiatives in the industry and within standard bodies aimed at better exploiting the potential that the Internet has for businesses. Leveraging these efforts, HP is promoting a comprehensive framework oriented towards making the e-service vision become a reality. The ability to expose services in a way that they can be automatically visible and accessible to potential customers is the focus of this service framework specification (SFS [4]). The work described in the next sections of this paper is based on such framework. The SFS defines standard business and technical conventions that allow e-services to dynamically interact with each other.

4. A model for service components

The model we propose for service components is based on the ideas of functional incompleteness, multi-party orchestration, and dynamic service composition [6, 9]. A service can be partially incomplete in terms of its implementation. Meta-information present in the electronic virtualisation layer for the service specifies the kind of support services needed, as well as the type of integration required to become fully functional. For an e-service, the focus moves from the access logic to the integration logic. The challenge for both service providers

and service consumers is to adopt an integration model based on business roles and behavioural descriptions.

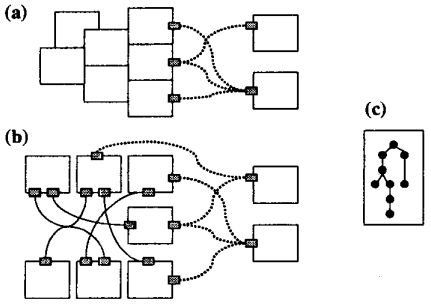

Figure 1: Process-based interaction

In a business transaction, the service consumer has to be informed about the kind of interaction process supported by the service provider. The idea is to expose the service delivery process as early as possible, so that both service consumer and service provider can better evaluate their operational compatibility. The impact of e-services on the design for e-business systems is captured in Figure 1. Existing systems are developed around object-oriented models, and different functions are isolated into different parts of the system. The problem is (Figure 1a) that different functional modules are hardwired to each other in an ad-hoc way. The idea is instead to move to a scenario (Figure 1b) in which different functional modules are kept separate. The interaction logic (Figure 1c) behind what then become service units is captured explicitly, and the distinction between internal and external service components is blurred.

Assuming a service offer organised around this model, the operational structure of the service itself can be designed with a new approach. First the need for specific support services is identified. Next the expected interaction processes with the potential service providers is identified. A specific service instance is available, only if the adequate support services can be found. The concept of adequacy is heavily based on operational compatibility, in order to ensure a smooth execution of the overall service. The implications on cost and availability are significant. The provider for an e-service component can focus on the implementation of the core aspects of the service. The e-service infrastructure will take care of the integration with the most suitable e-services to completely enable the new e-service. Integration logic coexists with business logic, still remaining two separate entities in terms of management and visibility.

5. E-Service Bean

As an implementation example of the component model proposed, we instrumented an EJB (Enterprise Java Beans [7]) platform with process-oriented componentisation capabilities [3, 5]. The work revolved around the implementation of a new type of EJB container, within which an XML-based process description file [2] can be used to model the observable behaviour of the bean. Clients and other beans will only be able to execute methods on a bean in this container if they are consistent with the process description. The outbound communication initiated by the bean is also monitored for compliance with the behavioural interface captured in the process. In line with the naming conventions for EJB, we refer to the new container as ESB (E-Service Bean).

```
<!DOCTYPE process_description [
        <!ELEMENT process_description
(roles,structure+)>
        <!ELEMENT structure
(type,elements,structure*,cons*)>
        <!ELEMENT con
(roles,method,exe_cons,exe_con*,ret_cons,ret_con*)>
        <!ELEMENT exe_con
(type,parameter,operator,value)>
        <!ELEMENT ret_con
(type,parameter,operator,value)>
        <!ELEMENT roles (#PCDATA)>
        <!ELEMENT type (#PCDATA)>
        <!ELEMENT elements (#PCDATA)>
        <!ELEMENT cons (#PCDATA)>
        <!ELEMENT ret_cons (#PCDATA)>
        <!ELEMENT exe_cons (#PCDATA)>
        <!ELEMENT parameter (#PCDATA)>
        <!ELEMENT operator (#PCDATA)>
        <!ELEMENT value (#PCDATA)>
        <!ELEMENT method (#PCDATA)>
]>
```

Figure 2: DTD for the Process Description Language

A bean models a service unit, and the process description captures the service delivery process deriving from the external interaction of the bean (Figure 2). Different roles can be involved in the delivery process behind the service implementation. The ESB container manages at run-time the behaviour of the entities playing these roles. When a bean is created, the roles involved can be partitioned into groups and assigned transparently to either client programs or other beans. The only interaction allowed is the one deriving from the process description (both inbound and outbound). The aim of our prototype was to implement a basic container that demonstrates this kind of protection for the beans. The container in which service beans are to be deployed has the following features not found in normal EJB containers:

- The bean provider can specify the service behaviour in a process description file (using XML) that is then enforced by the container. This means that the container will generate exceptions whenever a method is called in an incorrect way (at the wrong point in the process or with invalid parameters). Exceptions will also be generated in cases where a service bean invokes a method on another service bean that does not comply with the specified behaviour.
- A process can be specified to have a number of roles that can be played (Figure 2). Clients can create a bean, specifying the role(s) they want to play, or contact an existing bean to have a role/roles assigned to them. The container makes sure that a service cannot be started until all roles are assigned.
- The client can request role specific descriptions from a service bean to see what is required to do as the entity responsible for a specific role/roles.
- The system makes the state of each service instance persistent so that everything can be reconstructed in the event of a system crash.

The tasks performed by the system can be divided into two parts [7]. The first part is the creation of the home and remote object classes. The second part is the actual runtime handling of the beans, where the home objects are made available via JNDI allowing them to be created and used. The following sections describe the two parts of the container and how they function.

5.1 Creating the Home and Remote Objects

The JAR file containing the interfaces, process description and deployment descriptor are placed in the *hpcon\jars* subdirectory of the container installation directory. The *deploy* batch file is then executed with the location of the JAR file as the first argument and the name of the bean of the second argument e.g. *deploy C:\hpcon\jars\Cabin.jar Cabin*. The container then reads the manifest of the JAR file, finding the deployment descriptor.

The deployment descriptor contains information on the persistent fields of the bean and whether the bean contains references to other service beans. If the bean does contain such references, then any method invocations made using the references will be checked also against their behaviour specification. Once the information in the deployment descriptor has been read, the container can generate home and remote object classes. The home object is used by clients to create beans (partitioning the roles to be played into groups), request group specific remote objects, find remote objects, and destroy beans. The remote object is used by the client to make method calls on the bean. The *deploy* batch file then creates an instance of the home object which makes itself available via JNDI for clients to contact it.

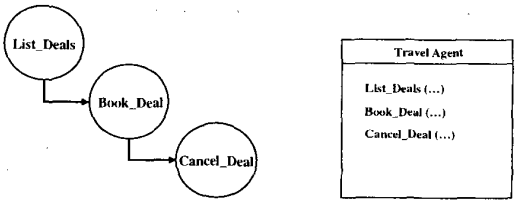

Figure 3: Service model for the Travel Agent bean

5.2 Runtime Handling of the Beans

The second part of the EJB containers work involves making the home object available to clients and monitoring the use of the bean object via the home and remote objects. The home class contains a constructor allowing a home object to be created from the command line. Once the home object is created it binds itself to the RMI Registry and makes itself available via JNDI. Remote interaction between the client and the container takes place via Java RMI [8]. The JNDI RMI standard extension is used so that the client can lookup home objects via JNDI.

The client creating a service bean can dynamically partition the roles to be played by clients into groups. Each client plays a specific group of roles, and it receives a remote object used to call methods on the bean. The client that creates the service bean gets a remote object for the first group of roles to be played. The other clients are assigned groups using the `assign` method in the home object. A `processUtility` object is instantiated by the home object when a service bean is created. When the `processUtility` object is constructed, it checks with the process description that the grouping of the roles it has been given by the home is correct. If the grouping is valid, an entry in the database for that type of bean is created. The remote object then uses the `processUtility` object to check the validity of method calls. The persistent fields of the service bean are written to the database after each valid method invocation by the `processUtility` object.

The remote objects can catch method invocations and return types to check that they are consistent with the process description. If they are not, a specific exception is thrown. The remote objects can also be used to catch outgoing calls from a bean. When a method invocation is made on the EJB remote object by the client, the remote object calls the `check_method` method of the `processUtility` object. This method uses the process description file to check if the method call (including the parameters) is valid. If the method call is not valid, the

result returned to the remote object contains specific error codes. The remote object therefore makes the method call on the actual bean if no error code was returned. When the return value is received from the method call, it is sent as along with other control information to the `check_return` method of the `processUtility` for final controls. The result is then returned to the client.

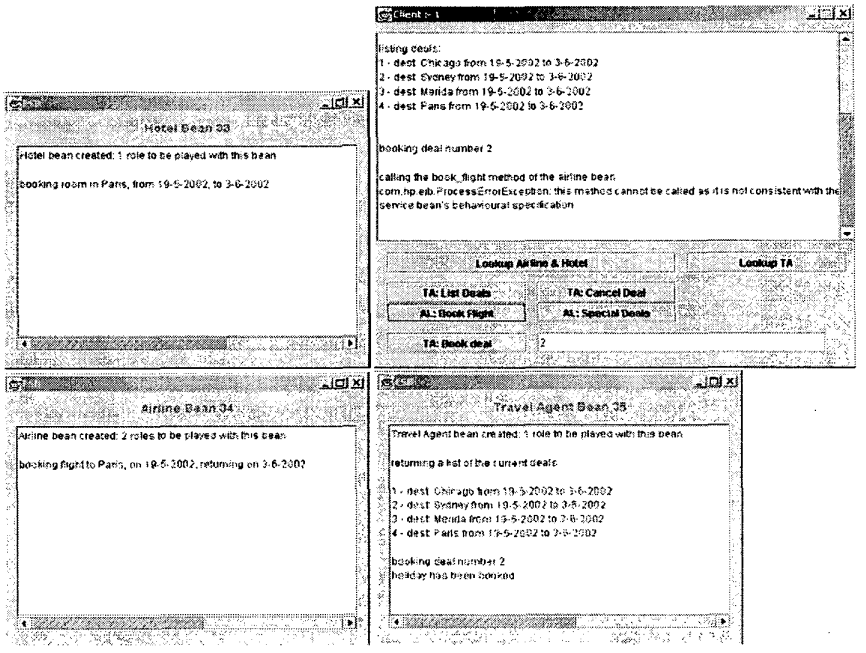

Figure 4: Component interaction mediated by ESB containers

6. Use example for the ESB container

The activity of the ESB container is illustrated in a scenario consisting of three very simple service beans, the Travel Agent bean, the Airline bean and the Hotel bean. The scenario shows the impact of external management on the interaction behaviour of the various components.

In the scenario (Figure 4), a client first interacts with the travel agent to list holiday deals. The client books a deal, and then cancels the deal. Everything is coherent with the behavioural model specified by the travel agent. When the client books a deal, the travel agent interacts with an airline and a hotel to book a flight and a hotel room for that holiday. The client can directly refer to the same airline to request information on a "special deals", which are budget flights with a price of less than £300. Though it is one of the methods exposed be the airline, the client is instead not allowed to do direct booking with the airline. When the client attempts the booking, the incompatibility between request and behavioural specification for the service is detected and the request rejected.

The behavioural interface of the travel agent bean (Figure 3) specifies that this bean can be created and used by one entity only, which means there is only one free role available to be played. In the animation of our scenario, the client plays this role. Once an instance of the travel agent bean is created, the client can invoke only the `list_deals` method. Once the list has been requested, a deal can be booked using the `book_deal` method. When the booking is requested, the travel agent bean calls the `book_flight` method of the airline bean and then the `book_room` method of the hotel bean. The client is now allowed to cancel the deal by calling the `cancel_deal` method. The ESB container prevents other method invocations from reaching the beans, as they do not conform to the specified behaviour of the components.

The Airline bean behavioural interface specifies two free roles, one of which will be played by the client and the other by the Travel Agent bean. The role played by the client will allow only one method to be invoked, the `special_deals` method, which must return an integer less than 300. The role played by the travel agent allows the invocation of the `book_flight` method. The Hotel

bean has only one free role, which is played by the Travel Agent and can be used to book rooms.

Figure 4 shows a basic client console and the monitor interface for the beans. The snapshot is taken immediately after the container has trapped a method invocation for the booking attempt from the customer to the airline. In this case the client has tried to invoke the `book_flight` method of the Airline bean, which is not available within the role the client is playing. The Airline bean is automatically shielded from the illegal request by the ESB container. Previous to the intercepted method invocations, the client called the `list_deals`.

7. Conclusions

E-business models often focus on the flexibility of the service offer. The capability to acquire efficiently the external resources required to satisfy specific demands is important, and electronic marketplaces play a key role in this process. Still, the quality and profitability of the service offer depends on the effective integration of external resources with internal business infrastructure. We propose that a service-oriented modularisation of e-business systems could reduce the gap between internal and external components behind a service implementation.

Based on the e-service vision, we propose a process-oriented model for the operational description of service components. Together with the foundational aspects of our proposal, in this paper we present a prototype infrastructure that instruments an EJB-platform (Enterprise Java Beans) with capabilities for the definition and implementation of e-service components.

8. References

[1] Blodget H. and McCabe E. *"The B2B market maker book"* Merrill Lynch & Co., 2000.

[2] Cagle K. *"XML developer's handbook"* Sybex, 2000.

[3] FIPA (2000) Foundation for Intelligent Physical Agents. http://www.fipa.org

[4] Hewlett-Packard (HP, 1999) *"E-Services"* http://e-services.hp.com

[5] Holligsworth D. *"The workflow reference model"*. Workflow Management Coalition (WfMC), TC00-1003, 1994.

[6] Marton A., Piccinelli G. and Turfin C. *"Service provision and composition in virtual business communities"*. Proc. 18th IEEE - ISRDS, Int. Workshop on Electronic Commerce. Lausanne, Switzerland, 1999.

[7] Monson-Haefel R. *"Enterprise Java Beans"* O'Reilly, 1999.

[8] Orfali R. and Harkey D. *"Client/Server programming with Java and CORBA"* Jon Wiley & Sons, 1997.

[9] Piccinelli G. and Lynden S. *"Concepts and Tools for E-Service development"*. Proc. 7th International Workshop HP OVUA, Santorini, Greece, 2000.

[10] Reilly B. and Block J. *"Next-generation E-Commerce processes and systems"* Electronic Commerce Strategies Report, GartnerGroup, 1997.

[11] RosettaNet *"RosettaNet"* http://www.rosettanet.org

[12] Schwartz D.G. *"Cooperating Heterogeneous Systems"*. Kluwer Academic Publisher, 1995.