A Service Component Deployment Architecture for e-Banking

3rd International IEEE Workshop on

Service Oriented Architectures in Converging Networked Environments (SOCNE 08)

José L. Ruiz*, Juan C. Dueñas**, Félix Cuadrado**

**Indra, **UPM*

jlrrevuelta@indra.es, {jcduenas, fcuadrado}@dit.upm.es

Abstract

Service-oriented architectures (SOA) foster software reuse and loose coupling by means of strongly restricting interactions to service contracts. Many companies have identified in SOA an opportunity not only to improve their productivity but also to promote and extend their B2B relationships. The ITECBAN project is a significant reference of this process within the Spanish banking market.

The scope of this paper is the implementation of the SOA paradigm in enterprise-class environments based on JEE technology. SCA (Service Component Architecture) and JBI (Java Business Integration) specifications have taken the first steps to support deployment of distributed services at the server-side Java platform. However, significant issues such as runtime service version management and context adaptation are still open. This paper presents a deployment solution for service-oriented JEE distributed systems based upon available specifications (SCA, OSGi and OMG Deployment and Configuration) and OSS implementations.

1. Introduction

Services are not by any means a novel concept. In economics, the notion of service has been used for ages. A service is just the non-material equivalent of a good. The main difference between goods (a.k.a. products) and services is the business model. In a product sale the customer becomes the owner of the business artefact. On the contrary, a service provider does not transfer the ownership of the business value. A service-oriented business model is related to service use (a.k.a. service consumption) and thus, service providers are free to make money out of the same resource with multiple customers [1]. The enforcement of a successful service-oriented business model requires the support of an effective SOA (Service Oriented Architectures) [2]. In an SOA service providers and service consumers must be loosely coupled. Service registries [3][4] and service discovery mechanisms [5][6][7] are alternatives for providing service location and service binding, which are preconditions to enable provider-consumer interactions. In B2B contexts, the need for interoperability between actors with different technology platforms has derived in a plethora of protocols and data formats fundamentally related to XML [8][9], which has finally brought consensus to the IT sector.

Unfortunately, the underlying heterogeneity is not an exclusive issue for B2B environments. In many companies, the internal IT infrastructure is a mixture of legacy systems and brand-new platforms. Consequently, business services are provided by the integration of software components scattered through a distributed and heterogeneous environment. Service deployment in this situation is a complex issue, too often solved by suboptimal solutions that include manual tasks, component replication and expensive management solutions. The aim of this paper is providing a service deployment architecture that adapts to the specifics of the deployment environment, considering service versioning and the capabilities of the nodes in the deployment domain.

This research work is carried out within the scope of the ITECBAN project. ITECBAN's mission is to provide a full-blown service-oriented solution (architecture, runtime and methodology) for the banking sector. In our opinion, the results obtained in ITECAN are a reference case-study to develop and validate enterprise-class service deployment solutions at other markets.

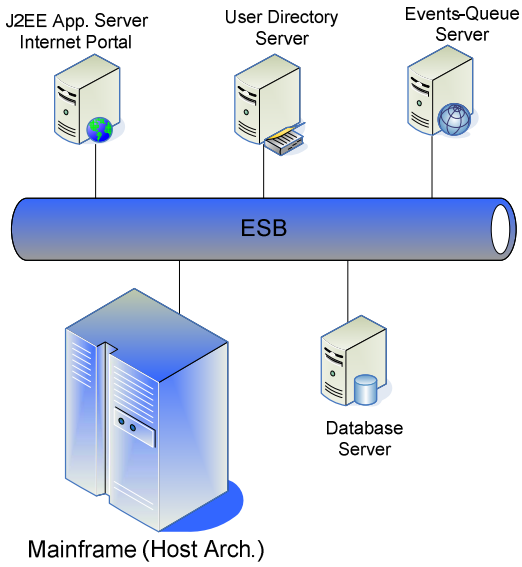

Figure 1 Target Deployment Environment

2. Deployment in the Banking Scenario

Banks are liable for whatever is done with their customers' savings and investments. Therefore, changes in the IT infrastructure are carefully controlled in order to deeply understand and measure the implications of the change. The life-cycle of systems in the banking sector is significantly longer than other business markets, such as telecommunications or the consumer electronics markets.

Consequently, it is quite common that business and IT services are supported by a heterogeneous infrastructure, where legacy systems and brand-new services coexist. And in many cases work together. In order to reduce risks, banks (as well as many other organizations) deploy services in two stages 1) at testing environments, with software and hardware configurations similar to production environments, but with a simplified data layer and 2) at production environments, which in some cases can include additional environments in order to carry out additional system tests.

COBOL and Hosts architectures (for short, composed of a jobs scheduler and a database running over mainframe architectures) have been the technological references in the banking market for ages. During the last decade Java Enterprise technologies have progressively gained momentum, especially related to e-banking portals.

Nowadays many e-banking services run over a distributed and heterogeneous architecture, where host architectures and JEE architectures cooperate with each other.

Several integration solutions have been used to cope with this situation [10]; event-driven architectures such as JMS or MQ-Series and lately, the ESB (Enterprise Service Bus). The ESB provides an architectural pattern for integration as 1) a runtime environment for services, 2) a service mediation manager and 3) as an inter-services message router.

As a message router the ESB is responsible for supporting a wide range of transports (SOAP/HTTP, SMTP, RMI/IIOP, FTP, JMS and so on), queuing and dispatching messages among services and carrying out data format transformations, which is in many cases required not only for service adaptations but also in the e-banking context for data confidentiality issues.

All in all, the ESB is a suitable candidate for the integration of heterogeneous systems and more important, for the enforcement of service-oriented architectures. Figure 1 presents a reference scenario of the target deployment environment that must be handled by the deployment architecture.

The following are the main requirements that the deployment architecture has to meet in order to provide an effective support for the deployment of e-banking services:

- It must support distributed deployment environments.
- It must support heterogeneous configurations in the deployment environment.
- It must support deployment operations for new services and also legacy systems. Given that the runtime environments are capable of executing the components being deployed.
- It must support the deployment of service-oriented components. Service orientation has been chosen as the technical reference for new systems and thus the deployment architecture must support the specifics of service artifacts deployment, which includes deployment in the ESB.
- It must support a wide range of implementation technologies and communication protocols. The distribution of service components across the deployment environment respecting resource allocation constraints, i.e. hardware resources, software configurations and networking capabilities.

- It must be version-aware with regards to service components. Dependencies management and change control are enforced by means of manipulating version tags. In addition, the runtime environment must be able to execute different service versions at runtime. The rationale for this is the long product life-cycle that is characteristic of the banking domain.
- It must support a fine-grained control of the deployment life-cycle at runtime. Switching-on and off a deployment environment means losing time. And thus money. The deployment architecture must provide capabilities to install, remove, update and (re)configure service units at the deployment environment without interrupting the operation of services which are not involved in the operation.

Many of the aforementioned requirements are related to the capabilities of the runtime environment. At production environments the newest elements are JEE application servers. Unfortunately, these runtimes are not suitable for SOA deployments. JEE servers are designed to execute independent and isolated applications. It lacks the service notion and the possibility of adding service components. In addition, though these systems support deployment replication in server clusters they are by no means targeted to the distribution of service components.

All in all, we must keep the pre-existing infrastructure, but it is obvious that we needed an ESB. ESB products basically implement either JBI (JSR-208) or SCA specifications. Because of the better support for non-Java technologies we decided to build the deployment architecture on top of SCA [11]. Next section presents a summary of the Service Component Architecture specification.

3. Service Component Architecture

SCA specifications aim at providing integration means for combining a wide range of implementation technologies in a distributed deployment environment. SCA's approach conforms to the SOA principles, i.e. restricting interactions to well-defined interfaces (business contracts) and providing flexibility with regards to implementation technologies (at the moment of this writing Java, C++, BPEL, Spring and COBOL are supported) and also to communication protocols (SOAP/HTTP, JMS, RMI/IIOP and JCA).

The focus of the SCA specification is service composition. For achieving this goal an XML-based

Listing 1. SCDL example

```
<composite
<component
   name="HelloWorldServiceComponent">
  <service name="HelloWorldService">
  <interface.wsdl
interface="http://helloworld#wsdl.interfac
e(HelloWorld)" />
  <binding.ws
uri="http://localhost:8085/HelloWorldServi
ce"/>
   </service>
   <implementation.java
     class="helloworld.HelloWorldImpl" />
</component>
</composite>


<component name="HelloClient">
 <implementation.java
   class="helloworld.HelloClient"/>
   <reference name="HelloWorldService">
<binding.ws
wsdlElement="http://helloworld#wsdl.port(H
elloWorldService/HelloWorldSoapPort)"/>
   </reference>
</component>
</composite>
```

language called SCDL (Service Component Description Language) is used. SCDL is basically an ADL (Architecture Description Language) that specifies service component assemblies.

A SCA component is the minimum description unit that can be described in SCDL. The implementation details of the SCA component are almost completely hidden in the description, but for an indication on the implementation technology, which is eventually used to deploy and run the SCA component.

The most relevant part of the SCA component descriptions has to do with the offered and required services (required services are called references in the SCA jargon). One service contract can be made available through many transport technologies; each transport offers a service binding (which includes information related to the protocols and the service location endpoint). Binding descriptions can be qualified with service policies using WS-Policy to express policy rules. Since services description is made explicit then it is possible for the deployment environment to provide techniques for automatic services resolution such as dependency injection. These features are available out-of-the-box for runtimes leveraged with technologies like the Spring framework.

SCA components can be aggregated into a SCA assembly, a.k.a. SCA composites. Listing 1 includes an

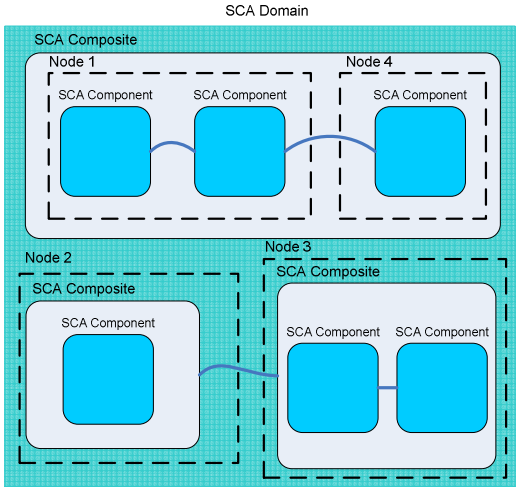

Figure 2 SCA Distributed Deployment

example of two SCA components, a *HelloWorld* service provider and a *HelloWorld* service client. The example showcases that the assembly includes wiring information that binds required service references to provided services (using Web Services in the given example).

In addition, the composite description may promote internal services at the composite level. Unsatisfied internal service references must be promoted and eventually resolved by the SCA runtime in order to execute the composite.

So far we have covered the capabilities of SCA for describing service components and creating assemblies. SCA specifications also cover packaging service components, which is required for remote component deployment. Basically, the packaging format is a ZIP file that contains implementation artefacts and the SCDL descriptors. Regarding distribution the specification provides some indications about the distribution of service components in a SCA domain. A domain represents a runtime, potentially distributed over a network of computing nodes. Deployment over a distributed domain is envisioned as shown in Figure 2. Service composites can be distributed across nodes within the domain, provided the bindings between components are established by remote transports.

4. Deployment Extensions to SCA

As such, the SCA specification covers most of the needs for the deployment architecture (see Section 2) in our target deployment environment (see Figure 2). However there are some issues missing with regards to our requirements and thus we had to build certain extensions to SCA in order to provide:

- ***Service versioning***. It is mandatory to support version management control at the deployment architecture. Therefore, service descriptions and service wirings must respect version information, because potentially there will be several service versions co-existing at runtime at the deployment target.
- ***Component distribution*** across the deployment target. The SCA specification leaves up to the SCA runtime providers the definition and mechanisms to carry out the distributed deployment. Our deployment architecture provides a solution for this issue based on node manager agents and a coordinator entity for the domain.
- ***Context adaptation***. Context adaptation is out of the scope for the SCA specification. In fact, this issue is once more related to the capabilities of the SCA runtime. Our aim was to adapt deployment operations to the available resources at the deployment environment. Therefore, we added this feature to our deployment architecture.

5. Deployment Architecture Description

First of all, our deployment architecture is based on the SCA specifications. It provides a distributed runtime SCA that is able to parse and process SCDL descriptors to carry out distributed deployments. We have provided some extensions to the SCDL descriptor in order to 1) provide required resources information and 2) to add version information related to services descriptions.

The first extension is based on the OMG Deployment and Configuration of Component-Based Distributed Applications [13] specification, for short OMG D&C. The objective of this extension is providing a basis to match service components resource requirements to the capabilities of the nodes that compose the deployment domain. This extension has been provided using XML Schema types conforming to the OMG D&C specification. [13]. Figure 3 shows the resource model data types. Each SCA component description has been extended to add an optional collection of required resources. These very same types are used to describe the capabilities offered by the nodes in the domain (an example is provided in Listing 2 and therefore it is possible to achieve a context-aware component distribution. The information shown in Listing 2 is

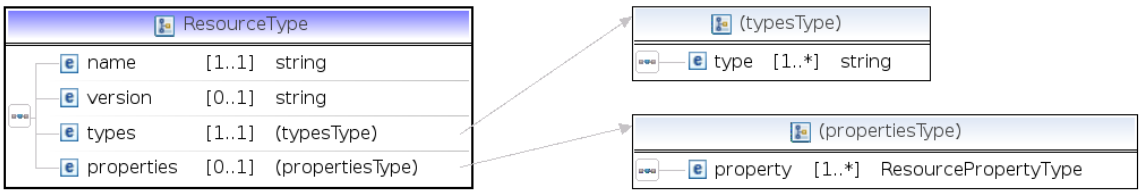

Figure 3 Resource Model

dynamically obtained by deployment infrastructure components called *NodeManagers*. The description of the node's resources could be done manually, but it would require a considerable effort. Furthermore, deployment target topology and nodes' resources change over time and we would need to redefine the domain information. Discovery protocols have helped us to create and update this information at runtime.

Each node included in the SCA domain is provisioned with a NodeManager component. *NodeManagers* are registered as dynamic services in the network using a DNS-SD (Service Discovery) implementation.

The SCA domain is controlled by one of the nodes, provisioned with the *TargetManager* component. The TargetManager is responsible for providing information regarding the domain configuration (such as the one shown in Listing 2) and coordinates the distributed deployment operations. Thanks to the DNS-SD protocol the target manager is aware of the nodes available in the domain, by means of tracking DNS-SD events related to *NodeManagers*.

Figure 4 shows a sample SCA domain composed of two nodes. According to the deployment architecture each node participates in the domain with an SCA runtime, which has been implemented using the OSGi platform. The main reasons for using OSGi at the core SCA runtime are as follows:

- OSGi provides a service-oriented programming model and a runtime that supports executing multiple service versions.
- OSGi is an extensible and portable execution environment.
- It provides a Java deployment API to govern deployment activities. We use its capabilities to install, uninstall, update and remove service components at runtime.
- OSGi can combine deployment for pure Java components with native libraries. In addition, with JSR-223 (Scripting for the Java Platform) the JVM can be used to execute components implemented in languages such as Ruby, Python or JavaScript.
- The advanced class-loading mechanisms included in OSGi enable a finer control of the low-level relationships of Java-based components. As the main focus of the deployment environment is the JEE platform this is an extra value for the deployment architecture.

The SCA runtime is leveraged with the *NodeManager* component. One of its main responsibilities is exporting the node's information according to the resource model. This is implemented using context gatherer services on top of OSGi. Tailored gatherer components are dynamically added and removed from

Listing 2. Node Resources Description

```
<node>
  <name>node1</name>
  <label>indra.domain/node1</label>
<resources>
 <resource>
  <name>memory</name>
<types>
  <type>RAM</type>
  <type>HARDWAREPLATFORM</type>
</types>
<properties>
 <property>
  <name>size</name>
  <value>64M</value>
 </property>
 <property>
  <name>freeNow</name>
  <value>31MB</value>
 </property>
</properties>
</resource>
<resource>
  <name>cpu</name>
  <types>
   <type>PROCESSOR</type>
   <type>HARDWAREPLATFORM</type>
  </types>
<properties>
<property>
<name>architecture</name>
<value>i386</value>
</property>
</properties>
<!—and so on…->
```

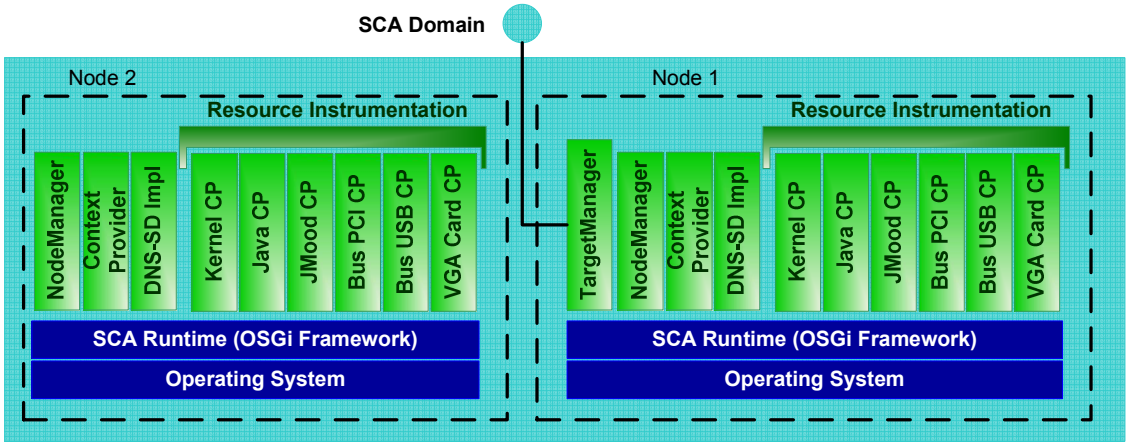

Figure 4 A sample deployment-enabled SCA Domain

the runtime. All of them are dynamically bound to the *ContextGatherer* component which exports this information to the *NodeManager* by means of using OSGi services.

Within the domain there is always one node that plays the role of the domain master. The master is leveraged with a *TargetManager* component (Node 1 in Figure 4). The *TargetManager* component exports a deployment service for the SCA deployment domain.

This completes the deployment architecture with regards to the deployment runtime. This infrastructure is complemented with a deployment planner service that is used to carry out deployment operations over SCA domains. The planner service prepares plans matching SCDL-extended descriptors to resources in the deployment environment. Plans are dispatched to the domain, using the deployment service offered by the TargetManager component. Finally, the *TargetManager* is responsible for carrying out the deployment actions within the domain.

6. Conclusions

This paper has presented the deployment architecture for service components for e-banking contexts with two particular characteristics: distribution and heterogeneity in the deployment environment. The chosen solution is based on industrial-strength specifications, mainly Oasis SCA, OMG D&C and OSGi. The main focus of the paper has been on the SCA-leveraged runtime. Because of space limitations the description of the server side of the deployment architecture, which controls the execution of the deployment operations, has been left out of the scope of this paper.

References

[1] Timmers, Paul (1998). Business Models for Electronic Markets. Electronic Markets, 8 (2), 1019-6781. Available at http://www.informaworld.com/10.1080/10196789800000016

[2] T. Erl, “Service-Oriented Architecture. Concepts, technology and design.” Ed. Prentice Hall., 2005

[3] OASIS, “Universal Description, Discovery and Integration (UDDI) specification”, v3.0, available at http://www.oasis-open.org

[4] M. Treiber, S. Dustdar, “Active Web Service Registries”, IEEE Internet Computing, Oct 07

[5] J Garofalakis, Y Panagis, E Sakkopoulos, A Tsakalidis *Web Service Discovery Mechanisms: Looking for a Needle in a Haystack?*,- International Workshop on Web Engineering, 2004DPWS

[6] F. Jammes, A. Mensch, H. Smit, “Service-Oriented Device Communications using the Devices Profile for Web Services”, Proceedings of the 2nd International IEEE Workshop on Service Oriented Architectures in Converging Networked Environments. Canada, May 2007

[7] E. Guttman, “Service location protocol: automatic discovery of IP network services”, IEEE Internet Computing, Jul/Aug 1999

[8] N. Mitra, Y. Lafon, ”Simple Object Access Protocol SOAP) ” Version 1.2 (Second Edition), W3C Recommendation, 2007, available at w3c.org

[9] P. Reveliotis and M. Carey. Your enterprise on XQuery and XML schema: XML-based data and metadata integration. Proc. of the 3rd Intl. Workshop on XML Schema and Data Management (XSDM), 2006.

[10] D.F. Ferguson, M.L. Stockton, “Service-oriented architecture: Programming model and product architecture”, IBM Systems Journal, 2005 DOI

[11] OSOA, Service Component Architecture Specification, version 1.0, available at http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications, March 2007

[12] Open Services Gateway Initiative. "OSGi Service Platform," Specification Release 4.0, August (2006).

[13] Deployment and Configuration of Component-based Distributed Applications Specification. OMG formal specification version 4.0 formal/06-04-02, 2006

Acknowledgements

ITECBAN is an IT innovation project partially funded by CENIT (a Spanish public R&D program). The authors are grateful to MITIC (Ministerio de Industria, Turismo y Comercio) and CDTI (Centro para el Desarrollo Tecnológico e Industrial) for supporting ITECBAN through CENIT.