Distributing OSGi services: The OSIRIS Domain Connector

Jose Manuel Marquez
Telvent
jose.marquez@telvent.abengoa.com

Javier Alamo
Telvent
javier.alamo@telvent.abengoa.com

Juan Antonio Ortega
University of Sevilla
ortega@lsi.us.es

Abstract

This paper presents a way of providing remote capabilities to OSGi services in order to build a distributed environment based on cooperative services. The OSGi Service Platform Specification delivers an open, common architecture for service providers, developers and operators to develop, deploy and manage services in a coordinated fashion. The OSGi Framework forms the core of the OSGi Service Platform Specification. It provides a general-purpose, secure, and managed Java framework whose extensible and downloadable units of deployment, called bundles, can export services and resources (such as packages and classes). However, the mechanisms of synchronization and communication of services allocated in different Service Platforms do not concern to the OSGi Service Platform Specification. This paper presents a new distributed service oriented architecture and its possible industrial applications and it benefits from previous European R&D projects in which the authors have participated.

1. Introduction

Nowadays the adaptation in information systems is an open problem, in particular, how to dynamically adapt different services to the user's needs. This is one of the main reasons of the huge impact that SOA has had. But the heterogeneity of these services makes necessary the definition of a consistent way of integrating their business logic through an intuitive and rich application interface.

The support of a wide variety of client devices is an obstacle that current information systems must solve. One of the most important tasks of this support is related to the adaptation of the presentation layer for rich Internet applications deployed on different devices, specially mobile (resource-constrained) devices. Besides, today more than 55% of all Web sites contain JavaScript. This dramatically affects the ability of people with disabilities to access Web content [1], making the adaptability a mandatory issue. In this context, XHTML 2 is evolving as an application middleware layer which can be used to generate (X)HTML content for a variety of mobile devices and user agents [2].

Information systems are constantly evolving and user needs are continuously changing, specially in contexts such as lifelong learning[1]. Patterns of learning, living and working are changing apace. This does not only means that individuals must adapt to change, but equally that established ways of doing things must change too [3].

How to dynamically deploy, integrate and orchestrate services is an interesting challenge that could have a major impact in the information systems focused on this changing reality. Projects like OSMOSE[2] and OSIRIS[3] address these problems within two different contexts respectively: stand alone and distributed services platform.

2. Scenario

The proposed scenario is given by an e-learning system built on top of an OSGi framework such as it is presented in [4]. In this scenario, educational contents, packaged as the IMS-CP specification defines, are delivered as OSGi bundles. These bundles are

[1] The European Commission and the Member States have defined lifelong learning, within the European Employment Strategy, as all purposeful learning activity, undertaken on an ongoing basis with the aim of improving knowledge, skills and competence.

[2] OSMOSE: Open Source Middleware for Open Services in Europe, was a project of the Eureka-ITEA program financially supported by the Spanish Ministry of Science and Technology through the PROFIT program. www.itea-osmose.org/

[3] OSIRIS: Open Source Infrastructure for Real-time Integration of Services, is a project of the Eureka-ITEA program financially supported by the Spanish Ministry of Industry, Commerce and Tourism through the PROFIT program. www.itea-osiris.org/

considered the nodes of a variable learning itinerary that can be adapted to each student's needs from the information provided by the rest of students. The adaption process uses Ant Colony Optimization and Bayesian Networks.

A smart LMS could recommend or impose the next course, taking into account the most recent results of the user in his/her passed courses and the most successful path taken by most of the users. This artificial intelligence based on the collective behavior of decentralized, self-organized systems has been named as swarm intelligence, an expression introduced in the context of cellular robotic systems [5] by Gerardo Beni and Jing Wang, inspired in the social behavior of animals [6].

Let us suppose that an enterprise wants to prepare a worker to employ him in a vacancy of higher responsibility. This vacancy requires a set of competences being owned by the applicant, and the way to achieve these competences is to pass some specific courses. With this goal, the company's pedagogical team has designed the following graph with several alternative learning paths (Figure 1) in which all the available learning paths a learner may follow is given. So, nodes represent courses and arcs are transitions between courses. Arcs rounded a curved arrow labeled with an ampersand (&) indicate that their children have to be crossed in the order expressed by the arrow (e.g. deep-first and from left to right) and finally, arcs rounded with a curved arrow labeled with a plus sign means that one of the children paths has to be chosen. Each node contains educational contents and exercises, usually embedded in web pages, and an evaluation test, which will send the final qualification of the student to the LMS.

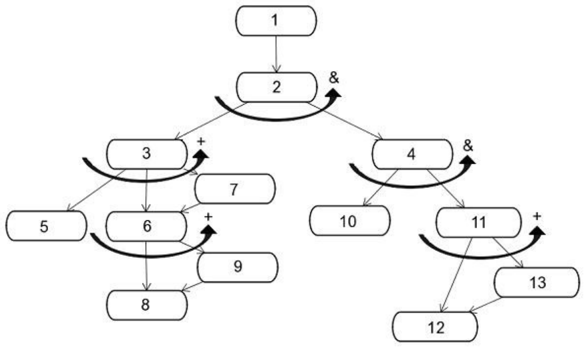

Figure 1: Sequencing graph with alternatives learning paths

Behold the stand alone context. Now, let us think in the system's architecture: How many requests the web server will have to attend? Which impact will have the calculation of the fitness function of each arc? Summarizing, how much CPU load and, consequently response time, will increase? Thus, in order to maintain a good scalability factor, all the process that have to be carried out in the server side to calculate the fitness function defined in [4] should be allocated in different nodes, then defining a distributed architecture, just like is proposed by the OSIRIS project. This distributed service oriented architecture permits the nodes to be continuously enhanced with new plug-ins (OSGi bundles), resulting in a distributed system with dynamically adaptive functionality.

The solution proposed consists of a set of OSGi nodes that acts like specialized processors and a one OSGi-based web server node which gets all the user requests and delegates the process of the learning graph in the rest of those specialized OSGi nodes within the same local area network. Besides, OSGi services may invoke a remote service in a transparent way, so that developers do not have to differentiate which kind of service are they going to use (local or remote). This approach, addressed in the OSIRIS project, has to overcome problems related with the connectivity middleware needed to inter-communicate services allocated in different nodes. In order to solve them, the OSIRIS Domain Connector was defined within the OSIRIS project and how to use it in the scenario presented in [4] is explained next in this work.

3. The OSIRIS Domain Connector

The goal of the OSIRIS project is to define and implement an across-domain open source service platform that will provide support for services provisioning, aggregation, delivery, dynamic adaptation to the context and lifecycle management. It also try to define a mechanism for integrating several OSGi Service Platforms within the same local area network so that services are able to invoke any other remote service in a similar way as OSGi defines for local services. This is the main reason which RMI has been chosen. The OSIRIS Domain Connector uses RMI to offer a transparent way of using OSGi services as if they were placed in the local OSGi framework. This idea holds the concept of the OSIRIS Domain.

3.1 Vocabulary

The OSIRIS Domain is composed of a set of nodes sharing a multicast address-port pair. These nodes contain an OSGI framework implementation with several remotely accessible services running on top of them. These are what we call **OSIRIS Services**.

As we have conceived it, an OSIRIS service is slightly different from an OSGI service. Within OSGi, any object can be registered as service, but in the purposed scenario, we need to be able of remotely invoking services allocated in different nodes, so a way of remote access has to be given to this kind of services. A way of achieving it could be just implementing the `java.rmi.Remote` interface. This could be enough to establish the basis in order to build a distributed system based on OSGi.

RMI (Remote Method Invocation) is part of the core Java API. It allows the programmer to call methods from a remote object without the need of socket handling, which reduce the overall complexity.RMI is used by the most widely accepted multicast Java tools, such as JGroups[4], in order to synchronize data among servers of the same cluster.

In a similar way, the OSIRIS Domain Connector (ODC) was developed to make possible having a cluster of OSGi nodes collaborating within the same local area network.

Each node must have the ODC service running in order to be automatically connected to the rest of the nodes. The ODC is based in the concept of "a virtual bus" made up of two properties: a multicast ip address and a port number, defined by the properties `odc.bus.ip` and `odc.bus.port` respectively. Using this bus all the nodes may send and receive messages (point to point messages or broadcast messages).

3.2 Features

- **Life cycle control**: The ODC provides an API to control the services' life cycle with methods for starting, stopping, registering, unregistering and publishing services.
- **Domain search**: The ODC also provides methods for looking up services and browsing (request all the service names) nodes of the domain. Developers may find services by service name, by node name or both.
- **Remote invocation**: It allows to get a remote reference of any OSIRIS service, in order to remotely invoke it.
- **Synchronization of information**: The ODC ensure the real-time synchronization of available services, sending broadcast messages containing the event occurred (new service registered, service started, service stopped, service published, node added, node removed from the domain, etc).
- **P2P Communication:** The ODC make possible the communication among several nodes using peer to peer messages. A simple chat tool could be implemented in an very easy way using the ODC.
- **OSGi-like implementation**: The way to register OSIRIS services is similar to the way of registering OSGi services. OSIRIS services are just a special kind of OSGi services. In order to accomplish with this premise, the OSIRIS Domain Connector registers several OSGI services, that can be used by developers for requesting OSIRIS services (through the `ServiceResolver` interface), registering its own OSIRIS services (using the `ServiceRegister` interface), etc. The OSGi service named `OSIRISService` provides references to mentioned interfaces, and it can be considered the entry point of the ODC.

The ODC uses RMI to send messages and to access remote services, so RMI `registry` has to be activated before registering and publishing remote services.

4. Using the ODC

4.1 Registering and resolving remote services

First of all, the `OSIRISService` instance have to be obtained querying to the bundle's context:

```
ServiceReference ref =
context.getServiceReference(
        OsirisService.class.getName());
OsirisService os = (OsirisService)
        context.getService(ref);
```

Using the OsirisSpace used by the object OsirisService we can get both ServiceRegister and ServiceResolver as explained next:

```
ServiceRegister register =
      os.getSpace().getRegister();
ServiceResolver resolver =
       os.getSpace().getResolver();
```

4.1.1 The ServiceRegister interface

The `ServiceRegister` interface define methods for adding and removing services, and for making them public to any node present in the service network

- **`addService`**`(Service impl)`
- **`removeService`**`(Integer instanceId)`
- **`publish`**`(Integer instanceId)`

[4] http://www.jgroups.org

- **unpublish**(Integer instanceId)

ServiceRegister interface offers the needed functionality for controlling the life cycle of services with remote communication capabilities.

4.1.2 The ServiceResolver interface

In the other hand, ServiceResolver define methods for discovering and getting instances of remote services:

- **findService**(String service)
- **findLocalService**(String service)
- **getNodes**()
- **getServices**(String node, String service)
- **browseNode**(String node)

Two first methods return the instance of the service whose name is passed as parameter. The first one finds it around the whole domain and the second method only look for the local node. The third method gets an array containing the names of all the nodes running in the domain. The getServices method returns all the instances compliant with the service name passed and running in the given node.

Finally, the last method returns the name of each service running in the given node.

ServiceResolver interface provides the needed functionality for searching the domain.

Both interfaces, ServiceRegister and ServiceResolver, are implemented by the same class, ServicesManager, which consequently is the entry point of ODC for any other bundle. In any case, third parties may code their own implementations.

4.2 How to implement an OSIRIS service

The only extra thing to be done while programming a bundle in order to implement an OSIRIS Service consists on implementing the Service interface, which extends from java.rmi.Remote and it have just only one method:

- **getName**(): returns the name of the service.

The main goal of this service is to be able of differentiating Osiris services from other common OSGi services when casting a ServiceReference instance. An important concept that developers should have in mind is that the Service interface extends the java.rmi.Remote interface, so they will have to throw RemoteException in every method you will define. Also, developers will have to make a client bundle that could be given to anybody who were interested in the use of that OSIRIS Service. This client bundle must contain all the remote interfaces and any class used on them. Just like developing an RMI application.

4.3 Invoking an OSIRIS service

In order to invoke an OSIRIS service, running on a remote OSIRIS node, the first thing to be done is to use the remote interface of the service to obtain the service name:

```
String name =
RemoteService.class.getName();
```

Then, this service name can be used with the class ServiceResolver to obtain a remote instance of the object and call its methods directly. There is several options for choosing the remote service from the right node.

a) If the name of the destiny node is specific then just request to ServiceResolver one of the instances running there:

```
resolver.getServices(node,service);
```

b) If the service is running on the local node, just indicate it to the ServiceResolver and all the local instances of the given service name will be returned:

```
resolver.getLocalServices(service);
```

c) When the remote node where service's instance remote will be obtained from does not mind, then the method findService may be used:

```
resolver.findService(service);
```

All these methods returns an instance of the required service (whose name is passed as a parameter) or throw NotSuchObjectException if any instance is found. The instance returned may be used as if it was a local one, invoking its methods normally.

5. Synchronization of information

In the proposed scenario we have a network with nodes that public a set of services (OSIRIS Domain). Each node must have information about the available services on the web. For that purpose each node has a table with the needed web information. This information must be synchronized so often[5] by a polling process. We must notify changes in the states of any OSIRIS service, to all the nodes just in runtime.

[5]To learn more about the frequency of synchronization process, please refer to [8]

This implies the design and implementation of a synchronization mechanism. This synchronization mechanism can be designed and implemented based on very simple bus. Using this bus, it can be created one or more multicast sockets that all the nodes must know in order to send and receive messages (point to point messages or broadcast messages).

The ODCActivator is the activator of the ODC bundle. It implements the BundleActivator interface. Its start method creates an OsirisSpace object and pass it as parameter to start an OsirisService. OsirisSpace represents the execution space of the ODC, while OsirisService is responsible of starting and stopping the services registry of the ODC. The OsirisService will be registered as OSGi service by ODCActivator after being started. This service is responsible of catching up all the existing remote instances of Osiris Services, and maintaining it in a well ordered structure. This structure has been named ServiceTable.

A ServiceTable object contains the information of the running instances of each service, allowing searches of these by node or by service. In order to be registered in the ServiceTable a Service have to implement the Service interface. This is a very basic interface and its unique purpose is to ease the identification of Osiris Services and its differentiation from other non specific Osiris services.

The ServiceResolver and ServiceRegister interfaces are focused on managing instances of services, that is adding, removing, publishing and unpublishing services, as well as searching them and getting references to any service in any node of the service network. These interfaces define the functionality needed to implement the OSIRIS Service Registry, so third parties may implement its own Service Registry. In any case, the ODC provides a default implementation in the ServiceManager class. Another functionality of the ODC consists on periodically polling the network in order to know how many nodes are connected and sending broadcast messages to advertise the changes produced by the management process.

The ServiceRegister interface define methods for adding and removing services, and for making them public to any node present in the service network. In the other hand, ServiceResolver define methods for discovering and getting instances of remote services.

Both interfaces, ServiceRegister and ServiceResolver are implemented by the ServiceManager class. This class manages the services registration and localization. In order to accomplish this task it uses a helper class for storing the information related with the different instances of services. This helper class is ServiceTable.

ServiceTable deals with Instances, both LocalInstances and RemoteInstances, and, if the node is public, it can be periodically caught up by the Synchronizer. The ServiceTable is constructed by the OsirisService for each node and passed to the ServiceManager and Synchronizer in order to have a synchronized copy of it in each node. Obviously, instances references to Osiris services, that is, they contain an attribute with the implementation of the Service interface. They also contain the RMI registry where the service resides.

The Synchronizer object is responsible of the synchronization among all the ServiceTables in the service network. To do it, Synchronizer trust on two controller classes: BusController which implements MessageListener in order to process incoming messages, and TimerController which implements TimerListener in order to trigger events for polling the service network and sending synchronization messages.

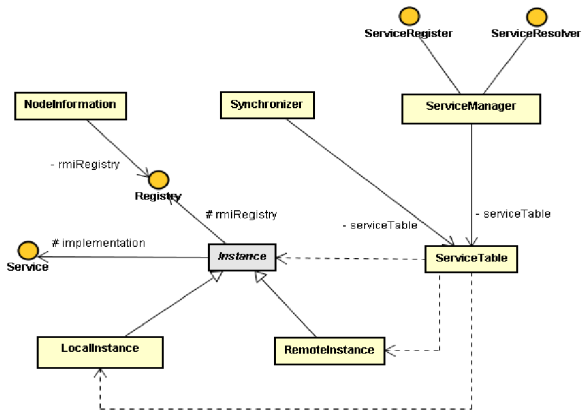

Figure 2: The Regservice's class diagram

6. Applying ODC in a e-learning scenario

With the ODC running in every node, an OSGi-based application may distribute its tasks around its local area network, making development of OSGi services easier for programmers. This approach is specially useful in those cases where execution of a huge amount of arithmetic operations and many simultaneous threads are considered critical system's requirements, when specialized functionalities have to be placed in separated and dedicated servers. Thus, very complex tasks, such as calculating the amount of pheromones to be dropped and the suitability factor of each arc, as it is explained in [4] can be carried out by

different processors, working simultaneously with the same goal: to calculate the fitness function. Thanks to the discretization of the time, described in a later work [7] by the same authors, synchronization of these tasks turn easier and make possible a collaborative work between that servers and the LMS.
ACO Server uses an Ant Colony Optimization algorithm in order to calculate the best path in a the learning tree of the Figure 1. It considers learners as ants moving from one node to another in the tree, dropping a variable amount of pheromones in the arc ij, depending of the level of success reached by the user in the course of the node i. The total amount of pheromones dropped by all the users in a concrete arch will be helpful for other users stayed in a node with more than one output arc they could choose.
By the other hand, the Bayesian Server applies the Bayes Theorem to calculate the most suitable path that a learner should take, taking into account the historical performance of his/her equals in previous academical courses/training experiences.
The results given by both servers will be added and it will return the final value of the fitness function for each arc. The system is finally made up of three OSGi nodes whose have been provided with the ODC bundle that offers them communication capabilities through the ODC bus. This bus is used mainly for sending multicast messages but also for requesting specific measures and data, taking advantages of the fast protocol used.
Scalability of the OSIRIS domain is granted by the ODC, because any OSIRIS node may join the bus, and consequently, it may be added to the OSIRIS domain at any time. for each arc taking the learning tree and historical data of the learner in the involved course (tracking information), taking the historical data and learner profiles as inputs, and other tasks

7. Conclusions and further work

ODC is only the base piece for transparent cooperation among the OSIRIS nodes. Other services may use its API for implementing more complex tasks. This is the case of ISIS, a sub-project that is currently being developed within the OSIRIS Project. ISIS is responsible of the high-level integration of services in order to share data. Data and events may be added in a declarative way, just in run-time, making easy the integration of services from different bundles in order to cooperate. This solution was designed to avoid dependencies among data consumers and data producers. These kind of dependencies differs from bundle dependencies because of they are not a matter of a Java package or Java Classes dependency but they depends of the existence of some kind of data. Some times, this kind of data cannot be predicted, and it depends only in the kind of data provided by the data producer. This problem will be solved by a data and event publication mechanism implemented by the ISIS Project.
Other future steps will be: to implement ISIS wrappers providing a http and Web Service-based access to the OSIRIS Domain's services, data and events through the Proxy Node, applying security policies based on digital certificates and identity federation.

8. References

[1] M. Birbeck, J. Axelsson, S. Pemberton, B. Epperson, S. McCarron, M. Ishikawa, A. Navarro, M. Dubinko. *XHTML™ 2.0*. W3C Working Draft (work in progress), 26 July 2006, http://www.w3.org/TR/2006/WD-xhtml2-20060726/

[2] W3C. *Roadmap for Accessible Rich Internet Application. (WAI-ARIA Roadmap)*. W3C Working Draft 4 February 2008. http://www.w3.org/TR/2008/WD-wai-aria-roadmap-20080204/

[3] Commission of the European Communities. *A Memorandum on Lifelong Learning*. Brussels, october 30th, 2000. http://ec.europa.eu/education/policies/lll/life/memoen.pdf

[4] José Manuel Márquez, Juan Antonio Ortega, Luis González-Abril, Francisco Velasco. *Creating adaptive learning paths using Ant Colony Optimization and Bayesian Networks*. In proceeding of 2008 IEEE World Congress on Computational Intelligence (WCCI 2008). Hong Kong, June 2008.

[5] G. Beni, J. Wang. *Swarm Intelligence in Cellular Robotic Systems*. In Proceedings of NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, June 26–30 (1989)

[6] P. Miller. *Swarm Theory*. National Geographic Magazine. July 2007

[7] José Manuel Márquez, Juan Antonio Ortega, Luis González-Abril, Francisco Velasco. D*efining adaptive learning paths in competence-oriented learning.* In proceeding of IADIS International Conference e-Learning 2008. Amsterdam, 22nd-25th July, 2008.

[8] J. Cho, H. Garcia-Molina. Synchronizing a database to Improve Freshness. *In Proceedings of 2000 ACM International Conference on Management of Data (SIGMOD) Conference*, May 2000.