Deployment of software services in the power distribution context

Philippe Lalanda, Antonin Chazalet, Vincent Lestideau

Deployment of software services in the power distribution context

Philippe Lalanda, Antonin Chazalet and Vincent Lestideau

Laboratoire LSR-IMAG, 220 rue de la Chimie
Domaine Universitaire, BP 53
F-38041 Grenoble, Cedex 9, France
{Philippe.Lalanda, Antonin Chazalet, Vincent Lestideau}@imag.fr

Abstract

Innovative machine-to-machine infrastructures have been recently defined to integrate IT applications and industrial applications. Many of them are based on Service-Oriented architectures. In this paper, we focus on three-tier architectures including OSGi gateways to connect field devices and the Internet. We present a deployment manager automating as much as possible the deployment operations.

Key words: Service-oriented architectures, service deployment, OSGi gateways, power distribution.

1. Introduction

Manufacturing enterprises have to face demanding new environments where market requirements are changing frequently, new technologies have to be regularly integrated and fast time-to-market has become a major business requirement. As a consequence, a broad range of industries — from manufacturing to utilities — must be able to seamlessly integrate application software that supports business services with control software implemented by field devices on the plant floor. Such computing elements have been separate until recently, primarily because of technical issues, including incompatible programming paradigms, network heterogeneity, differing time scales, and the lack of appropriate integration tools. With the Internet's emergence and the proliferation of smart communication devices, stronger coupling between previously autonomous activities is now possible.

When we examine the resources that numerous companies are investing in such technologies, we can see that the stakes are substantial. Indeed, linking business and operational processes will affect manufacturing enterprises along several dimensions. First, it will dramatically speed up decision-making by providing executives and/or decision software with accurate, up-to-date and appropriately formatted information, which is rarely the case today. Second, it will allow faster response to changes. Flexibility is today crucial in markets that are moving from mass production to a more customized production and where business decisions have to be implemented more rapidly than ever. Third, it will give the opportunity to significantly improve capital asset management and maintenance optimization which represent today an heavy cost for most companies. Finally, it paves the way for innovative e-services based on data regularly collected on the plant floor.

However, it is also clear that this goal of seamless integration is far from easy to achieve. It requires to build Internet-scale distributed systems in complex, heterogeneous environments characterized by stringent requirements regarding security, scalability and flexibility. In order to meet these requirements, innovative distributed architectures have been proposed recently. These architectures rely on the notion of service-oriented components which are distributed on the business and operational sites. This solution, adopted by major manufacturers (Schneider Electric, Siemens and ABB for instance), makes a better use of the available resources and is able to evolve nicely. But it is also complex. In particular, it introduces the need for remote deployment of software components.

The purpose of this paper is to present a service-oriented architecture and a deployment manager that has been developed in the power distribution context within the PISE[1] project. The paper is organized as follows. The coming section introduces a service-oriented architecture for power distribution. Section 3 presents a deployment manager automating the deployment activity. Section 4 presents an example. A conclusion summarizes this work and presents coming actions.

[1] PISE is supported by the French Ministry of Industry under the RNRT program. It is conducted by Schneider Electric in collaboration with France Telecom, Trialog, Grenoble University and the INRIA.

2. Service Oriented Architectures

Innovative machine-to-machine infrastructures have been recently defined to integrate distributed, heterogeneous applications. They primarily target IT applications (for B2B considerations for instance) but also IT and industrial applications [1, 2]. In the latter case, architectures are generally structured into three tiers (see Figure 1):

- The first tier corresponds to smart field devices that can communicate through field buses.
- The second corresponds to gateways that connect the devices to the Internet. Gateways perform local computation and mediation operations [3].
- The third corresponds to applications that run on IT servers.

Figure 1: Integration architecture.

We believe that Service-Oriented Computing (SOC) [4, 5] provides the level of flexibility and scalability that is needed in these architectures. SOC is based on the concept of service. A service can be defined as a particular resource offered by a software system that is made available to third parties. Services must declaratively define their capabilities and requirements in an agreed (standard) machine-readable format. Based on service specifications, automated service discovery, selection and binding can then be performed at run-time. The flexibility of the SOC approach essentially comes from this runtime, dynamic binding.

The SOC approach has been indeed recently used in architectures integrating IT and industrial applications. Different solutions have been proposed:

- The UPnP technology (see www.upnp.org) has been used to implement dynamic interactions between field devices that can be seen as service providers and requesters.
- The OSGi technology (see www.osgi.org) has been used at the gateway level in order to run flexible, dynamic applications.
- Web services (see www.W3C.org) has been used as the interaction protocols between applications.

In this paper, we focus on OSGi gateways. These gateways play a crucial role in the global architecture. They run service-oriented applications connecting devices and IT but also orchestrating the devices actions. Let us remind that OSGi is an open service platform defining a minimal component model, a small framework for administering components and a set of standard services. Components are packaged in bundles, which are the deployment units in the OSGi model. The framework also defines mechanisms that facilitate the dynamic installation, activation, deactivation, update, and removal of bundles.

Using a service-oriented framework like OSGi allows developpers to implement highly flexible applications where devices and applications can change over time. Unfortunately, it also brings a significant level of complexity. In particular, the deployment process is complex. It is, of course, necessary to regularly deploy new bundles on the OSGi gateways for maintenance purposes or to provide new services. This task rapidly becomes fastidious and error-prone. Deployment is actually an intellectually challenging task for several reasons. It has to take into account the current states of the gateways to deploy the best service implementations, to resolve the contract and code dependencies of the implementations, to share services if it makes sense, to stop services that are no longer used, to generate the appropriate life-cycle events, etc.

We argue that a deployment manager is needed in order to automate as much as possible these operations. Such a deployment manager has also to meet the following requirements:

- Security. Deployment actions can only be done by authorized persons. Downloaded files may be sensible and must be protected.
- Reliability. Deployment is an important process that must be done in a reliable and controlled fashion.
- Transparency. Although kept in the loop, the human deployer must concentrate on added value tasks regarding the deployment process. Technical details (network, administrative data, etc.) must be hidden.
- Standardization. Techniques implemented by the deployment manager must be based on standards.

3. Deployment manager

Schneider Electric has developed a distributed infrastructure allowing the development of added-value services using data generated by power distribution devices installed in customers' plants. The infrastructure implements a three-tier architecture like the one previously presented (see figure 1):

- the first tier corresponds to smart power devices,
- the second tier corresponds to smart OSGi gateways connecting devices and the Internet,
- the third tiers corresponds to Internet application servers implementing the business services.

The infrastructure integrates a large number of plants. This represents many devices and gateways. In addition, this number can change over time when new customers arrive. In the coming sections, we present a deployment manager that has been designed and tested in this context. Specifically, the manager automates the deployment of OSGi services on the smart gateways.

3.1 Overview

As illustrated on Figure 2, the deployment manager is structured into three main components:

- A « Collector » gathers contextual information that is kept at the gateways level. This includes information about the gateways (capabilities, hosted services, services status, etc.) and, possibly, about the communication infrastructure. Captured data are stored in a database.
- A « Planner » calculates deployment plans. Plan calculation is centralized and total which means that calculation is entirely made at the deployment manager level. Plans are stored in an XML file.
- A « Plan Manager » is responsible for the correct execution of the deployment plans.

The global behavior of the deployment manager is the following. At any time, the system administrator can specify a list of OSGi bundles to be deployed on a set of gateways. The specification is done through an XML-based editor and stored in a file called "Bundles.xml" which is submitted to the planner. Bundles are identified by their location (URL) whereas gateways are determined by a unique key (IP address).

The Planner also uses the « Repository.xml » file and the « Gateways.db » database to compute a deployment plan. The first file specifies the OSGi bundles which are available for deployment. In the Schneider Electric case, these bundles have been certified beforehand so that they can be safely installed on the customers' gateways. The description of an OSGi bundle is essentially made through its manifest which, in particular, provides information about its location and its dependencies. As we will see later on, the treatment of the bundles dependencies is one of the main tasks of the planner. This « Repository.xml » file is currently produced and updated by hand by the system administrator. On the contrary, the « Gateways.db » database is automatically filled by the Collector component. This database contains information about the current status of the OSGi gateways and maintains an history.

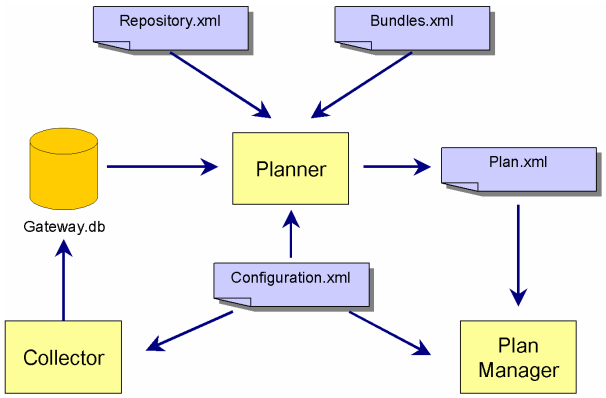

Figure 2. Architecture of the deployment manager

Given these inputs, the Planner calculates a deployment plan specifying all the bundles to be deployed, their destinations and a list of installation directives. At this step, dependencies have been resolved and the bundles can be activated after deployment. Then, the Plan Manager takes care of the realisation of the plan. We will see that part of the plan is executed at the gateway level for efficiency purposes.

The « Configuration.xml » file plays an important role in this architecture. It contains global parameters that are used by the three components. In particular, it specifies the localization of the « Repository.xml » file and « Gateways.db » data base, the communication protocols between the deployment manager and the gateways (RMI and HTTP/SOAP are today available), the frequency of the administration actions, etc.

3.2 Collector

The domains we are investigating are characterized by heterogeneous and very dynamic environments. In particular, the OSGi gateways differ from one customer to the next in terms of hardware and installed software. It is then of major importance for a deployment manager to know the current gateways characteristics in order to make the good deployment decisions. In the Schneider Electric

case, we have identified three types of information that intervene in the deployment process:

- The « system context » contains information about the operating system (OS) of the gateways. This includes the OS name, its version, the Java Virtual Machine (JVM) version, its computing state, etc.
- The « framework context » contains information about the OSGi framework installed on the gateway. This includes the framework identification (vendor, name), its version, etc.
- The « bundle context » contains manifest-like information on each installed or running bundle of the gateway.

These types of information are regularly collected on the gateways and used by the planner. The collection process has been implemented in accordance with the JMX standard. JMX is an extension of the JAVA language, developed by Sun MicroSystems (see http://java.sun.com/products/JavaManagement). It was introduced to support the administration of different kinds of resources. A resource may be an application, an object, a service, a device, etc. The only required condition is that the resources need to be modeled or instrumented, to match with the pattern of a manageable JAVA object. JMX defines a management architecture, an API for application development and a set of administration services. The architecture proposed by JMX is based on three levels:

- The first level defines how to model or instrument the resources. The instrumentation is based on objects called MBeans (Manageable Beans).
- The agent level specifies how to implement the agents, which control the MBeans and give them out accessible to administration's applications.
- The adapter level provides needed mechanisms, so that the distributed administration's applications can communicate with the agents.

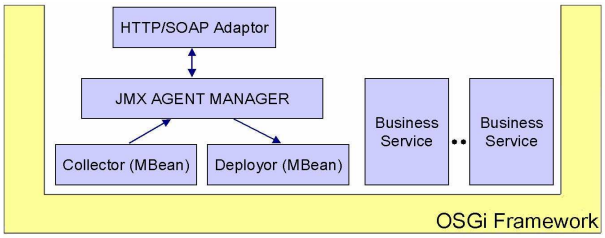

Figure 3. OSGi Gateway

As illustrated by Figure 3, we have implemented this three-level architecture in the OSGi context. We have developed an HTTP/SOAP adaptor for the communication with the deployment manager. This protocol was mandatory to meet the security policies set by the customers. Indeed, the industrial environments that we must take into account use routing solutions with access lists and firewalls (this means that, in our case, RMI-like protocols don't work and that only HTTP works). We have also developed a JMX MBean called collector which purpose is to collect the three types of information previously mentionned (system, framework and bundle contexts). Through the Java language and the OSGi framework, this MBean has access to all the necessary information. It is directly invoked by the deployment manager via an interface exposed by the JMX agent manager (provided by Sun MicroSystems). These different JMX elements are packaged as OSGi bundles so that they can be updated easily, like any other component run on the gateway.

3.3 Planner

Our planner is called GDF, for Generic Deployment Framework. It has been developed in the Osmose European project in order to automate the deployment of software services (see http://www.itea-osmose.org).

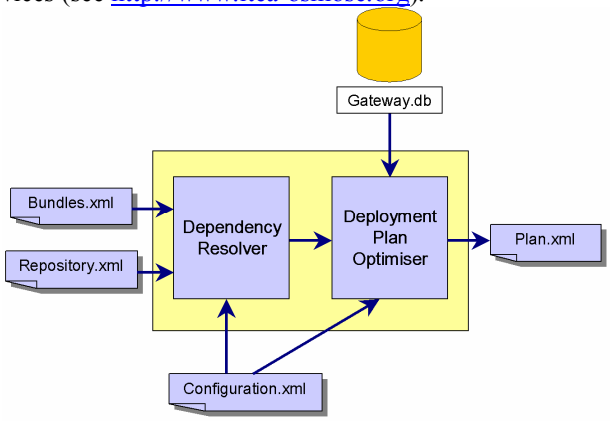

Figure 4. GDF Planner

GDF builds on the notion of dependency in order to express constraints on the services to be deployed. On that point, it can be related to the OMG proposal on the deployment and configuration of component-based distributed applications [6, 7]. Dependencies can be specified between services in order to express code dependencies or between services and execution platforms in order to express resources contraints (like the memory needed for a service to be properly executed). Dependencies resolution is platform independent and different algorithms can be used according to the deployer policies.

In our context, GDF is applied to the deployment of OSGi bundles. The challenge here is to allow the administrator to simply specify the bundles he needs to deploy without considering code dependencies and resource constraints. It is up to the planner to select the right bundles and to initiate administration actions on the gateways.

GDF works in two steps. First, it analyses the bundles specified in the « Bundles.xml » file and carries out a resolution of the import and export packages and import and export services using the « Repository.xml » file. This first step produces a temporary deployment plan comprising the whole set of bundles to be deployed. A plan is basically expressed in terms of instructions like install, start, update, etc. Then, the second step uses information about the gateways to optimize the temporary deployment plan. Several strategies can be used here for optimization. For instance, when bundles to be deployed are already running on a gateway, we may or may not re-install them (to take advantage of a newer version for example). Such strategies are not explicit today: we are actually working on innovative representation of such knowledge to better manage the optimization strategies [8].

3.4 Plan manager

The execution of the deployment plan produced by the planner is distributed in the sense that part of the plan is executed on the gateways. The purpose of the Plan Manager is to interpret the deployment plan and to send specific and optimal plans to the gateways. The role of the gateways is then to execute the OSGi instructions specified in the received plan.

Here again, the deployment process has been implemented in accordance with the JMX standard. More precisely, a JMX MBean called deployor has been installed on every OSGi gateway (see figure 3). It is invoked by the Plan manager through an interface exposed by the JMX agent manager. Then, it executes the plan, which is passed as a parameter, using OSGi instructions. Once this execution is made, the deployor MBean informs the Plan Manager of the state of the deployment plan execution.

The deployment process is controlled by the deployment manager: interfaces provided by the JMX agent manager allow it to follow the evolution of the process. If something goes wrong on a given gateway (uncompleted installation, communication failure, etc.) several strategies have been implemented regarding the whole deployment process. In general, the failure of one gateway does not stop the global deployment. Once again, strategies are not explicit today and we are working on solutions to make them more independent of the code.

Similarly to the collection process, the deployment process is initiated by the gateways. To be in accordance with most customers security policies, the gateways periodically calls the deployment manager to start the deployment process. The calling period is dynamic.

4. Example

As previously said, the deployment manager has been tested in the power distribution field. Specifically, it has been implemented on the top tier of the infrastructure with the Eclipse environment.

Let us now see an example of deployment that has been treated in the project. In this example, the administrator needs to deploy an Alarm service which purpose is to use data coming from two kinds of devices, namely PowerMeters and CircuitMonitors (see www.schneider-electric.com for specifications), in order to detect voltage dips at the electrical supply source. To do so, the administrator specifies in « Bundles.xml » the localization of a bundle implementing the service (say http://user-site/repository/voltage-dip-alarm.jar) and the IP address of the targeted gateways (say 129.88.103.25).

An interesting feature is that the bundle implementing the Alarm service needs two other services to run: the PowerMeter service and the CircuitMonitor service. These two services make use of a Modbus driver to collect data on the actual devices and to format them in a high level language. These dependencies are expressed in the manifest of the bundle implementing the Alarm service in the import service section (in this example, there is no import package dependencies).

Upon receiving the « Bundles.xml » file, our deployment planner identifies the dependencies and looks up for the missing packages in the « Repository.xml » file. In our example, they are stored in http://user-site/repository/PowerMeter.jar and in http://user-site/repository/CircuitMonitor.jar. Then, examining the « Gateways.db » database, the planner learns that the PowerMeter is already installed, but not started on the targeted gateway. It then produces the following deployment plan:

Install	http://user-site/repository/CircuitMonitor.jar
Install	http://user-site/repository/voltage-dip-alarm.jar
Start	http://user-site/repository/CircuitMonitor.jar
Start	http://user-site/repository/PowerMeter.jar
Start	http://user-site/repository/voltage-dip-alarm.jar

This plan is sent to the Mbean deployor of the targeted gateway. Then, it is executed. Appropriate bundles are sequentially installed and started. A status message is finally sent to the deployment manager that displays the result to the user.

5. Lessons learned and conclusion

The integration of IT and industrial applications is one of the great challenges of today's computing. It requires to build Internet-scale, distributed architectures made of interacting software components performing operations at the most efficient places (near the sensors in some cases). It also requires to meet stringent requirements regarding complexity management, security and flexibility.

The SOC paradigm appears as a promising way to meet these requirements. However, service-oriented computing is today technology driven and thus very hard to master. Deep technical knowledge is needed to design, implement, deploy and maintain service-based applications. Our experience in the PISE project is that most programmers do not take full advantage of the service approaches capabilities.

Another learned lesson is that service oriented architectures used to integrate IT and industrial applications are just too complex to be manually managed. In order to face the inherent complexity of such architectures, it is necessary to bring tools to help administrators to manage the applications life cycle from design to the maintenance [9].

In this paper, we have presented a deployment manager that automates the deployment of software components on OSGi gateways. This solution goes much farther than the usual script-based solutions that are generally encountered in the service deployment domain [10]. This manager, based on standard technologies, meets the requirements presented in section 2:

- Transparency. The all purpose of our solution is to hide low level technical aspects. In particular, the administrator does not have to deal with code dependencies and physical constraints which are resolved by the planner.
- Use of standard. The deployment manager is entirely based on open standards (JMX, HTTP, SOAP), which is also the case for the global infrastructure (JAVA, J2EE, OSGi).
- Security. A salient feature of our solution is the ability to adapt itself to the customers requirements (use of HTTP, gateways are the callers). We cannot however say that our infrastructure is perfectly secure. A lot of work is still needed. In particular, OSGi simply relies on Java security and we have found several limitations (use of call-back in the framework, interaction with native code, resource management, etc.).
- Reliability. The deployment manager brings fault tolerance capabilities for the deployment process. We are studying today the used of message-oriented middleware in order to improve the manager reliability at the communication level.

This deployment manager is currently tested within the PISE project on real settings: pilot sites distributed in several countries are used to assess its features.

Extensions are also needed. In particular, we would like to make service sharing possible at the gateway level. This implies that the deployment manager has to consider services running on a gateway and see if they can be shared by new applications. This raises important issues regarding service state management and, more generally, the management of application life cycle.

6. References

1 F. Jammes and H. Smit, Service-oriented paradigms in industrial automation, IEEE Transactions on Industrial Informatics, vol. 1, no 1, February 2005.

2 P. Lalanda, E-Services Infrastructure in Power Distribution, IEEE Internet Computing, May-June, 2005.

3 P. Lalanda, L. Bellissard and R. Balter, " Asynchronous Mediation for Integrating Business and operational Processes," IEEE Internet Computing, vol. 10, no. 1, 2006, pp. 56–64.

4 M. N. Huhns and M. P. Singh. Service-Oriented Computing: Key Concepts and Principles. IEEE Internet Computing,vol. 9:pages 75–81, Jan./Feb. 2005.

5 SECSE team, Toward service-centric system engineering, Int. Conf. on Service oriented computing, Trento, Italy, 2003.

6 V. Lestideau and D. Donsez, On-demand Service Installation and Activation with OSGi, Fourth Annual ObjectWeb Conference January, Lyon, France, 2005.

7 OMG RFP, Deployment and configuration of component-based distributed applications, http://www.omg.org/cgi-bin/doc?mars/2003-03-04, 2004.

8 P.Y. Cunin, V. Lestideau and N. Merle, ORYA: A strategy oriented deployment framework, 3rd International Working Conference on Component Deployment, November 2005, Grenoble, France

9 C. Marin, P. Lalanda and D. Donsez, "A MDE approach for power services development", Int. Conf. on Service Oriented Computing, Amsterdam, december 2005.

10 V. Talwar et al, Approaches for service deployment, IEEE Internet Computing, March-April, 2005.