Introducing the principles of the Event-Driven and Service-Oriented Architecture (SOA 2.0), and its role in the new interconnected world based on the cloud computing architecture paradigm, this book focuses on how the current and future SOA technologies provide the basis for the smart management of the service model provided by the Platform as a Service (PaaS) layer. In this new context, the concept of "service" is widely applied to the hardware and software resources available in the new generation of Internet.

This book will explore the various evolutions of the middleware communication layer, designed as an adaptation layer intended to hide the complexity of application component distribution. The evolution of the middleware layer will be discussed, with a particular focus on the main paradigm that has guided the design of this kind of system: the Service-Oriented Architecture (SOA) paradigm.

The authors take a practical approach to the topic, called project-based learning, in order to avoid a sequential theoretical presentation of definitions and concepts, which enables them to demonstrate how Cloud Computing IT Platforms and SOA concepts can be efficiently applied when developing current and next generation enterprise applications.

Primarily intended for network and software engineering students, engineers, researchers and experts in service-oriented, event-driven and cloud-computing architectures, this book is also suitable for anybody interested in learning about the evolution of distributed computing environments.

Ernesto Exposito is Associate Professor at the INSA of Toulouse and a researcher at the LAAS laboratory of the CNRS, France. His teaching and research activities include designing, modeling and developing service-oriented, component-based and ontology-driven autonomic transport and middleware communication services.

Codé Diop is a Network and Telecommunications engineer. He is currently preparing his PhD in Service-Oriented Architectures at LAAS-CNRS. His research interests concern QoS management, service-oriented architecture, enterprise service bus, autonomic computing and cloud computing.
Smart SOA Platforms in Cloud Computing Architectures
Dedicado a mis padres, Francisca y Ernesto Expósito

Ma nguíkoymaysama ay wadjour Mame Awa Syak Serigne Diop
Smart SOA Platforms in Cloud Computing Architectures

Ernesto Exposito
Codé Diop
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>ix</td>
</tr>
<tr>
<td>Introduction</td>
<td>xv</td>
</tr>
<tr>
<td><strong>Chapter 1. ESBay Case Study</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1. ESBay: use case description</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1. System overview</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2. Functional requirements</td>
<td>2</td>
</tr>
<tr>
<td>1.1.3. Other requirements</td>
<td>4</td>
</tr>
<tr>
<td>1.2. yPBL inception phase</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1. Functional requirements</td>
<td>5</td>
</tr>
<tr>
<td>1.2.2. Non-functional requirements</td>
<td>6</td>
</tr>
<tr>
<td>1.2.3. Requirements matrix</td>
<td>8</td>
</tr>
<tr>
<td>1.3. Summary</td>
<td>9</td>
</tr>
<tr>
<td><strong>Chapter 2. Service-Oriented and Cloud Computing Architectures</strong></td>
<td>11</td>
</tr>
<tr>
<td>2.1. State of the art of service-oriented architectures</td>
<td>11</td>
</tr>
<tr>
<td>2.1.1. Communication middleware solutions</td>
<td>12</td>
</tr>
<tr>
<td>2.1.2. New orientations for integration and interoperability</td>
<td>18</td>
</tr>
<tr>
<td>2.1.3. Mediation is the solution</td>
<td>22</td>
</tr>
<tr>
<td>2.1.4. SSOAPaaS 1.0 Cookbook</td>
<td>30</td>
</tr>
<tr>
<td>2.2. Evolution of enterprise integration with the event-driven architectures</td>
<td>30</td>
</tr>
<tr>
<td>2.2.1. Event-driven architecture paradigm</td>
<td>30</td>
</tr>
<tr>
<td>2.2.2. Event-driven and service-oriented architecture (EDSOA)</td>
<td>33</td>
</tr>
<tr>
<td>2.2.3. SSOAPaaS 2.0 cookbook</td>
<td>34</td>
</tr>
<tr>
<td>2.3. Performance and scalability of SOA platforms</td>
<td>34</td>
</tr>
</tbody>
</table>
2.3.1. ESB mechanisms for scalability and performance management .............................................. 34

2.4. Smart management of SOA platforms ................................................................. 37
   2.4.1. Cloud computing ............................................................... 37
   2.4.2. Autonomic computing ...................................................... 40
   2.4.3. SSOAPaaS 3.0 cookbook .................................................. 41
   2.4.4. SPaaS cookbook ............................................................ 42

2.5. Summary .............................................................................................. 42

CHAPTER 3. SPaaS 1.0 COOKBOOK .................................................................. 47

3.1. SPaaS 1.0 overview .............................................................................. 47
3.2. Creation of virtual IT infrastructure ......................................................... 48
   3.2.1. Creation of virtual machine hosting the Proxmox ...................... 49
   3.2.2. Installing Proxmox on a VMWare virtual machine .................... 52
   3.2.3. Testing and exploring the Proxmox installation ......................... 55
   3.2.4. Creation of Proxmox virtual components ................................. 57
   3.2.5. Maintenance of the platform .................................................... 62
3.3. Extending the platform ........................................................................ 63
   3.3.1. Cloning the platform ............................................................ 63
   3.3.2. Extending Proxmox virtual appliance templates ...................... 65
3.4. Managing the platform .......................................................................... 66
   3.4.1. Monitoring the Proxmox server and virtual containers using the PVE Web-GUI ................................................................. 67
   3.4.2. Monitoring the Proxmox server and virtual containers using the Proxmox API ................................................................. 69
3.5. Scaling the platform .............................................................................. 72
   3.5.1. Creating a cluster ................................................................. 73
   3.5.2. Virtual component migration ................................................... 74
3.6. Autonomic management of the platform ................................................... 76
3.7. Summary .............................................................................................. 77

CHAPTER 4. SSOAPaaS 1.0 COOKBOOK ......................................................... 79

4.1. SSOAPaaS 1.0 overview ..................................................................... 79
4.2. Using the SPaaS 1.0 .......................................................................... 80
4.3. Adding integrability and interoperability support ..................................... 81
   4.3.1. Creation of an enterprise service bus virtual container ................ 81
   4.3.2. Creation of an application server virtual container .................... 86
   4.3.3. Creation of a database server virtual container ......................... 91
   4.3.4. Creation of an e-mail server virtual container ........................... 93
   4.3.5. Managing OpenESB binding components .................................. 97
   4.3.6. Managing OpenESB service engines ....................................... 100
   4.3.7. Netbeans IDE/Connect to OpenESB installation ....................... 103
Contents vii

4.4. Illustrating integrability and interoperability support of an ESB........ 105
4.4.1. Integrating an application server.............................. 105
4.4.2. Integrating a database server in OpenESB..................... 111
4.4.3. Integrating a mail server in OpenESB......................... 123
4.5. Summary .................................................. 131

CHAPTER 5. SSOAPaaS 2.0 COOKBOOK .................................. 133
5.1. SSOAPaaS 2.0 overview........................................... 133
5.2. Using the SSOAPaaS 1.0........................................... 134
5.3. Adding availability support......................................... 135
5.3.1. Creation of a message-oriented middleware virtual container...... 136
5.3.2. Illustrating the availability support............................. 139
5.4. Adding proactivity support.......................................... 151
5.4.1. Enabling a complex event processing (CEP) engine............... 151
5.4.2. Illustrating proactivity support ................................ 154
5.5. Summary .................................................. 160

CHAPTER 6. SSOAPaaS 3.0 COOKBOOK .................................. 163
6.1. SSOAPaaS 3.0 overview........................................... 163
6.2. Using the SSOAPaaS 2.0........................................... 164
6.3. Adding manageability support....................................... 165
6.3.1. Creation of a monitoring virtual container....................... 166
6.3.2. Deploying the Jolokia agent and creation of a monitoring client 167
6.4. Illustrating manageability support................................... 170
6.4.1. Glassfish administration console-based monitoring.............. 170
6.4.2. JMX console-based monitoring.................................. 173
6.5. Illustrating scalability support..................................... 175
6.5.1. Cluster of ESB instances........................................ 175
6.5.2. Federation of ESB instances.................................... 179
6.6. Autonomic management of the SOA platform....................... 182
6.7. Summary .................................................. 183

Conclusion and Perspectives ........................................... 185

Bibliography ........................................................... 189

Index ................................................................. 193
The complexity involved in the development of the initial distributed applications was considerable in the context of the original Transmission Control Protocol/Internet Protocol (TCP/IP) network model. This complexity has been increasing rapidly with the evolution and diversity of transport and network services. For this reason, developers working directly over the transport layer application programming interface (API) need a high level of expertise on all the services offered by the large set of transport protocols as well as on how these various services can be combined with the underlying services offered at the IP network level.

For instance, a developer of a web application intended to operate on a mobile terminal should include, within the application logic, the selection of an adequate transport protocol to be used under different network context situations. In other words, the developer should consider not only the static situations where the web application can operate connected via WiFi or cellular networks, but should also implement the adequate logic to cope with the handover when moving from one network to another, or even when the terminal gets disconnected.

Today, developers of distributed applications want to concentrate on the application logic in order to be more productive and to reduce development delays and implementation bugs. To achieve this, they need to make an abstraction of all the details of the communication system. This means that they do not need to cope with the direct selection and configuration of the transport and network services or how to deal with the details on the transmission of data between the distributed components of the application.
Audience

This book can be used by network experts, but it is actually intended for another kind of audience: i.e. anybody interested in learning about the solutions that make it possible to make an abstraction of the transport and network layer services when developing distributed systems. We will consider this audience to be interested in information technology (IT) solutions applied to distributed applications at intra-enterprise and inter-enterprise levels.

In particular, this book will introduce various evolutions of the middleware communication layer that has been designed as an adaptation layer intended to hide the complexity of the distribution of applications components. We will introduce the evolution of the middleware layer and, in particular, we will concentrate on the main paradigm that has deeply impacted the design of this kind of system, the service-oriented architecture (SOA) paradigm. Even if this is not a new or revolutionary concept, after several years working in this area we have realized that the complexity involved in the SOA paradigm is so high that it is usually badly understood. SOA is often reduced to the basic concept of Web services (WS), losing the huge advantages of the paradigm that are urgently required to build agile and flexible distributed systems. This is particularly important in the area of cloud computing architectures, where delivering infrastructures, platforms or software following a service orientation is a fundamental principle.

Approach

Our goal is to demonstrate, based on a practical use case, how cloud computing IT platforms and SOA concepts can be efficiently applied when developing current and next generation enterprise applications. In order to avoid a sequential theoretical presentation of definitions and concepts, we have decided to propose a well-probed practical approach named project-based learning. This approach is based on the rationalization of concepts and the acquisition of competences based on the analysis, design and construction of a real-world project.

We have selected a case study inspired by the very well-known eBay online marketplace. This consumer-to-consumer distributed system use case will allow us to introduce all the components necessary for building a smart SOA platform. An initial implementation of this platform will be achieved on the basis of the fundamental pillar of the SOA: the enterprise service bus
Preface xi

(ESB), which is aimed at guaranteeing integrability, interoperability and extensibility non-functional requirements. We will consider this initial phase as the development of the fundamental SOA concepts that will be integrated in a first release of the proposed platform solution named Smart SOA Platform as a Service 1.0 (SSOAPaaS 1.0). Based on additional non-functional requirements, mainly related to a better decoupling or reduction of direct dependency between the distributed components as well as the needs for proactivity properties, a new version of the platform will be developed under the name of SSOAPaaS 2.0. The SSOAPaaS 2.0 enriches the previous platform by including a second fundamental pillar commonly called event-based systems for event-driven architectures (EDA). In a final development, additional non-functional requirements, in terms of manageability and scalability aspects, will be considered within the SSOAPaaS 3.0 solution. The extension of the SSOAPaaS 3.0 solution with self-management properties will make the proposed platform smart to achieve the goal of this book: designing and developing a smart platform for SOA-based systems.

A specialized IT infrastructure needs to be provided to easily install and deploy the required components for the different versions of SSOAPaaS. A solution based on virtual and cloud technologies facilitating the design and implementation of an efficient, portable and extensible infrastructure will be developed. This platform, named Smart PaaS (SPaaS), provides the required infrastructure management functionalities in order to support the three releases of the Smart SOA Platforms.

**Book structure**

This book is organized into six chapters, an Introduction, and Conclusion and Perspectives:

In the Introduction, we provide a global introduction including the main goals of this book and the applied project-based learning methodology.

Chapter 1 presents the case study that introduces the main requirements guiding the proposed three generations of SOA platforms.

Chapter 2 is intended to introduce the basic concepts of SOA, EDA, cloud computing and autonomic computing. It will present the basic concepts of communication middleware, the solutions for enterprise application integration and SOA, and will introduce the WS and ESB
technologies. The description of an important evolution in the world of enterprise integration represented by the message-oriented and EDA paradigms will then follow. Likewise, the basic concepts illustrating how these paradigms can enhance integration solutions by providing decoupling and proactivity functionalities between distributed components will be discussed. Moreover, an introduction to virtualization and cloud computing architectures will be presented to show how non-functional requirements such as manageability and scalability can be addressed by using advanced virtualization and cloud computing strategies. Complexity of manual management of SOA platforms will be illustrated to introduce the autonomic computing framework that will be followed to implement our smart platform solutions.

Chapter 3 presents a first cookbook composed of a set of recipes guiding the development of the SPaaS solution aimed at developing the smart IT infrastructure needed for installing and deploying the required components for the different versions of SSOAPaaS platforms.

Chapter 4 develops a cookbook that shows how SOA paradigm and technologies satisfy interoperability, extensibility and integrability non-functional requirements by means of the SSOAPaaS 1.0 platform.

Chapter 5 presents a cookbook that is intended to illustrate the concept of message-oriented middleware (MOM) and the use of messaging systems (i.e. Java messaging service (JMS)) to provide asynchronous communication channels (i.e. point-to-point or publish/subscribe). Moreover, the concept of complex event processing (CEP) will also be demonstrated. This chapter shows how the message-oriented and event-driven paradigms and technologies have been able to satisfy availability and proactivity non-functional requirements due to the SSOAPaaS 2.0 platform.

Chapter 6 presents a final cookbook focused on the modern generation of SOA platforms within a global and interconnected extended enterprise world. A set of recipes showing how non-functional requirements such as manageability and scalability can be implemented within the SSOAPaaS 3.0 platform will be presented.

Finally, the Conclusion and Perspectives summarize the evolution of communication middleware and the role played by SOA platform generations in satisfying integration, interoperability and performance needs
for large networked systems. Current and future challenges and perspectives involved in the design and development of future smart and autonomic SOA platforms in cloud computing architectures will be described. Moreover, the guidelines for smart self-configuration, self-provisioning and self-optimization strategies guiding the next generation of platform as a service solution will be presented.

Figure 1.1 presents the overall book structure and the various topics to be discussed.

![Book Structure Diagram]

**Figure P.1. Book structure**

In order to demonstrate how SOA concepts can be efficiently applied when developing the modern generation of enterprise applications, we have proposed a project-based learning methodology, based on a case study and allowing the introduction of the foundations of distributed systems design and development. We will cover SOA and EDA and how the cloud and autonomic computing paradigms play a fundamental role in modern distributed architectures.
We will illustrate, using a practical cookbook-based approach, how the fusion of all these technologies and approaches can enable the construction of smart SOA platforms in order to be able to satisfy a large set of non-functional requirements including integrability, interoperability, availability, proactivity, manageability, scalability, portability, extensibility and security.

Ernesto EXPOSITO and Cédé DIOP
May 2014
In this introduction, the motivation for applying the service-oriented architecture (SOA) paradigm to distributed systems is introduced. Moreover, the project-based learning methodology, named yPBL, as well as the case study that will be followed in the next chapters will be presented.

I.1. Evolutions of distributed systems

The complexity involved in the development of the first generation of networked applications was relatively important in the context of the original Transmission Control Protocol/Internet Protocol (TCP/IP) network model [EXP 12]. This first generation of networked applications was mainly represented by centralized client/server or distributed peer-to-peer applications (see Figure I.1). Examples of these applications are file transfer, Web browsing, audio/video on demand, interactive audio and video conferencing, etc.

Part of these applications have been statically implemented to directly use the fully reliable and fully ordered service offered by TCP or the non-reliable and non-ordered service offered by the User Datagram Protocol (UDP) over the basic best-effort service provided by the Internet Protocol (IP). More recent applications asking for more complex Quality of Service (QoS) requirements (e.g. delay, jitter, bandwidth, reliability and order) have been able to take advantage of more specialized services at transport (e.g. datagram congestion control protocol (DCCP), stream control transmission protocol (SCTP) and multipath transmission control protocol (MPTCP)) or network (e.g. IntServ, DiffServ and, MPLS) layers. Moreover, an important
number of application and session layer protocols have been developed in order to facilitate common networked functions such as session control, multimedia data transfer and presentation.

Consequently, the complexity related to the development of distributed systems has been increasing rapidly mainly due to such evolution and diversity of transport, network and higher layers. Today, developers working directly over the transport layer application programming interface (API) need a high level of expertise on all the services offered by the large set of transport protocols. Moreover, the developers also require a good understanding of how these transport services can be combined with the underlying services offered at the IP network level.

Let us illustrate this complexity with an example: the development of a Web-based mobile application. If a developer needs to access the transport layer API directly, it will be necessary to include, within the application logic, the selection of the adequate transport protocol to be used not only when the application is launched but also under different network context situations. It means that the developer should consider not only static situations where the Web application can operate connected via WiFi or
cellular networks, but should also implement the adequate logic to cope with the handover when moving from one network to another or even when the terminal gets disconnected and reconnected.

In order to cope with this complexity, important efforts have been invested to facilitate network functionalities implementation allowing developers of distributed applications to concentrate on the application logic. It means, for instance, that developers should not need to cope with the selection and configuration of the communication system or how to deal with errors, delays or any other unexpected events during the transmission of data between the distributed components of the application.

The results of these efforts have been translated in the development of the second generation of networked applications based on a new layer between the transport and the application layers called communication middleware (see Figure I.2).

![Figure I.2. Second generation of networked applications](image)

The communication middleware layer can be defined as an adaptation layer aimed at hiding all the complexity related to the distribution of application components from an applications’ developers. In other words, developers of networked applications (i.e. multi-tiered applications including client, server and backend distributed enterprise components) should be able
to interact with any distributed component as if it were located in the same execution environment. Moreover, communication middleware solutions can also hide heterogeneity of components by providing uniform and standard interfaces and making an abstraction of the implementation languages and environments of the components (e.g. Java EE, .NET, C and C++). Solutions include Remote Procedure Calls, Distributed Objects, Message-Oriented, Resource-Oriented, etc. Service-Oriented solutions have been developed in order to facilitate the implementation of networked applications that can be completely agnostic of the services provided at the transport and network layers.

The SOA paradigm promotes the design of distributed systems based on elementary services that can also be composed to implement complex composite business processes.

Initial integration efforts in SOA architectures were based on enterprise application integration (EAI) solutions. EAIs are aimed at offering integration of services between enterprise applications. This solution is implemented based on a server acting as a central hub and offering a common API for application integration. The server also implements the required proprietary links to communicate with non-standard third-party solutions. EAIs implement an architectural model defined as “hub-and-spoke”. The main drawback of the EAIs is the high dependency on the performance of the central hub (i.e. potential scalability problems resulting from bottlenecks and risks of global system fault).

In order to cope with the drawbacks of the EAI, the enterprise service bus (ESB) paradigm was proposed. The ESB integration paradigm follows the SOA approach and proposes a framework aimed at allowing synchronous or asynchronous communication between consumers and providers by implementing the well-known bus communication topology used in computer and networks architectures. An ESB performs as a mediator facilitating the provision and consumption of services [CHA 04]. In contrast to centralized EAI solutions, the use of an ESB increases the availability, reliability, performance and scalability and facilitates maintenance (e.g. by including better performing or more adapted services) and evolution (e.g. by including new services or modifying the orchestration logic). ESBs can be clustered or federated in order to cope with the scalability and fault tolerance limitations of the EAIs. Likewise, the ESBs are based on a common normalized messaging approach, thus coping with the interoperability
limitations of the EAIs. The mediation between service consumers and providers is achieved by the message routing functionalities provided by the ESBs.

ESBs operate at the backend tier and facilitate the collaboration between providers and consumers of services by providing a mediation service. This mediation simplifies the generally static and complex point-to-point communication between N providers and M consumers by offering the adequate interoperability, transport and routing of message functionalities. Likewise, the underlying complexity at transport and network layers, as well as the protocol-specific formats and data models at the middleware communication layers, is hidden by the ESB by providing a uniform and standard Web service-based API and data model.

This approach has been followed by small, medium and large distributed applications. An example of a large distributed application operating over a communication middleware is the well-known eBay consumer-to-consumer system. eBay is able to process billions of transactions per day with the participation of thousands of distributed components. eBay IT infrastructure is based on an ESB middleware, which is able to perform as a mediator between the various middleware solutions and distributed components.

This book is mainly intended for those who are interested in acquiring knowledge and skills in the area of IT infrastructures applied to this kind of distributed system at intra-enterprise and inter-enterprise levels.

In particular, this book will introduce the evolutions of the middleware communication layer that has played an important role as an adaptation layer hiding the complexity of the distribution of applications components [HO 03]. We will introduce the evolution of the middleware layer; in particular, we will concentrate on the main paradigms that have deeply impacted the design of this kind of system: the SOA [KRA 04, MAC 06, ERL 09] and event-driven architecture (EDA) [VAN 06, TAY 09] paradigms.

Our goal is to demonstrate, based on a practical use case, how SOA concepts can be efficiently applied when developing the modern generation of enterprise applications. Moreover, we will present an important evolution to the SOA platforms represented by the cloud and autonomic computing paradigms. We will illustrate how the fusion of all these technologies and
approaches can enable the construction of smart SOA platforms able to satisfy a large set of non-functional requirements including integrability, interoperability, availability, proactivity, manageability, scalability, portability, extensibility and security.

In order to avoid a sequential theoretical presentation of definitions and concepts, we have decided to propose a well-probed practical approach named project-based learning. This approach is based on the rationalization of concepts and the acquisition of competences based on the analysis, design and construction of a real-world project. The following section describes this learning methodology.

I.2. yPBL: a project-based learning methodology

The software engineering (SE) domain involves complex software development processes demanding a high level of knowledge and expertise from analysts, designers, developers and architects in diverse areas including project management skills, communication, design and development. Moreover, the large diversity of software design and development approaches and paradigms as well as the accelerated development of new software technologies ask for a continuous knowledge acquisition process.

This is particularly important in the area of distributed systems design and development where the networking deployment models have been continuously changing since the beginning of the Internet, encompassing centralized client/server models and multi-tiers, peer-to-peer, overlays, or cloud computing distributed models. *Software engineers and architects working on designing and developing distributed systems need to face a constant context evolution. This evolution includes both the communication systems point of view (i.e. new protocols and services at transport, network layers, and communication middleware levels) and the applications point of view (i.e. enterprise applications needs in terms of integration, interoperability, scalability, multi-tenancy, etc.). In this evolving and complex context, it is not easy to acquire the required design and development knowledge and skills of IT infrastructures.*

In the area of the software engineering process, several methodologies have been proposed in order to efficiently support members of development teams to design and implement software products. Unified process (UP)
methodologies are very well known in the world of software engineering by providing an efficient process based on an incremental and iterative sequence of phases [JAC 99]. The unified process includes phases targeting specification and analysis of requirements, design of the software solution and implementation, test, integration and deployment of the software product. Phases are planned and executed in incremental iterations where in each increment new customer requirements can be added within the process. Likewise, bug detection and corrections as well as requirements change requests can be added in each iteration. As agreed in the software management plan, stable or experimental software products can be released at the end of the iterations. UP has been specialized in new context-specific methodologies such as the rational unified process (RUP) [KRO 03], enterprise unified process [AMB 05] or the agile unified process [AMB 02].

Another interesting specialization known as the two tracks unified process (2TUP) has been proposed to face the reality of continuous changes of requirements and technologies that represents an invariant reality in software engineering [ROQ 04]. The 2TUP process, also known as the “y” process due to its graphical representation, proposes a differentiation of two tracks for the unified process, the first (left) track represents the functional and non-functional requirements related to the software product and the second (right) track represents the technical solutions (e.g. technology, environment and platforms). This separation of concerns helps software engineers to concentrate on discovering and specifying the requirements that need to be satisfied (left track) while allowing them to explore and select the technologies that could be used to build the software solutions (right track). Once the functional, non-functional and technical requirements have been identified and specified, both functional and technical tracks can be merged in order to produce the software design specification. From this point, the software product can be developed, tested, integrated and deployed. This sequence of parallel and serialized activities will be executed within the incremental and iterative process proposed by the UP method. Benefits of this interesting methodology have been demonstrated by its application in many industrial and research software projects.

In order to face the challenge of efficiently acquiring the required knowledge and the expected competences in the complex SOA domain, we have decided to follow a software engineering approach driven by a project or case study that will provide the fundamental requirements that will drive the readers during the whole learning process.