Process-Driven SOA

Patterns *for* Aligning Business *and* IT

Infosys® Press

Carsten Hentrich
Uwe Zdun

Process-Driven SOA

Patterns *for* Aligning Business *and* IT

Carsten Hentrich
Uwe Zdun

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN AUERBACH BOOK

Infosys® Press

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works

Printed in the United States of America on acid-free paper
Version Date: 20111007

International Standard Book Number: 978-1-4398-8929-9 (Hardback)

Library of Congress Cataloging-in-Publication Data

Hentrich, Carsten.
Process-driven SOA: patterns for aligning business and IT / Carsten Hentrich, Uwe Zdun.
p. cm. -- (Infosys press)
Includes bibliographical references and index.
ISBN 978-1-4398-8929-9 (hardback)
1. Service-oriented architecture (Computer science) I. Zdun, Uwe. II. Title.

TK5105.5828.H46 2012
658.4'038--dc23 2011029099

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Process-Driven SOA

Patterns *for* Aligning Business *and* IT

Infosys® Press

In an initiative to promote authorship across the globe, Infosys Press and CRC Press have entered into a collaboration to develop titles on leading edge topics in IT.

Infosys Press seeks to develop and publish a series of pragmatic books on software engineering and information technologies, both current and emerging. Leveraging Infosys' extensive global experience helping clients to implement those technologies successfully, each book contains critical lessons learned and shows how to apply them in a real-world, enterprise setting. This open-ended and broad-ranging series aims to brings readers practical insight, specific guidance, and unique, informative examples not readily available elsewhere.

PUBLISHED IN THE SERIES

.NET 4 for Enterprise Architects and Developers
Sudhanshu Hate and Suchi Paharia

Process-Centric Architecture for Enterprise Software Systems
Parameswaran Seshan

Process-Driven SOA: Patterns for Aligning Business and IT
Carsten Hentrich and Uwe Zdun

Web-Based and Traditional Outsourcing
Vivek Sharma and Varun Sharma

IN PREPARATION FOR THE SERIES

Scrum Software Development
Jagdish Bhandarkar and J. Srinivas

Software Vulnerabilities Exposed
Sanjay Rawat, Ashutosh Saxena, and Ponnapalli K. B. Hari Gopal

Preface

The business environment is in a constant state of flux. Increasing connectedness due to the Internet and globalization is narrowing the gap between cause and effect, leading to rapid feedback loops. Inability to analyze and adapt to these feedback loops makes an organization irrelevant, leading to stagnation and decay. In such a fast-paced environment, agility is of strategic importance to enterprises as they need to continuously optimize their operations and accelerate innovation. This is the only way they can remain relevant and competitive and grow profitably. An approach that is "business process driven and services enabled" provides a great foundation for such continuous change. While business processes are key drivers of value in any organization, services provide a foundation on which business processes can be reconfigured to handle the changes in the business environment. Today, service-oriented architecture (SOA) is being used for achieving strategic objectives across multiple industries, across geographies.

For consumer-facing enterprises, the Internet and mobility are driving significant change. More than 75% of the world is connected using mobile devices, and more than 25% use the Internet. This interconnectivity is leading to accelerated flow of information, and demands for personalization are rapidly increasing. This requires enterprises to partner and integrate with others to continuously deliver new products, services, and experiences. Such integration can quickly lead to complex interdependencies in supply chain and associated systems. These can make future changes difficult, time consuming, and slowly unviable—making the enterprise less amenable to change. A service-oriented approach is important for development of integrated but loosely coupled systems whose components can evolve independently.

In addition, with increasing competition it is becoming paramount to provide unified and integrated experiences through multiple channels—Web, mobile, TV, callcenter, and so on. A services-based approach has become the standard approach for such initiatives.

The rise of emerging economies is opening new growth opportunities for large enterprises. However, these markets require lightweight processes and systems that can scale and evolve rapidly. Enterprises that have built systems using SOA principles are able to rapidly evolve an existing system to serve these newer engines of growth.

Enterprises in asset-heavy industries and logistics are leveraging sensor networks, intelligent decision support systems, and cloud to increase information visibility and improve operational decision making. Intelligent and cloud-based sensor networks create new opportunities owing to the integration of the physical and the virtual worlds. Advanced analytics based on real-time events and sentiments enable new forms of remote services with improved business models for consumer engagement. Also, many experts are propagating new revenue enhancement and cost reduction opportunities leveraging the cloud. Interestingly, for all those key driving forces an architectural foundation that aligns business and information technology (IT) and one that enables flexibility would be a key element. Systems built on SOA principles are more amenable to supporting these requirements.

These developments logically influence a trend within companies toward building smarter organizations. Organizations need to learn, adapt, collaborate, and simplify their internal structures to cope with the pace of these developments. A recent analysis of 75 companies across 12 industries globally found that the companies with a lower level of complexity grew revenues 1.7 times faster than their peers on average. Complexity holds four

times the predictive power for revenue growth than company size. Inevitably, successful companies will need to reengineer their platforms and renovate them on a sustained basis. As a result, architecture initiatives are targeting simplification, standardization, modularization, and harmonization. The pressure on flexibility is getting so strong that it is no longer possible not to invest in these architectural innovations to stay competitive. Systems built on SOA principles are more amenable to supporting these requirements.

The SOA has been around for a while; however, interpretations of designing and implementing SOA are manifold. In this book an approach is presented that has actually evolved as a set of successful architectural patterns researched and discovered since 2002. It appears that the key architectural concepts that evolved as successful principles over the years remain independent of actual technologies being used and have proven to be timeless. This book elaborates SOA principles that are foundational for creating an adaptive integrated system—allowing tomorrow's enterprises to continuously evolve processes and systems as the business requirements change.

The patterns presented in this book form timeless and proven solutions to these challenges of business-IT alignment, continuous business optimization, and accelerated innovation. The authors proclaim an approach for architectural flexibility based on SOA that is a mandatory building block in many of the business initiatives aimed at transformation, optimization, and innovation.

From a business transformation perspective, the trends mentioned require an approach that focuses on six key dimensions: ensure adoption and accountability, optimize (simplify) processes, visualize information, orchestrate (global) capabilities, strengthen strategic partnerships, and maximize return on assets. To measure the business value of those transformational initiatives, one has to map value levers to operational levers, map those to processes, and map processes to change enablers. Operational levers form operational business goals that are directly related to processes. It becomes clear that at the architectural core of those transformation initiatives there must be an approach that is business process driven and that relates process design to business goals. That way, it is aligning business and IT. The patterns in this book follow exactly this principle.

From an innovation perspective, we can summarize five key trends in information management: knowledge-driven systems that make use of advanced analytics, distributed (cloud and grid) computing, digital convergence, sensor networks, and mobility. SOA itself is a basis for all these trends by virtue of the need for modular and flexible access to IT fulfilled by SOA.

From the business optimization perspective, an approach focused on measuring, managing, and optimizing business processes is needed. That means an architectural approach is required that makes use of business process management as a discipline. The content of this book also serves this purpose well.

From the trends discussed, it is clear that organizations need an architectural foundation that enables continuous optimization, accelerated innovation, and rapid transformation. That is exactly what this book provides. It provides the architectural basis in the form of proven patterns that have passed the test of time and are rightly positioned to form the stable core of future-ready enterprise architecture.

Subu Goparaju
Head of Infosys Labs

Acknowledgments

It is our pleasure to thank the many people who supported us in creating this book, as either our collaborators or colleagues, by sharing their knowledge with us or by reviewing and commenting on our work.

We wish to thank Andreas Grunewald, who has worked on the implementation of the pattern language walkthrough example presented in Chapter 4. We thank Till Köllmann, who is the coauthor of the paper, "Synchronization Patterns for Process-Driven and Service-Oriented Architectures." Some reworked patterns from this paper are included in this book. There are a number of coauthors of research papers related to the content of this book: Wil van der Aalst, Paris Avgeriou, Schahram Dustdar, and Vlatka Hlupic. Thanks to the IBM Software Group, especially Lucie Schellberg, for supporting this work. We would also like to thank Infosys for its strong support.

Special thanks for reviewing the material in this book go to Paris Avgeriou, Gregor Hohpe, and Till Schümmer, who all provided in-depth reviews for some of the contents.

In addition, we presented parts of the contents of this book at various European Conference on Pattern Languages of Programs (EuroPLoP) and Conference on Pattern Languages of Programs (PLoP) pattern conferences. Many thanks go to all participants of the EuroPLoP and PLoP workshops who commented on our work. Special thanks go to our EuroPLoP shepherds who provided detailed feedback on our EuroPLoP and PLoP papers: Andy Longshaw, Wolfgang Keller, Eduardo B. Fernandez, and Stephan Lukosch.

Contents

1

Introduction

What This Book Is About

This book is about a special kind of service-oriented architecture (SOA): *process-driven SOA*. To start and to set the scope of this book, let us consider an initial, rather technical, definition of SOA in general: SOA is an architectural concept in which all functions, or services, are defined using a description language and have invokable, platform-independent interfaces that are called to perform business processes [CHT03, Bar03]. Each service is the endpoint of a connection, which can be used to access the service. Communication among services can involve simple invocations and data passing or complex activities of two or more services.

This view of SOA, even though it is rooted in well-known concepts and technologies, such as middleware concepts and systems, is limited to technical aspects. In our experience, one of the key issues in engineering SOAs is the integration of organizational and software model building to provide a conceptual foundation for designing *business-driven SOAs*. In this view, at the core of SOA is a business-oriented approach to architecture, but SOA, as it is described in the first paragraph, seems rather to focus on technical issues. A foundation for an integrated engineering approach seems yet to be missing. Still, many fundamental issues arise due to the separation of business-oriented and technical viewpoints. One solution that has been proposed to solve these problems is process-driven SOAs, that is, SOAs that are based on business processes and utilize process technologies such as business process engines. Our perspective of business-information technology (IT) alignment refers to this context. In this book, we focus on this special kind of SOA and describe time-proven solutions for process-driven SOAs in pattern form.

Please note that many other flavors of SOAs have been proposed, such as SOA concepts focusing on the middleware or Web service aspects, component architectures, enterprise architectures, governance, or even embedded systems or mobile systems. In this book, we only touch on them, if they are linked to process-driven SOAs, but no further.

So, what is a process-driven SOA, and what are the major challenges we address in this book?

Since 2002, workflow technology [LR00] has evolved as an important technological approach for business process management (BPM) [Pri03]. Today, a process engine takes the role of a process controller within the enterprise architecture. The goal of the process engine is to coordinate and distribute tasks in a heterogeneous system environment. In recent years, this approach has been strongly adopted in the SOA area to support *service orchestration*. An important driver for this trend is that IT strategies have changed and demand more support from IT technologies, such as SOA and workflow technology, to flexibly *change and adapt business processes.*

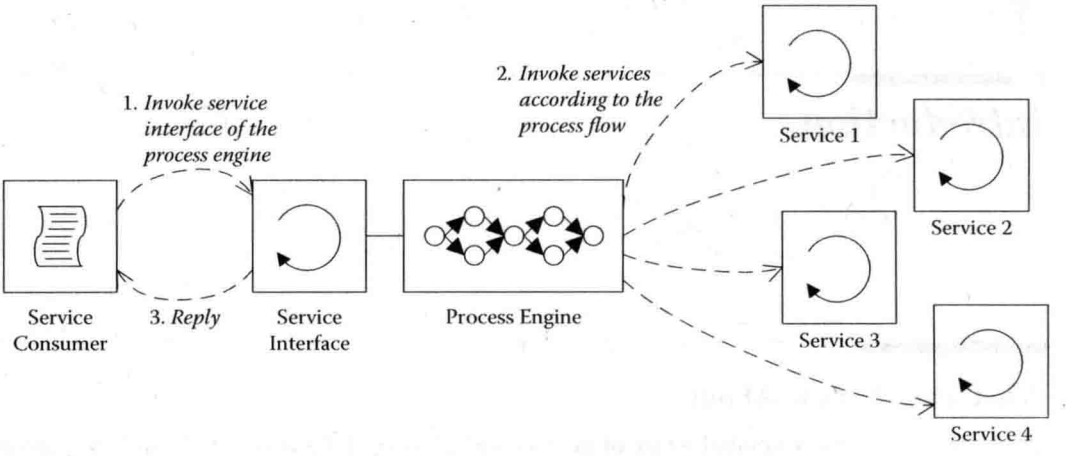

FIGURE 1.1
Basic architecture of a process-driven SOA.

In an SOA, which results from the combination of service-based middleware and process engines, services are invoked to perform business processes. We call this kind of architecture a *process-driven SOA*. In it, the services describe the operations that can be performed in the system. The process flow orchestrates the services via different activities of the process. The operations executed by activities in a process flow thus correspond to service invocations. The process flow is executed by the process engine.

This basic architecture of a process-driven SOA is illustrated in Figure 1.1. A service consumer invokes the service interface of the process engine. Then, the process engine executes the process flow. Some of the activities in the process flow invoke services. Hence, these services are orchestrated according to the process flow. Finally, when the process flow terminates, the process engine replies to the service consumer.

Because—in principle—the service interface of a process engine and the interface of any other service cannot be distinguished, arbitrarily complex architectures can be composed this way. For example, in the example diagram in Figure 1.1, any of the services invoked by the process engine could itself hide another process engine. This process would then be invoked as a subprocess of the invoking process.

A process is a behavioral model expressed in a process modeling language, such as the Business Process Modeling Notation (BPMN), event-driven process chains (EPCs), the Business Process Execution Language (BPEL), or the jBPM Process Definition Language (jPDL), to name but a few existing process modeling languages. Some process models, such as those described in BPEL, are executable and can be instantiated and managed by a process engine. On a process engine, multiple process instances of one or more processes are typically running in parallel. Processes usually work on business data stored in business objects.

Please note that not all processes are executable. For example, processes modeled in languages such as BPMN or EPC are usually not directly executable. Instead, they model the processes from a business perspective only. To make them executable on a process engine, they must be translated into an executable process language such as BPEL or jPDL. This usually means, among other things, adding technical details to the process models.

This book focuses on an overall and integral architectural perspective for process-driven SOA systems. That is, from the perspective taken in this book, it is actually not enough to

concentrate on the process engine itself but to view it as an integral part within a larger software architecture.

Process-driven SOAs are introduced into an organization for various reasons. Some exemplary strategic goals for introducing process-driven SOAs are

- flexible business process control,
- integration of legacy systems functionality within business processes,
- automatic invocation of services,
- connectivity to various interfaces,
- scalability,
- automated measurement and monitoring of business processes,
- processes as reusable components,
- business-to-business (B2B) integration, and
- business process automation.

To achieve these strategic goals, architectures must be designed accordingly, balancing both technological and business-oriented forces. To balance these forces, this book introduces time-proven practices presented using the concept of *software patterns* as guidelines for the software design. These software patterns cover the whole design space of designing a process-driven SOA and guide you step by step to a complete architecture and software design, covering the various aspects discussed.

In this context, it is important to note that designing a nontrivial process-driven SOA involves many difficult design and architectural decisions. Examples are as follows:

- Different kinds of processes exist, long-running, business-oriented and short-running, technical processes. How to best integrate them, and how to map them to execution platforms? How to best design the processes to achieve the business goals and fulfill the technical requirements?
- An SOA has many different stakeholders, such as business analysts, management, software designers, architects, and developers, as well as many different types of models these stakeholders need or want to work with. How to present each of them with the best view on the models they need for their work?
- A realistic process-driven SOA contains many systems that need to be integrated, such as various process engines, services, and back-end systems, running on heterogeneous technologies and platforms. How to perform integration in a way that is maintainable and scalable?
- Often, business processes and services within an organization and within different cooperating organizations need to interact with each other and with other external events. How to synchronize all these processes, services, and events?
- Business processes and services need to work with data. How to best integrate the data of the organization into the process-driven SOA?

This book introduces a pattern language that deals with issues such as process modeling, execution, integration, synchronization, and so on. Its main goal is to help solution architects, as well as process and service designers, to master the challenges in designing a stable and evolvable process-driven SOA. In this sense, the book aims to bring together the business and IT worlds.

Target Audience

This book is primarily written for *SOA solution architects* and *SOA designers* as it primarily provides in-depth guidelines on how to cope with various architectural and design decisions that you encounter when designing and implementing a process-driven SOA system. The decisions are described in the form of software patterns that provide a practical guideline to address the key problems in this field using time-proven solutions.

Another primary audience is *SOA developers* who need to realize the service architectures and designs described in this book. For them, the book primarily provides detailed descriptions of the solution for the pattern. That is, considerations of the solution, detailed examples, and case studies illustrating the patterns are provided.

Another primary audience is the *business stakeholders* involved in process-driven SOA projects, such as SOA business analysts or SOA managers, who want to understand the technical solutions applied in a process-driven SOA project. For them, probably mainly the overview parts of the chapters are interesting—not all the technical details.

For all audiences named, the patterns provide a common terminology that can be used for the solutions described in the pattern descriptions throughout their process-driven SOA projects.

A secondary audience is *students* learning about process-driven SOA or *researchers* working on process-driven SOA topics. While this book is no textbook or research-oriented book, it still can be useful for these audiences as a practical introduction to basic and advanced topics in the process-driven SOA field. The material in this book has been used in a number of tutorials and courses on the process-driven SOA topic as educational material. It also has been used in various ways in a number of research projects.

The patterns in this book have emerged from, and are primarily applicable in, the following industries: insurance, banking, media/telecommunication, and automotive/manufacturing. They are validated in this context. Our primary target audience for this reason is people from these industries, although the patterns may also apply to other contexts than those listed. The patterns have not been applied and validated in the embedded systems context.

Software Patterns

This book is based on the concept of *software patterns*. Software patterns provide concrete guidelines for the software design [BHS07b]. They capture reusable design knowledge and expertise that provide proven solutions to recurring software design problems arising in particular contexts and domains [SB03]. Software patterns are a systematic reuse strategy [SB03]: In a pattern, successful software models, designs, or implementations that have already been developed and tested are documented so that they can be applied to similar software problems. Thus, software patterns in the first place help to avoid the costly cycle of rediscovering, reinventing, and revalidating common software artifacts. Patterns provide "timeless" knowledge in the sense that it lasts longer than concrete technologies or implementation concepts [BHS07b, Ale79].

The full power of patterns is only achieved, however, if they are used in the context of each other to form a web of related patterns, more commonly known as a *pattern language*

[Cop96, BHS07a, BHS07b]. A single pattern can only solve a single problem. Patterns in a pattern language, in contrast, build on each other to generate a system [BHS07b]. Hence, a pattern language has a language-wide goal, and it solves the prevalent problems in a particular domain and context [Cop96, BHS07b]. The purpose of the language is to guide the user step by step to reach this goal. A pattern language achieves this by specifically focusing on the pattern relationships in this domain and context. In the domain and context, patterns provide a design vocabulary [BCM+96, BHS07b, GHJV94] that can be used among developers or designers to ease their interaction. In addition, pattern languages also provide the "big picture" of the domain [BHS07b].

Originally, the concept of patterns was invented by Christopher Alexander [Ale77, Ale79], who used the concept of recurring problems and corresponding solutions in the context of towns, buildings, and construction. The basic idea of patterns is that problems arise due to conflicting forces in a given context, and solutions resolve these conflicting forces. For this reason, the pattern approach provides a flexible method for finding appropriate solution concepts, as there is no static and linear process to follow, but rather flexible and context-dependent solutions are created using patterns. Alexander defined a pattern as follows [Ale77]:

> Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution. As an element in the world, each pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves. As an element of language, a pattern is an instruction, which shows how this spatial configuration can be used, over and over again, to resolve the given system of forces, wherever the context makes it relevant.

The concept of patterns has been successfully applied to software system and architecture design [GHJV94, BMR+96, SSRB00, BCM+96, Cop96, VSW02, HW03, VKZ04, KJ04].

The Appendix provides a detailed overview of patterns related to the patterns presented in this book and provides short thumbnails of each external pattern referenced in this book.

Pattern Form and Pattern Chapter Structure

Several formats for specifying patterns have been proposed that are specialized to the purposes of different types of patterns. In this book, we introduce different types of patterns. For this reason, a pattern format has been chosen that suits all these types. The format consists of seven logical parts: pattern name, context, problem, problem details, solution, solution details, and examples (including some known uses of the patterns).

This core of the pattern descriptions appears in the pattern texts according to this format (presented in more detail in the following discussion). Some other parts of the patterns, such as pattern relationships and bigger case studies, are rather presented in other places in the pattern chapters because they are shown together for a number of related patterns.

When pattern names are referenced in the text, they are written in SMALL CAPS font.

Each pattern chapter starts with a *narrative introduction* to the contents of the chapter. This narrative introduction gives an overview of the patterns described in the chapter

and defines the role of each pattern in the pattern language. That is, *related patterns* of the patterns described further in the chapter are mostly described in these narrative introductions. Also, the *common context* of the patterns in a process-driven SOA and in the pattern language is described here. These pattern relationships illustrate how the pattern works in conjunction with other patterns and describe their interplay. In addition, some examples of known uses are given for illustration.

In most chapters, the pattern descriptions follow. Sometimes, between the pattern sections we provide another *narrative* section.

Each pattern has a common appearance in the text (as outlined in the box below). The pattern starts with the *pattern name*. The pattern name has the task to identify the pattern. Pattern names make it possible to use patterns as a common terminology that can be used by architects and designers (i.e., in a software project team or in a community). Then, the sections are outlined and explained in the following box.

PATTERN NAME

The pattern starts with a description of the *context* of the pattern. The context leads to the problem description.

A question summarizes the *problem* of the pattern.

The *detailed problem description* follows. It contains detailed explanations of the addressed problem. Most important, the conflicting *forces* are described in this section. Forces are important factors that influence the design decision for or against the use of a pattern.

A rather short *solution* description follows that outlines the pattern's time-proven solution in one paragraph.

An *illustration* follows to sketch the pattern's solution.

The *detailed description of the solution* follows right after the short solution description. Different figures or models can be used to illustrate the solution. Central to the solution is that it resolves the conflicting forces in the context. Also, some central relationships of a pattern to other patterns are described, but only if they are needed for describing the solution. All other pattern relationships are described in the narrative sections before.

The consequences that result from applying the pattern are also described in this section. There are usually positive and negative consequences. The expected consequences are important as input for the design decision when and how to apply the pattern.

EXAMPLES

Finally, some examples illustrate the pattern. Some examples are artificial but resemble our real-world experiences. Others are taken from known uses of the pattern that are listed as examples of real-world incarnations of the pattern. The known uses illustrate how and where the pattern has been successfully applied in the past.