Introduction 1

1.1 Enterprise Application Software in the Age of Globalization

Without a doubt, today's companies are facing increased competition as a result of globalization. In their daily fight for survival with their competitors, their reaction to change must be swift, yet sustained. The following quote from Charles Darwin is particularly valid in this context:

It is not the strongest of the species that survives,
nor the most intelligent that survives.
It is the one that is the most adaptable to change.

It is a company's ability to find effective solutions in response to changes that will ultimately guarantee its survival. Information technology (IT) is an important contributory factor here. Efficiently providing new software solutions is not the main problem, thanks to enormous advances in software engineering:

– Model-driven software development,
– The standardization of communication and interface technologies,
– The modularization of business functions in the form of reusable services,
– Productivity improvements for software developers due to highly integrated development environments,
– The deployment of agile development methods.

As far as the total cost of ownership (TCO) is concerned, it is the maintenance, enhancement, and adaptation of software to changing conditions that are proving to be much greater hurdles. The adaptability of software systems is becoming ever more important, but efforts to reflect this in software architectures are being thwarted by time and cost pressures. This is particularly true for software that needs to integrate existing business functions with new applications spanning multiple systems and applications in line with the service-oriented architecture (SOA) approach. Service-oriented architecture originally promised to leverage the potential of increased agility, lower costs, reusability, and a rapid implementation of business strategy into software, but results in projects have been disappointing. Anne Thomas Manes has even predicted the death of SOA in her now almost legendary blog (Manes 2009). There are surely many reasons why results have fallen short of expectations, and this is no doubt at least partly due to the fact that SOA does not have, indeed has never had, a precise definition. Another reason, perhaps not immediately obvious, is that developers, when creating applications in the service-oriented world, draw on the expertise they have gained while programming tightly coupled solutions. Unfortunately, architectures that function perfectly well for tightly coupled stand-alone applications are completely unsuitable for loosely coupled, distributed applications. It doesn't matter if you are dealing with data types (working with only one data type system in contrast to working with several data type systems), communication between systems, or transaction behavior: a distributed environment requires different solutions. This means that developers need to change their way of thinking; as with any paradigm shift, a new approach is necessary. Be it a change from machine language programming to structured programming, from structured programming to object-oriented programming, or the current trend towards developing service-oriented architectures; at each stage of this evolution the degree of abstraction increases, and developers must adopt and adapt to the new paradigms if they are to fully exploit their potential. Clinging to old habits in this new world is more of a hindrance than a help. Process-driven applications or composite applications (composites for short)—I use these terms synonymously in this book—are no exception; there is no longer sole control over the resources involved (for example, database systems or ERP systems). Rather, we work in heterogeneous system landscapes where we are tasked with combining existing services and new business logic; application developers need to be much more aware of the problems that this presents.

The list of requirements goes on: Flexibility is another aspect that must not be underestimated. We can illustrate this by looking at two specific requirements:

1. Need for changes within the process-driven application itself, particularly changes to business processes
2. Ongoing changes within the system landscape on which the process-driven application operates

Changes to Business Processes Business processes, understood in the classic sense as "a series of specified operations (activities, tasks) executed by humans or machines to achieve one or more objectives" (see for example Hammer and Champy 1993), play a special role in process-driven applications. To differentiate themselves from their competitors, companies must constantly fine-tune those business processes that give them a competitive edge. Inevitably, successful processes are copied by competitors, so the competitive lead is short-lived. Let us look at an example from the airline industry: self-service check-in terminals. The airline that first introduced this idea could offer their customers the advantage of shorter waiting times (leading to higher customer satisfaction), and the airline itself benefited from personnel savings leading to lower costs. Today, almost all airlines use this service, so the competitive advantage has disappeared. Companies also have to change their processes as quickly as possible to adapt to changing market conditions. Whatever the trigger for the change, the consequences are always the same: A new innovation cycle compels companies to optimize their business-critical processes yet further, to ensure their continued success. The quicker these changes can be implemented, the higher the chances of a successful outcome. To ensure that software has this flexibility and agility, readiness for change must underpin the architecture of process-driven applications.

Changes in the System Landscape Process-driven applications must likewise be prepared for a constantly changing system environment. It is one of the central characteristics of composite applications that they function in a heterogeneous system environment. Reusability, as we have already met in SOA, plays an important role here as well. However, we should not try to reinvent the wheel. If established, functioning business logic is already in place, then it should be used accordingly. This does present a particular challenge though, since system landscapes are not stable. When designing process-driven applications, you must take into account from the very start that changes to landscapes are inevitable. There are at least three reasons to expect that system landscapes will continue to change in the future, perhaps at an even faster rate:

1. System consolidations
2. Company mergers
3. Software as a service (SaaS), on-demand, cloud

Let us examine these three points in more detail. IT departments in companies are subject to enormous cost pressure. Executive management demands constant cost optimization from IT managers. *System consolidations* are a proven tool for reducing costs; the more systems and functions that can be pooled, the greater the cost-saving potential. For this reason alone, system landscapes will continue to change and develop. For process-driven applications, this means that calls that were previously made to several systems now have to be processed by a reduced number of systems.

On one side of the coin, retiring systems and moving diverse processes to standard software does indeed simplify the IT landscape. The flip side of this coin is *company mergers*. In order to reach growth targets, companies supplement their organic growth with corporate takeovers. Fusing companies together in this way swells the number of systems and applications in the landscape. In such cases, the composite application must be adapted to the increased number of services in the new environment.

Let us also not forget the newest trends: *software as a service* (SaaS), on-demand, and cloud providers. These offer IT managers further optimization potential by providing cost-effective business services. You can now farm out non-critical functions, thus contributing towards an optimized IT cost structure.

However, this scenario does demand a higher degree of flexibility from the architecture of process-driven applications. All of these scenarios, be it system consolidations, the addition of new systems, or the outsourcing of functions, belong to the daily challenges that process-driven applications must face, and must be considered as an integral part of their design right from the start. In other words, in spite of the havoc they wreak on system landscapes, profitable innovation processes are not something that companies are willing to forgo. Nor should they have to. With the right architecture, they can rest assured that changes to their system landscape, which are both inevitable and necessary if a company is to innovate, will have almost no negative impact on process-driven applications. Their capacity for innovation is not impaired.

Process-Driven Applications for SMEs and ISVs So far, this introduction may have given the impression that process-driven applications are only a matter for large companies, but this is certainly not the case. Even though constantly changing system landscapes are less of an issue for small businesses and midsize companies (SMEs) (although SaaS will surely leave its mark), the same principles still apply: Only successful processes that differentiate them from their competitors will guarantee survival. The demands placed on them in this respect are just the same as for large enterprises, and the readiness for change in application architecture discussed above is just as vital.

It is not just within companies that the new generation of process-driven applications has an important role. Independent software vendors (ISVs) can also use composite applications to gain entry to lucrative new markets. The business model for ISVs usually focuses on developing solutions as extensions to the products of well-known vendors such as SAP, Oracle, or Microsoft that can only connect to these standard solutions (think back to the example mentioned in the preface of an application for managing software licenses). The problem with this business model is obvious: Even if a customer shows interest in the vendor's business solution, a business deal is not possible quite simply because the customer does not meet the technical requirements. That is, the customer does not use the standard application required and has no plans to do so in the near future. Many potential customers are ruled out at the outset, because the solution stipulates a particular vendor. If software could be created that could be adapted to a customer's system landscape, this would open up a whole new avenue of possibilities for ISVs. Process-driven applications meet exactly this requirement. Once created, their architecture allows them to be adapted to the customer's landscape, without affecting the actual business solution. This is only possible if we can make the business application and the system landscape as independent of one another as possible. This book discusses in detail how this goal can be achieved.

Design and Implementation Approach for Process-Driven Applications The attributes, problems, and scenarios mentioned above form the starting point for this book. The emergence of the SOA idea and the accompanying hype around it led to a flood of books and articles elaborating on the various aspects of this topic.

Unfortunately, they did not manage to actually solve the challenges at hand; in my opinion this is largely due to, among other things, the bottom-up approach for interface determination and the strong focus on services and their reusability. In this book, we will propose a top-down methodology instead and shift the focus to the business processes that are implemented.

Another element I felt was missing was a detailed and comprehensive description of the architecture of business-oriented (process-driven) applications and their actual implementation, exploiting the full potential of the service-oriented idea and keeping the SOA promise. To provide such a description, the individual aspects that are relevant to process-driven applications, such as loose coupling, transaction handling using compensation, error tolerance, the role of service repositories, the requirements that services need to meet to support process-driven applications, and so on, need to be combined into a harmonious ensemble. To this aim, this book sets out a practical architecture proposal that takes these aspects into account and asserts the separation of business and technical processes. This book is also a plea for sustainable architectures that are robust yet flexible, that can adapt swiftly and effectively, in a way that exactly reflects management's business strategy. With a well thought-out architecture, the time lag between the announcement of a new (business) strategy and its implementation can be reduced to a minimum.

I will also consider new possibilities for the implementation already mentioned. I will explain how the latest version of Business Process Model and Notation (BPMN) (version 2.0) can be used for exactly this purpose, since there have been interesting new developments as regards the implementation of collaborative but also integration-centric processes. I will examine these new developments with reference to actual examples, and assess their potential and their limitations.

To summarize, the design and implementation approach presented in this book is based on the following three pillars:

1. The use of a *top-down methodology* (I also use the term *process-driven methodology*) to determine the main components/artifacts of the solution. The business processes are undoubtedly the drivers of the solution; hence the term *process-driven* in the title of the book. Throughout the book I also refer to solutions created in this way as *process-driven applications*, to emphasize this point.
2. A *sustainable architecture* (the same as a *process-driven architecture*) for process-driven applications that separates the actual business processes at the level of the end application and the technical processes of the so-called service contract implementation layer. The assignment of tasks to the two layers is clearly defined, as is their interaction with each other. The separation by a service contract protects the business processes from the constant changes at the system level.
3. The *use of Business Process Model and Notation (BPMN) throughout* for modeling as well as for implementing all process components at the business application level and at the level of the service contract implementation layer. As you will also see, a differentiation can be made in the service contract implementation layer between stateful and stateless processes.

1.2 Book Structure

To help achieve our goals of flexibility, scalability, and fault tolerance, this book will explain the concept of the process-driven application in more detail (Chap. 2). We will look at the typical properties of these applications and how they differ from other application types such as tightly coupled stand-alone programs, project-specific solutions, and pure integration applications. I will use examples to illustrate the ideas that composite applications are based on.

Once you have an understanding of process-driven applications, Chap. 3 moves on to the basic architecture of composite applications and the method for determining the main components of a composite (processes, user interfaces, data in the form of business objects, the required and provided services, and the service contract itself). Chapter 4 looks at an actual implementation of a process-driven application. BPMN, the standardized graphical notation for modeling business processes, plays an important role here, as already mentioned, and will be discussed in detail. As well as covering the theory, I will also use an example to demonstrate that the desired effect can actually be achieved. I will use a purchasing process as an example to demonstrate the implementation in software using SAP Process Orchestration, briefly describe the main implementation steps, and evaluate the experiences.

Process-driven applications encompass your critical business processes, they handle distributed architectures and a host of potential error situations. Still, to ensure that they can match the robustness and reliability provided by conventional tightly coupled applications, they need more than just a solid architecture. There are numerous additional technical concepts that you can use to help stabilize the overall solution, such as optimized locking in the involved systems, idempotence support in services, transaction processing using compensation, a troubleshooting concept close to the participating back-end systems, and so on. Chapter 5 investigates how these additional measures interact and how you can model them using BPMN.

Chapter 5 also introduces typical BPMN process fragments (patterns), which you can use to connect process-driven applications to systems in IT landscapes. In this context I will also present an enhancement proposal for BPMN that will make it possible to model integration processes in particular even more precisely. Chapter 5 continues with a discussion of the different approaches for increasing the flexibility of the process-driven application itself, and also the integration with the back-end systems. Business rules and analytical applications have an important role here.

Chapter 5 concludes with an analysis of the current discussions about structured and unstructured processes (keyword: adaptive case management), as well as their possible effect on process-driven applications.

Chapter 6 sums up the findings of the book and concludes by examining how customers can enhance composites like a product, without making any modifications. I will briefly cover the new developments that are emerging in this area.

The appendix contains an overview of the BPMN elements and an excursus on different approaches to managing services.

At this point, I would once again emphasize that this book does **not** address all facets of the development of service-oriented software. Neither is it a comprehensive introduction to BPMN. There is already excellent literature available on this topic, such as *BPMN 2.0* by Thomas Allweyer (Allweyer 2010b) and *Real-Life BPMN: Using BPMN 2.0 to Analyze, Improve, and Automate Processes in Your Company* by Jakob Freund and Bernd Rücker (2012). In his book *BPMN Method & Style* (Silver 2011), Bruce Silver describes very well how to implement BPMN correctly on the business side and how, in this way, you can distribute even complex processes into manageable components that can communicate with each other.

The focal point of this book is how to create an architecture for business applications that span multiple business processes, based on a process-driven methodology and producing solutions that exploit the full potential of SOA, and how to implement it using BPMN. You can, of course, deviate from the architecture proposed, and develop your applications using only selected parts of the recommended approaches. The book informs architects who choose to do so of the possible disadvantages if they deviate from the solution described here, thus enabling them to make an informed decision about how best to proceed.

I have intentionally left out subjects that have little relevance for the architecture of a process-driven application, such as the following:

- Organizational aspects of an SOA
- Service lifecycle
- Version management
- SOA and security
- Enterprise Service Bus
- SOA governance
- SOA project management
- Searching for services

These are covered quite adequately as individual topics in the SOA literature. In this regard, I would refer you to the (in my opinion, best) book about SOA, entitled *SOA in Practice*, by Nicolai Josuttis (Josuttis 2007). I can also strongly recommend *Enterprise SOA—Service-Oriented Architecture Best Practices* (Krafzig et al. 2004) by Dirk Krafzig, Karl Banke, and Dirk Slama.

1.3 SOA and Process-Driven Applications

Process-driven applications certainly share many properties, values, and goals with SOA, and you will encounter many commonalities throughout this book. However, the two approaches differ fundamentally with regard to the following: the *focus* of the approach, and *how* the applications and, in particular, the service contract are created. I am often asked what the actual differences are between the two approaches, so I would like to address this important aspect at the very start of the book. However, I must insert a disclaimer before I begin my comparison: Since SOA does not have an exact definition, but is rather an architecture paradigm (Josuttis 2007), all I can do is compare process-driven applications with *my interpretation of SOA* (and this applies to all comparisons with SOA in this book). And this is the crux of the matter: SOA is a moving target and each person has a different interpretation of one point or another. Even SOA experts find it difficult to pin down the term SOA and its characteristics. This is evident from the SOA manifesto (Erl et al 2009), which was formulated by industry thought leaders who came together "to hash out a formal declaration of the vision and values behind SOA and service orientation" (Erl 2013). Thomas Erl describes the process of negotiating the wording for the SOA manifesto. One participant, Nicolai Josuttis, published a German brochure called *Das SOA-Manifest. Kontext, Inhalt, Erläuterung* (Josuttis 2010), in which he writes about the particular difficulties they faced when searching for suitable formulations for the SOA manifesto. On page 7 of his brochure he explains:

> "Discussions were quickly reduced to the question of whether we could live with a specific formulation. We found that it was still possible for us to have different opinions about one point or another, because we could interpret the formulations that we decided on in different ways."

It is exactly this room for interpretation that is my main criticism of SOA, and this is probably where it differs most from the process-driven applications discussed in this book. Whereas SOA leaves a great deal open to interpretation (which, unfortunately, leads to unnecessary and time-consuming debate about the basic principles of the approach), I want this book to be as practical and unambiguous as possible. I want a specific application category (process-driven application) with a specific application architecture (process-driven architecture) to solve a specific problem (process-driven application development based on a distributed IT landscape), with a specific method (process-driven methodology).

However, since the SOA manifesto is the closest thing we have to a characterization of SOA, I do make reference to it in the following comparison. The SOA manifesto itself has three sections: a preamble, a value system, and the SOA principles upon which the aims of SOA as well as the value system are based. The preamble introduces the terms *service orientation* and *SOA*, and sets out the aims of SOA. The preamble is followed by a list of value statements, in which the authors compare and prioritize particular values. The final section sets out 14 guiding principles that underlie the statements in the preamble and the value system.

1.3.1 Commonalities

First, I'd like to look at the commonalities. We can keep this section relatively short, since I would go along with nearly all of the statements in the SOA manifesto. For example, I agree completely with the following sentence in the preamble:

We have been applying service orientation to help organizations consistently deliver sustainable business value, with increased agility and cost effectiveness, in line with changing business needs.

This sums up the aims beautifully, and concurs fully with those of process-driven applications. Let me also take a few examples from the value system. I agree with these as well, and also rate them as particularly important (this does not imply in any way that the values that I do not mention are wrong or insignificant):

Business value over technical strategy

Strategic goals over project-specific benefits

Flexibility over optimization

Evolutionary refinement over pursuit of initial perfection

There is also a degree of agreement in the principles. The following principles are also very important for process-driven applications:

Products and standards alone will neither give you SOA nor apply the service orientation paradigm for you.

SOA can be realized through a variety of technologies and standards.

Separate the different aspects of a system that change at different rates.

Reduce implicit dependencies and publish all external dependencies to increase robustness and reduce the impact of change.

We can apply these principles unchanged to process-driven applications. Simply replace SOA with PDA (process-driven application). You can already see that we can draw many parallels between SOA and PDA. There are of course differences as well, as discussed below.

1.3.2 Differences

I touched on the two main differences at the beginning of this subchapter: setting the focus and how, on this basis, the resulting applications and the required service contract result. Let us take SOA first. A focus on services is clearly expressed at several points in the SOA manifesto. The preamble states:

Service-oriented architecture (SOA) is a type of architecture that results from applying service orientation.

Josuttis expands on this in his commentary (loosely translated):

"Apart from stating the obvious, that is, that the focus is on services or service-related concepts, it is very difficult to come up with an exact definition (of service orientation)."

This concentration on services is a recurrent theme throughout the manifesto. Other examples:

Shared services over specific-purpose implementations.

Identify services through collaboration with business and technology stakeholders.

Maximize service usage by considering the current and future scope of utilization.

At every level of abstraction, organize each service around a cohesive and manageable unit of functionality.

This focus on services is intended to create a foundation upon which processes spanning multiple systems can *then* be built. Josuttis expresses this as follows (Josuttis 2010, p. 5, loosely translated):

"It is about creating a useful and appropriate *foundation* for business processes that are distributed across multiple systems or entire system landscapes."

These statements bring me to the second aspect, where I see a significant difference. The above quotations imply a bottom-up procedure: Only once services are established can you work on innovative processes. Accordingly, the SOA literature lends great importance to matters such as identifying services, selecting the right services and interfaces, service management, service governance, and the reusability of services defined in this way.

Process-driven applications, on the other hand, put the focus firmly on the new business processes to be implemented. This inevitably results in a different procedure: top-down, starting from the business processes (I call this *process-driven methodology*). This top-down approach also defines the service contract, which ultimately determines the external representation of the process-driven application in the form of interfaces, and encompasses the business functions that are provided and those that are required. These interfaces contain only the fields that are absolutely necessary, from the application perspective. The data types of the interface fields are based on a canonical data model. An important point to consider here is that, to start with, the interfaces of the service contract are not even expected to be reusable. They are tailored individually to the process-driven application. This embodies the objective of process-driven applications to be able to reuse the business functions themselves, but not their interfaces. In other words, the interfaces of the back-end system themselves do not play a role in the process-driven application. It is the business functions that are important. How they are actually called, is secondary. Our application requirements first meet the interfaces of the back-end systems in the form of the service contract at the service contract implementation level that implements the service contract. It is in this layer that mapping takes place.

This important aim, to reuse *business functions* and not *interfaces*, is summarized in the following note:

Note
Process-driven applications build on the reuse of existing business functions, but not on the direct reuse of the associated interfaces (for example, WSDL).

Therefore, process-driven applications are not at all concerned with the issues that are so essential in SOA (see the list above). Of course, they are based on the reuse of existing business functions, but they do not need to reuse the service interfaces at the level of the end application. Therefore, the service contract neutralizes all existing interfaces with the application. This method leads to different content and a different task distribution within the resulting architecture. At first glance, the application architecture may look very similar (the separation into layers and the distribution of the functions that are to be performed across these layers is crucial in both SOA and process-driven applications), but it differs considerably in its details. Consequently, I will talk of a *process-driven architecture*, to differentiate it from SOA. To sum up, process-driven applications represent a different method, which results in a different architecture with a different distribution of tasks within it, a different approach to services and, therefore, a different management. The book contains detailed discussions of all of these topics:

– Procedure: Sect. 3.1
– Architecture and distribution of functions: Sects. 3.3, 3.3.3 and 3.3.5
– Services: Sects. 3.2.6 and 3.3.1
– Management of services in repositories: Sect. 3.4

Finally, I would like to draw your attention to yet another positive effect between SOA and process-driven applications. If you have already established SOA in your company, a process-driven application will definitely benefit from the services that have been developed as a result of the SOA initiative. After all, they simplify the implementation of the service contract implementation layer significantly; the effort invested in their development is certainly not wasted.

1.3.3 Process-Driven Application, Process-Driven Architecture

The term *process-driven architecture* is not new. It has appeared in many articles and scientific publications (Margulius 2005; Strnadl 2006; Togliatti 2008, to name but a few) in which the authors recognize the significance of business processes in application development. However, these authors treat the business process part simply as a *complementary enhancement* to the service-oriented architecture; an SOA is a prerequisite, upon which you can implement the business processes. This is discussed by Krafzig et al. (2004) in Appendix A.2 entitled “BPM and the Process-Enabled SOA”. The term *process-driven architecture* is used, but essentially this is an SOA that is used as a basis for implementing processes (we can express this as a formula: process-driven architecture = BPM + SOA). Whether this alone justifies a new label is another matter. In my opinion, it does not adequately reflect the influence of the business processes as the drivers of the design of the architecture (and therefore the application), as described in my book. For me, the terms *process-driven application*, *process-driven architecture*, and *process-driven methodology* form a unit and one cannot exist without the others. I want to use this book to explore the differences from the approaches mentioned above, and elaborate on the details of the methodology, the architecture, and the applications that result. Let us begin.