

Executable business processes in BPMN 2.0

# Activiti IN ACTION

Tijs Rademakers  
Ron van Liempd



MEAP



MANNING



**MEAP Edition  
Manning Early Access Program  
Activiti in Action MEAP version 2**

Copyright 2010 Manning Publications

For more information on this and other Manning titles go to  
[www.manning.com](http://www.manning.com)

## Table of Contents

### **Part I: Introducing BPMN 2.0 and Activiti**

1. BPMN 2.0: what's in it for developers?
2. Introducing the Activiti tool stack

### **Part II: Implementing processes with Activiti**

3. Working with the Activiti process engine
4. Implementing a BPMN 2.0 process
5. Deploying a BPMN 2.0 process

### **Part III: Adding core logic to BPMN 2.0**

6. Applying core BPMN 2.0 constructs
7. Adding workflow to a BPMN 2.0 process
8. Integrating services with a BPMN 2.0 process

### **Part IV: Extending BPMN 2.0**

9. Ruling the business rule engine
10. Adding documents to a BPMN 2.0 process
11. Business events and BPMN 2.0

### **Part V: Managing and monitoring processes**

12. Managing the Activiti process engine
13. Reporting about the status of the process

### **Appendices**

- A. BPMN 2.0 language overview
- B. Activiti tool stack

## 1

## *BPMN 2.0: what's in it for developers?*

This chapter covers

- Introducing the world of BPM
- Designing processes with BPMN 2.0
- Implementing a BPMN 2.0 process with Activiti

We are about to start our journey in designing business processes with BPMN 2.0 and implementing these processes with the open source BPM platform Activiti. But before we explore these topics, we need some background into business process management (BPM).

The definition of business process management (BPM) is really broad, and BPM vendors are broadening the term even further every day. Because we can't (and don't want) to cover the full spectrum of what is covered by BPM, this chapter defines the boundaries what we'll cover in this book. You'll experience that this book is not about the theory behind for example business processes, business rules, business activity monitoring and straight through processing. This book will show how to develop and deploy business processes with BPMN 2.0 and the Activiti process engine. So this chapter shows we'll talk *No Fluff just Stuff*.

But before we dive into code examples starting in section 1.5, we'll first take a look at the topic of BPM. Once we have a good idea of this broad world, you'll be introduced in the BPMN 2.0 specification and why it's such an important industry standard. Then the theoretical foundation for this book is presented and we'll look into BPMN 2.0 from a pure developers perspective. That'll provide a good introduction before we finally start developing a simple process and run it with the Activiti process engine.

But we can't start developing BPMN 2.0 processes before we get a clear understanding of BPM and the wide range of business theories, but also different technologies it covers.

## 1.1 Taking a closer look at BPM

We already mentioned a couple of times that BPM covers a wide spectrum. And that's because BPM has a high goal that can be summarized as improving processes continuously and promoting efficiency and effectiveness. You can imagine that it covers a lot of different roles and players, including management, end users, business analysts, information analysts, architects, developers and system controllers, to be able to succeed in achieving these goals.

Goals like promoting efficiency and effectiveness are typical targets that the management of an organization tries to achieve. BPM can be regarded as a management discipline and therefore it's obvious that these kind of goals are part of the targets set by implementing BPM. In this book we don't focus on the management side of BPM, although we fully comprehend the importance of it. We'll focus on the technical aspects of BPM with process engines and business process management suites, which we'll discuss in more detail in section 1.2.

Our starting point and the central component within BPM is a business process. Simple examples of business processes are a vacation request process, or a book order process. Such a process consists of several activities that eventually result in you receiving a vacation request confirmation or the book you bought. Let's look at an example book order process in figure 1.1.

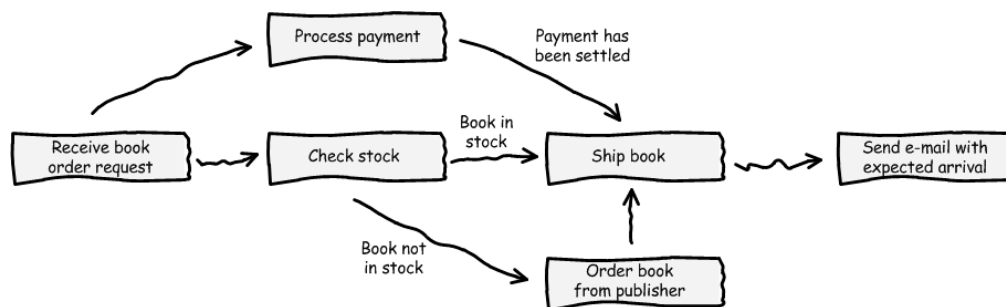


Figure 1.1 An example book order business process that processes the book payment and eventually ships the book to the customer.

The book order business process consists of six activities which may need to be executed. Once the book order request is received, the payment is processed to make sure the money is received and the stock of the book is checked. When the book is in stock, it can be shipped to the customer and a confirmation e-mail is sent. However, when the book is not in stock it needs to be ordered from the publisher of the book before it can be shipped. So the activity *Order book from publisher* is an optional activity, which is only executed when the condition *Book not in stock* is met.

As you can see this business process is fairly simple and incomplete. For example, what should happen when the process payment fails? Error handling is one of the challenging topics when developing business process. How to deal with error handling is covered in chapter 4.

Another part which the example doesn't cover is how the process will be triggered when the book is ordered from the publisher and has arrived at the book store. Because there are a lot of additional activities involved, like sending the order to the publisher and following up to the publisher when the book does not arrive in time, the "Order book from publisher" activity could be modeled as a sub process. Sub processes are a good solution to for example abstract a main process flow from all the details and to structure process logic for the purpose of reuse. In chapter 5 we look into sub processes in more detail.

To be able to implement even a simple business process as the given example a number of steps have to be performed. We'll now look at these important steps in the BPM life-cycle.

### **1.1.1 Walking around the BPM life-cycle**

The work to create a fully functional business process can be grouped together in five categories which are often referred to as the BPM life-cycle, which is shown in figure 1.2.

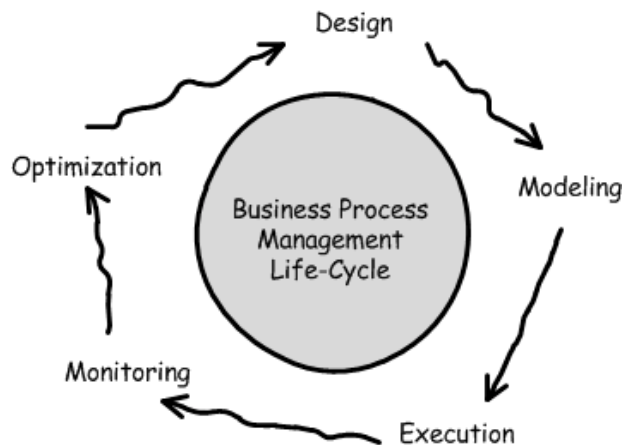


Figure 1.2 The five steps of the business process management discipline: Design, Modeling, Execution, Monitoring and Optimization.

Each of these five steps have an important job to do:

- Design; the first step contains the activities to define the business process. This means that the high-level activities are identified, possible organizational changes are discussed, service level agreements are defined and process details such as actors, notifications and escalations are specified.
- Modeling; in this step the business process is fully specified and validated. The

process flow is formalized by using for example BPMN, process variables are defined and candidate services are identified that can be used to execute an activity. To validate the business process *what-if* scenarios are performed with process simulation.

- Execution; the modeled business process is implemented in a business process application, often a business process management system (BPMS) such as Activiti. Technical details still need to be added to the business process to be able to execute it. And the process is implemented with a process language like WS-BPEL or BPMN 2.0.
- Monitoring; the processes are monitored for business goals that are defined by key performance indicators (KPIs). Examples of KPIs are the average number of orders received in a day should be between 20 and 30, and the time to send a proposal to a customer based on a web inquiry should not be more than 8 hours.
- Optimization; based on new insights, changing business requirements and monitoring results, the implemented business processes will need to be optimized. When the optimization phase is done, the business process goes into the design phase again and completes the circle.

The BPM life-cycle shows that implementing business processes results in an ongoing process due to the ever changing business environment and need for optimization. However, in practice it very much depends on the business environment and the ability of a business to execute how fast the BPM life-cycle is run through. In some business, it may take years to complete the cycle, in others it can be weeks or even days. In this book we'll focus on the execution step of the BPM life-cycle as this book is targeted at getting business processes running on the Activiti process engine.

To be able to work yourself through the BPM life-cycle, we first must establish a common vocabulary so we can communicate with BPM language.

## **1.2 Speaking the BPM language**

We already spoke about business processes and the BPM life-cycle, but there is a lot of other terminology used in the BPM space. Therefore we will look at the most important parts of the BPM language in this chapter to get up to speed for the rest of this book.

### **1.2.1 Getting the big picture**

First, let's look at the big picture of the BPM terminology by looking at a typical BPM architecture. Such an architecture is implemented by a business process management system or suite if you like (BPMS). There are a lot of software vendors who provide BPMS software (such as IBM, Oracle, Tibco, Intalio and of course Alfresco with Activiti), but all their suites contain similar tools. Figure 1.3 provides an overview of a typical BPM architecture.

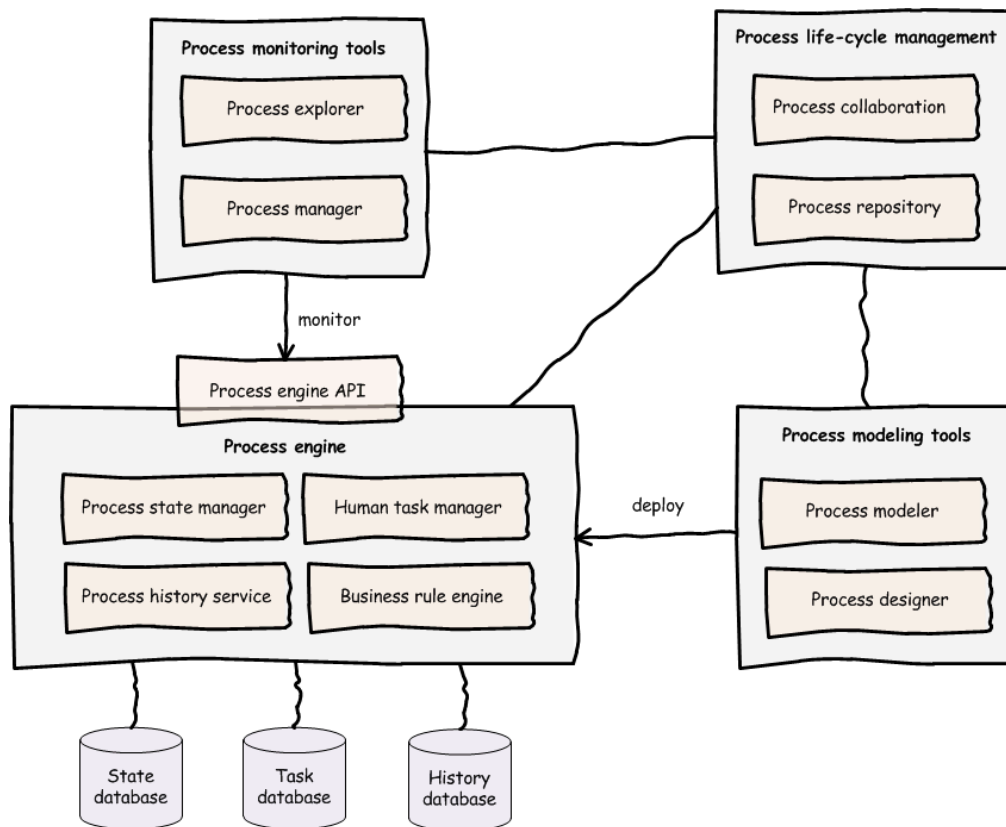


Figure 1.3 An overview of a typical BPMS architecture, with a process engine in its core and surrounded by process modeling, monitoring and life-cycle management tools.

There are quite a lot of components mentioned in figure 1.3, but the core of a BPMS is without doubt the process engine. In most chapters of this book we'll therefore discuss this part of a BPMS in detail. The process engine is responsible for handling the process state for example. Because processes can run for days, weeks or even months it's vital that the process state is persisted in a database. In case of hardware or software failure, the process state can always be recovered based on the process state retrieved from the database.

In the rest of this section we will look at each of the four core components of a BPMS architecture, first the process engine, then the process modeling tools, third the process monitoring tools and eventually the process life-cycle management tools.

### 1.2.2 Exploring the core process engine

A process engine is the foundation for a business process management suite. The process engine is capable of executing the business process model with quality aspects such as



availability, performance and robustness in mind. In the next subsections we discuss the core parts of the process engine, the process state manager and the human task manager. The business rule engine integration will be discussed in chapter 9 and we'll be talking about the process history service in chapter 12 and 13.

#### **PROCESS STATE MANAGEMENT**

One of the common requirements of a business process is that it must be able to run over a longer period such as a day or a week. In that period the business process must be capable of accepting requests and handling events and progress a step further in the business process definition. A simple implementation of a process engine supporting this requirement would keep the running business processes in memory until it eventually reaches its end state. This would however lead to a very high consumption of memory when a lot of business processes are running.

Therefore a core capability of a process engine is to have good process state management. A common solution for managing the process state is to have the current state of a process stored in a database at certain points in a process execution. When we talk about the process state you can think of the following parts of a process:

- Process variables
- The current activity in the process execution, often named the process execution token.
- Assigned users / groups in a user task
- The start and end time of executed process activities

Because the process state is stored in a database, the process engine can also resume running processes after a server shutdown, due to hardware failure or regular maintenance. So the process state manager is a vital component of the process engine.

#### **HUMAN TASK MANAGER**

Another important component of a process engine is a human task manager to support workflow functionality. With workflow functionality, we mean the creation of user or group assigned tasks during process execution. In BPMN 2.0 this can be implemented with the user task construct.

The human task manager should be capable of assigning tasks to specific users or groups of users. This also means that an API must be available to query the human task manager for tasks available to a specific user. A task can have at least three different states.

1. The task is unassigned, which means that the task is available to every user that is part of the user or group criteria specified in the task configuration. User or group criteria define the users who can claim a user task. Often, there is support for LDAP queries to fill-in these criteria.
2. The task is assigned, which means the task is assigned to one and only one user. This happens when a user has claimed the task or when the task was assigned directly to the user as defined in the task criteria. An assigned task will not appear on other

user's task lists. Of course administrators and managers still can perform commands on this tasks in exceptional cases.

3. The task is completed, which means the task is completed by the user who has claimed the task in the first place.

### **1.2.3 Modeling and execution tools**

It's great to have a process engine to run your business processes, but a BPMS also need modeling and development tools to create the business processes. Before BPMN 2.0 it was easier to differentiate between modeling and development tools, because modeling tools used a modeling notation like BPMN 1.x and development tools used a process execution language like WS-BPEL or jPDL for jBPM.

With BPMN 2.0 we now have a standard that provides a modeling notation as well as a process execution language. This means that it's not necessary any more to convert and transform a BPMN 1.x business process model to WS-BPEL. But as we will see in section 1.3.3, there's still a difference between the level of detail used in modeling and execution steps from the BPM life-cycle. A modeling tool has to support features to do high-level modeling and a development tool must be able to create a process execution model that can be directly deployed to a process engine. There may be tools which support modeling as well as development functionality, but we'll discuss the functionality separately here.

#### **MODELING A BPMN BUSINESS PROCESS**

A BPMN modeling tool should support the definition of business processes by business and information analysts. This means that in a modeling tool we don't want to be restricted too much by the formal BPMN 2.0 specification, but the focus should be to support the user in the definition of the business process model in a user-friendly way.

In addition, it would be nice to see simulation functionality in the modeler to be able to see if the modeled business process executes as expected. There are pretty advanced simulation tools available that can calculate maximum throughput and can identify possible bottlenecks in a business process.

#### **DEVELOPING A BPMN BUSINESS PROCESS**

The development tool takes a BPMN business process defined in the modeler as starting point or you can decide to develop a business process from scratch if you want to. A development tool is often dedicated to support a specific implementation of a process engine. Where a modeler tool can and should be vendor independent, a development tool can contain vendor specific details and have support for vendor specific functionality inside a process engine.

Important functionality for a development tool is to be able to add for example error handling to a process, to couple service tasks to web services via WSDL and Java classes, and to couple user tasks to LDAP related queries. Furthermore, it must be possible to run a process engine from within the development tool and deploy processes on it for testing and debugging purposes. Finally it must support unit testing, so a developer is able to perform some basic unit tests before a process is deployed to a central process engine.

### **1.2.4 Managing and monitoring the process engine**

Management functionality for a process engine means that an administrator must be able to query the process engine for deployed process definitions and look into version information. In addition, running process instances can be queried to look into specific states of processes when end users have questions about it. Also for user tasks it must be possible to claim and complete certain user tasks when there is a need for this.

In general, the deployed process definitions, process instances and user tasks must be available for an administrator. This also includes the process engine databases, which are a core part of the process engine. So the database model and its tables should be easily accessible for administration purposes.

Monitoring functionality is more targeted at specific business events that occur in a process instance. For example, when a large order enters an order process, it may be interesting for specific managers to receive a signal. Another more technical example is that a process instance is inactive for more than a week, it may need attention to see if the process is still running normally. It's obvious that management and monitoring is a very important part in the business success of process engines and BPM in general.

### **1.2.5 Collaborating on business processes**

Collaboration tools are a central part of a lot of IT projects in general. Collaboration tools include for example source code repositories, code review tools and wiki pages. But in BPM projects it's even more important due to the complexity and the large amount of different roles and persons in such a project.

BPMN provides a good foundation for collaboration because it supports high level business modeling understood by managers, it also supports detailed business modeling understood by business analysts, architects and designers and with BPMN 2.0 it also support process execution understood by architects, developers and administrators.

A BPM collaboration tool must provide a view on the different process model repositories of a BPM project. There are model, code and issue tracking repositories which must be bundled in one or more views depending on the end user. It's also very important that everybody working on a business process is identifiable with his role and work in this collaboration tool. It must be clear which person must be contacted for specific questions.

We rushed through the major components of the typical BPM architecture we showed in figure 1.3. But we'll take a closer look at every part in the rest of this book. For example we'll look at the process engine in chapter 3, discuss the human task manager in chapter 7 and look at the modeler, development and collaboration tools in chapter 2. For now it's good to have heard about the vocabulary, as we will be using it a lot in this book.

With a foundation of BPM knowledge and vocabulary in our minds it's time to get introduced into BPMN 2.0 and its history.

## **1.3 Evolution to BPMN 2.0**

Now that we established a solid foundation of BPM knowledge and terminology it's time to look at a language to implement a business process, which is the Business Process Model and

Notation (BPMN) 2.0. Before we start looking at the BPMN 2.0 language constructs it's good to know a bit about the history of this OMG standard.

### 1.3.1 Wasn't there a standard called WS-BPEL?

Right! From a developer's perspective, the first industry standard for implementing business processes was the Web Services Business Process Execution Language (WS-BPEL) specification. While BPMN 1.0 was already standardized and widely used by information and business analysts starting from 2004, WS-BPEL was the first BPM language that was used by developers to run processes on a process engine. In figure 1.4, the timeline of the WS-BPEL standard is shown.

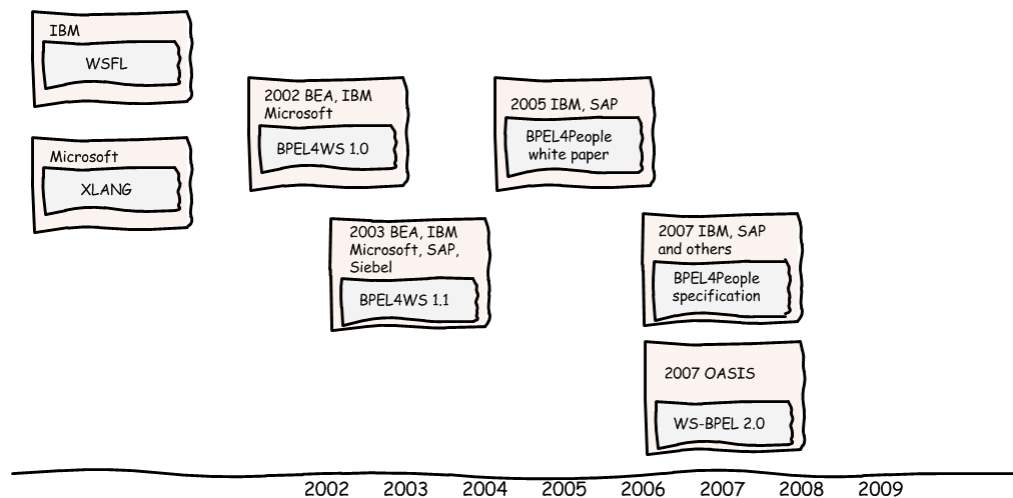


Figure 1.4 The timeline of the WS-BPEL 2.0 standard which was the successor of the BPEL4WS, WSFL and XLANG process languages.

The timeline shows that we already had executable process languages with the web services flow language (WSFL) by IBM and the XLANG specification by Microsoft. But these languages were not used by other software vendors, so the traditional vendor lock-in scenario was there. It took until 2002 before BEA, IBM and Microsoft made the business process execution language for web services (BPEL4WS) publically available. The purpose of team of software vendors was to standardize version 1.1 of BPEL4WS at OASIS. But it was not before 2007 until OASIS standardized the specification and renamed it to WS-BPEL 2.0.

Although the WS-BPEL 2.0 standard is quite successful at defining an execution model for business processes, important constructs were lacking. One important part is human task or workflow support, that is used to allocate work to a group of people or an individual. In figure 1.4 the BPEL4People specification is included, because this add-on specification to WS-

BPEL 2.0 does provide this functionality. But BPEL4People is not standardized and not fully embraced by BPM software vendors.

Another construct lacking in WS-BPEL is cyclic control flow. That sounds a bit complex, but it's nothing more than looping back to a previous activity in a process. In WS-BPEL this cannot be done, other than using a while loop with all kinds of difficult conditional logic. But let's not stick too long in the past, let's look at the new standard for implementing business processes, BPMN 2.0.

### 1.3.2 And then there was BPMN 2.0

While WS-BPEL was standardized in 2007, BPMN 1.0 was already standardized by the Business Process Management Initiative (BPMI) in 2004. BPMN 1.x is widely used as a modeling notation for business processes. So we as process developers, may have received a BPMN 1.x model for requirements or documentation purpose from a information or business analysts. But then we had to convert those models into an execution language such as WS-BPEL. But as figure 1.5 shows, now we have BPMN 2.0 to the rescue.

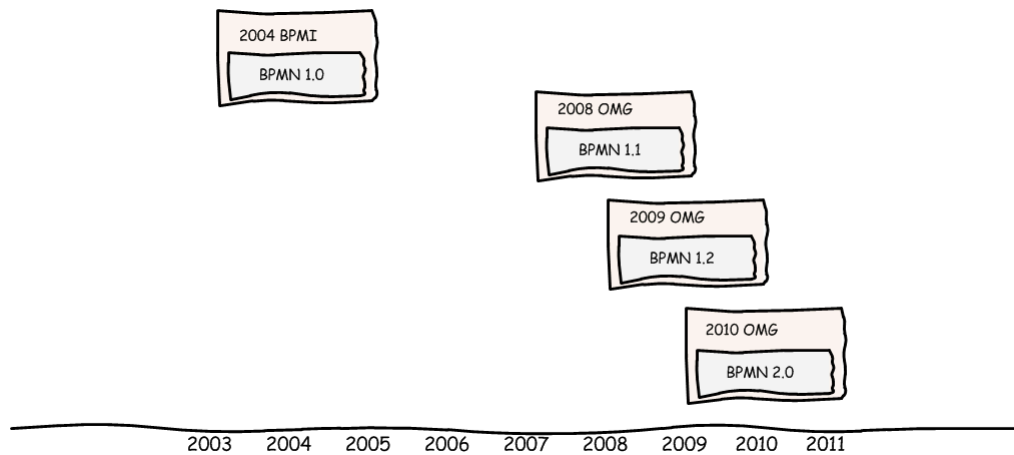


Figure 1.5 The timeline of the BPMN 2.0 standard which was the successor of the BPMN 1.x modeling notations.

So now we have a standard for modeling business processes AND implementing a process execution model. There's a real opportunity for business oriented people and developers to speak with the same vocabulary and to share business models without the need of conversion.

And because the BPMN 2.0 standard provides the opportunity to bring business and IT closer together, there's a real need for collaboration tools. A business process will be defined and implemented by a lot of people with different backgrounds. So it is a real challenge to

provide a toolset where each individual can do their job. In chapter 2, we'll look at Activiti Cycle, which provides such a platform.

With the history of BPMN 2.0 in our minds, we can now look at the elements of the language itself, and start modeling.

### **1.3.3 Getting your head around all the BPMN 2.0 constructs**

You only need to take a quick look at the BPMN 2.0 specification at the Object Management Group (OMG) at [www.omg.org/spec/BPMN/](http://www.omg.org/spec/BPMN/) and it becomes obvious that it's a quite substantial specification with around 550 pages and over a 100 of BPMN 2.0 constructs. It can be overwhelming to getting started with BPMN 2.0 and to comprehend the basics of the specification. Therefore, it's important to start with structuring the BPMN 2.0 in different groups of modeling detail.

An important advocate of grouping the constructs is Bruce Silver with his book BPMN Method & Style. The book is a very good read to getting started with modeling BPMN 1.x and 2.0 processes. In addition, Bruce categorizes the BPMN constructs into 3 different levels:

- Level 1 is described as descriptive BPMN, which can be used for high-level modeling with a restricted palette of BPMN constructs.
- Level 2 can be used for detailed modeling including event and exception handling and is described as analytical BPMN. It uses a wide range of BPMN constructs.
- Level 3 is the execution model of BPMN (which is new in BPMN 2.0) that can be used to be deployed on a process engine.

With these different levels in mind, it's easier to start with BPMN by using the level 1 group of BPMN constructs. Another important advocate of categorizing BPMN constructs is the Workflow Management Coalition (WfMC). The WfMC, and Robert Shapiro in particular, categorized the BPMN 2.0 constructs in four different categories (see [http://www.wfmc.org/standards/Update%20on%20BPMN%20Release%202.0\\_final.ppt](http://www.wfmc.org/standards/Update%20on%20BPMN%20Release%202.0_final.ppt) for more details), which are shown in figure 1.6.

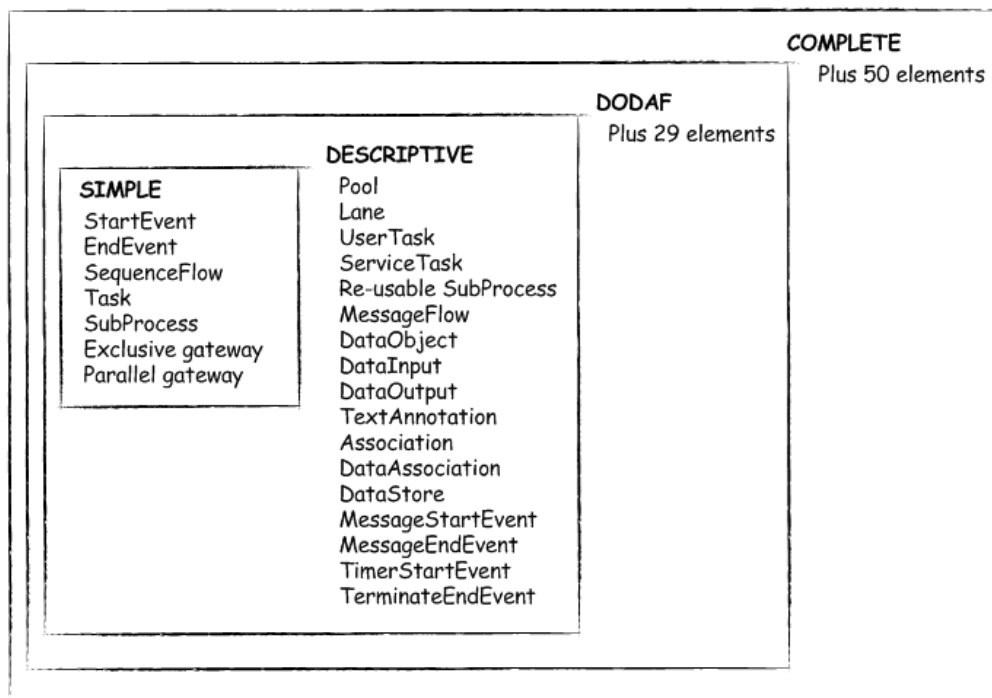


Figure 1.6 A categorization of BPMN 2.0 constructs by the WfMC. The simple category can be used for high level business modeling without a lot of restrictions. The descriptive category can be used for more detailed modeling by business and information analysts.

The categorization of WfMC as shown in figure 1.6 is similar to the level groups of Bruce Silver. The descriptive category can be compared to level 1, DODAF to level 2 and the complete palette to level 3. So the main difference is the simple category that can be used for very high-level modeling of business processes. And even then you can question if a vital construct of business process modeling like pool and lane should not belong to the simple category as well. But it's obvious that the level 1 palette of Bruce Silver and the descriptive category of WfMC are better starting points than the complete palette of the BPMN 2.0 standard.

Now we understand the history of BPMN 2.0, it's time to start looking at the actual BPMN constructs and do some modeling!

## 1.4 Introducing BPMN 2.0 from a developers viewpoint

To become familiar with the important constructs of BPMN, we'll build up the detail level of modeling similar to the categorization of WfMC and Bruce Silver. First, we start with a high-level model example and then we dive into a more detailed process. Level 3 of the Bruce

Silver categorization will be discussed in section 1.5, when we will implement our first process with the Activiti process engine.

### 1.4.1 High-level modeling with BPMN 2.0

In section 1.1 we looked at a sample book order business process. In figure 1.1 the book order process was modeled without a real model notation. With the simple or level 1 palette in mind, we'll take another look at the book order process and convert it into a real BPMN 2.0 business process model.

This means that we have to add a more formal notation to describe the book order process. So in the BPMN 2.0 book order process we'll use start and end events, parallel gateways, pools and tasks. Figure 1.7 shows the book order process modeled with a simple subset of the BPMN 2.0 palette.

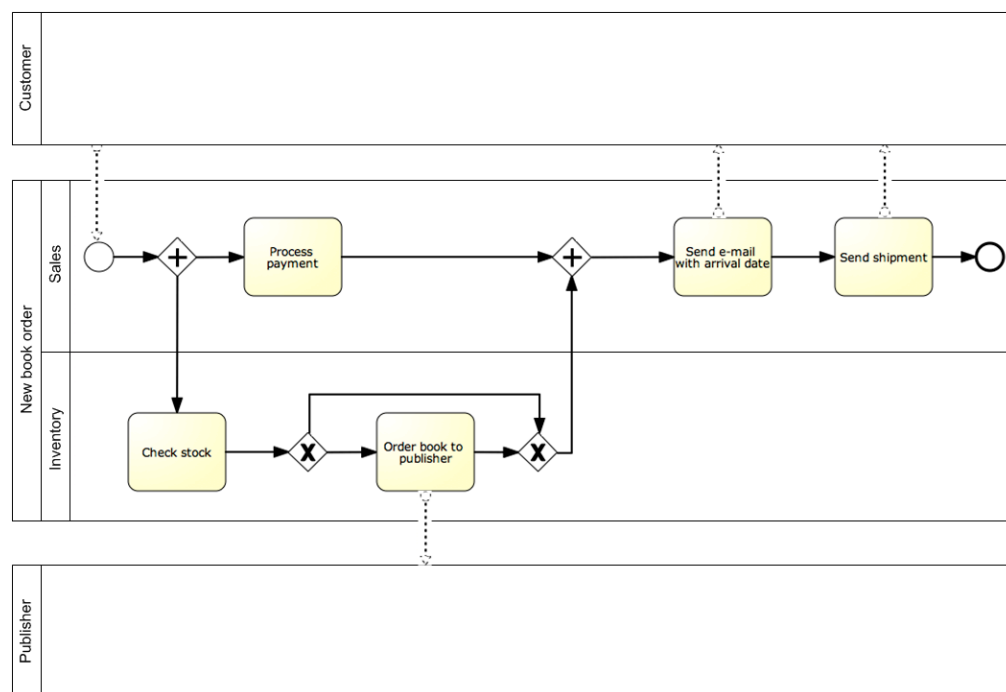


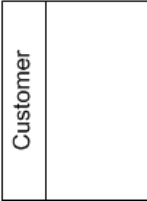
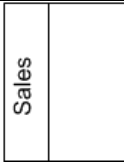
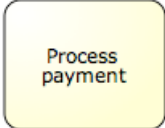





Figure 1.7 A high level BPMN 2.0 model describing the book order process with a simple subset of the BPMN 2.0 construct palette.

Before discussing the process in more detail let's first look at the individual BPMN 2.0 constructs in table 1.1.

Table 1.1 Overview of the BPMN 2.0 constructs used in figure 1.7.



BPMN 2.0 icon	BPMN 2.0 name	Description
	Start event	A start event is the trigger to start a new process instance.
	End event	An end event is the last step before the process instance is completed. The end event has a thicker circle border line than the start event.
	Pool	A pool represents the container for the activities of a process. Best practice is to use the process name for the pool name.
	Lane	A lane represents a role within a process model. In most cases this is a organizational unit or a role definition.
	Task	A task is a piece of work that has to be executed as part of the process definition. A task can be an automated activity or a manual activity.
	Parallel gateway	A parallel gateway is used to indicate that activities can be executed simultaneously or that all incoming activities must be completed before the process progresses to the next activity.
	Exclusive gateway	An exclusive gateway is used for conditional logic. Based on a condition only one of the outgoing sequence flows will be followed.
	Message flow	A message flow is used to send a signal or message from one pool to another. It may not be used to connect activities within



Sequence flow

one pool.

A sequence flow connects activities, gateways and events to each other within one pool. It therefore represents the orchestration of the process definition.

Now we know the meaning of the individual BPMN 2.0 constructs let's walk through the full process model. One of the eye-catching differences between the model we've drawn in section 1 and figure 1.7 is the use of pools and lanes. In figure 1.7 there are three pools: Customer, Book store and Publisher. The pools describe the different roles that perform activities in the business process. In the book store pool there are two lanes that characterize the different organizational units within the book store. Because in this example the bookstore is a small company, there's only a sales and an inventory organizational unit.

In this business process model we focus on the book store, but we could also include the process activities that are necessary for the publisher to complete the order process. The process is started with a order request by a customer, which is pictured here as a message flow (the dashed line) initiating a start event (a circle).

When the process is started two tasks should be completed, which are the process payment and the check stock tasks. These tasks can be executed in parallel and therefore a parallel gateway is modeled after the start event. After the stock is checked an exclusive gateway is used for the conditional logic of the book being in stock or not. When the book is not in stock it's ordered from the publisher by an additional task.

When the book is in stock, either because it was already or by an extra order via the publisher, the book is ready for shipment. But before the book will be shipped we must be sure that the payment has been successfully completed. So a parallel gateway is used to join the tasks, meaning that the process will go further before these tasks have been completed.

After the parallel gateway two additional tasks are executed that ship the book to the customer and inform the customer about the arrival date before the process is completed by an end event. As you see the high level model doesn't contain stuff like error handling or the definition of the type of a task. That's what we will add in the next section.

### **1.4.2 Detailed process modeling**

Although a book ordering process may seem simple at first hand, when looking at it in more detail a lot of process logic is needed. In this section we'll focus on detailing the process payment task by adding validation and error handling logic. This also means we'll need BPMN 2.0 constructs that are part of the descriptive or level 2 palette. In figure 1.8 the sub process payment is shown, which is a more detailed model of the process payment task of figure 1.7.

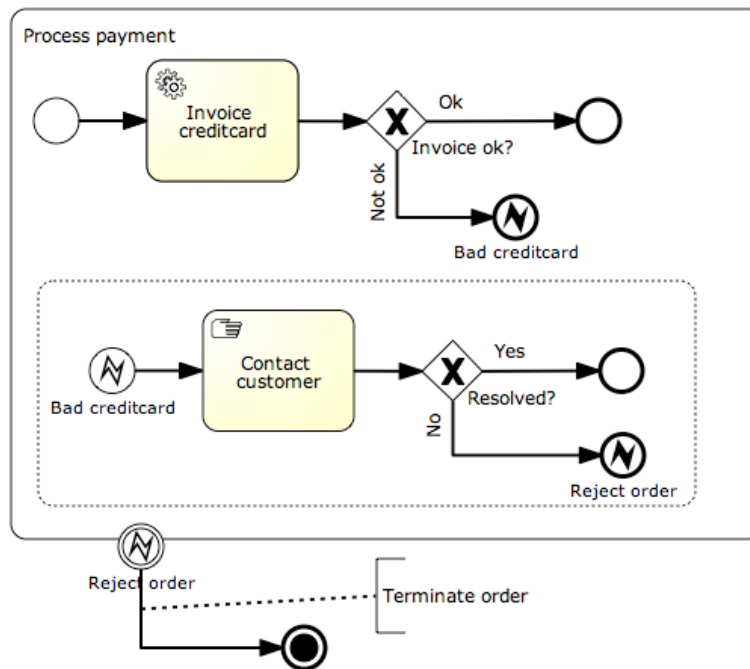




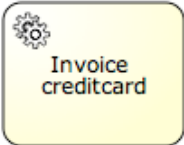
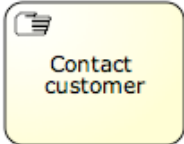
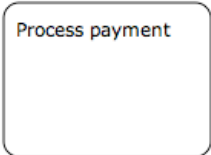
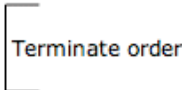


Figure 1.8 The extracted process payment sub process from the book order process model. The sub process shows the use of error end and start events and the use of a terminate end event.

As you can see we use a number of additional BPMN 2.0 constructs in the process model of figure 1.8. So let's first look at these extra element definitions in table 1.2.

Table 1.2 Overview of the additional BPMN 2.0 constructs used in figure 1.8.

BPMN 2.0 icon	BPMN 2.0 name	Description
 Bad creditcard	End error event	An error end event is a specific kind of end event, which can be used to throw an error inside the process model definition.
 Bad creditcard	Start error event	A start error event can be used to catch a specific error thrown by an error end event, for example within a sub process.

 Reject order	Intermediate error event	An intermediate error event can be used to indicate a fault on a task or a sub process.
	Terminate end event	A terminate end event is a special kind of end event that causes the process to be terminated. If a terminate end event is used in a sub process it only causes the sub process to be terminated, not the parent process.
	Service task	A service task is a specific type of task, which represents an automated activity. A service task can be a web service call or for example a Java class invocation.
	User task	A user task is a specific type of task, which represents a manual activity. A user task can be claimed and completed by a configured individual or group of users.
	Sub process	A sub process is a compound activity, which can contain multiple other activities including tasks, gateways and events. A sub process can be embedded in the parent process or be a standalone process model that can be invoked by the parent process via a call activity.
	Text annotation	A text annotation can be used to add documentation to specific elements of the process model.

In figure 1.8 we made the tasks more specific by adding a type identifier. For example the check credit card task is modeled as a service task, because the validity of the credit card can be checked by invoking a web service. We also added a user task to indicate that the task has to be performed by a human. The contact customer activity is a user task, because an employee of the book store has to get in contact with the customer to solve the bad credit card problem.

In the process payment sub process shown in figure 1.8, we see a couple of other new BPMN 2.0 constructs. First note that a sub process always starts with a start event. First the

credit card information is validated by invoking an automated task. Then we check the outcome of this credit card validation with an exclusive gateway. In case the credit card validation was successful the payment is finished and the sub process is ended.

When the credit card validation didn't succeed an error end event throws an exception. This exception is caught within the sub process by the error start event handling the bad credit card exception. In this case the customer is contacted personally by an employee of the book store to see if the credit card information was not received in a good manner or that the customer can pay in another way. If the payment can be settled with the customer a normal end event in the exception handler is reached and the sub process is completed by executing the finish payment service task.

However, if the payment can't be settled another error end event throws a reject order exception. This exception is not handled within the sub process, but with a error boundary event. This error boundary event handles the exception by forwarding it to a terminate end event, which causes the whole book order process to be terminated immediately.

We covered a lot of the most important parts of the BPMN 2.0 palette in this section. Of course we haven't been able to discuss the full palette and by no means we talked about every BPMN 2.0 construct in detail. But this should provide you with a good start in BPMN 2.0 modeling and we'll be discussing more details of BPMN 2.0 throughout the rest of the book. First it's time for some action by implementing your first process with the Activiti tool stack and this is also your first introduction into the XML side of BPMN 2.0.

## **1.5 Action!**

We've done enough of talking about why BPMN 2.0 is bridging the gaps between business and IT and why it's a very interesting specification for us developers. In this section we'll get our hands dirty on our first process, implemented with the Activiti tool stack. In chapter 2 we'll provide a detailed overview of every tool that's part of the Activiti tool stack, so for now we'll start with a short background of the Activiti project. Then we're going into combat by downloading and installing Activiti and implementing a BPMN 2.0 process.

### **1.5.1 Getting to know Activiti**

The Activiti has a clear origin in the jBPM project of JBoss. The former lead developers of jBPM, Tom Baeyens and Joram Barrez, created a vibrant community around this process engine and they created a process virtual machine sub-project that enabled jBPM to be used for multiple process languages. Only recently jBPM started with the implementation of the BPMN 2.0 specification in addition to the jPDL supported process language.

At that point in time Alfresco was using jBPM in their document management system for the implementation of foremost workflow related functionality. But they were looking for a process engine with a more liberal open source license. Eventually Alfresco decided to create a liberal Apache licensed open source process engine themselves and contacted Tom and Joram. Activiti was born!

The Activiti project started off at a very fast pace and succeeded to do a monthly release of the tool stack until the stable 5.0 release in December 2010. Like the jBPM engine, Activiti

has a solid foundation with a process virtual machine. The process virtual machine ensures that Activiti will be future proof for new versions and other process languages. One of the main differences between jBPM and Activiti is that Activiti has been built with a focus on a BPMN 2.0 compliant process engine. So Activiti is a new project, but it's built with a lot of years of experience from the jBPM project. That makes the foundation of Activiti rock solid and ready for production usage.

Before we are moving to installing the Activiti tool stack, let's first take a quick look at the components that are part of the suite in figure 1.9.

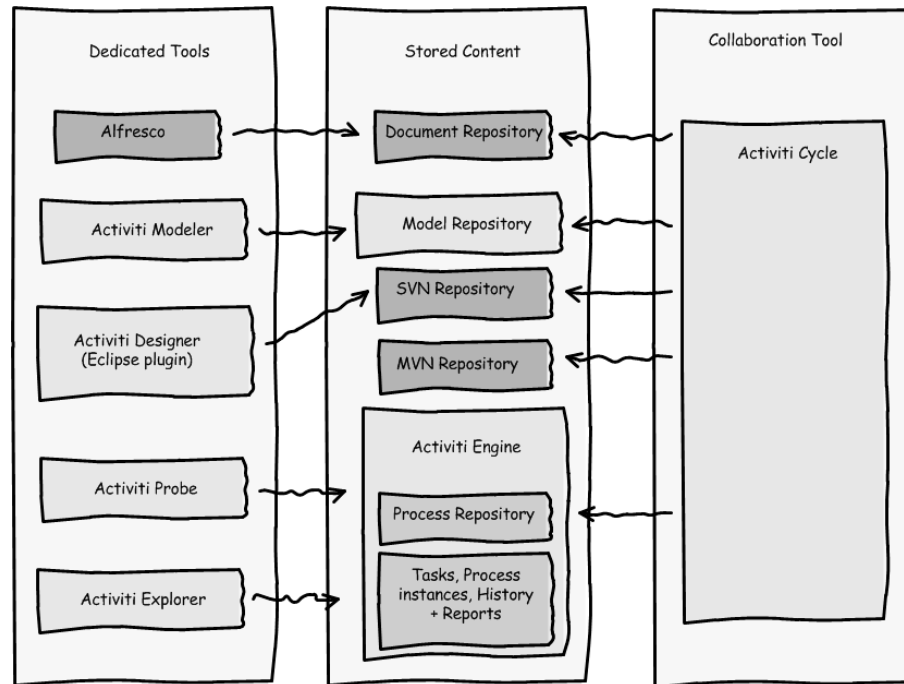


Figure 1.9 An overview of the Activiti tool stack, with in the center the Activiti process engine and on the right and left side the accompanying modeling, design, monitoring and management tools.

The center of the Activiti tool stack is of course the process engine we already discussed a bit. But a business process management solution is not ready for primetime with only a good process engine. The surrounding toolset is equally important.

With the Activiti modeler, the business and information analyst are capable of modeling a BPMN 2.0 compliant business process in a web browser. This means that business processes can easily be shared; no client software is needed before you can start modeling. The Activiti designer is an Eclipse based tool, which enables a developer to enhance the modeled

business process into a BPMN 2.0 process that can be executed on the Activiti process engine.

In addition to the modeling and design tools, Activiti also provides a web based management tool with Activiti probe. In Activiti probe you can get an overview of the deployed processes and even dive into the database tables underneath the Activiti process engine. The Activiti explorer gives users the possibility to interact with the deployed business processes. You can for example get a list of tasks assigned to you by running processes. The explorer additionally provides functionality to claim and finish such a task with flexible task forms where the necessary information about the task assignment can be shown and edited.

In addition to the common functionality a business process management suite like Activiti must provide, there's also Activiti cycle. We discussed the topic of bridging the gap between business and IT quite thoroughly in this chapter. To be able to work in a project team with both business as well as IT people, collaboration tools are vital. Activiti cycle provides such a collaboration tool and enables users to view a business process from a model, design as well as a XML perspective.

Now that we know a bit more about Activiti let's get this business process management suite installed on your development machine.

### **1.5.2 Installing the Activiti tool stack**

Tom Baeyens and his team have been able to provide a smooth installation process. The first thing you have to do is to point your web browser to the Activiti website at <http://www.activiti.org>. There you will be guided to the latest release of Activiti via the download button. Just download the latest version and unpack the distribution to a logical folder such as:

```
C:\activiti (Windows)
/opt/activiti (Linux or Mac OS)
```

When working with other process engines downloading the distribution, which is also a lot larger than the couple of MBs of Activiti, is only the beginning of a long and complex installation procedure. With Activiti, there's a setup directory which contains an Ant build file that installs the full Activiti suite. The directory structure of the distribution is shown in figure 1.10.

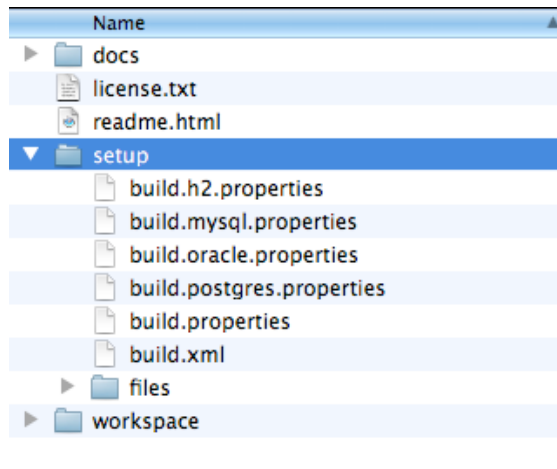


Figure 1.10 The directory structure of the Activiti distribution with the setup directory and the Ant build.xml file as the main parts for the installation procedure.

Before you go further with the installation procedure, make sure that you have installed a Java 5 SDK or higher, pointed the JAVA\_HOME environment variable to the Java installation directory, and installed Ant (<http://ant.apache.org>). Shortcuts to the Java SDK and the Ant framework are also provided on the Activiti download page.

The last thing to make sure is that you have an internet connection available without a proxy, because the Ant build file will download additional packages. If you are behind a proxy, make sure you configured the Ant build to use that proxy.

When you open a terminal or command prompt and go to the setup directory shown in figure 1.11, you only have to run the simple `ant` command (or `ant demo.start`). This will kick off the Ant process, which will look for a `build.xml` file in the setup directory. The installation that now takes includes the following steps:

1. A H2 database is installed to `/apps/h2` and the H2 database is started on port 9092.
2. The Activiti database is created in the running H2 database.
3. Apache Tomcat 6.0.x is downloaded and installed to `/apps/apache-tomcat-6.0.x`, where x stands for the latest version.
4. Demo data including users, groups and business processes are installed to the H2 database.
5. The Activiti engine, REST API, explorer, probe, cycle WARs are copied to the webapps directory of Tomcat.
6. The modeler is downloaded separately and installed in the Tomcat webapp directory.
7. Tomcat is started with the Activiti toolstack.



8. A couple of web browsers are started with Explorer, Probe, Cycle and Modeler.

When the Ant script is ready, you have the Activiti toolstack installed and also running. That's not bad for about a minute of installation time. The Ant build file is not only handy for the installation of Activiti, but also for doing common tasks like stopping and starting the H2 database (`ant h2.stop`, `ant h2.start`) and the Tomcat server (`ant tomcat.stop`, `ant tomcat.start`) and for re-creating a vanilla database schema (`ant db.drop`, `ant db.create`). So it's worth the time to look at the Ant targets in the Ant build file.

The installation of Activiti consists foremost of several web applications being deployed to a Tomcat server with Activiti libraries and a ready-to-use H2 database with example processes, groups and users already loaded. For the full picture figure 1.11 shows the installation result in a schematic overview.

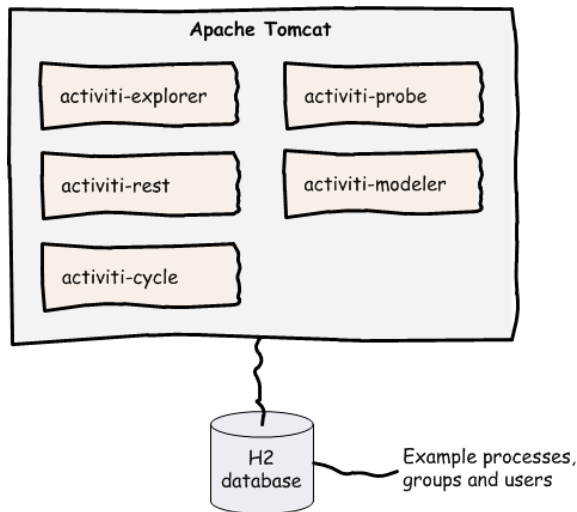


Figure 1.11 An overview of the installation result of the Activiti tool stack, including a running Tomcat server and H2 database with all the Activiti web applications already deployed.

To verify if the installation has succeeded the web applications listed in table 1.1 should be available via your favorite web browser. You can use the user `kermit` with password `kermit` to login to these applications.

Table 1.1 The URIs of the web applications available for you after the installation of Activiti

Application name	URI	Short description
Activiti Probe	<a href="http://localhost:8080/activiti-probe">http://localhost:8080/activiti-probe</a>	The Probe application provides an admin console on the engine. Use this tool to verify your installation.

Activiti Explorer	<a href="http://localhost:8080/activiti-explorer">http://localhost:8080/activiti-explorer</a>	The Explorer application can be used to work with the deployed processes. This is a good starting point to try the example processes.
Activiti Cycle	<a href="http://localhost:8080/activiti-cycle">http://localhost:8080/activiti-cycle</a>	Activiti Cycle is a collaboration tool that can be use to browse the process repositories.
Activiti Modeler	<a href="http://localhost:8080/activiti-modeler">http://localhost:8080/activiti-modeler</a>	The Modeler application can be used to model BPMN 2.0 business processes.

By trying the Activiti Probe application, you can verify if the installation was successful. After logging in, you should get a message saying that everything is running fine. Working with demo processes is fun, but it's even better to try out your own developed business process.

### 1.5.3 Implementing your first process in Activiti

In this chapter we referenced a book order process a couple of times. Now we have our BPM environment ready to be used, let's try and implement a simplified version of this business process. We could use the Activiti Modeler to first model the process, and the Activiti Designer to implement and deploy the process, but it's better to start off with a BPMN 2.0 XML document for learning purposes. So no drag and drop development, but get ready for some XML hacking.

We already introduced a lot of BPMN 2.0 constructs in section 1.4 and we also looked at a BPMN 2.0 XML document there. So let's create a clean XML document for the book order process with a start and end event as shown in code listing 1.1.

#### Listing 1.1 bookorder.simple.bpmn20.xml document with only a start and end event

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"          #1
    targetNamespace="http://www.bpmnwithactiviti.org">                    #2

    <process id="simplebookorder" name="Order book">
        <startEvent id="startevent1" name="Start"/>
        <sequenceFlow id="sequenceflow1" name="flow"                      #A
            sourceRef="startevent1" targetRef="endevent1"/>
        <endEvent id="endevent1" name="End"/>
    </process>
</definitions>
```

**#1 Root element of BPMN 2.0 XML**

**#2 The process namespace**

**#A Connecting start to end event**

A BPMN 2.0 process definition always starts with a `definitions` element #1 with a namespace of the OMG BPMN specification. Each process definition must also define a namespace, we have defined here a `targetNamespace` #2 of the book's website. Activiti also

provides a namespace which enables us to use Activiti extensions to the BPMN 2.0 specification as we will see in chapter 3. We can now run this very simple process to test if we have the process defined correctly and the environment set-up in the right manner.

To test the process, you have to create a Java project in your favorite editor. In this book we'll use Eclipse for the example description, because the Eclipse Designer is only available as an Eclipse plug-in. You can also download the source code from the book's website at Manning and import the examples from there. After your Java project is created, the Activiti libraries have to be added to the Java build path. The source code of the book uses Maven to retrieve all the necessary dependencies. The project structure of the example code is explained in detail in chapter 3. If you are already familiar with using Maven you can use the `mvn eclipse:eclipse` command or the m2eclipse Eclipse plugin to get the dependencies referenced from your Eclipse project, otherwise you can read the first section in chapter 3 how to setup the examples project.

With the dependencies in place you can add a class file with the name `BookOrderTest` to your Java project. The `SimpleProcessTest` class should contain one test method which is shown in code listing 1.2.

#### Listing 1.2 First example of a JUnit test for a Activiti process deployment

```
public class SimpleProcessTest {

    @Test
    public void startBookOrder() {
        ProcessEngine processEngine = ProcessEngineConfiguration           #1
            .createStandaloneInMemProcessEngineConfiguration()           #1
            .buildProcessEngine();                                         #1

        RuntimeService runtimeService =
            processEngine.getRuntimeService();
        RepositoryService repositoryService =
            processEngine.getRepositoryService();
        repositoryService.createDeployment()                               #2
            .addClasspathResource("bookorder.simple.bpmn20.xml")         #2
            .deploy();                                                     #2

        ProcessInstance processInstance =                                #3
            runtimeService.startProcessInstanceByKey(                     #3
                "simplebookorder");                                         #3
        assertNotNull(processInstance.getId());
        System.out.println("id " + processInstance.getId() + " " +
            processInstance.getProcessDefinitionId());
    }
}
```

**#1 Create the Activiti engine**

**#2 Deploy the simplebookorder process definition**

**#3 Start the bookorder process instance**

In just a few lines of code we are able to start-up the Activiti process engine, deploy the book order process XML file from code listing 1.1 to it, and start a process instance for the deployed process definition.

The process engine can be created with the `ProcessEngineConfiguration` #1, which can be used to start the Activiti engine and the H2 database, in this case an in-memory one. There are different ways to start-up an Activiti engine and we'll explain the options in detail in chapter 3. By the way, Activiti can also run on other database platforms than H2, like Oracle or PostgreSQL.

The next important step in code listing 1.2 is the deployment of the `bookorder.simple.bpmn20.xml` file we showed in code listing 1.1. To deploy a process from Java code we need to access the `RepositoryService` from the `ProcessEngine` instance. Via the `RepositoryService` instance we can add the book order XML file to the list of classpath resources to deploy it to the process engine #2. The process engine will validate the book order process file and create a new process definition in the H2 database.

Now it's easy to start a process instance based on the newly deployed process definition by invoking the `startProcessInstanceByKey` method #3 on the `RuntimeService` instance, which is also retrieved from the `ProcessEngine` instance. The key `bookorder` which is passed as process key parameter should be equal to the process `id` attribute from the book order process of code listing 1.1. A process instance is stored to the H2 database and a process instance `id` is created that can be used to as a reference to this specific process instance. So this identifier is very important.

Before you run the `startBookOrder` unit test, make sure that the `bookorder.bpmn20.xml` file is available in the source folder of the Java project. If that's the case you can run the unit test and the result should be green. In the console you should see a message like:

```
id 3 simplebookorder:1
```

This message means that the process instance `id` is 3 and the process definition that was used to create the instance was the `simplebookorder` definition with version 1. Now we have the basics covered, let's implement a bit more complex book order process, so we can use the Activiti Explorer to claim and finish a user task for our process.

### 1.5.4 Implementing a simple book order process

It would be a bit of a shame to stop chapter 1 with an example that contains just a start and an end event. So let's enhance our simple book order process with a script task and a user task, so we can see a bit of action on the Activiti engine. First the script task will print an ISBN number that will be given as input to the book order process when started. Then a user task will be used to handle the book ordering manually. So this is not the most efficient process, but it's good for learning about BPMN 2.0 and Activiti.

Activiti provides the possibility to use the scripting language you want, but Groovy is supported by default. So we'll use a line of Groovy to print the ISBN process variable. In code listing 1.3 a revised version of the book order process of code listing 1.1 is shown.

#### Listing 1.3 A book order process with a script and user task

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
  targetNamespace="http://www.bpmnwithactiviti.org">

  <process id="bookorder" name="Order book">
```

```

<startEvent id="startevent1" name="Start"/>
<sequenceFlow id="sequenceflow1" name="Validate order"
  sourceRef="startevent1" targetRef="scripttask1"/>
<scriptTask id="scripttask1"                                #A
  name="Validate order"
  scriptFormat="groovy">
  <script>
    out:println "validating order for isbn " + isbn;          #1
  </script>
</scriptTask>
<sequenceFlow id="sequenceflow2" name="Sending to sales"
  sourceRef="scripttask1" targetRef="usertask1"/>
<userTask id="usertask1" name="Work on order"                #2
  <documentation>book order user task</documentation>
  <potentialOwner>
    <resourceAssignmentExpression>
      <formalExpression>sales</formalExpression>           #B
    </resourceAssignmentExpression>
  </potentialOwner>
</userTask>
<sequenceFlow id="sequenceflow3" name="Ending process"
  sourceRef="usertask1" targetRef="endevent1"/>
<endEvent id="endevent1" name="End"/>
</process>
</definitions>
#A Definition of a script task
#B Assign task to sales group
#1 Print the isbn
#2 Definition of a user task

```

With the two additional tasks added to the process definition, the number of lines of the XML file grows quite a bit. In the next chapter we'll show the Activiti designer, which abstracts you from the XML file when designing the process.

The script task contains a `out:println` variable `#1`, which is a Groovy reserved word within the Activiti script task for printing text to the system console. Also notice that the `isbn` variable can be used directly in the script code without any additional programming.

The user task `#2` contains a potential owner definition, which means that the task can be claimed and completed by users that are part of the group sales. When we run this process in a minute, we can see in the Activiti Explorer that this user task is available in the task list for the user `kermit`, who is part of the sales group.

Now that we added more logic to the process, we also need to change our unit test. One thing we need to add is a process variable `isbn` when starting the process. And to test if the user task is created, we need to query the Activiti engine database for user tasks that can be claimed by the user `kermit`. Let's take a look at the changed unit test in code listing 1.4.

#### Listing 1.4 A unit test with a process variable and user task query

```

public class BookOrderTest {

    @Test
    public void startBookOrder() {
        ProcessEngine processEngine = ProcessEngineConfiguration
            .createStandaloneProcessEngineConfiguration()
    }
}

```

```

        .buildProcessEngine();

        RuntimeService runtimeService =
            processEngine.getRuntimeService();
        RepositoryService repositoryService =
            processEngine.getRepositoryService();
        TaskService taskService = processEngine.getTaskService();           #1
        repositoryService.createDeployment()
            .addClasspathResource("bookorder.bpmn20.xml")
            .deploy();

        Map<String, Object> variableMap = new HashMap<String, Object>();
        variableMap.put("isbn", "123456");
        ProcessInstance processInstance =                                   #2
            runtimeService.startProcessInstanceByKey(                       #2
                "bookorder", variableMap);                                   #2
        assertNotNull(processInstance.getId());
        System.out.println("id " + processInstance.getId() + " "
            + processInstance.getProcessDefinitionId());
        List<Task> taskList = taskService.createTaskQuery()                 #A
            .taskCandidateUser("kermit")                                     #A
            .list()                                                         #A
        assertEquals(taskList.size(), 1);
        System.out.println("found task " + taskList.get(0).getName());
    }
}

```

**#1 Get a TaskService instance**  
**#2 Start a process with a variable**  
**#A Find tasks available for kermit**

The `BookOrderTest` unit test has been changed to start a process instance with a map of variables `#2` that contains one variable with a name of `isbn` and a value of `123456`. In addition, when the process instance has been started, a `TaskService` instance `#1` is used to retrieve the tasks available to claim by the user `kermit`. Because there is only one process instance running with one user task we test that the number of tasks retrieved is 1.

Also note that we are not using the in-memory database anymore, but switched (`createStandaloneProcessEngineConfiguration`) to the default standalone H2 database that's installed as part of the Activiti installation procedure. So before running the unit test make sure the H2 database is running (`ant h2.start` or `ant demo.start`). Now we can run the unit test to see if our changes work. In the console you should see a similar output like:

```

validating order for isbn 123456
id 112 bookorder:1
found task Work on order

```

The first line is printed by the Groovy script task in the running process instance. The last line confirms that one user task is available for claim for the user `kermit`. Because a user task is created we should be able to see this task in the Activiti Explorer. Confirm that Tomcat has been started (`ant tomcat.start` or `ant demo.start`).

Now point your browser to <http://localhost:8080/activiti-explorer> and login with the user `kermit` and the same password. When you click on the link "Unassigned tasks" you should a

screen with one user task with the name of "Complete order" like the screenshot shown in figure 1.12.

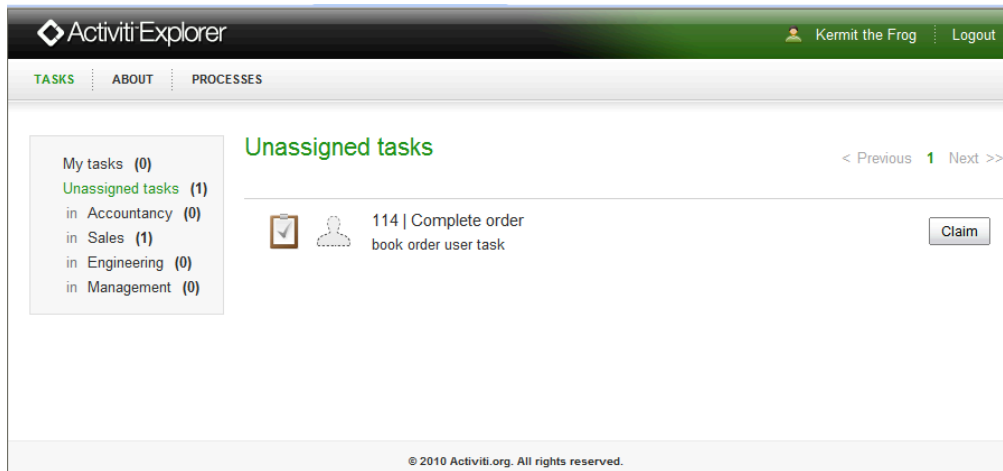


Figure 1.12 A screenshot of the Activiti Explorer showing the user task of the simple book order process.

For sake of completeness you can claim the user task and see that it becomes available in the "My tasks" page. There you can complete the tasks, which triggers the process instance to complete to the end state. We can now also take a quick look at the Activiti Probe application to look for this process instance in the Activiti process database table. So go to <http://localhost:8080/activiti-probe> and login again with the user kermit. Then go to "Database" and the "ACT\_HI\_PROCINST" database table as shown in figure 1.13.

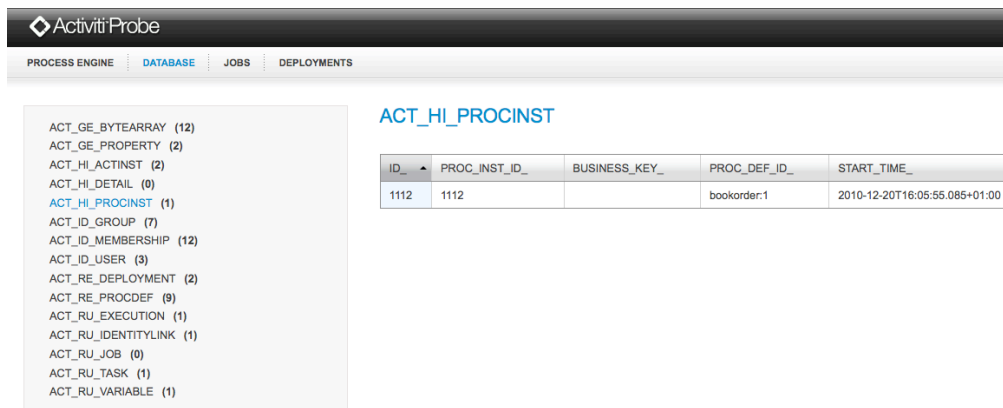


Figure 1.13 A screenshot of the Activiti Probe application showing the process instance database table, which contains the completed simple book order process.

Activiti Probe provides us with a web view on the database tables of the Activiti engine. In the historic process instance table (ACT\_HI\_PROC\_INST) you can see information like the start and end time of the process and the name of the last activity. More information about the tables that can be viewed with Activiti Probe can be found in chapter 3.

This completes our first journey of the Activiti toolstack. In the coming chapters we'll take a more detailed look at the Activiti toolstack and how to use the Java API of Activiti to for example create processes or retrieve management information. So hold on tight as we are about to expose you to more powerful functionality of BPMN 2.0 and Activiti.

## **1.6 Summary**

This chapter was really packed with information! We started with a gentle introduction into business process management and BPM vocabulary. We also saw a bit of history when we talked about WS-BPEL, BPMN 1.x and eventually BPMN 2.0. At that point it was time to take a closer look at the way you can do modeling with BPMN 2.0 and we looked at different categorization strategies of WfMC and Bruce Silver to make BPMN 2.0 understandable and usable for different users.

And finally we got acquainted with the Activiti tool stack and we were able to implement a simple book order process using a script and user task. And what's even more interesting, we were able to start the Activiti process engine, deploy our book order process, start a process instance and do some unit testing on it with just a couple lines of Java code. It's obvious that Activiti provides us with a powerful API and toolset. In the next chapter you will be introduced in more detail into the different parts of the Activiti tool stack.