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 version 7

For more information on this and other Manning titles go to
www.manning.com

Table of Contents

1 Introducing the Activiti framework

This chapter covers

- Introduction into Activiti
- Installing the Activiti framework
- Implementing a BPMN 2.0 process

You are eager to start learning about Activiti and start coding. This chapter we'll get you up and running in 15 minutes. First we'll take a closer look at the history of the Activiti framework and compare the functionality with its main competitors jBPM and BonitaSoft. Then we'll take a look at the different components of the Activiti tool stack including a Modeler, Designer and a REST web application.

Before we dive into code examples starting in section 1.4, we'll first make sure the Activiti framework is installed correctly in section 1.3. So at the end of this chapter you've a running Activiti environment and with a deployable example.

But we can't start developing before we get a clear understanding of the Activiti framework and understand the architecture that is built around a state machine.

1.1 Getting to know Activiti

When you start working with a new framework it's always good to know a little bit of the background of the project and to understand the main components. In this section we'll be doing exactly that.

1.1.1 A little bit of history

The Activiti project was started in 2010 by the former founder and the core developer of jBPM (JBoss BPM), respectively Tom Baeyens and Joram Barrez. The goal of the Activiti project is crystal clear: built a rock-solid open source BPMN 2.0 process engine. In the next

chapter we'll talk in detail about the BPMN 2.0 specification, but in this chapter we'll focus on the Activiti framework itself and getting it installed and up-and-running with simple examples.

Activiti is funded by Alfresco (known for its open source document management system Alfresco, see http://www.alfresco.com), but Activiti acts as an independent open source project. Alfresco uses a process engine to support things like a review and approval process for documents. This means that the document has to be approved by one user or a group of users. For this kind of functionality Activiti is integrated into the Alfresco system to provide the process and workflow engine capabilities needed to offer support for these processes. By the way, because jBPM was used in the past to provide this process and workflow functionality, jBPM is also still included in Alfresco but may be deprecated at some point in time.

Besides running the Activiti process engine in Alfresco, Activiti is built to run standalone or embedded in any other system. In this book we'll focus on running Activiti outside the Alfresco environment. Although, we'll discuss the integration opportunities between Activiti and Alfresco in detail in chapter 12.

In 2010 the Activiti project started off at a very fast pace and succeeded to do a monthly(!) release of the framework. In December 2010 the first stable and production-ready release (5.0) was made available. The Activiti developer community, including companies like SpringSource, FuseSource and Mulesoft, has since then been able to develop new functionality at a frequent basis. In this book we'll also explore this contributed functionality like the Spring integration (chapter 3) and the Mule and Apache Camel integration (chapter 9).

But first things first, what can you do with a process engine? Why should you use the Activiti framework. Let's discuss the core component named the Activiti Engine.

1.1.2 The very basics of the Activiti Engine

In section 1.2 we'll look at the full stack of components that the Activiti framework provides. But in the essence of a BPMN 2.0 process engine framework is the capability to run and manage processes and workflow tasks. The Activiti Engine implements the BPMN 2.0 specification and is able to deploy process definitions, start new process instances and run BPMN 2.0 tasks etc.

However at its core the Activiti Engine is a state machine. A BPMN 2.0 process definition consists of elements like events, tasks and gateways that are wired together via sequence flows (think of arrows). When such a process definition is deployed on the process engine and a new process instance is started, the BPMN 2.0 elements are executed one by one. This is very similar to a state machine, where there is an active state and based on conditions the state execution progresses to another state via transitions (think again of arrows). Let's look at an abstract figure of a state machine and how it's implemented in the Activiti Engine in figure 1.1.

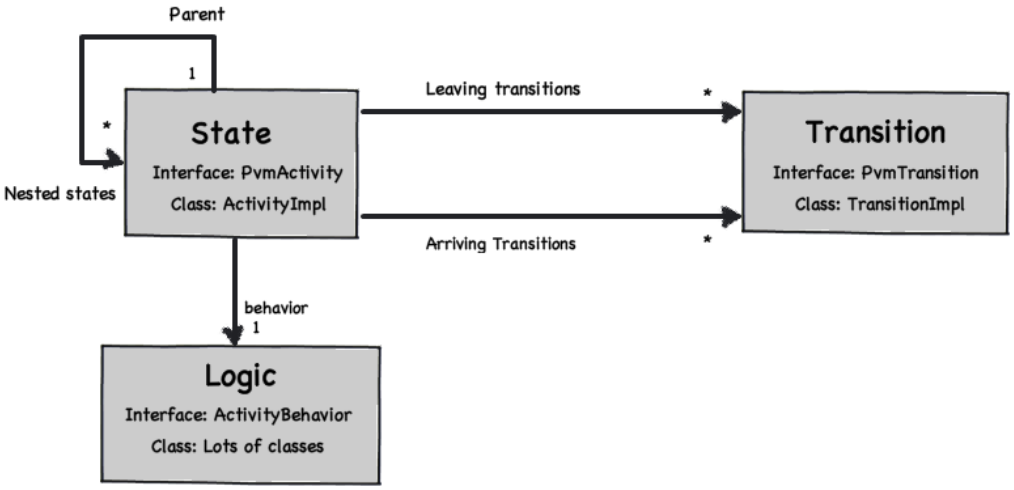

Figure 1.1 An abstract overview of a state machine and how it's implemented in the Activiti Engine. States have leaving and arriving transitions and can be nested. In addition they contain logic implemented with the ActivityBehavior interface.

In the Activiti Engine most BPMN 2.0 elements are implemented as a state. And they are connected with leaving and arriving transitions, which are called sequence flows in BPMN 2.0. Every state or corresponding BPMN 2.0 element can have a piece of logic attached to it that will be executed when the process instance enters the state. In figure 1.1. you can also look up the interface and implementing class that is used in the Activiti Engine. As you can see the logic interface `ActivityBehavior` is implemented by a lot of classes. That's because the actual logic of a BPMN 2.0 element is implemented there.

So when you'll see a complex BPMN 2.0 example in the rest of the book, remember that in essence it's a rather simple state machine. Now let's look at a couple other open source process engines that offer similar functionality as Activiti, and look what are the differences.

1.1.3 Knowing the competitors

When you're interested in an open source process engine like Activiti, it's always good to know a little bit more about the competing open source frameworks. Because of the origin of the main developers of Activiti lies in the JBoss BPM or jBPM framework, there's also some controversy surrounding this discussion. It's obvious that jBPM and Activiti share a lot of the same architectural principles, but there are also enough differences. We'll only discuss the two main open source competitors of Activiti, which are:

- JBoss BPM or jBPM. An open source process engine that first supported the custom jPDL process language, but since version 5.0 supports BPMN 2.0. The jBPM project has merged with the JBoss Drools project (an open source business rule management framework) and replaced Drools Flow as the rule flow language for the Drools

framework.

- Bonitasoft. An open source process engine with support of the BPMN 2.0 process language. The main differentiators of Bonitasoft are the large set of supported elements and the integrated development environment.

Let's discuss the similarities and differences between Activiti and the two competitors in a bit more detail.

ACTIVITI AND JBPM

Activiti and jBPM have a lot in common. They are both developer-oriented process engine frameworks built around the concept of a state machine (see section 1.1.2). And because jBPM 5 is also implementing the BPMN 2.0 specification a lot of similar functionality can be found. But there are a number of differences that are important to mention, see table 1.1.

Table 1.1 Main differences between Activiti and jBPM

Description	Activiti	jBPM
Community members	Activiti has a base team consisting of Alfresco employees. In addition, companies like SpringSource, FuseSource and MuleSoft provide resources on specific components. And of course there are individual open source developers committing to the Activiti project.	jBPM has a base team of JBoss employees. In addition there are individual committers.
Spring support	Activiti has native Spring support, which makes it very easy use Spring beans in your processes and use Spring for JPA and transaction management.	jBPM has no native Spring support, but you can use Spring with additional development effort.
Business rules support	Activiti provides a basic integration with the Drools rule engine to support the BPMN 2.0 business rule task.	jBPM and Drools are integrated on a project level and therefore there's native integration with Drools on various levels.
Additional tools	Activiti (see section 1.2) provides a modeler (Oryx) and designer (Eclipse) tool to model new process definitions. But the main differentiator is the Activiti Explorer, which provides an easy-to-use web interface to start new processes, work with tasks and forms and manage the running processes. In addition it provides ad-hoc	jBPM also provides a modeler based on the Oryx project and a Eclipse designer. With a web application you can start new process instances and work with tasks. The form support is limited.

	task support and collaboration functionality.	
Project	Activiti has a strong developer and user community with a solid release schedule of 2 months. Its main components are the Engine, Designer, Explorer and REST application.	jBPM has a strong developer and user community. The release schedule is not crystal clear and some releases have been postponed a couple of times. The Designer application is (at the moment of writing) still based on Drools Flow and the promised new Eclipse plug-in keeps getting postponed.

It's always difficult to compare two open source frameworks in an objective manner, certainly when this book is about Activiti. In this overview is by no means the only view on the differences, but it provides a number of points that you can use if you have to make a choice between these frameworks. Next up is the comparison between Activiti and Bonitasoft.

ACTIVITI AND BONITASOFT

BonitaSoft is the company behind the open source BPM product Bonita Open Solution. There are a number of clear differences between Activiti and Bonitasoft:

- Activiti is developer-focused and provides an easy-to-use Java API to communicate with the Activiti Engine. Bonitasoft provides a tool-based solution where you can click-and-drag your process definition and forms.
- With Activiti you are in control over every bit of the code you write. With Bonitasoft the code is often generated from the developer tool.
- Bonitasoft provides a large set of connectivity options to a wide range of third-party products. This means it's easy to configure a task in the developer tool to connect to SAP or to query a particular database table. With Activiti the connectivity options are also very broad due to the integration with Mule and Camel, but more developer-focused.

While both frameworks focus on supporting the BPMN 2.0 specification and offering a process engine, they took a very different implementation angle. Bonitasoft provides a development tool where you can draw your processes, configure and deploy them without needing to write one line of code. This means that you are not in control over the process solution you are developing. Activiti provides an easy-to-use Java API that will need some coding, but in the end you can easily embed it into an application or run it on every platform you would like.

So Activiti is obviously not the only open source process engine capable of running BPMN 2.0 process models. But it's definitely a flexible and powerful option that we'll discuss in lots of detail in this book. First let's look at the different components of the Activiti framework.

1.2 The Activiti tool stack

The core component of the Activiti framework is of course the process engine. The process engine provides the core capabilities to for example execute BPMN 2.0 processes and create new workflow tasks. But the Activiti project contains a couple of additional tools on top of the Activiti Engine. Figure 1.2 shows an overview of the full Activiti tool stack.

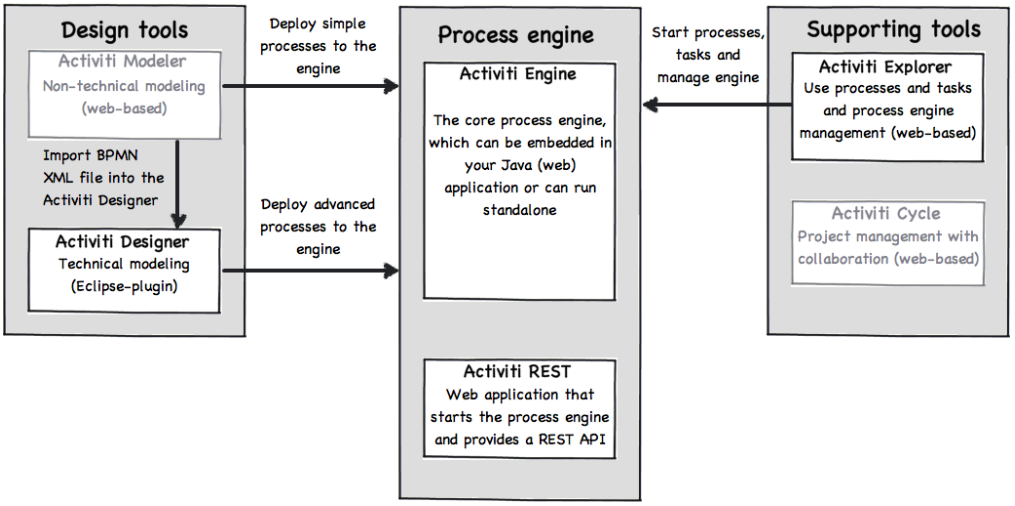

Figure 1.2 An overview of the Activiti tool stack, with in the center the Activiti process engine and on the right and left side the accompanying modeling, design and management tools. The grayed-out components are add-ons to the core Activiti framework.

Let's quickly walk through the different components listed in figure 1.1. 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 plug-in, 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. You can also run unit tests, add Java logic and create deployment artifacts with the Activiti Designer.

In addition to the design tools, Activiti also provides a number of supporting tools. With Activiti Explorer you can get an overview of deployed processes and even dive into the database tables underneath the Activiti process engine. But besides the process management capabilities you can also interact with the deployed business processes. You can for example get a list of tasks assigned to you. And you can start a new process instance and look at the status of that newly created process instance in a graphical diagram.

The last supporting tool is Activiti cycle, which provides a project management and collaboration tool. With Activiti Cycle you can keep an overview of the status of your various business process projects and get a good overview of all the different artifacts involved.

Lastly, there's the Activiti REST component, which provides a web application that starts the Activiti process engine when the web application is started. In addition it offers a REST API to be able to communicate remotely with the Activiti Engine. Let's summarize the different components in table 1.2.

Table 1.2 An overview of the different components of the Activiti tool stack

Component name	Short description
Activiti Engine	The core component of the Activiti tool stack that provides the process engine capabilities like the execution of BPMN 2.0 business processes and creating workflow tasks.
Activiti Modeler	A web-based modeling environment for creating BPMN 2.0 compliant business process diagrams. This component is donated by Signavio, who also provide a commercial licensed modeling tool, named the Signavio Process Editor.
Activiti Designer	An Eclipse plugin that can be used to design BPMN 2.0 compliant business processes with the addition of Activiti extensions such as a Java service task and execution listeners. You can also unit test processes, import BPMN 2.0 processes and create deployment artifacts.
Activiti Explorer	A web application that can be used for a wide range of functionality that involves interacting with the Activiti Engine. You can for example start new process instances and get a list of tasks assigned to you. In addition you can perform simple process management tasks like deploying new processes and retrieving the process instance status.
Activiti Cycle	A web-based project management and collaboration tool that provides an easy way to administer all your business process projects. Cycle provides capabilities to get a clear overview of all the artifacts involved in the business process project.
Activiti REST	A web application that provides a REST interface on top of the Activiti Engine. In the default installation (see section 1.1.3) the Activiti REST application is the entry point to the Activiti Engine.

Now that we know the different components of Activiti let's get the framework installed on your development machine.

1.3 Installing the Activiti framework

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)
/usr/local/activiti (Linux or Mac OS)
```

This is not the beginning of a long and complex installation procedure. With Activiti, there's a setup directory that contains an Ant build file that installs the Activiti framework. The directory structure of the distribution is shown in figure 1.3.

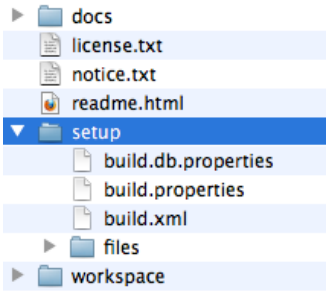

Figure 1.3 The directory structure of the Activiti distribution with the setup directory and the Ant build.xml file as the main parts for the installation procedure.

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 a current version (>= 1.8.x) of 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 (more info can be found on http://ant.apache.org/manual/proxy.html).

When you open a terminal or command prompt and go to the setup directory shown in figure 1.3, you only have to run the `ant` command (or `ant demo.start`). This will kick off the Activiti installation process, which will look for a `build.xml` file in the setup directory. The installation 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 REST and Activiti Explorer WARs are copied to the webapps directory of Tomcat.
6. Tomcat is started with the Activiti toolstack.
7. A web browser is started with the Activiti Explorer application.

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 two web applications being deployed to a Tomcat server and a ready-to-use H2 database with example processes, groups and users already loaded. Figure 1.4 shows the installation result in a schematic overview.

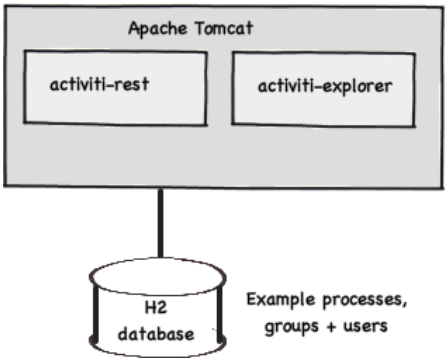

Figure 1.4 An overview of the installation result of the Activiti tool stack, including a running Tomcat server and H2 database with the two Activiti web applications already deployed.

Notice that we didn't yet install the Activiti Modeler, Designer and Cycle applications. These components are not part of the installation script and have to be installed separately. We'll discuss how to do this in chapter 3. 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 URI of the Activiti Explorer and REST web applications available for you after the installation of Activiti

Application name	URI	Short description
Activiti Explorer	http://localhost:8080/activiti-explorer	The Explorer application can be used to work with the deployed processes. This is a good starting point to try the example processes.
Activiti REST	http://localhost:8080/activiti-rest/service	The REST application can be used to have remote access to the Activiti engine via a REST interface. For all available REST services you can look in the Activiti user guide that can be found on the Activiti website.

By trying the Activiti Explorer application, you can verify if the installation was successful. After logging in and clicking on the Process tab, you should get a list of the examples processes that are deployed on the Activiti Engine. Working with demo processes is fun, but it's even better to try out your own developed business process.

1.4 Implementing your first process in Activiti

Let's try and implement a simplified version of a book order 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'll keep things simple and if you don't understand every construct already, don't be worried we'll discuss the BPMN 2.0 elements in more detail in chapter 2. In code listing 1.1 a starter for the BPMN 2.0 XML definition of the book order process is shown with only a start event, an end event and a sequence flow to connect the two.

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"                                      #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 the start and end event

A BPMN 2.0 XML 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 with the book's website as its attribute value. Activiti also provides a namespace, which enables us to use Activiti extensions to the BPMN 2.0 specification as we will see in chapter 4. We can now run this very simple process to test if we have correctly defined the process definition 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 4. 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 4 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 the process engine is started with an in-memory H2 database. There are different ways to start-up an Activiti engine and we'll explain the options in detail in chapter 4. 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 of the book order process, so we can use the Activiti Explorer to claim and finish a user task for our process.

1.4.1 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 been given as input to the book order process when started. Then a user task will be used to handle the book ordering manually.

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 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 chapter 3 we'll show the Activiti designer, which abstract you from implementing XML file and provides a drag and drop type of process development.

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();

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

    Map<String, Object> variableMap =
        new HashMap<String, Object>();
    variableMap.put("isbn", "123456");
    identityService.setAuthenticatedUserId("kermit");                    #A
    ProcessInstance processInstance =                                    #2
        runtimeService.startProcessInstanceByKey(                        #2
            "bookorder", variableMap);                                   #2
    assertNotNull(processInstance.getId());
    List<Task> taskList = taskService.createTaskQuery()                  #B
        .taskCandidateUser("kermit")                                     #B
        .list()                                                          #B
    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 Set authenticated user to kermit
#B 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 the H2 database should be 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 to:

```
validating order for isbn 123456
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 `Queued` you should see one task in the group `Sales`. When you click on this `Sales` group you should see a screen with one user task with the name of `Work on order` like the screenshot shown in figure 1.5.

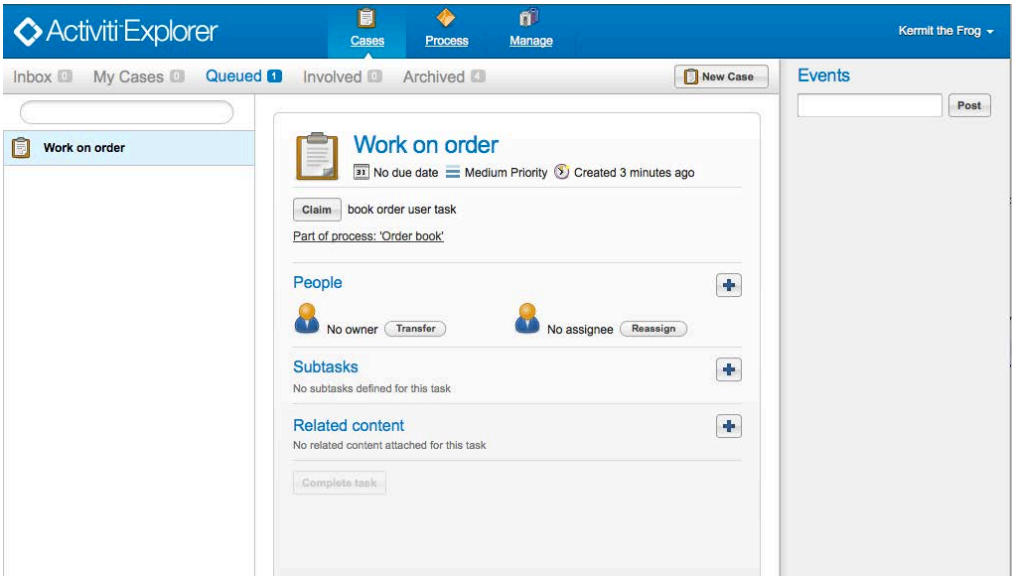

Figure 1.5 A screenshot of the Activiti Explorer showing the user task of the book order process.

For sake of completeness you can claim the user task and see that it becomes available in the `Inbox` page. There you can complete the task, which triggers the process instance to complete to the end state. But before you do that you can click on the process link `Part of process: 'Order book'` to see details about the running process instance as shown in figure 1.6.

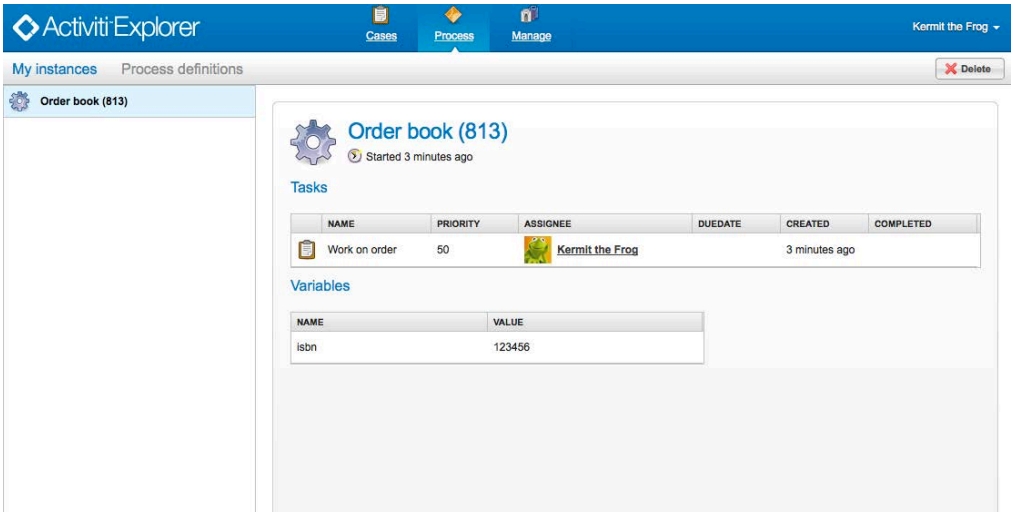

Figure 1.6 A screenshot of the Activiti Explorer application showing the details of a running process instance with open user tasks and its variables.

In the process instance overview you can get the details about the user tasks that are not yet completed and the process variables of the running instance. The Activiti Explorer contains a lot more functionality, which we'll discuss throughout the book starting in chapter 3.

This completes our first journey in the Activiti framework. 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. But first we'll look in more detail at BPMN 2.0.

1.5 Summary

In this chapter we started with a gentle introduction into Activiti, its history and its competitors. And 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. We were also 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 to run our processes. But how can we model and implement theses processes? The BPMN 2.0 specification is the foundation for the Activiti Engine and to get ready for the examples in the rest of the book we'll discuss the details of BPMN 2.0 in the next chapter.