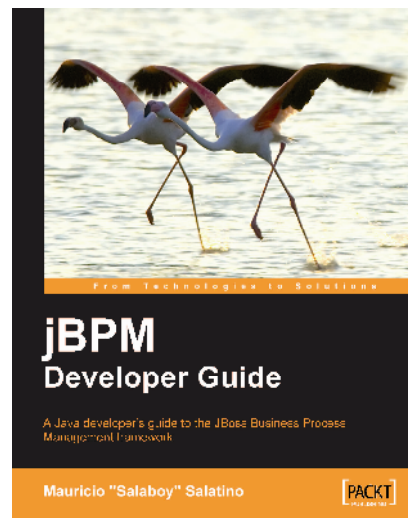




jBPM Developer Guide

Mauricio "Salaboy" Salatino



Chapter No. 5

"Getting Your Hands Dirty with jPDL"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.5 "Getting Your Hands Dirty with jPDL"

A synopsis of the book's content

Information on where to buy this book

About the Author

Mauricio Salatino (a.k.a. *Salaboy*) has been a part of the Java and open source software world for more than six years now. He's worked with several technologies (such as PHP, JSP, Java SE, Java ME, and Java EE) during these years and is now focused on JBoss frameworks. He got involved with the JBoss Drools project about a year and a half ago as a contributor, gaining a lot of experience with the open source community and with multiple technologies such as JBoss jBPM, JBoss Drools, Apache RIO, Apache Mina, and JBoss Application Server.

During 2008 he dictated the official jBPM courses for Red Hat Argentina several times, and he was involved in several JBoss jBPM and JBoss Drools implementations in Argentina. He was also part of the Research and Development team of one of the biggest healthcare providers in Argentina, where he trained people in the BPM and Business Rules field.

Mauricio is currently involved in different open source projects that are being created by the company he co-founded, called **Plug Tree** (www.plugtree.com), which will be released in 2010. Plug Tree is an open source based company that creates open source projects and provides consultancy, training, and support on different open source projects.

For More Information:

www.packtpub.com/jboss-business-process-management-jbpm-developer-guide/book

Mauricio is an Argentinian/Italian citizen based in Argentina. In his free time he gives talks for the JBoss User Group Argentina (www.jbug.com.ar), that he co-founded with a group of local friends. He also runs his personal blog about JBoss, jBPM, and JBoss Drools, that was originally targeted to Hispanic audiences but is now aimed at an international audience and receives more than five hundred questions per year.

I would like to thank my family for always being there to support my decisions and adventures, my new and old friends who have helped me during this process, all the Packt Publishing staff who have guided me during these months of hard work; and last but not least, the open source community guys who are always creating new, interesting, and exciting projects.

For More Information:

www.packtpub.com/jboss-business-process-management-jbpm-developer-guide/book

jBPM Developer Guide

You are reading this because you are starting to get interested in the open source world. This book is especially for Java architects and developers with a free mind, who want to learn about an open source project. The fact that jBPM is an open source project gives us a lot of advantages, but it also comes with a big responsibility. We will talk about both—all the features that this great framework offers us and also all the characteristics that it has, being an open source project.

If you are not a Java developer you might find this book a bit harder, but it will give you all the points to understand how the open source community works.

I would like to take you through my own history, about how I discovered jBPM so that you can identify your situation right now with mine. Take this preface as an introduction to a new field—integration. It doesn't matter what your programming skills, experiences, and likes (user interfaces, code logic, low level code, simple applications, enterprise applications, so on) are, if you are a courageous developer you will like to tackle down all types of situations at least once.

With the myriad of web technologies these days, it's not a surprise that the new developers' generation starts building web applications. I have been working in the software development field for approximately six years now. I used to spend most of my time creating, developing, and designing web-based applications. I have also learned more "low level" languages such as C and C++, but in the beginning I could not make money with that. So, PHP and JSP were my first options. Although it was challenging I realized that I could not create bigger projects with my knowledge about JSP and PHP. The main reason for this, in my opinion, is that bigger projects become unmanageable when you start having web pages that contain all your application logic. At that point I recognized that I needed to learn new paradigms in order to create bigger and scalable applications. That is when I switched to Java Enterprise Edition (version 1.4), which provides us with a componentized way to build applications in order to be able to scale and run our applications on clusters and with all these features about high availability and fault tolerance. But I was not interested in configuring and making environmental settings, I just wanted to develop applications. An important point in my career was when I started getting bored as I had to spend hours with HTML and CSS frontend details that I did not care about. So, I looked for other frameworks like JSF, which provides a componentized way to build UIs and newer frameworks like JBoss Seam/web beans (JSR-299) that have intimate relationships with the EJB3 specification, but once again I had to check for HTML and CSS details for end users. I think that the fact that I used to get bored with HTML and CSS is one of the biggest reasons why I got interested in integration frameworks. When I use the word *integration*, I mean making heterogeneous

For More Information:

www.packtpub.com/jboss-business-process-management-jbpm-developer-guide/book

applications work together. Most of the time when you are doing integrations; the user interfaces are already done and you only need to deal with backends and communication stuff. That was my first impression, but then I discovered a new world behind these frameworks. At this point two things got my attention: the open source community and the theoretical background of the framework. These two things changed my way of thinking and the way I used to adapt to a new open source framework. This book reflects exactly that. First we'll see how we can adapt all the theoretical aspects included in the framework and then move on to how we can see all these concepts in the framework's code. This is extremely important, because we will understand how the framework is built, the project direction, and more importantly how we can contribute to the project.

I have been involved with the open source community for two years now, working with a lot of open source frameworks and standards that evolve every day. When I got interested in jBPM I discovered all the community work that is being done to evolve this framework. I wanted to be part of this evolution and part of this great community that uses and creates open source frameworks. That is one of the main reasons why I created a blog (<http://salaboy.wordpress.com>) and started writing about jBPM, I also cofounded the JBoss User Group in Argentina (<http://www.jbug.com.ar>) and now Plug Tree (<http://www.plugtree.com>), an open source-based company. With these three ventures I encourage developers to take interest in new frameworks, new technologies and the most important thing, the community.

What This Book Covers

Chapter 1, *Why Developers Need BPM?* introduces you to the main theoretical concepts about BPM. These concepts will lead you through the rest of the book. You will get an idea of how all the concepts are implemented inside the jBPM framework to understand how it behaves in the implementations of the projects.

Chapter 2, *jBPM for Developers*, introduces the jBPM framework in a developer-oriented style. It discusses the project's main components and gets you started with the code distribution.

Chapter 3, *Setting Up Our Tools*, teaches you to set up all the tools that you will be using during this book. Basic tools such as Java Development Kit and the Eclipse IDE will be discussed. It will also provide you with a brief introduction to Maven2 here to help you understand how to build your projects and the framework itself. At the end of this chapter you will see how to create simple applications that use the jBPM framework.

For More Information:

www.packtpub.com/jboss-business-process-management-jbpm-developer-guide/book

Chapter 4, *jPDL Language*, introduces the formal language to describe our business processes. It gives you a deep insight in to how this language is structured and how the framework internally behaves when one of these formal definitions is used.

Chapter 5, *Getting Your Hands Dirty with jPDL*, gets you started with working on real-life projects. You will be able to create your first application that uses jBPM and define simple processes, using the basic words in the jPDL language.

Chapter 6, *Persistence*, sheds light on the persistence service inside the jBPM framework, which is one of the most important services to understand in order to create real-life implementations using this framework. The persistence services are used to support the execution of long-running processes that represent 95% of the situations.

Chapter 7, *Human Tasks*, describes the human interactions inside business processes, which are very important because humans have specific requirements to interact with systems and you need to understand how all this works inside the framework.

Chapter 8, *Persistence and Human Tasks in the Real World*, mainly covers configurations to be done for real environments where you have long-running processes that contain human interactions. If you think about it, almost all business processes will have these requirements, so this is extremely important.

Chapter 9, *Handling Information*, helps you to understand how to handle all the process information needed by human interactions inside the framework, as the human interactions' information is vital to get the activities inside our business processes completed.

Chapter 10, *Going Deeply into the Advanced Features of jPDL*, analyzes the advanced features of the jPDL language. This will help you improve your flexibility to model and design business processes, covering more complex scenarios that require a more advanced mechanism to reflect how the activities are done in real life.

Chapter 11, *Advanced Topics in Practice*, provides us with practical examples on the topics discussed in the previous chapters. This will help you to understand how all the advanced features can be used in real projects.

Chapter 12, *Going Enterprise*, introduces the main features provided by jBPM to run in enterprise environments. This is very important when your projects are planned for a large number of concurrent users.

For More Information:

www.packtpub.com/jboss-business-process-management-jbpm-developer-guide/book

5

Getting Your Hands Dirty with jPDL

In this chapter, we will practice all the conceptual and theoretical points that we have already discussed in the previous chapters. Here we will cover the main points that you need in order to start working with the jBPM framework.

This chapter will tackle, in a tutorial fashion, the first steps that you need to know in order to start using the framework with the right foot. We will follow a real example and transform the real situation into requirements for a real jBPM implementation.

This example will introduce us to all the basic jPDL nodes used in common situations for modeling real world scenarios. That's why this chapter will cover the following topics:

- Introduction to the recruiting example
- Analyzing the example requirements
- Modeling a formal description
- Adding technical details to our formal description
- Running our processes

We have already seen all the basic nodes that the jPDL language provides. Now it's time to see all of them in action. It is very important for the newcomers to see how the concepts discussed in previous chapters are translated into running processes using the jBPM framework.

The idea of this short chapter is to show you a real process implementation. We will try to cover every technical aspect involved in development in order to clarify not only your doubts about modeling, but also about the framework behavior.

For More Information:

www.packtpub.com/jboss-business-process-management-jbpm-developer-guide/book

How is this example structured?

In this chapter, we will see a real case where a company has some requirements to improve an already existing, but not automated process.

The current process is being handled without a software solution, practically we need to see how the process works everyday to find out the requirements for our implementation. The textual/oral description of the process will be our first input, and we will use it to discover and formalize our business process definition.

Once we have a clear view about the situation that we are modeling, we will draw the process using GPD, and analyze the most important points of the modeling phase. Once we have a valid jPDL process artifact, we will need to analyze what steps are required for the process to be able to run in an execution environment. So, we will add all the technical details in order to allow our process to run.

At last, we will see how the process behaves at runtime, how we can improve the described process, how we can adapt the current process to future changes, and so on.

Key points that you need to remember

In these kind of examples, you need to be focused on the translation that occurs from the business domain to the technical domain. You need to carefully analyze how the business requirements are transformed to a formal model description that can be optimized.

Another key point here, is how this formal description of our business scenario needs to be configured (by adding technical details) in order to run and guide the organization throughout its processes.

I also want you to focus on the semantics of each node used to model our process. If you don't know the exact meaning of the provided nodes, you will probably end up describing your scenario with the wrong words.

You also need to be able to distinguish between a business analyst model, which doesn't know about the jPDL language semantics and a formal jPDL process definition. At the same time, you have to be able to do the translations needed between these two worlds. If you have business analysts trained in jPDL, you will not have to do these kind of translations and your life will be easier. Understanding the nodes' semantics will help you to teach the business analysts the correct meaning of jPDL processes.

Analyzing business requirements

Here we will describe the requirements that need to be covered by the recruiting team inside an IT company. These requirements will be the first input to be analyzed in order to discover the business process behind them.

These requirements are expressed in a natural language, just plain English. We will get these requirements by talking to our clients—in this case, we will talk to the manager of an IT company called MyIT Inc. in order to find out what is going on in the recruiting process of the company.

In most cases, this will be a business analyst's job, but you need to be aware of the different situations that the business scenario can present as a developer. This is very important, because if you don't understand how the real situation is sub-divided into different behavioral patterns, you will not be able to find the best way to model it.

You will also start to see how iterative this approach is. This means that you will first view a big picture about what is going on in the company, and then in order to formalize this business knowledge, you will start adding details to represent the real situation in an accurate way.

Business requirements

In this section, we will see a transcription about our talk with the MyIT Inc. manager. However, we first need to know the company's background and, specifically, how it is currently working. Just a few details to understand the context of our talk with the company manager would be sufficient.

The recruiting department of the MyIT Inc. is currently managed without any information system. They just use some simple forms that the candidates will have to fill in at different stages during the interviews. They don't have the recruiting process formalized in any way, just an abstract description in their heads about how and what tasks they need to complete in order to hire a new employee when needed.

In this case, the MyIT Inc. manager tells us the following functional requirements about the recruiting process that is currently used in the company:

We have a lot of demanding projects, that's why we need to hire new employees on a regular basis. We already have a common way to handle these requests detected by project leaders who need to incorporate new members into their teams.

When a project leader notices that he needs a new team member, he/she will generate a request to the human resources department of the company. In this request, he/she will specify the main characteristics needed by the new team member and the job position description.

When someone in the human resources team sees the request, they will start looking for candidates to fulfill the request. This team has two ways of looking for new candidates:

- By publishing the job position request in IT magazines
- By searching the resume database that is available to the company

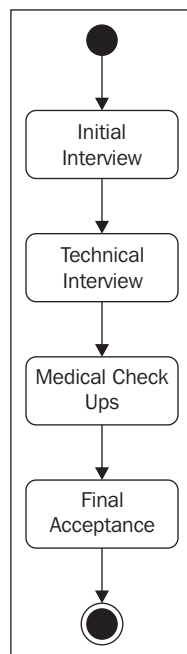
When a possible candidate is found through these methods, a set of interviews will begin. The interviews are divided into four stages that the candidate needs to go through in order to be hired.

These stages will contain the following activities that need to be performed in the prescribed order:

- **Initial interview:** The human resources team coordinates an initial interview with each possible candidate found. In this interview, a basic questionnaire about the candidate's previous jobs and some personal data is collected.
- **Technical interview:** During the technical interview stage, each candidate is evaluated only with the technical aspects required for this particular project. That is why a project member will conduct this interview.
- **Medical checkups:** Some physical and psychological examinations need to be done in order to know that the candidate is healthy and capable to do the required job. This stage will include multiple checkups which the company needs to determine if the candidate is apt for the required task.
- **Final acceptance:** In this last phase the candidate will meet the project manager. The project manager is in charge of the final resolution. He will decide if the candidate is the correct one for that job position. If the outcome of this interview is successful, the candidate is hired and all the information needed for that candidate to start working is created.

If a candidate reaches the last phase and is successfully accepted, we need to inform the recruiting team that all the other candidate's interviews need to be aborted, because the job position is already fulfilled.

At this point, we need to analyze and evaluate the manager's requirements and find a graphical way to express these stages in order to hire a new employee. Our first approach needs to be simple and we need to validate it with the MyIT Inc. manager. Let's see the first draft of our process:



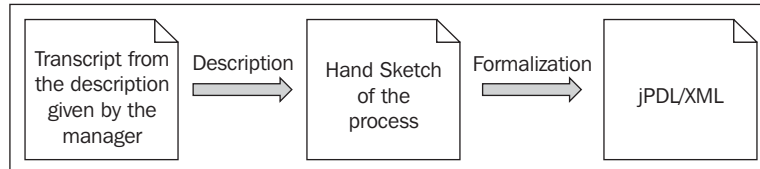
With this image, we were able to describe the recruiting process. This is our first approach that obviously can be validated with the MyIT Inc. manager. This is our first draft that tells us how our process will appear and it's the first step in order to define which activities will be included in our model and which will not. In real implementations, these graphs can be made with Microsoft Visio, DIA (Open Source project), or just by hand. The main idea of the first approach is to first have a description that can be validated and understood by every MyIT Inc. employee.

This image is only a translation of the requirements that we hear from the manager using common sense and trying to represent how the situation looks in real life. In this case, we can say that the manager of the MyIT Inc. can be considered as the stakeholder and the **Subject Matter Expert (SME)**, who know how things happen inside the company.

Once the graph is validated and understood by the stakeholder, we can use our formal language jPDL to create a formal model about this discovered process.

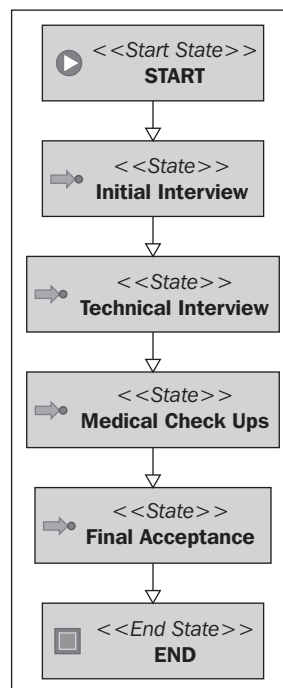
The idea at this point, is to create a jPDL process definition and discard the old graph. From now on we will continue with the jPDL graphic representation of the process. Here you can explain to the manager that all the new changes that affect your process will go directly to the jPDL defined process.

Until now our artifact has suffered the following transformations:



The final artifact (the jPDL process definition) will let us begin the implementation of all the technical details needed by the process in order to run in an execution environment.

So, let's analyze how the jPDL representation will look for this first approach in the following figure:




At this point we don't add any technical details, we just draw the process. One key point to bear in mind in this phase is that we need to understand which node we will use to represent each activity in our process definition.

Remember that each node provided by jPDL has its own semantics and meanings. You also need to remember that this graph needs to be understood by the manager, so you will use it in the activity name business language. For this first approach we use state nodes to represent that each activity will happen outside the process execution. In other words, we need to inform the process when each activity ends. This will mean that the next activity in the chain will be executed. From the process perspective, it only needs to wait until the human beings in the company do their tasks.

Analyzing the proposed formal definition

Now that we have our first iteration that defines some of the important aspects described by the MyIT Inc. manager, some questions start to arise with relation to our first sketch, if it is complete enough, or not. We need to be sure that we represent the whole situation and it defines the activities that the candidates and all the people involved in the process need, to fulfill the job position with a new employee. We will use the following set of questions and their answers as new requirements to start the second iteration of improvement.



The idea is that each iteration makes our process one step closer to reflecting the real situation in the company. For reaching that goal, we also need to be focused on the people who complete each activity inside our process. They will know whether the process is complete or not. They know all the alternative activities that could happen during the process execution.

If we see the proposed jPDL definition we can ask the following questions to add more details to our definition.

- How about the first part of the description? Where is it represented? The proposed jPDL process just represents the activities of the interviews, but it doesn't represent the request and the creation of the new job position.
- What happens if the candidate goes to the first interview and he/she doesn't fulfill the requirements for that job position?
- How many medical checkups are done for each candidate?
- What happens if we fulfill the job position? What happens with the rest of the candidates?

These questions will be answered by the MyIT Inc. manager and the people who are involved in the process activities (business users). The information provided by the following answers will represent our second input, which will be transformed into functional requirements. These requirements need to be reflected in our formal description once again. Let's take a look at the answers:

How about the first part of the description? Where is it represented? The proposed jPDL process just represents the activities of the interviews, but it doesn't represent the request and the creation of the new job position.

Yes, we will need to add that part too. It's very important for us to have all the processes represented from the beginning to the end. Also, you need to understand that the interviews are undertaken for each candidate found, and the request to fulfill a new job position is created just once. So the relationship between the interviews and the new employee request is 1 to N, because for one request, we can interview N candidates until the job position is fulfilled.



What happens if the candidate goes to the first interview and he/she doesn't fulfill the requirements for that job position?

The candidate that doesn't pass an interview is automatically discarded. There is no need to continue with the following activities if one of the interviews is not completed successfully.

How many medical checkups are done for each candidate?

All the candidates need to pass three examinations. The first one will check the physical status of the candidate, the second will check the psychological aspects, and the third one will be a detailed heart examination.

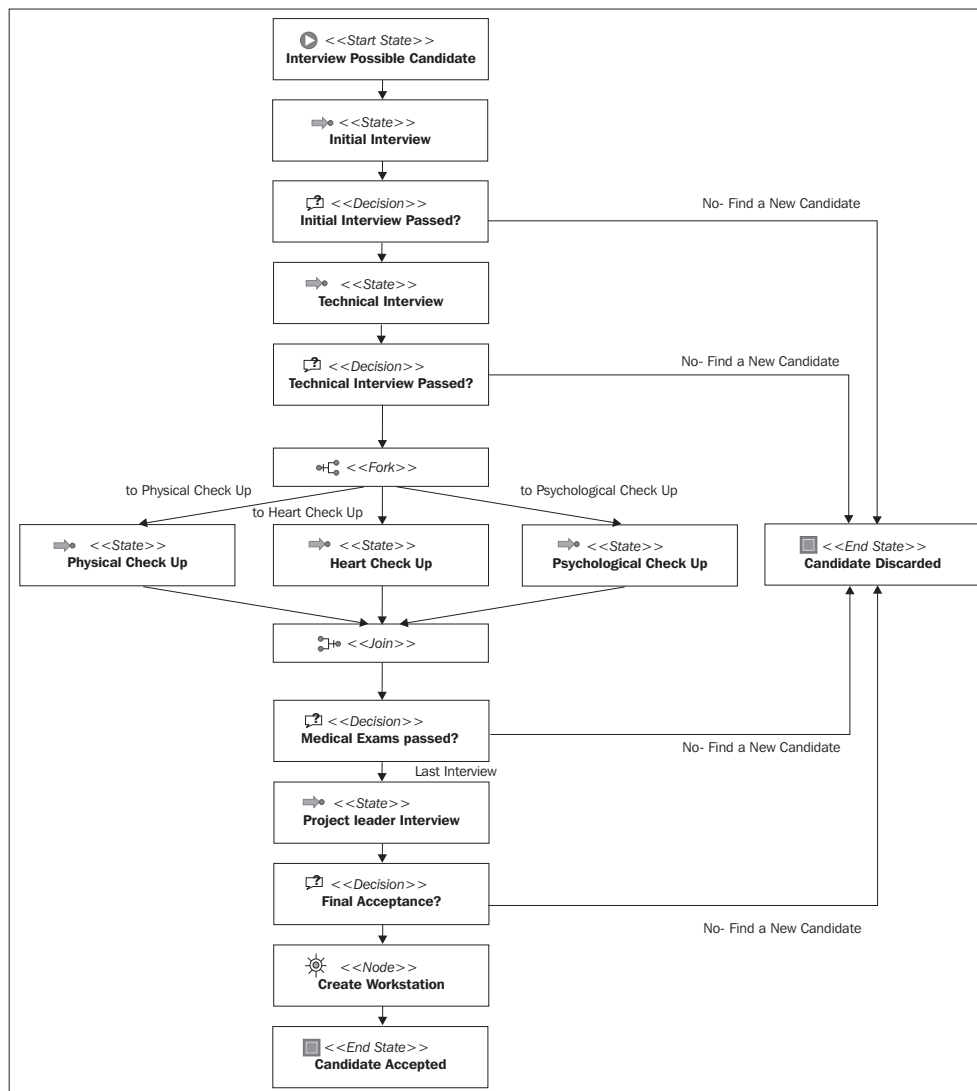
What happens if a candidate fulfills the job position? What happens with the rest of the candidates?

If one of the candidates is accepted, all the remaining interviews for all the other candidates need to be aborted.

Refactoring our previously defined process

Now with the answers to our questions, we can add some extra nodes to represent the new information provided. Take a look at the following image of the process, you will find new nodes added, and in this section, we will discuss the reason for each of them.

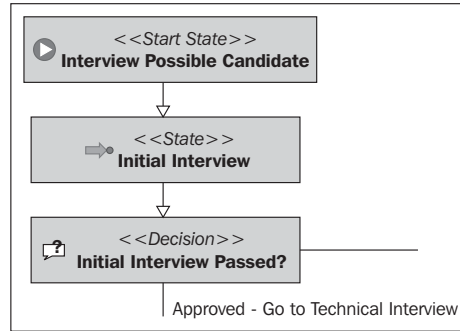
The new proposed process is much more complex than the first, but the main idea is still intact. You need to be able to understand the process flow without problems. It's more complex just because it represents the real situation more closely.



For More Information:

www.packtpub.com/jboss-business-process-management-jbpm-developer-guide/book

In this section, we will review all the nodes added at each stage of the process. With this, you will have a clear example to see how a real situation is translated to a specific node type in our process definition and how we will iteratively add information when we find more and more details about the situation.



If you take a look at the **CandidateInterviews** process (`/RecruitingProcess/src/main/resources/jpdl/CandidateInterviews/processdefinition.xml`), you will see that the first node (**Start State** node) doesn't have the default name `Start/start-state1`. Here, I have chosen a more business-friendly name, *Interview Possible Candidate*. This name looks too long, but it says precisely what we are going to do with the process. Here a possible candidate was found and the process will interview him/her in order to decide if this candidate is the correct one for the job.

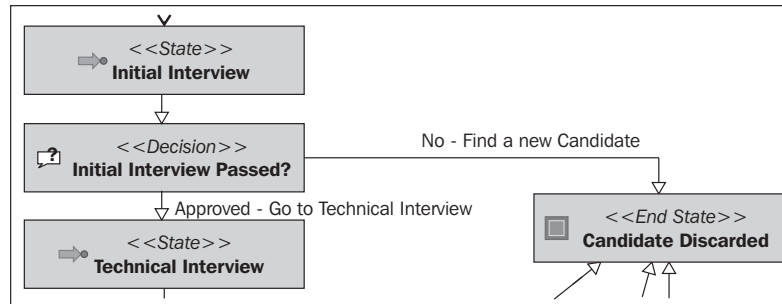


Using the business terminology will help us to have a more descriptive process graph that can be easily validated by the stakeholders.

The second node called **Initial Interview** will represent the first interview for each selected candidate. This means that someone in the recruiting team will schedule a face-to-face meeting with the candidate to have the first interview. If you take a close look at the process definition graph or the jPDL process definition XML, you will find that for this activity, I have chosen to use a **State node**. I chose this type of node, because the activity of having an interview with the candidate is an external activity that needs to be done by a person and not by the process. The process execution must wait until this activity is completed by the candidate and by the recruiting team. In the following (more advanced) chapters, we will see how to use a more advanced node to represent these human activity situations. For now, we will use state nodes to represent all the human activities in our processes.

Once the **Initial Interview** is completed, an automatic decision node will evaluate the outcome of the interview to decide if the candidate must be discarded, or if he/she should continue to the next stage of the process.

This will look like:




Just for you to know, this is not the only way to model these kinds of situations, feel free to try other combinations of nodes to represent the same behavior.

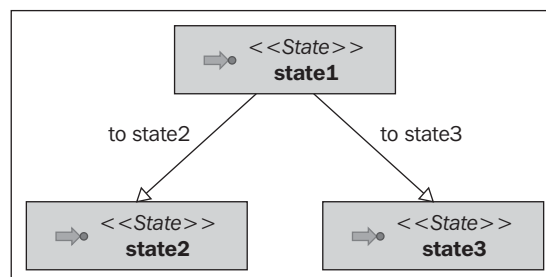
The decision node is used to decide for which transition the process will continue its execution. This decision node can define N (in this case, only two) leaving transitions, but at runtime, just one will be chosen to continue.

Remember that the **Decision** node takes a transition based on the following two evaluation methods:

- Using an EL expression
- Using a delegated class, implementing the method interface `DecisionHandler`

No matter which method we choose, the information that is used to make a decision needs to be set before the node was reached by the process execution. In this situation, the information used in the evaluation method of the decision node will be set in the state node called **Initial Interview** as the interview outcome.

 Another way you can use to model this situation is by defining multiple leaving transitions from the state node.





This approach writes the following logic inside an action of the state node:

1. The logic used to determine which transition to take is based on the interview outcome.
2. Explicitly signals one of the N leaving transitions defined based on the logic outcome.

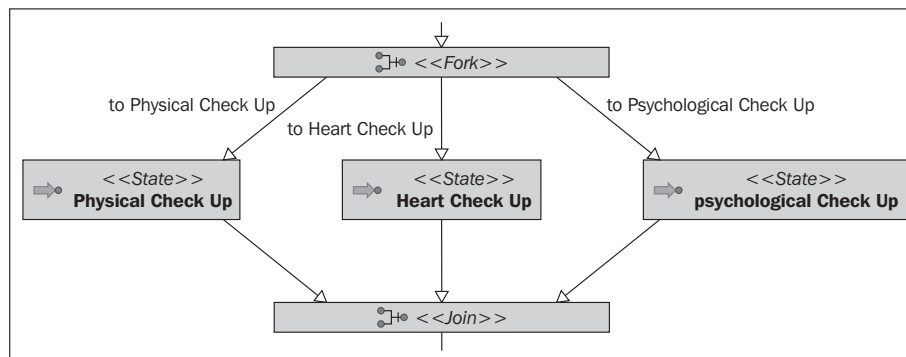
This approach tends to be more difficult to maintain than a detached decision node that handles all that logic. Basically, it is up to you to decide how to model these kinds of situations.

The pattern of using a state node and then a decision node, to decide if the previous activity is completed, with the desired outcome, is applied throughout all the process stages in order to decide if the candidate can continue or not, based on each activity's outcome.

The next stage described in the process definition looks exactly the same as the first one. The **Technical Interview** looks exactly the same as the **Initial Interview** stage. It also includes a decision node to evaluate the outcome of this specific interview.

If the candidate passes/approves the first two interviews, some medical examinations need to be taken in the third stage.

As these check ups have to be done in different buildings across the city, taking advantage of the fact that all of them are independent from each other, a **fork** node is used to represent this temporal independence. Take a look at the following image:



Here we need to understand that the **Fork** and **Join** nodes are used to define behavior, not to represent a specific activity by itself. In this situation, the candidate has the possibility to choose which exam to take first. The only restriction that the candidate has is that he/she needs to complete all the activities to continue to the next stage. It is the responsibility of the **Join** node to wait for all the activities between the **Fork** and **Join** nodes to complete before it can continue with the execution.

This section of the modeled process will behave and represent the following situations:

- When the process execution arrives at the fork node. (Note that the fork node doesn't represent any one-to-one relationship with any real-life activity. It is just used to represent the concurrent execution of the activities.)
- It will trigger three activities, in this case, represented by state nodes. This is because the checkups will be done by an external actor in the process. In other words, each of these activities will represent a wait state situation that will end when each doctor finishes each candidate's checkup and notifies the outcome of the process.
- In this case, when the three activities end, the process goes through the join node and propagates the execution to the **Decision** node to evaluate the outcome of the three medical checkups. If the candidate doesn't have three successful outcomes, he/she will automatically be discarded.



We use the fork node because the situation behaviors can be modeled as concurrent paths of execution. A detailed analysis of the fork node will take place in the following chapters. But it's important for you to play a little bit with it to start knowing this node type. Try to understand what we are doing with it here.

Describing how the job position is requested

In the previous section, we find all the answers to our questions; however, a few remain unanswered:

- How is the first part of the process represented? How can we track when the new job position is discovered, the request for that job position is created, and when this job position is fulfilled?
- Why can't we add more activities to the current defined process? What happens when we add the *create request*, *find a candidate*, and *job position fulfilled* activities inside the interview process?

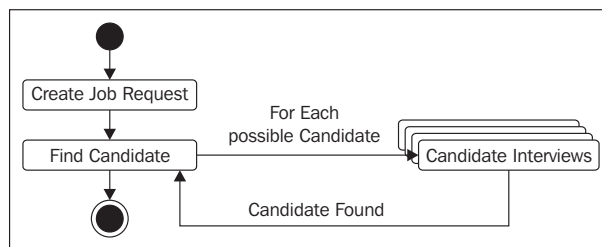
The answers for these questions are simple. We cannot add these proposed nodes to the same process definition, because the interview process needs to be carried out (needs to be instantiated) once for each candidate that the recruiting team finds. Basically, we need to decouple all these activities into two processes. As the MyIT Inc. manager said, the relationship between these activities is that a job request will be associated with the N-interviews' process.

The other important thing to understand here, is that both the processes can be decoupled without using a parent/child relationship. In this case, we need to create a new interview's process instance when a new candidate is found. In other words, we don't know how many interviews' process instances are created when the request is created. Therefore, we need to be able to make these creations dynamically.

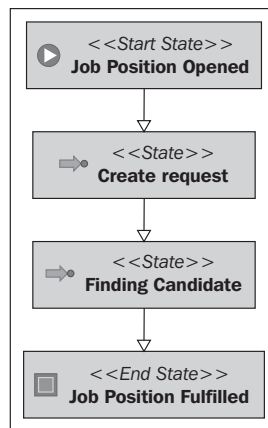
We will introduce a new process that will define these new activities. We need to have a separate concept that will create an on-demand new candidate interviews' process, based on the number of candidates found by the human resources team. This new process will be called "Request Job Position" and will include the following activities:

- **Create job request:** Different project leaders can create different job requests based on their needs. Each time that a project leader needs to hire a new employee, a new instance of this process will be created where the first activity of this process is the creation of the request.
- **Finding a candidate:** This activity will cover the phase when the research starts. Each time the human resources team finds a new candidate inside this activity, they will create a new instance of the candidate interviews' process. When an instance of the candidate interviews' process finds a candidate who fulfills all the requirements for that job position, all the remaining interviews need to be aborted.

We can see the two process relationships in the following figure:



If we express the Request Job Position process in jPDL, we will obtain something like this:



In the following section, we will see two different environments in which we can run our process. We need to understand the differences between them in order to be able to know how the process will behave in the runtime stage.

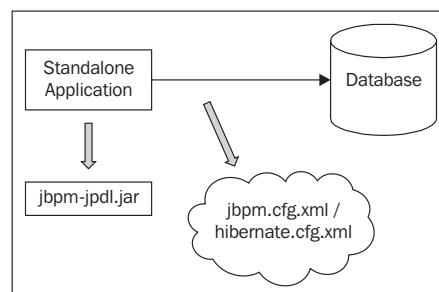
Environment possibilities

Based on the way we choose to embed the framework in our application, it's the configuration that we need. We have three main possibilities:

- Standalone applications
- Web applications
- Enterprise application (this will be discussed in Chapter 12, *Going Enterprise*)

Standalone application with jBPM embedded

In **Java Standard Edition (J2SE)** applications, we can embed jBPM and connect it directly to a database in order to store our processes. This scenario will look like the following image:



For More Information:

www.packtpub.com/jboss-business-process-management-jbpm-developer-guide/book

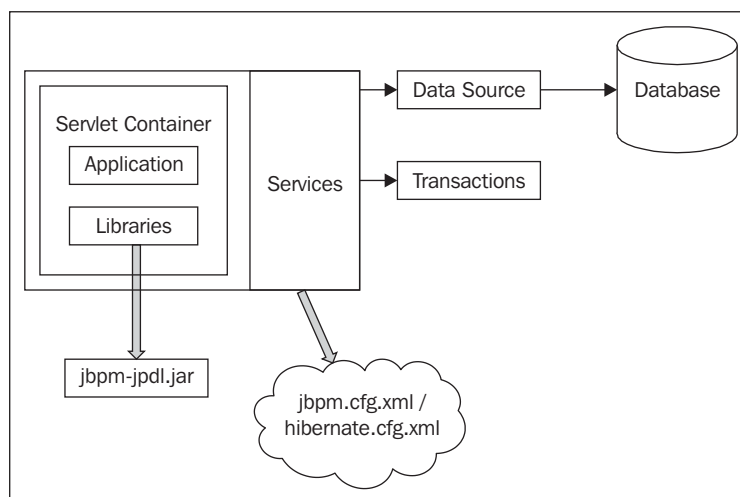
In this case, we need to include the jBPM JARs in our application classpath in order to work. This is because our application will use the jBPM directly in our classes.

In this scenario, the end users will interact with a desktop application that includes the `jbpm-jpdl.jar` file. This will also mean that in the development process, the developers will need to know the jBPM APIs in order to interact with different business processes.

It's important for you to know that the configuration files, such as `hibernate.cfg.xml` and `jbpm.cfg.xml` will be configured to access the database with a direct JDBC connection.

Web application with jBPM dependency

This option varies, depending on whether your application will run on an application server or just inside a servlet container. This scenario will look like:



In this case, we can choose whether our application will include the jBPM JARs inside it, or whether the container will have these libraries. But once again, our application will use the jBPM APIs directly.

In this scenario, the end user will interact with the process using a web page that will be configured to access a database by using a JDBC driver directly or between a `DataSource` configuration.

Running the recruiting example

In this section, we will cover the first configuration (the standalone one). This configuration can also be used to develop a web application. We will see that one has to test the whole process, which has been recently executed in order to see how it behaves.

This process will live only in memory, and when the thread that starts it dies, all the changes in that process will be lost. In other words, this process will start and end in the same thread, without using the database access to store the process status.

Running our process without using any services

In this section, we will see how our two processes will run using JUnit, so that we can test their behavior. The idea is to know how to move the process from one state to the other, and also to see what is really going on inside the framework.

Feel free to debug the source code provided here step by step, and also to step into jBPM code to see how the jBPM classes interact in order to guide the company's activities.

In this test, we will see how our two processes are chained logically in order to simulate the real situation. By "logically", I mean that the two processes are manually instantiated when they are needed. It is important to notice this, because there are situations where the process can be automatically instantiated, which is not the case here.

Take a look at the project called `/RecruitingProcess/`. You will find both the process definitions under `/src/main/resources/jpdl`. If you open the test called `RecruitingProcessWithoutServicesTestCase` located inside `/src/test/org/jbpm/example/recruiting/`, you will see a long test that shows how the process behaves in a normal situation.

Here we will explain this execution in order to understand the expected behavior and how this can be checked using JUnit asserts.

Normal flow test

If you take a look at the method called `test_NormalFlowWithOneCandidate()` inside the test case, you will see that we are trying to execute and test the normal flow of our defined process. We will simulate the situation where a new job position request is created. Then in our test, a new candidate is found. This will mean that a new candidate interview process will be created to evaluate if the candidate will get the job or not.

This is a simple but large test, because the process has a lot of activities. I suggest you to take a look at the code and follow the comments inside it.

In a few lines, you will see the following behavior:

1. A new job position request is created. This will happen when a project leader requires a new team member. This will be translated to an instance of the **Request Job Position** process. Basically, we parse the jPDL XML definition to obtain a `ProcessDefinition` object and then create a new `ProcessInstance` from it.
2. Now we need to start this process. When we start this process, the first activity is to create the request. This means that someone needs to define the requisites that the job position requires. These requisites will then be matched with the candidate's resume to know if he/she has the required skills. The requests (requisites) are created automatically to simulate the developer job position. This is done inside the `node-enter` event of the "Create Request" activity. You can take a look at the source code of the `CreateNewJobPositionRequestActionHandler` class where all this magic occurs.
3. When this request is created, we need to continue the process to the next activity. The next activity is called "Find Candidate". This activity will be in charge of creating a new process instance for each candidate found by the human resources team. In the test, you will see that a new candidate is created and then a new instance of the **Candidate Interviews** process is created. Also, in the test, some parameters/variables are initialized before we start the process that we created. This is a common practice. You will have a lot of situations like this one where you need to start a process, but before you can start it, some variables need to be initialized. In this case, we set the following three variables:

- `REQUEST_TO_FULFILL`: This variable will contain a reference to the process instance that was created to request a new job position.
 - `REQUEST_INFO`: This variable will contain all the information that defines the job request. For example, this will contain the profile that the candidate resume needs to fulfill in order to approve the first interview.
 - `CANDIDATE_INFO`: This variable will contain all the candidate information needed by the process.
4. Once the variables are set with the correct information, the process can be started. When the process is started, it stops in the "Initial Interview" activity. In this activity, some data needs to be collected by the recruiting team, and this again is simulated inside an action handler called `CollectCandidateDataActionHandler`. In this action handler, you will see that some information is added to the candidate object, which is stored in the `CANDIDATE_INFO` process variable.
 5. The information stored in the `CANDIDATE_INFO` process variable is analyzed by the following node called "Initial Interview Passed?" that uses a decision handler (called `ApproveCandidateSkillsDecisionHandler`) to decide whether the candidate will go to the next activity or he/she will be discarded.
 6. The same behavior is applied to the "Technical Interview" and "Technical Interview Passed?" activities.
 7. Once the technical interview is approved, the process goes directly to the "Medical Checkups" stage where a fork node will split the path of execution into three. At this point, three child tokens are created. We need to get each of these tokens and signal them to end each activity.
 8. When all the medical examinations are completed, the join node will propagate the execution to the next decision node (called "Medical Exams passed?"), which will evaluate whether the three medical check ups are completed successfully.
 9. If the medical exam evaluation indicates that the candidate is suitable for the job, the process continues to the last stage. It goes directly to the Project leader Interview, where it will be decided whether the candidate is hired or not. The outcome of this interview is stored inside a process variable called `PROJECT_LEADER_INTERVIEW_OK` inside the `ProjectLeaderInterviewActionHandler` action handler. That process variable is evaluated by the decision handler (`FinalAcceptanceDecisionHandler`) placed inside the "Final Acceptance?" activity.

10. If the outcome of the "Final Acceptance?" node is positive, then an automatic activity is executed. This node called "Create WorkStation" will execute an automatic activity, which will create the user in all the company systems. It will generate a password for that user and finally, create the user's e-mail account. It will then continue the execution to the "Candidate Accepted" end state.
11. In the "Candidate Accepted" node, an action is executed to notify that the job position is fulfilled. Basically, we end the other process using the reference of the process stored in the variable called `REQUEST_TO_FULFILL`.

I strongly recommend that you open the project and debug all the tests to see exactly what happens during the process execution. This will increase your understanding about how the framework behaves. Feel free to add more candidates to the situation and more job requests to see what happens.

When you read Chapter 6, *Persistence*, you will be able to configure this process to use the persistence service that will store the process status inside a relational database.

Summary

In this chapter, we saw a full test that runs two processes' definitions, which are created based on real requisites. The important points covered in this chapter are:

- How to understand real-life processes and transform them into formal descriptions
- How this formal description behaves inside the framework
- How to test our process definitions

In the next chapter, we will cover the Persistence configurations that will let us store our process executions inside the database. This will be very helpful in situations where we need to wait for external actors or events to continue the process execution.

Where to buy this book

You can buy jBPM Developer Guide from the Packt Publishing website:
<http://www.packtpub.com/jboss-business-process-management-jbpm-developer-guide/book>

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our [shipping policy](#).

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.



www.PacktPub.com

For More Information:

www.packtpub.com/jboss-business-process-management-jbpm-developer-guide/book