Degree project 10 Swedish credits

Workflow Management System in Java environment

Ángel Rodríguez García
September 2007
English Title:  
Workflow Management System in Java environment

Abstract:  
This degree project deals with the study about the development of workflow applications in a Java programming environment and the developing of an application that manages a claim process. The structure of the report contains four main sections: introduction, theory, design and implementation.

In the introduction, all the background and basic information needed is described. The theoretical part explains the concepts related to the project needed for good understanding. During the design and implementation parts, all the processes of developing the application, the difficulties and solutions found are described. Finally in the last part the Conclusions of this project and the future investigations are shown.

Language/Språk:  
English/Engelska

Approved by/Godkänd av:  
______________________________
Fredrik Jönsson
Examiner/Examinator         Date/Datum
Summary

The present report introduces the reader to the workflow topic and illustrates the process of building a workflow management system with Java based technologies. The workflow and business process management systems take a formal description of business processes (process definition) as an input. This formal description is usually written in a process definition language and maintains the state of instances of the mentioned process.

The business problem that the system in this degree project solves is the management of claims in an insurance company. Each claim is an instance of the process definition. The claims are submitted by the customers and every claim follows a process depending on its attributes. The process has states and in every state there is a user with a task to carry out. When the user finishes his task, the system makes the instance to flow to the next state and generates the task for the next user.

The solution chosen for the system is the development of a server side application embedding a middleware workflow framework. The server side application will follow the model-view-controller paradigm using Java Server Pages (JSP) and Java Server Faces technologies for the view and Java Beans for the logic in the model. The data in the model is provided by the workflow framework which uses a database for storing the information. The framework that will be used is the JBoss jBPM. jBPM is an open source framework which provides support and documentation and a graphical process designer for Eclipse and also a starter’s kit with a pre-configured JBoss server ready to use.

The application will be used via the browser. It provides an identification login page and when the user involved in the claim process is logged in it will show a list with the outstanding tasks of this user. The user can click on one task and the form of this task will be open letting the user complete it. When a task is completed it disappears from the list. The customers can register themselves in the system. When a customer logs in he will see the different claim forms and he can choose one for filling in. When the customer submits the claim a new instance of the process is created and a task for the user in the first state is generated. The system has an administrator that can manage the users and also the claim instances and tasks.

This project has as a goal to transfer the theoretical benefits of the use of workflows and business process management into a real problem. At the end of the project with this system the insurance claim process will be automated. The project can also be used to guide in the process of construction a Java based workflow management system.
CONTENTS

1. Introduction ......................................................................................1
   1.1. Background ...............................................................................1
   1.2. Purpose ...................................................................................2
   1.3. Method ...................................................................................2
   1.4. Restrictions ............................................................................4

2. Theory................................................................................................5
   2.1. JBoss jBPM ............................................................................5
   2.2. Process definition modelling .....................................................6
   2.3. Application interface .................................................................7
   2.4. Deployment .............................................................................9
   2.5. Software testing .......................................................................10

3. Design .................................................................................................12
   3.1. Process modelling .....................................................................12
   3.2. Application design ....................................................................15

4. Implementation..................................................................................18
   4.1. First steps .................................................................................18
   4.2. Server-side application ..............................................................18
   4.3. Process definition .......................................................................25
   4.4. Settings ....................................................................................30

5. Testing ...............................................................................................31
   5.1. Black box testing .......................................................................31
   5.2. White box testing .........................................................................32

6. Problems/solutions ........................................................................33

7. Conclusions ......................................................................................34

8. Suggestions for future investigations ..............................................35

References ..............................................................................................36

Appendix A
Appendix B
1. Introduction

1.1. Background

The current project will deal with the investigation in the emergent area of workflow systems and business process management (BPM) applications and the development of an example application using this technology that solves the management of claims in a fictitious insurance company. The Java technology provided by Sun Microsystems is one of the most important and most used software platforms nowadays; for that reason I chose to focus this project on how to develop a workflow application with Java technology.

The workflow technology started around ten years ago, but it is now when it is starting to became a popular solution due to its characteristics make them suitable for many situations. There is a very clear difference between concurrency in a workflow and java concurrency. A workflow or business process always contains a description of a state machine. When a state machine is created it enters the initial state. After that, signals can be applied to the state machine, causing the state machine to move to a new state. The traditional state machine is not sufficient to express real life processes, real life processes can have multiple concurrent paths of execution but many parts of a business application can be expressed in terms of a state machine, like in this case handling insurance claims. These are some form of execution that span a long time. When using Java-only, the good old top-down approach can't be applied because Java doesn't support persistent wait states. These kinds of processes include long wait states in which your server application is waiting for someone else or something else to initiate continuation of the overall execution. With Java (or any other imperative programming language) you just don't have a way to program a persistent wait state. Hence you can’t express the overall process in Java. The solution is to use a workflow language in which you can express processes. [3]

A workflow management system is a software component that takes as input a formal description of business processes and maintains the state of processes executions. We can define a business process as a set of linked activities that create value by transforming an input into a more valuable output. A process definition is a formal description of a business process or procedure and a process instance is one execution of a process definition. In workflow applications there is a process definition which describes the states of the business process and how does the data flow. The term workflow is more often heard in software developer communities, some authors maintain the similarity with the term business process management as synonym words but a BPM has a broader scope and it can be defined as a supercharged workflow that has sophisticated flow design through process modeling and analysis. [5]

Introducing workflow in an organization delivers benefits on the software development level as well as the business level. One of the best options of usage is embedding a workflow/BPM engine or framework in our project, we will use it like a Java library, as an application component, hiding it from the application users. The workflow engine is an execution environment which assists or performs the coordination of processes and activities. There are many open source workflow framework projects with we can provide support and maintainability to our program but the main benefit of this choice is the reusage (not reinventing wheels & hot water). [3]
1.2. Purpose
According to the previous reasons my purpose is to develop a server side application following the paradigm model-view-controller and using for the view (interface) Java Server Pages (JSP) and Java Server Faces, creating then a web application interface and embedding a business process management (workflow) framework that would let deploy in the server’s database the process definition of the business process problem we want to manage. Then the application has to deal with instances of the process, making them flow over the states following the specifications from the process definition as well as the input from the users and the context in each case.

The application is supposed to be used by multiple and different users, logically all users involved in the process have a different point of view of the interface because each one is working in different nodes or states of the process so consequently the program needs to provide a login interface where the users can identify themselves with an username and a password. As well when one user logs in the system the interface will show a list with the tasks which are outstanding to carry out by that user. The application will let the user select one from the list and the information from this task will be displayed for the user to complete his assignment on it. The costumers will have a different view, first they can register to use the application and for them the different claim form will be show letting the choose one to fill in the form and submit the new claim.

When a new insurance claim is created in the system it will automatically generate a task for the user assigned in the first task node (state) of the claim process. Then when this user carries his task out the instance claim will flow depending of the actual situation to the next node or back to the previous one, being placed then in the task list of the user in charge of that state. In this way the process instance (claim) will circulate through the problem workflow until it arrives to the end or final state and it is finished. In this case it will be that the insurance claim is declined or approved. The finished tasks are going to remain stored in the system database, so then it will be possible to access them from the application.

The application will also have an administration separated part where the program admin will have a different perspective for:

- Creating new users in the system.
- Updating and modifying the information from the actual users.
- Eliminating users.
- Searching for claim entering specific values.
- Monitoring status of claims.
- Modifying variable values and fields from claims.

The purpose of this application is to transfer the theoretical benefits of the use of workflows into a real problem like the management of insurance claims in a company. At long last with the developed program the fictitious company will have automated the tasks in handling and resolving the claims.

1.3. Method
For choosing a fitting method for this project, a pre-study of the different possibilities for embedding a workflow management system within a Java project was made. Considering the deadlines, the size of this project and its investigation purpose, I have came to the conclusion that the appropriate way to tackle this problem is to use a middleware workflow engine which
allows us an easier programming due to the considerable abstraction of the intern operation that it provides.

JBPM framework is the engine which I have finally chosen for several reasons. First of all it is an open source project in other words its source code and the code of any other project developed from it is made available to the general public with relaxed or non-existent intellectual property restrictions. That facilitates and makes faster the learning stage needed before you can really start to develop the program because you have access to the source of many projects and also the engine own source code. JBPM is designed to support multiple process languages and environments but especially it provides a truly process-oriented programming model with its Process Definition Language (jPDL). Another important thing is that it also provides a visual process designer for the Eclipse platform that simplifies the jPDL development making possible to draw the main structure of the process although the details must be written with the editor. Due to it the process definition of the claim problem will be expressed in the jPDL language.

The JBoss jBPM starter’s kit provides all the tools necessaries for work with jBPM. It includes an application server and the workflow engine embedded with a pre-configuration ready for deploy, the Eclipse plug-in for the visual process designer, a sample web application that connects with the server and all the documentation, source code and samples.

The idea is to develop the web application interface using as a template this sample web application because then this will reduce the risks and problems in the connection to the server and to make shorter the developing time, something very important to consider in this project due to the tight developing time we have. The method I will follow for the application construction is the Model-View-Controller paradigm which provides a complete division of the project into layers. MVC helps to reduce the complexity in architectural design, and to increase flexibility and reuse.

Another important part of this project is the process definition. As it was mentioned before it will be expressed using the jPDL definition language. Our task is study carefully the problem and then we need to represent it in a process archive. Here it is going to be very helpful the visual process designer provided for the Eclipse platform. The basis of a process definition is a graph that is made up of nodes and transitions. That information is expressed in an xml file called processdefinition.xml. Each node has a type like e.g. state, decision, fork, join…. Each node has a set of leaving transitions. There are also expressed the swim-lanes used in this process and also the schema and the mechanism to package all the process definition related files into a process archive.

Once the process is completely defined the next step is to deploy it in the application server. The deployment process consists of several interrelated activities that make the software system available for use, in our case when the process definition is correctly deployed in the server database system, it is accessible for the web application interface, being it able to create process instances of the process definition and store them on the server database. There are many ways to deploy a process in the server but one advantage of the jBPM framework is that we can do it from the Eclipse graphical process designer that allows us to connect to the server when this one is running and send to it the process definition, automating the deployment like that.
1.4. Restrictions

There are some factors in all software developing projects which one must consider for establishing the main goals of the project and making appropriate restrictions. Among those factors we have to mention the time deadlines and the human resources that restrict rather the time we can spend developing the program. Following this reasoning we have to consider apart extra functionalities as restrictions that would be appropriate for future versions and to focus ourselves in doing well the main goal of our project.

The following are some of the restrictions that this project has, with them I want to clear up the goal points mentioned in the purpose section:

- The management of the users will only be possible from the administration part by the system admin. Then the costumers can only enter their information when they register but modifying this information only can be done by the administrator.

- According to the investigation purpose of this project and that the chosen problem was a fictitious example the application will keep as generic as possible making then possible an easy re-usage for more concrete specifications.

- The application has to be developed from the point of view of a possible change in the company structure or the problem work path, being then possible undeploy the previous process definition and deploy the developed new one without any problem in the system operation. This will facilitate the maintenance and the future reuse.
2. Theory

2.1. JBoss jBPM

JBoss (a division of Red Hat) is an open source Java Enterprise Edition (Java EE) based application server implemented in Java. Because it is Java-based, JBoss is cross-platform, usable on any operating system that supports Java. JBoss implements the full Java EE suite of services. JBoss pioneered the Professional Open Source business model where the core developers of projects make a living and offer their services. Between JBoss-related projects there is JBoss jBPM the framework and workflow engine I have chosen.

The name is an acronym of java Business Process Management. jBPM is a JBoss Enterprise Framework that delivers workflow, business process management (BPM), and process orchestration in a scalable and flexible product. As well as it supports others process definition languages it also provides an outstanding process-oriented programming model with its Process Definition Language (jPDL). The jPDL language blends the best of both Java and declarative programming techniques and enables developers to structure their software around an easy to understand process graph. Mention that jPDL has minimal dependencies and can be used as easy as using a java library. But it can also be used in environments where extreme throughput is crucial by deploying it on a J2EE clustered application server. It also can be configured with any database and it can be deployed on any application server.

The framework is designed to support multiple process languages and environments including jPDL for workflow and human task management, BPEL for web services orchestration, and page flow constructs for embedded workflow in web applications. This ability to support multiple process languages is a key differentiator for enabling both process developers and web application developers to utilize a unified platform for creating a wide range of applications. This is an advantage in this case for adding possible extra features including web services in future versions.

JBoss jBPM includes an Eclipse-based visual designer that simplifies jPDL development. You can add it as a plug-in to your installed Eclipse software or download a new installation of Eclipse with the jBPM IDE already integrated and configured. The JBoss jBPM visual designer supports both a graphical and an XML code view of the business process or workflow being developed enabling people with different levels of process definition programming skills use it. The structural part of the claim process problem can be easily outlined with the predesigned nodes, tasks and transitions from the designer palette as well changing the view to the source code view type directly in jPDL language the conditions and elements that cannot be made only with graphical programming.

JBoss jBPM includes an assortment of pre-built process jPDL nodes or states that are used to declaratively build process graphs. These nodes provide the necessary process-building functionality such as start, task, fork, join, and decision. These nodes provide default process-aware execution as well as enable Java code to seamlessly plug in to extend the process logic and address the needs of the application.

One of the differences between jBPM and other open source frameworks is the support and documentation. The jBPM is a wide project with an important number of users. On the official web page you can find a complete documentation from all the versions including the Javadocs, download different versions of the product for different type of users and purposes.
The JBoss jBPM Starters Kit contains everything needed to execute JBoss jBPM, with the exception of a JDK. There is also a user forum section where you can post your doubts and problems with the jPDL language or with any part of the framework engine.

### 2.2. Process definition modelling

A process definition represents a formal specification of a business process and is based on a directed graph. This graph is composed of nodes and transitions. Every node in the graph is of a specific type that defines its different runtime behavior. A process definition has exactly one start state but it can have more than one end states. A token is one path of execution and it is the runtime concept that maintains a pointer to a node in the graph. A process instance is one execution of a process definition. When a process instance is created, a token is created for the main path of execution. This token is called the root token of the process instance and it is positioned in the start state. A signal orders a token to continue graph execution. When receiving an unnamed signal, the token will leave its current node over the default leaving transition. As well as much real problems the process can have more than one token of execution, for example after a fork node the path of execution will be divided and it will have two tokens that propagate parallel tasks. When a transition-name is specified in the signal, the token will leave its node over the specified transition. After the token has entered a node, the node is executed and then depending on its programmed behavior it will continue the graph of execution. A node that does not propagate execution will behave as a wait state.

**Process graph**

The basis of a process definition is a graph that is made up of nodes and transitions. That information is expressed in an xml file called processdefinition.xml. Each node has a type like e.g. state, decision, fork, join, etc. Each node has a set of leaving transitions. A name can be given to the transitions that leave a node in order to make them distinct.

**Nodes**

Each node has a specific type. The node type determines what will happen when an execution arrives in the node at runtime. jBPM has a set of pre-implemented node types that you can use. Alternatively, you can write custom code for implementing your own specific node behavior. Each node has 2 main responsibilities: First, it can execute plain java code. Typically the plain java code relates to the function of the node. E.g. creating a few task instances, sending a notification, updating a database … Secondly, a node is responsible for propagating the process execution. The node has some options for propagating the execution: not propagate the execution (then it behaves as a wait state), propagate it over one of the following transactions, create new path of execution (for example the fork node) or end the path of execution. Beside the pre-implemented node types the model is open for developers to create new nodes behavior and use them in a process. That makes the difference with the traditional workflow systems much more closed and limited.

**Transitions**

Transitions have a source node and a destination node. The source node is represented with the property from and the destination node is represented by the property to. A transition can optionally have a name, but if there is more than one possible transition on one node they need to be named. Note that most of the jBPM features depend on the uniqueness of the transition name. If more than one transition has the same name, the first transition with the given name is taken. The default transition is the first transition in the list.
**Actions**

Actions are pieces of Java code that are executed upon events in the process execution. The main view you will have from the process definition is the graph, the graph is an important part that describes the program requirements. But the graph is just one view (projection) of the program that is being built. It hides many technical details. Actions are a mechanism to add technical details outside of the graphical representation. Once the graph is in place, it can be complemented with actions. This means that Java code can be associated with the graph without changing the structure of the graph. The main event types are entering a node, leaving a node and taking a transition. The actions can be associated with an event and when it occurs the code of the action is executed.

The jBPM engine will fire events during graph execution. This occurs when jBPM calculates the next state, and then the list of actions is executed. [6]

**2.3. Application interface**

As you could read before the interface of the application will be developed as a dynamic web application. When downloading the complete package of the JBoss jBPM framework, you can find within the documentation the source code of a simple web application whose purpose is showing the user the basis of using the jBPM framework libraries inside a web application and how to interact with the server workflow engine. This application shows how to access to the process definitions that are deployed in the server and how to use jBPM classes for creating tasks and instances for the process. The mentioned sample web application has been developed with Java Server Pages technology and Java Beans. In order to take advantage of this implementation that simplifies a bit the procedure of developing a server side application because all the libraries from the engine are correctly embedded I will enlarge this application sample adding all the functionalities that I need for my project but continue using the same technology. Then I can reuse the Java beans already implemented for the monitoring part because with those beans you can access the instances of a process definition and the tasks of a process instance.

**Model View Controller Paradigm**

Model-view-controller is an architectural pattern used in software engineering and which is follow in the creation of the server-side application in this project. In large applications is common to split them into separate layers: presentation (UI), domain logic, and data access. With this pattern the presentation layer is further separated into view and controller. [12]

The model is the domain specific representation of the information and the logic added to manipulate that information. Usually a persistent storage mechanism like databases is used to store the data. Due to our application is not a pure web application, it’s a workflow management system. Then the model consists in the entity beans (which provide the logic), the workflow framework (which also provides logic) and the database used by the framework to store the data of the process instances, task instances, etc.

The view renders the model into a form suitable for interaction, typically is a user interface. For one single model there can exist multiple views for different porpoises. The model has not direct knowledge of the view. In our project the view are the Java Server Pages that are also currently implemented using the Java Server Faces technology.

The controller processes and responds to events, typically user actions that may invoke changes in the model. In other words it translates interactions with the view into actions to be
performed by the model. In the project the role of the controller is taken by the java servlets like in general in every Java Enterprise Edition project.

In conclusion, by decoupling models and views, MVC helps to reduce the complexity in architectural design, and to increase flexibility and reuse.

**Java Server Pages**

Java Server Pages is server-side technology that has been created as an extension to the Java servlet technology that was developed by Sun Microsystems. This Java technology allows software developers to dynamically generate HTML, XML or other types of documents in response to a Web client request. The technology allows Java code and certain pre-defined actions to be embedded into static content. [11]

JSPs have dynamic scripting capability that works in tandem with HTML code, separating the page logic from the static elements (the design and display of the page) to help make the HTML more functional. The JSP syntax adds additional XML-like tags, called JSP actions, to be used to invoke built-in functionality. Additionally, the technology allows for the creation of JSP tag libraries that act as extensions to the standard HTML or XML tags. Tag libraries provide a platform independent way of extending the capabilities of a Web server.

A JSP is translated into Java servlet before being run and it processes HTTP requests and generates responses like any servlet. However, JSP technology provides a more convenient way to code a servlet. Translation occurs the first time the application is run. A JSP translator is triggered by the .jsp file name extension in a URL. JSPs are fully interoperable with servlets. You can include output from a servlet or forward the output to a servlet and a servlet can include output from a JSP or forward output to a JSP. [1]

Inside the JSP container is a special servlet called the page compiler. This page compiler turns a servlet container into a JSP container. When a .jsp page is first called, the page compiler parses and compiles the .jsp page into a servlet class. This makes very slow the navigation the first time but then when all the classes are compiled it is faster. If the compilation is successful, the jsp servlet class is loaded into memory. On subsequent calls, the servlet class for that .jsp page is already in memory; however, it could have been updated. Therefore, the page compiler servlet will always compare the timestamp of the jsp servlet with the jsp page. If the .jsp page is more current, recompilation is necessary. With this process, once deployed, JSP pages only go through the time-consuming compilation process once.

JSP is not a replacement for servlets. Rather, JSP technology and servlets together provide an attractive solution to web scripting/programming by offering platform independence, enhanced performance, separation of logic from display, ease of administration, extensibility into the enterprise, and most importantly, ease of use. [11]

As a whole with the Java Server Faces framework we use the Java Server Pages for the creation of the view in the project. Then the jsp pages are going to be composed by HTML code, Java code inside the scripts and Java Server Faces tags for the view implementation.

**Java Server Faces**

Java server faces is a Java-based Web application framework that simplifies the development of user interfaces for Java EE applications. Unlike other traditional request-driven MVC web frameworks, JSF uses a component-based approach. The state of UI components is saved
when the client requests a new page and then is restored when the request is returned. JSF uses Java Server Pages for its display technology, as we are going to use them in our project but JSF can also accommodate other display technologies.

JSF includes a set of APIs for representing user interface components and manage their state, handling events and input validation, defining page navigation and supporting internationalization and accessibility. Also two libraries of jsp custom tags for expressing a Java Server Faces interface within Java Server Pages. Managed beans and sever side event model. [12]

**Java Beans**

Enterprise Java Beans (EJB) is a server-side component that encapsulates the business logic of an application. This technology is an architecture for both using in building components in Java. The EJB specification was originally developed in 1997 by IBM and later adopted by Sun Microsystems (EJB 1.0 and 1.1) and enhanced under the Java Community Process. [12]

This architecture supports the features of software reuse, component models, and object orientation. One of the most important features of JavaBeans is that it does not alter the existing Java language. If you know how to write software in Java, you know how to use and create Beans.

While JavaBeans were originally introduced as client-side, GUI-builder-friendly components, there is nothing in the JavaBeans specification that limits their use to the client-side or GUI. In fact, it's fairly common to use JavaBean components with a JSP. It's also easy and useful. At the bare minimum, a JavaBean is an object that has a public no-argument constructor and follows the set/get paradigm. This means that there is regularity in the get and set methods.

We are going to reuse and extend some beans from the jBPM sample page like the process definition bean or the process instance bean. Then for each module in our application it will be a managed bean to provide the logic to access the data with its corresponding scope.

### 2.4. Deployment

When the application is completed (also during construction for testing) we need to build it and deploy it in the server. For the compilation and building we are going to use the tool from apache Ant which works with XML scripts to automate the process facilitating then this task which has to be completed a lot of time during the application construction. The other deployment we need to do in the project is the process definition deployment. Once the process definition is written we need to deploy it in the server, then the workflow engine can work with it. This process is much easier thanks to the Eclipse graphical process designer which has a tab exclusively created for the process deployment. When choosing this tab we can introduce the server settings like address and port and test the connection. Once the connection is successfully tested we can easily deploy the process definition with just press one button. Then it is ready to be used by our application.

**Ant**

Apache Ant is a software tool for automating software build processes. It is similar to make but is written in the Java language, requires the Java platform, and is best suited to building Java projects.
This tool has the advantage that it doesn’t depend on the commands from the shell in the different operating systems because is based in configuration files written in XML language and Java classes to make the different tasks during the build process. Being it an ideal multi-platform tool.

For using Ant it is only necessary to have a binary distribution of the tool and to have installed a version of the Java development kit JDK higher than the 1.4. Actually only the folders bin and lib from the ant distribution are necessary for executing it.

Even though Ant is a multi-platform tool there are some previous settings that are different in every operating system. Due to this project is going to be developed in the windows platform those settings will be briefly explained. First add the bin folder of the Ant distribution to the path and set the variable ANT_HOME to the place where Ant has been installed. And also set the environment variable JAVA_HOME to the path where JDK is installed. These settings are made from the command shell as it is shown below:

- set ANT_HOME=c:\ant
- set JAVA_HOME=c:\Archivos de programa\Java\jdk1.5.0_02
- set PATH=%PATH%;%ANT_HOME%\bin

Ant works with build-files that are written in XML. Every build-file has one project and at least one target. Every target can contain multiple tasks that are code fragments for executing. One project can also contain diverse properties. Each of those properties has a name and a value and is used for setting values to the tasks attributes.

For the Ant execution is enough with typing ant in the command line when this one is situated in the folder with the build.xml file. For using another build-file it must used with the parameter –buildfile and the name of the new build-file. [12]

2.5. Software testing

The nest stage after the implementation stage is the software testing. Software testing is the process used to measure the quality of the developed software. Usually, quality is constrained to topics like correctness, completeness and security but it can also include more technical requirements. The correctness is the minimum requirement of software. The testing includes, but is not limited to, the process of executing a program or application with the intent of finding errors. This process can give a criticism or comparison with the software specifications but it can never completely establish the correctness of the software.

There are many different types of test used through the testing process. The two main types of testing are white box and black box testing but it is also coming into common usage the grey box testing in the recent years.

**Black box test**

The black-box test is a testing method in which test data are derived from the specified functional requirements without regard to the final program structure. Because only the functionality of the software module is tested, black-box testing also mainly refers to functional testing. A testing method emphasized on executing the functions and examination of their input and output data. The tester treats the software under test as a black box, only the inputs, outputs and specification are visible, and the functionality is determined by observing the outputs to corresponding inputs.
White box test
Contrary to black-box testing, software is viewed as a white-box, or glass-box in white-box testing. The structure and flow of the software under test are visible to the tester. Testings are made according to the details of the software implementation. Test cases are derived from the program structure. White-box testing is also called glass-box testing, logic-driven testing or design-based testing. White box testing has the intention of checking all the possibilities the code can cover, like executing each line of code at least once (statement coverage), execute every branch statements (branch coverage), or cover all the possible combinations of true and false condition predicates (Multiple condition coverage).

Verification and validation
Software testing is used in association with verification and validation (V&V). Verification is the checking or the testing of items, for checking the correspondence with the software specifications. Because the software testing is not only for finding bugs and eliminating them, it is also a tool for verification and validation. Software testing is just one kind of verification, there are also other kinds which uses techniques like reviews or inspections. Validation is the process of checking if the software fits with what has been specified. [12]
3. Design

The design is the process of application of different techniques and fundamentals with the objective of defining the model or models of the program we want to develop with enough details to allow the correct implementation later. The design is one of the most important parts in every project and sometimes in large projects the most important one. Usually the design takes about the 40% of the time dedicated to the project when the implementation and testing takes about the 20% each part. It also depends on the size of the project even though in smaller projects like this one it has a considerable influence on the implementation stage. [12]

The design has some theoretical principals which can facilitate this task and some are progressive refining, in other words to go from the general idea to the small details, information concealment which is to hide the non relevant elements, modularity which consists of dividing the project in modules as small as possible to facilitate its development afterwards and abstraction in other words to try to abstract from the concrete characteristics having then a easy overview of the main problem. The application of the mentioned principals during the design will contribute to the creation of quality software and lead the project to success.

The first decision which must be taken is to decide what we need to model during the design. There are to distinguishable parts in the project than can be dealt independently. One is the process definition which needs to be modelled before its construction. So starting from the explanation of the business problem which correspond with the insurance claim we can model the state based structure as well as the respective swimlanes. The other distinguishable part of the project that needs to be modelled is the server-side application where the view can be modelled with web map diagrams, also the flow pages structure and the logic and data composed by the Java beans and database tables.

3.1. Process modelling

The process modelling lies in the representation of the business problem into a workflow model or business process model. For a better understanding of this part it is necessary to have a detailed knowledge about the business problem which will be modelled.

The claim process is being treated here starts when the insurance costumer pretends to claim to the company about any incident or accident cover by his insurance policy. This process is the general one so it is the same for all the types of claim. In the fist step the costumer fills out the claim form with his personal data and the appropriate information about the incident such as the type of incident, the date when it happened, a detailed description and if he is sending corroborated documents like a police report. When the costumer submits the claim it arrives to a reviewer who has to check if the information sent is complete. If not then the claim goes back to the costumer and he has to complete which the reviewer requires and submit it again. Otherwise it is accepted as valid and it starts the process. Then it arrives to an evaluator who will decide if a report from an expert is necessary for the claim review. If it is then the claim goes to the expert who elaborated the required inform and sends back the claim. Otherwise it skips this stage and goes directly to the review which is divided into two different typed depending on the amount sued by the costumer. If the amount is smaller than 1000€ then the claim just has a simple review by a reviewer. Otherwise the flow starts tow different ways, one goes to a reviewer for an incisive review and the other parallel has two stages. One stage with an inspector who checks the fraud story of the costumer and another stage for check the
costumer’s credit. After both different reviews the claim always arrives to the decision stage where it is approved or rejected. When is rejected it goes to the secretary department for sending a letter o e-mail with detailed information and if it is accepted it has two parallel process, one in the secretary department for sending an informative letter or e-mail and another in the accountant department for carrying out the claim payment.

Workflow process models as are known by many names, but because of their appearance are most commonly referred to as “swimlane diagrams”. A swimming pool might be divided lengthwise into swimlanes. Just as each swimmer is expected to stay in his or her swimlane, each “actor” with a “role” in the process has his or her own swimlane. A box represents a task or step in the process, and is placed in the swimlane of the responsible actor. Arrows connecting the boxes indicate the sequence and flow of the steps.

\[ \text{Figure 1: Swimline model diagram which describes the business process and the actors rolls within the process.} \]

Since the swimlane diagrams are a good modelling tool for process definitions and easy understandable, they are not enough to express completely the process. They empathize in the
tasks, the actors and the interaction of them with other actors. One good complement for the swimlane diagrams in the process modelling is the state chart diagrams. State Diagrams describe the flow of one process instance over several use cases. A state diagram also describes all possible states in the process and the possible paths to get from one state to another.

Figure 2: State based diagram from the business process which describes its flow of execution.
The states are represented as rounded rectangles, there are also two special types of state: the start state represented by a solid circle and the end state represented by a solid circle surrounded with a circle. All the states are connected together by transitions that point from a source to a destination state. The black rectangles where more than one transition start represents parallel process, where both transitions are taken. There are also conditions represented by diamond-shaped square which establish the path that the process instance will flow.

3.2. Application design
As it was said in the theory part the pattern followed during the application is the MVC, model-view-controller which divides clearly the application into three differentiable parts. The view here is the application interface. As it is a web interface, the structure of the interface can be modelled using a web map diagram where all the parts that the application will have are shown using a schematic view.

Another part of the view that be modelled is the pages structure. Here the page files and the connection between them will be defined. Due to the application is developed using Java Server Pages technology the view is making up by .jsp files as it is shown in the diagram below.

![Diagram of the application's page structure](image)

Figure 3: Pages diagram of the application which describes which files make up each module and the flow logic between them.
The previous diagram is being made from the diagram you see below. This web map diagram describes the different parts of the application interface and its functionalities.

Figure 4: Web map of the application interface where all the different parts are defined.
With the two previous diagrams the structure and functionalities from the view part are modelled. In the figure 3 the rounded rectangles represent jsp files and the arrows the path that the application can follow from each one. Being possible the return where the arrows have double end. As well it is displayed that some pages contains other as it is the case of the header and menu which are included in the rest of the pages. In the figure 4 the interface is modelled from the functionality point of view showing the main parts of the application and the subparts that each one has. Some graphs are used in this diagram do represent web page elements like: web links, login form or other forms, lists or complex list with links embed.

The logic of the application is another important part that must be modelled. The logic is contained in the java beans. The java beans are object from java classes and here the classes that the application will need in the different areas will be listed explaining the functionality of each one.

There are some bean classes in the sample web of jBPM that are going to be reused in our project. Those beans are ProcessDefinitionBean.java that gets the process definitions deployed in the server as well gets the process instances from a previously set process definition and also starts a new process instance. And also the MonitoringBean.java which can list the instances of a process definition and also show the tasks with it variables and corresponding values. Besides the mentioned beans some other classes will be needed in each part of the application. Here they are listed with a briefly explanation of its managed task:

- **UserBean**: This bean will be used during the login process for checking the username and password provided and during the hold session to identify who is logged in and access to its information. As well it has the user properties that are set during the user self registration it will add the new user to the system and set the user as logged in afterwards.

- **HomeBean**: This bean will get the list of the task from the user that is logged in for listing them in the home page. It also will set the task selected before going to task.jsp. When the user is a costume this beans provides the method to start a new instance on the process definition.

- **AdminBean**: This bean contains an object of the next three classes and is used for the system's user administration.

- **NewUserBean**: Here the properties are set with the values of the new user and it has a method that inserts these values in the system users table.

- **UpdateUserBean**: As the class mentioned before this one has similar properties and also two methods, one that gets the old values from the database of the user is being modified and another method that changes the values setting the new ones in the table.

- **DeleteUserBean**: This one has a property for the username and a method that deletes this entrance from the system users table in the database when it is called.

- **TaskBean**: This beans list the variables and values from the task selected to be shown in the task.jsp page and provides the methods for finishing the task or for save the changes made in the task but keeping it in the task list of the user.

- **ProcessInstanceBean**: This bean is used in the monitoring part together with the mentioned monitoring bean. Here the values of the process instance can be inspected and also the list of the tasks from this instance (claim) can be got.

- **ErrorBean**: This bean is uses during the hold session and it provides a property where the message errors can be set when an exception occurs and a method to list this messages that can be called from the error page.
4. Implementation

Once the design is completed the implementation of the claim manager starts. This transition is not always clearly differentiated. In this concrete case it is because the project can be divided in to two different and quite independent parts the process definition and the application. So then the implementation of any of them can be started before to finish with the design of the other one. This is the process I have follow making parallel implementation of both parts.

The implementation stage contains all the steps followed in the construction in practice, all the parts that compose the project with different aspects of them and how do they interact with each other and other some important details of the implementation process. It also describes the important files in each part and the problems found during the implementation and how they where solved.

4.1. First steps

Now that it is described what we have to do there are some decisions to take before we start to implement the different parts in the application. For the implementation of the claim process definition the tool is already chosen but we haven talked yet about which tool or tools use to implement the server-side application.

Due to the project is composed by different kind of files like html files (the .jsp files), Java files (the beans), some XML files and some other files then the best option is to develop the project with the Eclipse platform because it provides auto completion and code recognition for many different languages including the mentioned before. Also because the web interface in the .jsp files it is not complex it is not necessary to develop it with a graphical tool and it can be done programming the html and faces code.

Another thing that wasn’t mentioned during the design is how the data is stored and how we use the database. The workflow engine works with the Hypersonic database and it has some tables where it stores the data of the process definitions, process instances, tasks, variables etc. The only thing that the application needs is a table for the users because the rest is done by the workflow engine. The fist what I thought was to create a new table for the users but the engine also has a table for the jBPM users and then the username must be the same in both and changing the username could generate problems and inconsistencies in the program. So I thought it is better to use the jBPM users table adding some extra columns for store users information because all the users of the application are actors for the engine and we would avoid the inconsistencies. Not because is easier since, as creating a new table is the same work as alter one existing adding extra columns.

4.2. Server-side application

The main part of this project is the server-side application. This kind of applications uses the active web pages technology. In this case the technology used is the Java Server Pages. The application is composed mainly by .jsp files and .java files besides all the configuration files and the .jar libraries. Now I’m going to explain with details the parts of the application and the main files that they include.
The application starts with index.jsp that just redirects to login.jsp because the first step for access to the system is provide a valid user-name and password (see figure 7 in Appendix A). The login.jsp file consists in a form with an imputText and an imputSecret where the user introduces the name and the pass. And a command-bottom which action is “#{userBean.login}”. The userBean is one of the session beans used in the program, they are specified in the faces.config.xml file and it has a session scope. The bean are normal java files, the userBean includes besides other things two properties the username and the password and the method login that access to the database to check the values given. The login method returns a String which can be “home” or “login”, then depending on the navigation rules specifies in the faces.config.xml the application will follow to the adequate page.

All the pages have two common parts the header and the menu. The header is inheader1.jsp and it consists in the logo of the claim manager on the top at the left and also on the right some links and labels. Always it is saying if you are logged in the system or not and as who you are logged in. When the user is logged in there is one link that says “log in as another user” this redirects to login.jsp. When you are not logged in, there is also other link that redirects to the registration page. It is also showed here a link to the help.

```<%
UserBean userBean = (UserBean) session.getAttribute("userBean");
if ((userBean!=null) && (userBean.getUserName()!=null)) {
    <
    You are logged in as <%= userBean.getUserName() %> <
    } else {
    <
    You are not logged in. | <b><a href="registration.jsp"> Registration</a></b>| <b><a href="login.jsp"> Login as another user</a></b>
    <
    %>
```

The code above shows how it is implemented embedding the plain java code in the html code. First the userBean is get from the session accessing to its attributes and then depending if it is null or not it shows different links and labels with the login information.

The menu is in header2.jsp and it shows a menu row on the left with the different parts of the system. Every kind of user has different permits so in the same way as it was shown before the userBean is accessed to check which kind of user is logged in. When the type is U (normal user), C (costumer) or V (visitor) then the menu only shows a link to the home page but when the type is A (system administrator) also the links to the administration page and the monitoring page are shown.

**The registration**

The costumers can register themselves to use the system. In the login part when you are not logged in, you can see a link to the registration page. The registration page is in registration.jsp and it consist in a form with some imputText and imputSecret that the user has to fill in with his information including the policy number, the name, a username, a password and his e-mail address (see figure 8 in Appendix A). Each input field is connected with the properties of the userBean and also all of them are required in the form. The
The code above corresponds with the method register() from the class UserBean. There we can see how the new user is introduced in the user table. First the query is created for the SQL command INSERT using the values of the prosperities which were set from the jsp inputs. Then the database connection is accessed from the jbpm current context and the method executeQuery is called. Afterwards the user is registered in the system so in the application he should be already logged in. That part is made setting the Actor in the jbpm context and the username in the userBean. If the query execution doesn’t generate a SQLException the method returns “success” otherwise it returns “error”.

```java
public String register()
{
    String reg = "INSERT INTO JBPM_ID_USER
    (CLASS_,NAME_,EMAIL_,PASSWORD_,USER_) VALUES ('C','" +
    regUserName+"','"+regEmail+","+regPass+","+regName+")";

    try {
    //access to the database connector and execute query
    JbpmContext.getCurrentJbpmContext().getConnection().createStatement().executeQuery(reg);
    //set the JBPM actor and the current user
    JbpmContext.getCurrentJbpmContext().setActorId(regUserName);
    userName = regUserName;
    userPass = regPass;
    userClass = "C";
    reset();
    return "success";
    } catch (SQLException e) {
    //if any exception occurs return error
    return "error";
    }
}
```

**The home page**

The home page is the main view that the user has from the application. When the user logs in the page he will see is the home page home.jsp. Bearing in mind the different kind of users the system has every kind of user will have different parts and functionalities in the home page.

The costumers will have two principal parts in the home page. The first one is a list with the different process definitions that the system manages (different types of claim). At the moment there is only one process developed, the general claim process but more process definitions that describes other kind of claim process can be deployed in the server. What the costumer sees is a table where each row corresponds with one process definition and it has some columns with the process information (id, name, description …) and also the name of the first task that is “create a new claim” and the costumer can click in the label to open the form for creating a new claim and submit it in the system. This label is a command-link that call the method homeBean.startProcessInstance() with the parameter of the process definition id. This method will redirect to task.jsp where the corresponding task form is shown (see figures 9 and 10 in Appendix A).
The code above shows how this table is created. This is not exactly the code used, there are more columns in the table but with this two is enough to explain how it works. The dataTable has associated the value of one method. This method returns a list of objects from the class processDefinition. For each object in the list one row is created and each column shows one property of the object. The first column is as well a link that starts a new process instance.

The other part that the customer sees is a list with messages from the users in the claim process. The structure is mainly the same as the task list that the normal users have. When the customer submits a claim form a new process instance is created and it is possible that the customer takes part again in one of the tasks. In the general case when the reviewer checks the information and it is not complete the customer has to complete it. So in this list the mentioned task will appear with a link that will open again the claim form to complete it.

The users that take part in the claim process have one principal view in the home page. That is a task list with the outstanding tasks that the user has to complete. The administrator of the system has a similar home page as the normal users. The way that this list is shown is similar to the way mentioned before for the process definition list. A data table is also used and it shows information about the task like the process it corresponds to, the task name, the date it was generated, who generated the task, the id number and a description. The task id is also the link that calls homeBean.selectTaskInstance() and opens task.jsp to complete the task (see figures 11 and 12 in Appendix A).

When the user clicks on one of his outstanding task from the list, a form will open to let the user complete the task. In the process definition the task contains a set of variables that are global for the hold process. When a task is programmed the set of variables that take part in this task are selected. The form that is opened when the user clicks on the task shows this set of variables with their correspond values. For each variable the user has different access permits like only reading, reading and writing or also that the user must introduce a value. A taskBean is created and this bean contains all the information of the task in the process instance. The bean has a method that returns a list of formParameters objects. For each one of this form parameters the name is shown and an entry with the value of the variable. The entry will have different write properties depending on the variable properties in the task. After the completion of the task there are one or more possible transitions to follow. For each transition a command-button is shown. Then the user can decide the way that the claim has to follow
after he completes his task. It may happen that the user opens a task but he doesn’t complete it. Then on the down par of task.jsp there are always two buttons, one button for saving the changes made in the task and another to cancel without saving. When the user saves the changes the task is still not finish so it will continue standing on the outstanding task list of the user. Below we can see part of the code of task.jsp where the buttons for each possible transition are shown. All the buttons will have the action saveAndClose() but with a different availableTransition set.

```jsp
<c:when test="${!empty taskBean.availableTransitions}"
    Task Options:
    <c:forEach var="availableTransition" items="${taskBean.availableTransitions}"
        <c:set var="availableTransition" scope="request" value="${availableTransition}"
        <h:commandButton id="transitionButton" action="#{taskBean.saveAndClose}" value="#{availableTransition.name}"/>
    </c:forEach>
</c:when>
```

The administration page

The application provides extra functionalities for the system’s administrator and one of these functionalities is the administration page where the admin can manage and control the system’s users. The administration page is located in admin.jsp and can only be access from the menu navigator that is on the left. The only user that is able to see this link is the admin so this part is hidden to the rest of users. The user’s administration is divided in four parts. The first one is for creating new users, the second one is for listing all the users in the system, the third one for modify the information stored about a user that already exists and the last one for deleting users from the system (see figure 13 in Appendix A)

For the creation of a new user by the admin there is just a link that redirects to new_user.jsp where there is a form similar to the one explained before for the registration. The difference is that here the administrator can choose the type of user he is creating between normal user and costumer user. Furthermore the process it follows is very similar to the registration process (see figure 14 in Appendix A)

The users listing procedure has similar structure. There is also a link that redirects to another page, list_users.jsp (see figure 15 in Appendix A). In this page there is a dataTable similar to the ones shown before witch is connected with the method getUserList() from the class UserBean. This method returns a list of object MyUser which just contains the properties of the user. Then each property is connected to a column in the table and for each object from the list one row is created. In the method getUserList() the process used to get a list with the users is the following: first the query with the Select SQL command is stored in a String. Then the database connector is get from the JBPM context and the query is executed. This execution returns a ResultSet and this one is read row by row creating a new MyUser object each time and setting on it the info from each row of the ResultSet. Then these objects are stored in the list that is finally returned.

For updating the information of the users there is a small form in the administration page composed by an inputText and an actionButton where the admin can introduce the username from the user who wants to modify and press the button (see figure 16 in Appendix A). The inputText is associated with the property username from the updateUserBean. When the
button is presses the text from the input is set to the bean’s property and the button calls the method setValues(). That method makes a select in the users table from the database and gets the old values of the user’s info. The values got from the database are set in the bean’s properties. The method returns a string and if everything goes well it returns update and with the navigation rules the program will follow to the page update_user.jsp. The page mentioned before contains almost the same as the page for creating a new user but in this case the inputTexts are associated to the properties of the updateUserBean so they have already the values set before and the admin can see the old information and modify what he wants. When the information is update the admin press the button update info and this one calls the method update() from the bean. This method creates a SQL update table query from the values got in the properties and executes it in the database connector. If no SQLException is generated then it will return “success” so the program will follow to the success.jsp page otherwise the result will be “error” following then to error.jsp.

The last part allows the admin to delete a user from the system. There is a small form similar to the one before where the admin can introduce the username and then press the button delete. This button has associated the method delete() from deleteUserBean. What this method does is to delete the user from the users table in the database constructing a SQL delete query and executing it with the database connector. Mention that the user table has integrity relations so the users will not be able to delete if they have process instances associated. So is task for the admin to delete his claim instances from the monitoring area before he can delete the user. Another possibility is to change the user type so then the user can not log in to the system and the claims information is still stored in the database. Other possibility considered before was to make cascade deleting, in other words to delete everything associated to the user when deleting the user. But I discarded this option because it must be important for the company to keep all this information so then only when it is strictly necessary the claim instances will be deleted from the system.

The monitoring page

The monitoring page is with the administration page the part of the application exclusively accessible for the system’s administrator (see figure 17 in Appendix A). Here the admin can do monitoring tasks like list the instances of a process definition, look for a particular instance (claim) giving the value of any of its variables, inspect the status of one claim and all the task associated to it and also to delete all the process definitions deployed before. The reasons that make this part exclusively for the admin is that he can access from here to all the information stored and that the changes he can make affect to the stability of all the system so only an appropriate user can do that.

The first element in the monitoring area allows the admin to delete the process definitions previously deployed. This only should be used in the case that the admin want to make a complete cleaning of the system because if in the case that the process that a claim follows changes and they want to apply the changes to the system then it is only necessary to modify the old process definition and deploy it. It will overwrite the previous one and the version will be changed.

In the second part the admin can supervise the state of all the claims in the system. When the command link is pressed the method showProcessDefinitions of the monitoringBean is called. This method redirects to process_definitions.jsp (see figure 18 in Appendix A) and sets a list in the bean with the process definitions deployed. That page shows this entire list in a table with some properties like the number of instances. The admin can click in the number of
instances of any of them. When he clicks in one all these claim instances will be shown. The method showProcessInstances from processDefinitionBean is called and this method gets from the database all the instances using his id. These instances are stored in a list of the bean and then the bean set itself to the session beans in the JBPM context so it will be accessible from the web application. Afterwards it returns “processinstances” that makes the application to follow to process_instances.jsp. This page contains a datatable which is connected with the bean mentioned before so all the claim instances from the list will be shown. In the table the property ID of the instances is a command link that opens inspect_instance.jsp where all the information of the instance is shown (see figure 20 in Appendix A). This module of the application was already made in the sample page that JBPM provides. I have included in the web application changing a few thing to make it more suitable for the project because like this I’m reusing code and the functionality that it provides is very complete. So I thought that the admin should have a monitoring perspective in the application and this module also with the advanced search module is generic enough to embed them in my application.

The other monitoring part is the search part. Here the admin can look for claim instances providing some values of the claim (see figure 19 in Appendix A). There are two types of search, the advanced search and the search by ID. With the advanced search the admin is redirect to search_instances.jsp where the user can introduce the name of one variable and one value choosing “equal” or “is like” and the method search instances is called. This method makes a complex selects in the tables of the framework and sets a list with maximum 100 instances that fit with the search requirements. Then the method returns back to the same page and if the list is not empty the down part of the page is shown. There is a table where the claim instances of the list will be collocated showing some or their values. For every claim the admin can click in the link that the ID has and the method inspectInstance() will be called. The method will put that process instance in the session beans and return “inspectinstance” which redirect to inspect_instance.jsp. There all the information of the claim instance is shown. On the top the fields will show the values of the properties of the instance. Below there is a small table that shows all the tasks associated to the instance and the admin can see if they are completed or not. When clicking in one of the task from the list bellow will appear the variables from the task and the values of them. Also the admin can change the values of the variables in the small form, in one input he should write the name and in the other the new value and press the command button. I’m not going very deep in the complete operation of this part because the module was already developed so here is more important the functionality than the operational process.

The search by ID is the simplified search of the one explained before. The admin inserts the ID of the claim and the method inspectInstance() is directly called. Then if the claim is fond the page explained before will be shown will the complete information. Otherwise the search error will be shown to the admin if there are no claims with that ID number.

The faces configuration
The faces configuration is stored in the xml file faces-config.xml. In this case the faces configuration file has only two types of staff. The managed beans then the application will have and also all the transition rules which set the flowing way after the methods in the Java Beans. The beans are Java objects which store the logic of the application. More or less each module in the application has one bean because dividing simplifies the programming of the application and also because each bean has different scope. For specifying a managed bean in the faces configuration file there are a few thing that must be written. As you can see in the code below the name that the object will have is the first. Then the complete package route of
the Java class must be set. Also the type of scope the bean will have is necessary. The scope express a bit the life of the bean, if the scope is session the bean’s values are accessible during the hold session of the user but for example when the scope is request the bean only stays during the request. The beans can have as properties other beans and these properties can be set to managed beans as you can see in the last part of the code. The class HomeBean has two properties, one TaskBean and one UserBean and the beans previously specified taskBean and userBean are set here as its properties so they are accessible from the homeBean.

```xml
<managed-bean>
  <managed-bean-name>homeBean</managed-bean-name>
  <managed-bean-class>webapp.bean.HomeBean</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>taskBean</property-name>
    <value>#{taskBean}</value>
  </managed-property>
  <managed-property>
    <property-name>userBean</property-name>
    <value>#{userBean}</value>
  </managed-property>
</managed-bean>
```

The navigation rules express the path that the applications flow. When a command button call a method the method will return a String. This string will specify with the rule to with page to flow. As it is shown the rule must specify from which view it will flow and the different cases with the correspondent view destinations.

```xml
<navigation-rule>
  <from-view-id>/new_user.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/success.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>error</from-outcome>
    <to-view-id>/error.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

### 4.3. Process definition

Once the interface application is ready the other important part in this project is the process definition, with the process definition the application is able to trace the flow of our business problem.

As it was mentioned before, the tool used for the developing of the process definition was the Eclipse platform with the plug-in graphical process designer from JBoss JBPM. For solving the possible incompatibilities that upgrading the plug-in could produce in the Eclipse it was decided in favour of taking the clean installation of Eclipse that you can download from JBoss page with the plug-in already installed and configured.

The first step is to create a new process project. Its name will be General Claim Process. In the package explorer we open for editing the file `processdefinition.xml`. At this moment is very useful the states diagram created during the design because choosing the tab “diagram” a palette with nodes and transitions appears and we have to draw the main structure of the claim
The process has one start state node where the token is placed when an instance is generated and one end state node where the token/s arrive/s when the flow of execution is completed. Following the structure of the states diagram, we choose for each state the adequate type of pre-configured node from the palette. When one state is clearly associated with a task carried out by a user then we choose the task node which later can associate a task to a user throw the swim-lanes. In the case of the decisions there are mainly two different possibilities. When the decision can be take automatically by the system we choose node type decision, like when the claim with an amount higher than 1000€ follow different path than the others do. In the case that the decision has to be taken by a user we choose the task node, like when for a claim we need to decide if it is necessary a report elaboration by an expert. When the flow of execution is divided following two parallels paths it is necessary to use a node type fork that can be joined with two or more nodes. Breaking then the token into two or more independent tokens that need to be joined after the parallel process is finished. Because of that when there is only one end state and we use a fork node, it is necessary to use a join node that combines the tokens into one and when the first one arrives to the join node it waits there until the other arrives without propagating the execution. All the nodes must have a name adequate with the task or actions are taking place in. For the completion of the structure of the process definition, the nodes need to be joined using the transition from the palette, the transition determines the path that the token will follow within the nodes.

**Figure 5:** print-screen of the eclipse graphical process designer with the view of the diagram structure and the palette of nodes and transitions.
When the basic structure of the process is drawn then we can programme the tasks, actions and decisions. For each task node doing right click in the diagram we can choose to create a new task. It is possible to develop the almost complete task in the graphical designer, but it is also a good choice to do it from the source view typing the task in JPDL language. One of the important parts of a task is the controller. In the controller there are defined the variables that are used in that task, for example the first task of the process is for the customer and it is to complete the claim form with the corresponding information and submit the claim. So in this case the variables will be the name, insurance policy number, type of incident, etc. For each variable is also possible to defined the permissions that the user has, for example if he only can read it or if he can modify it. It is also possible to mark the variable as a required value, and then the task can not be completed if the user doesn’t give it a value. The variables are easily added to the controller with the add button and filling out the name and mapped name.

```xml
<task name="Review and Decide Claim" swimlane="deepreview">
  <controller>
    <variable name="Name" access="read" mapped-name="Name of the insured"/>
    <variable name="Surname" access="read" mapped-name="Surname/s"/>
    <variable name="PolicyNumber" access="read" mapped-name="Insurance policy number"/>
    <variable name="IncidentType" access="read" mapped-name="Type of incident"/>
    <variable name="Date" access="read" mapped-name="Accurate date"/>
    <variable name="Amount" access="read" mapped-name="Amount sued"/>
    <variable name="Description" access="read" mapped-name="Description"/>
    <variable name="Docum" access="read" mapped-name="Sending corroborable documents"/>
    <variable name="ExpertReport" access="read" mapped-name="Expert Report of the Incident"/>
    <variable name="Approved" access="read,write,required" mapped-name="Approve Claim?"/>
    <variable name="Reasons" mapped-name="Reasons"/>
  </controller>
</task>
```

The table above shows the code in JPDL for the task “Review and Decide Claim” with all the parts mentioned before. This code is inside the code of the node the task belongs to. For more detail see the Appendix B.
Figure 6: View of the properties of the task explained above showing the controller part with the variables used in this task.

All the nodes that have the type task-node have at least one task implemented inside. Those are the task implemented in the process with a briefly explanation of each one:

- **Create new claim**: here the customer fills in the form and submit the claim.
- **Verify information**: the info of the form is verified and accepted if it is adequate.
- **Complete info**: if the information is not enough the customer has to complete it.
- **Evaluate claim info**: here it is decided if a report from an expert is needed or not.
- **Elaborate incident report**: the appropriate expert prepares a report when it is required.
- **Review and decide claim**: those are two tasks one is for the small claims and the other for the bigger claims. The main review is done here.
- **Review client credit**: when the claim is big the credit from the client is checked.
- **Check client fraud history**: also only with big claims the fraud history is investigated.
- **Approve or reject**: according to the previous users informs the claim is decided.
- **Send rejection letter**: if the claim is rejected a letter with the reasons is sent to the customer.
- **Send inform letter**: when the claim is approved the customer also receives an informative letter.
- **Pay claim**: when the claim is approved it has to be paid to the customer.

Each task belongs to a user. This liking is made via the swim-lanes. The swim-lane can be assigned to an expression or a handler. For assigning it to the user I have chosen to do it via the expression: user (user-name). Where username is the name that the user has in the system. Then when the task is generated it goes to the task list of that user.

The code below is an example of how to declare an expression swimlane. In this project there are one swimlane for each user in the system because one user can have more than one task.
In the task node must be specified inside the task which swimlane has assigned. The list of the different swimlane in the process is:

- **Costumer**: this one places the linking with any user who starts the claim (costumers).
- **Checking**: it links with the user “reviewer” who checks the information when the claim is submit and according with the informs accepts or rejects each claim.
- **Evaluate**: it links with the user “evaluator” who decides if any report is needed.
- **Report**: it links with the user “expert” who elaborates the expert’s inform.
- **Normal-Review**: it links with the user “evaluator1” who reviews the simple claims.
- **Deep-Review**: it links with the user “evaluator2” who reviews the big claims.
- **Check-Credit**: it links with the user “accountant” who checks the costumer’s credit.
- **Fraud-History**: it links with the user “inspector” who researches about possible previous frauds from the costumer.
- **Letters1**: it links with the user “secretary1” who has to send rejection letters.
- **Letters2**: it links with the user “secretary2” who has to send the informative letters.
- **Payment**: it links with the user “clerk” who takes the responsibility of the payments.

When a task can be automated, in other words when it is not necessary that any user carries it out and the system can do it by itself then it is used a decision node instead of a task node. In the decision node are written the conditions that make the token to flow one u other transitions after the evaluation. The process has a decision node when the claim is accepted as valid by the evaluator and it is going to the process of reviewing then the amount claimed is evaluate and depending if it is higher or lower than 1000€ the execution will follow different ways.

```xml
<decision name="Claim Importance">
  <transition name="&lt; 1000 $" to="Review Claim">
    <condition expression="#(contextInstance.variables.Amount lt 1000)"/>
  </transition>
  <transition name="&gt;= 1000 $" to="fork1">
    <condition expression="#(contextInstance.variables.Amount >= 1000)"/>
  </transition>
</decision>
```

Above there is the code for the decision. In JPD when a node has more than one leaving transitions then it is necessary to put condition expressions inside the transitions, otherwise the first would be the default one which the token would take. In the order written the expressions are getting evaluated and when the value is true the execution follows that transition, otherwise the next expression is evaluated in this way until one is true.

When the process definition is completed then it is time for deploying it at the server. From the Eclipse graphical process designer we can do the deployment. First of all the server must be running. Then in the Eclipse we go to the tab-view deployment and there we choose the files we want to deploy: `processdefinition.xml`, `gpd.xml` and `processimage.jpg` are default marked. The server name and the server port are also required. For testing is the Eclipse can connect to the server press the bottom Test Connection and if the test its successful the process definition can be deployed pressing Deploy Process Archive. Now it is deploys and ready to access from the application.
4.4. Settings

Once we have the application finish and also the process definition has been completed there are besides the testing some previous settings that must be done before the application is ready to use.

After the claim manager application is being compiled and built it has to be deployed in the server. The built files as the war file are located in a folder called built inside the project folder. The starters kit from jBPM comes with a JBoss application server pre-configured where the application can be easily deployed locating the files in the deploy folder and change some settings in the start.bat script file which makes the server running.

At least the admin user has to be in the system otherwise it is not possible to log in and insert the rest of the users. For this task we use the jmx console from jBPM. First the server has to be running. For starting the server is enough with executing the file start.bat which is in the server folder. When it is running the jxm web console can be accessed with the browser typing in the address the server address and port (now just http://localhost:8080/) and /jmx-console. From the jmx console the hypersonic database can be managed starting the bean database manager. Then a database manager starts and it has an entry where it is possible to type the SQL queries and execute them. Then using the SQL command Insert we insert in the table JBPM_USER_ID a new user which username must be “admin”. Then the application has an admin user registered and can be accessed.

Another thing that must be done is the process definition deployment. Now together with the server the jBPM workflow engine is also running but there is no process definition deployed in the database. From the Eclipse graphical process designer we open the general claim process definition. Then in the tab “deploy” the address from the server and the port must be enter and afterwards press the button test connection. If the connection is successfully tested the just with pressing the deploy button the process definition of the general claim process is deployed when any error message is shown.

The last step to make the complete application ready is to insert the users that take part in the general claim process. This step can be made as it was done before for the admin directly with the database manager but it is easier and quicker to do it from the application. As it was said before during the process implementation, the process definition has different swimlanes which assign a task to a user. When this swimlanes are created the username has to be given. This username corresponds with the username from the table JBPM_USER_ID that is the one used for the system users. Then starting the claim manager application with the browser (http://localhost:8080/claim-manager) and logging in as admin we have to insert all the process users. For that in the administration area we choose create a new user and we fill in the form with the user information providing the username exactly as it was set in the swimlane. The rest of the information like names, e-mails can be changed later by the admin. Now the process definition requires that these users exist to work properly.
5. Testing

The testing is an important part of every software project because its porpoise is not only to find bugs and solve them. With the testing we also make sure that the application works properly and it is a verification that the software developed carries out with the specifications. As it was said during the theoretical part the combination of white box testing and black box testing check almost every feature of the software. In the following sections the different tests applied to the project are shown with their corresponding results.

5.1. Black box testing

With the black box testing only the functionality of the software is tested. The different modules of the application are treated as black boxes, being visible only the inputs and outputs and checking the results with the specifications.

Due to it is not possible to check all the different possibilities a set of different tests the most generic possible and that covers mainly the specifications have been done. In the table bellow there are the results of the different test with the inputs, the desired outputs and if the test was successful.

<table>
<thead>
<tr>
<th>Input / Action</th>
<th>Desired output / Result</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>A new user registers itself in the Claim Manager.</td>
<td>The users information is inserted into the users table and a success message is shown and he is now logged in.</td>
<td>Yes</td>
</tr>
<tr>
<td>A user tries to register itself with a policy number already registered.</td>
<td>The user can not register and an error message explaining the reason is shown.</td>
<td>Yes</td>
</tr>
<tr>
<td>The client user creates a new claim and submits the form.</td>
<td>There is now a new process instance and the first user in the claim process has a new task in his outstanding tasks list.</td>
<td>Yes</td>
</tr>
<tr>
<td>The reviewer sends the form back for completing the info.</td>
<td>The costumer who submitted the claim in gets the form in his message list.</td>
<td>Yes</td>
</tr>
<tr>
<td>A claim with amount value of 200 passes the first two states.</td>
<td>The evaluator1 gets a new task in his list for a simple claim review.</td>
<td>Yes</td>
</tr>
<tr>
<td>A claim with amount value of 1050 passes the first two states.</td>
<td>The evaluator2 gets a new task in his list for a deep claim review and also the accountant gets a new task for review client’s credit.</td>
<td>Yes</td>
</tr>
<tr>
<td>A claim passes all the states in its flow.</td>
<td>The process instance stays in the system with all the tasks associated done.</td>
<td>Yes</td>
</tr>
<tr>
<td>The administrator creates a new user of the system.</td>
<td>The user’s information is inserted into the users table and a success message is shown.</td>
<td>Yes</td>
</tr>
<tr>
<td>The administrator modifies the information of a system’s user.</td>
<td>The new information is inserted in the user’s table and a success message is shown.</td>
<td>Yes</td>
</tr>
<tr>
<td>The administrator tries to delete a user with claims associated.</td>
<td>The user can not be deleted and an error informative message is shown.</td>
<td>Yes</td>
</tr>
<tr>
<td>The administrator tries to delete a user without any claims associated.</td>
<td>The user is deleted from the table in the database and a success message is shown.</td>
<td>Yes</td>
</tr>
<tr>
<td>The admin search for a claim with a given id number.</td>
<td>The correct claim is shown with its associated information.</td>
<td>Yes</td>
</tr>
</tbody>
</table>
The admin search for a claim with a concrete value of amount.  
The correct claim is shown with its associated information.  
Yes

The admin search for a claim with a value that none claim has.  
No claim with this value is founded and a search error message is shown.  
Yes

The admin ends one task that was pending for one process instance.  
This task disappears from the task list of his user and a new task is generated for the next state and added to the list of the corresponding user.  
Yes

The admin modifies the value of the variable “approved” in one claim.  
The new value is saved in the database and a success message is shown.  
Yes

The admin deletes all the process definitions deployed in the workflow system.  
There are no process definitions deployed and no instances can be created. The customer sees an empty list of different claim forms.  
Yes

The results show us that the functionalities tested work properly but the black box tests are never enough to say that the software functionality is correct but are a good tool to check that the software apparently keeps the specification.

5.2. **White box testing**

Contrary to the black box testing in the white box testing the structure and the flow of the software are visible for the tester and the cases are derived from the program structure. The white box testing tries to test the integration and the units or modules of the software.

To test the unit level implies to test every small part of the application. In this case the units can be considered the different bean classes. The unit test has the intention of checking all the possibilities that the code can cover, like the execution of every line of code or every branch of code at least once and also to cover all the different possibilities and combinations in the loops. Considering the quantity of code we have in all the classes to take this testing policy is unviable due the fix time we have for the project. So every class has been tested separately with java units. These units are just small classes that use the methods of the classes we are testing. Since all the possibilities in every loop couldn’t be programmed executing all the lines of code then the cases chosen in the test units were agreed to the logical cases that can occur during the normal use of the application.

Together with the test of every class separately the white box testing includes integration test. This testing checks the integration of the modules of the application working together. After checking the units separately and the integration with them and also with the black box test showed in the previous part we have an acceptable vision of the application reliability.
6. Problems/solutions

First during the project design and after during the implementation some difficulties and problems arise and had to be solved in some way. In this section some of this problems will be explained and also the solution that was taken and why was it taken. Just those problems that are a bit illustrative for this project or interesting in some way will be described.

When a costumer wants to use the application, first he has to register in the system. For this registration he clicks on the link that there is at the starting page. Then the application flows to the registration page. During this process the userBean that identifies the user who is logged in is null because nobody logged in and this thing involved some exceptions in the server due to the userBean is also used for identifying the actor in the workflow system. The solution founded was to create a new user called visitor and set the user bean with this username for the registration process. This user has zero permits so it can not access the system and its only function is to make the bean not null during the registration.

Another difficulty founded during the implementation is the problem with the users table in the database. It was already mentioned before that the application just needs a table for store the user’s information and at first this table was going to be a new table with the username field connected with the jBPM users table. But this would cause inconsistencies in the future if any user is modified in one table and the changes are not transfer on cascade to the other table. For avoiding any possible inconsistence in the database we just choose to use the jBPM users table for storing the information of the system users since, as every user is also user of the workflow engine. This choice involved the modification of that table adding some more columns to it.

Another problem arisen when choosing the scope of the managed beans. In concrete the bean used during the process of modification the information of a user. At first sight it looks that the scope of this bean should be request since it is used only during this process. But the difference is that it is used two times, one for showing the old information from the user and the next one for making the changes in the information. If the scope is just request the information is lost after the admin sees the details of the user. Then for the changes update the information is taken from the form but we need other information that is not shown like the users id which can not be changed so for that we made the scope session and the bean is cleaned every time after it is used.

As you could read before in the theoretical part the Ant tool used for compiling and building the project needs some setting before it can be used. These settings are mainly adding some paths to the environment variable. But the thing is that every time that the computer was switched off these settings disappear and the Ant couldn’t be executed any more. Bearing in mind that during the implementation the application had to be compiled and built a lot of times for testing and checking changes made it was very annoying to type this settings in the command line again all the time. So for avoiding this process and loose also less time I made a script .bat file that executes the commands to make the setting and can be used very easy and fast just making double click and executing it.
7. Conclusions

Once the project reaches the finishing line and the development process is completed, it is time to write about the conclusions that this working time has provided. This section will list the most relevant conclusions.

One of the goals of this project that was mentioned at the beginning in the purpose section is to transfer the theoretical benefits of using workflow and business process management systems to a real problem. We can see that the management of the claims is now automated for the users in the insurance company and the application simplifies the problem reducing the time and work that was necessary before. The users involved in the claim process have now organised their task being also able to have a quick access to the tasks data.

The administrator has a complete control of the whole application from the users to the claim instances. He is able create, modify, delete or inspect all users, claim instances and the tasks of every instance. The admin can also end task without the permit from the owner and also to clean the system undeploying the process definitions.

The claim manager improves the service that the company offers to the costumer. The application let the costumers to contact the company for the claim in a very easy way. The costumer can easily register and submits his claim without any inconvenience. The system also reduces the costs for the company since the claim is submitted in the process directly by the costumer avoiding any intermediary like could be for example via telephone, letter or e-mail.

Another conclusion that should be taken into consideration is the investigation purpose of the project whose intention besides the teaching that have given to the developer it is also useful for other developers who are interested in getting started with this topic. The project provides the basics concepts and principal ideas of workflow management. The project can be used also as a manual for starting with the workflow systems development because it describes the way to embed a workflow framework in an MVC application and it explains how these workflow engines works.
8. Suggestions for future investigations

Now that the developing time is over and the program covers the specifications that where exposed at the beginning of the project, the developer has a very deep knowledge about the project and the possibilities it has so it is now when the ideas for improving it come up. In this section some of these ideas will be explained for possible future versions of the application due to now the developing time is completed and it wouldn’t be possible to carry them out.

The e-mail support is one of these ideas. When the project started and the workflow framework was chosen we didn’t consider the e-mail support as a requirement of the framework we were going to use. But during the developing of the application the possibility of that the users receive an e-mail when a new task is assigned to them came out. And also for the registration that when a new costumer registers itself an e-mail is sent to the address he provided for checking if it was correct and also for informing the costumer that the registration was successful. But the version of the jBPM framework we used doesn’t support e-mail and it was too much work and no time left to develop it. The easiest for a future version is to update jBPM to the version 3.2 that is now stable (when the project started was beta so we chose the 3.1) and that version supports e-mail being easily implemented to send an e-mail to an actor when a task is generated and it also includes mail nodes for the process definition so other possible situations like the mentioned before with the registration can be solved.

One interesting extra feature for the project would be that the process instances could also include documents or files. This change would automate completely the business process avoiding any kind of paper during the process. For example when the expert elaborates the report that he could upload it in a PDF file to the system and the claim instance would have it associated. The part of uploading the file and storing it in the system is very easy but the difficulties for its implementation would appear in the task.jsp page because it is generic and dynamic for all the different tasks. That’s why this feature was not included thinking in the developing time that would be necessary to use.

Another recommendation for a future version is to upgrade the workflow framework to the jBPM version 3.2. This recommendation comes from the JBoss official page when they made stable the version 3.2 because there are some bugs in the previous version that can make the system instable. One of this errors appears when sometimes the condition expressions if the transition don’t get evaluated. This can cause that the default transition is followed when it should be a different one. To upgrade the framework is not that easy as it sounds so that’s why it was not done due to the developing time wouldn’t permit to finish on time. And also due to the investigation nature of this project wasn’t considered as a priority.
References

Books:
    This book contains a description of how to use Java Server Pages and Servlets for web application development.

    This book teaches how to use the XML technology within Java projects.

Articles:
    This article gives a wide overview about the actual state of the business process management and workflow technology.

    This article gives an introduction of Business Process Management, its history, and selected vendors.

    This document offers an explanation of workflow and how it actually works and how it relates with Java.

WWW:
    JBoss jBPM Official Website

    JBoss tutorial site

[8] [http://jira.jboss.com/](http://jira.jboss.com/)
    Jira project official site

    Eclipse official site

    Apache official site

    Java official site

    The free encyclopedia
Appendix A                  Application’s print-screens

Figure 7: Login page.

Figure 8: Costumer’s registration page.
Figure 9: Customer's view when logged in.

Figure 10: Customer creating a new claim.
Figure 11: Reviewer’s view of the application with his task list.

Figure 12: View of one of the tasks, verifying claim form.
Figure 13: Admin’s view of the user administration page

Figure 14: Administrator creating a new user.
Figure 15: List of all the system’s users.

Figure 16: Administrator editing the information of one user.
Figure 17: Admin’s view of the monitoring part.

Figure 18: List of the claim instances in the system.
Figure 19: Advanced claim search page.

Figure 20: Claim instance inspection page where all its information is shown.
<xml version="1.0" encoding="UTF-8"/>

<process-definition xmlns="urn:jbpm.org:jpdl-3.1" name="GeneralClaim">
  <swimlane name="costumer"></swimlane>
  <swimlane name="comprobate">
    <assignment expression="user(reviewer)"></assignment>
  </swimlane>
  <swimlane name="report">
    <assignment expression="user(expert)"></assignment>
  </swimlane>
  <swimlane name="evaluate">
    <assignment expression="user(evaluator)"></assignment>
  </swimlane>
  <swimlane name="normalreview">
    <assignment expression="user(evaluator1)"></assignment>
  </swimlane>
  <swimlane name="deepreview">
    <assignment expression="user(evaluator2)"></assignment>
  </swimlane>
  <swimlane name="checkcredit">
    <assignment expression="user(accountant)"></assignment>
  </swimlane>
  <swimlane name="fraudhistory">
    <assignment expression="user(inspector)"></assignment>
  </swimlane>
  <swimlane name="letters1">
    <assignment expression="user(secretary1)"></assignment>
  </swimlane>
  <swimlane name="letters2">
    <assignment expression="user(secretary2)"></assignment>
  </swimlane>
  <swimlane name="payment">
    <assignment expression="user(clerk)"></assignment>
  </swimlane>
  <start-state name="Create Claim">
    <task name="Create New Claim" swimlane="costumer">
      <controller>
        <variable name="Name" access="read,write,required" mapped-name="Name of the insured"></variable>
        <variable name="Surname" access="read,write,required" mapped-name="Surname/s"></variable>
        <variable name="PolicyNumber" access="read,write,required" mapped-name="Insurance policy number"></variable>
        <variable name="IncidentType" access="read,write,required" mapped-name="Type of incident"></variable>
        <variable name="Date" access="read,write,required" mapped-name="Accurate date"></variable>
        <variable name="Amount" access="read,write,required" mapped-name="Amount sued"></variable>
        <variable name="Description" access="read,write,required" mapped-name="Description"></variable>
        <variable name="Docum" access="read,write,required" mapped-name="Sending corroborable documents?"/>
      </controller>
    </task>
  </start-state>
</process-definition>
<transition name="continue" to="Claim Importance"></transition>
</task-node>
<task-node name="Expert Report">
  <task name="Elaborate Incident Report" swimlane="report">
    <controller>
      <variable name="Name" mapped-name="Name of the insured"></variable>
      <variable name="Surname" mapped-name="Surname/s"></variable>
      <variable name="PolicyNumber" mapped-name="Insurance policy number"></variable>
      <variable name="IncidentType" mapped-name="Type of incident"></variable>
      <variable name="Date" mapped-name="Accurate date"></variable>
      <variable name="Amount" mapped-name="Amount sued"></variable>
      <variable name="Description" mapped-name="Description"></variable>
      <variable name="Docum" mapped-name="Sending corroborable documents?"></variable>
      <variable name="ExpertReport" access="read,write,required" mapped-name="Expert Report of the Incident"></variable>
    </controller>
  </task>
  <transition name="" to="Evaluate Info"></transition>
</task-node>
<decision name="Claim Importance">
  <transition name="&lt; 1000 E" to="Review Claim">
    <condition expression="#{contextInstance.variables.Amount lt 1000}" />
  </transition>
  <transition name="&gt;= 1000 E" to="fork1">
    <condition expression="#{contextInstance.variables.Amount &gt;= 1000}" />
  </transition>
</decision>
<task-node name="Review Claim">
  <task name="Review and Decide" swimlane="normalreview">
    <controller>
      <variable name="Name" access="read" mapped-name="Name of the insured"></variable>
      <variable name="Surname" access="read" mapped-name="Surname/s"></variable>
      <variable name="PolicyNumber" access="read" mapped-name="Insurance policy number"></variable>
      <variable name="IncidentType" access="read" mapped-name="Type of incident"></variable>
      <variable name="Date" access="read" mapped-name="Accurate date"></variable>
      <variable name="Amount" access="read" mapped-name="Amount sued"></variable>
      <variable name="Description" access="read" mapped-name="Description"></variable>
      <variable name="Docum" access="read" mapped-name="Sending corroborable documents?"></variable>
      <variable name="ExpertReport" access="read" mapped-name="Expert Report of the Incident"></variable>
      <variable name="Approved" access="read,write,required" mapped-name="Approve Claim?"></variable>
      <variable name="Reasons" mapped-name="Reasons"></variable>
    </controller>
  </task>
  <transition name="done" to="Approval"></transition>
</task-node>
<fork name="fork1">
  <transition name="review" to="Incisive Review"></transition>
  <transition name="extra actions" to="Review Credit"></transition>
</fork>
<task-node name="Incisive Review">
  <task name="Review and Decide Claim" swimlane="deepreview">
    <controller>
      <transition name="continue" to="Claim Importance"></transition>
    </task-node>
  </task>