JBoss Process Server Demo Guide

Overview

For years, Java programmers have baked business rules and process logic into Java code. There is something terribly wrong with having to re-write and re-deploy code every time a business process or rule changes but BPM and rules technology is known as expensive, niche technology. JBoss, a division of Red Hat, has broken the high cost barrier with an Open Source BPM and Rules Engine offering. Pragmatic, Java developer friendly, enterprise grade BPM and Rules Engine technology is now available to the masses. By combining the leading J2EE Application Server (JBoss Application Server), the leading Open Source BPM solution (jBPM) and the leading Java based Rules engine (Drools), Red Hat has forever changed the BPM and Rules Engine landscape.

Problem

JBoss Rules has garnered a fair amount of fan-fair by providing a very good, general purpose, Java rules engine framework. Unfortunately, JBoss Rules, also known as Drools, doesn't provide an effective rules management mechanism. Many development groups struggle with the following:

- How should the JBoss Rules binaries be packaged/deployed to the server? It's easy to bundle the JBoss Rules binaries in each rules engine enabled EAR, but it's far more likely that several applications will share a single set of JBoss Rules binaries.
- How should the POJO (Plain Old Java Object) model be packaged/deployed? Again, it's likely that several applications will need to make calls to the same rules engine instance and it makes sense to have a single POJO model that is shared.
- How should rules be version controlled?
- How should rules be deployed? Ideally, rule deployment will not necessarily mandate application re-deployment. It would be nice to GUI enable rules management/deployment.
- How should rules be stored for application usage? Caching a Rule Base makes sense.
- How does one test a rule set? This is important because rules will reference/manipulate POJO attributes. If a rule in the rule set has a typo or references a POJO attribute that isn't currently deployed, the rule will not compile. In this scenario, testing a list of DRLs prior to deployment makes sense.

JBoss jBPM is the leading Open Source BPM framework for Java applications. Like JBoss Rules, it's POJO based and very good at handling business process definition, via a visual designer. JBoss jBPM also delivers process orchestration/management via a straightforward Java API that back-ends to a RDBMS (Relational Database Management System) for process version control and long running process support. But, jBPM is missing the following:

- How should the jBPM binaries be packaged/deployed to the server? It's easy to bundle the jBPM binaries in each BPM enabled EAR, but it's far more likely that several applications will share a single set of jBPM binaries and RDBMS tables.
- How should the POJO (Plain Old Java Object) model be packaged/deployed? Again, it's likely that several applications will need to make calls to the same business process instance and it makes sense to have a single POJO model that is shared.
- How should processes be version controlled? JBoss jBPM versions runtime processes in a database, but it's also a best practice to version control process definitions in a traditional source control system.

Most applications can benefit from a unified BPM and Rules solution, and the two are often confused and misused because they are tightly coupled. The one constant in IT is change. JBoss Process Server provides the means to change business processes and business rules without changing java code.

Solution

The missing piece to this puzzle involves three simple steps:

- Create a process server configuration that includes all JBoss Rules and jBPM binaries. This process server configuration is basically JBoss Application Server pre-configured for BPM and Rules based applications.
- Create a simple process server web console that supports rule/process browsing, testing and deployment. The web console is backed by a source control system, so all deployable BPM processes and rules are version controlled.
- Create a persistent rule engine cache so that rules can be cached in memory, deployed to a single server or cluster and pre-loaded when the application server is bounced.

Assumptions

By following the KISS(Keep It Simple Stupid) model, we can leverage robust rules management right here, right now. Having said that, there are a few key assumptions that are required to deliver the goods with this approach quickly, efficiently and effectively.

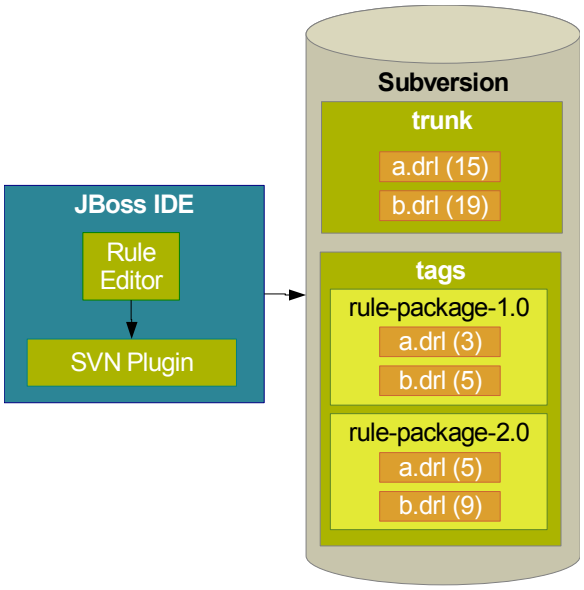
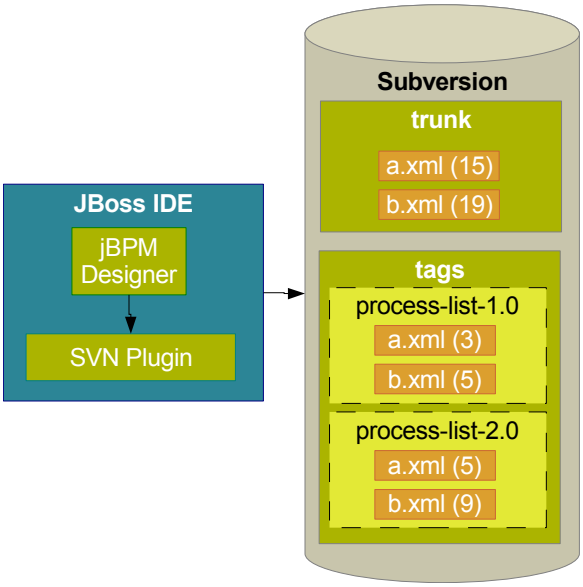
- Subversion is chosen as the source control system for this solution. That's because it's better than CVS, it's easy to setup, open source, and there are good Java APIs to interface with the source control system.
- A simple process administration console will suffice.
- JBoss Application Server is required and while this solution can be ported to WebSphere, Weblogic and standalone Tomcat, it is not portable without some code changes.

This approach has its limitations, but there's no need to over engineer and postpone BPM and Rules enablement. Here's what we're avoiding:

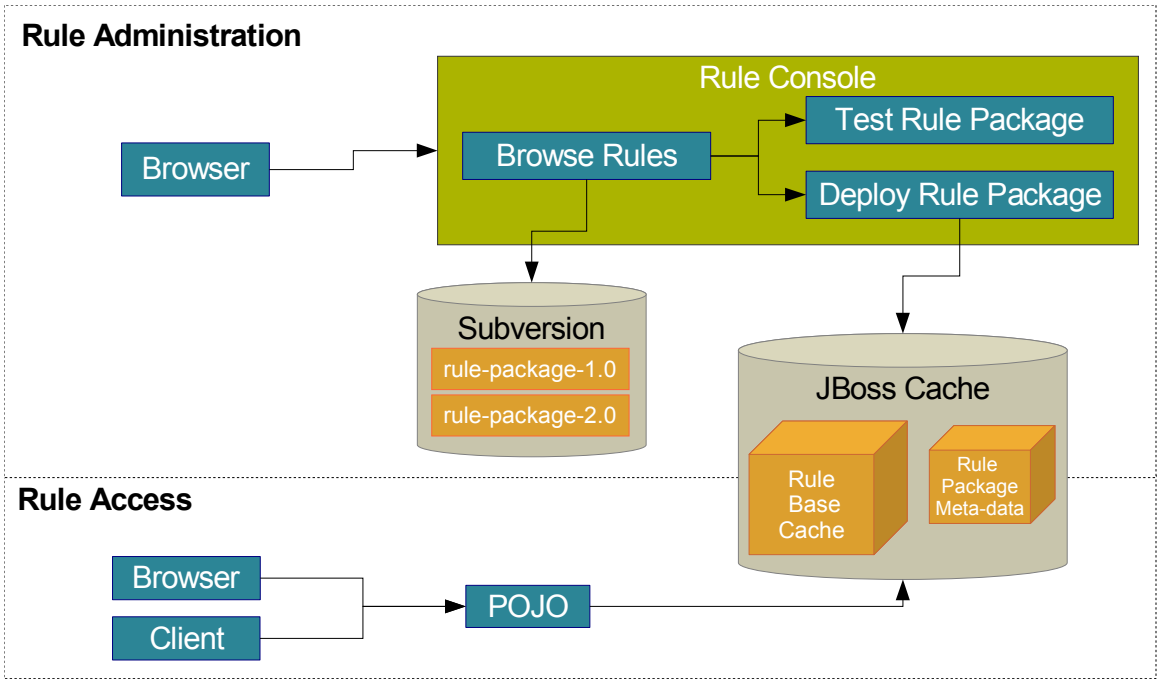
- We don't want to re-write SVN, CVS or any other version control system. Let's just leverage what's freely available and known to work.
- Spend a lot of time developing features that fall outside of the 80/20 Rule. Remember, we're attempting to solve the “Problem“ line items and nothing else. So, while rule re-use and rule-level version support might be “nice to haves”, they are by no means inhibiting our adoption of JBoss Rules.
- A one-size-fits all solution. This solution mandates JBoss Application Server and Subversion for process/rule version control. This solution ignores WebSphere, Weblogic and standalone Tomcat deployments.

Process Server Demo Authoring and Version Control

The diagram below denotes the business rule and process authoring and version control. The process server leverages existing tools for rule/process authoring and source control. JBoss IDE 2.0 contains the BPM and Rules Engine plugins needed to author/debug business rules and processes. Subclipse is available from http://subclipse.tigris.org/ and should be used as the source control client.

Rules	BPM
	
Two different streams of source control are leveraged. The “trunk” repository path is for the active development stream. The “tags” repository path is for grouping a package of rules, called a “rule-package”, that should be deployed as a single group. The “tags” repository will likely have file versions that differ from the “trunk” DRL versions. By deploying tagged versions, the Rule Administrator can easily switch out sets of business rules.	BPM uses the same “trunk” and “tags” scheme as Rules, but there are a couple subtle differences. Business processes are defined in XML files, not DRL files. And, while business processes can also be grouped for browsing, but they are not deployed one at a time. The process-list construct is is a browse only construct.

Rule Engine Administration and Access



Rule Administration

For rule administration, a single page console is leveraged. This console provides rule browsing from the Subversion “tags” repository location. Remember, a Subversion tag is just a group of DRL files. One or more rules can be selected and viewed via the console. If desired, rules can be tested prior to deployment. Deploying rules will create a cached RuleBase instance. JBoss Cache with persistence is used so that application server re-starts do not require manual rule re-deployments. If the DRL file(s) reference a DSL, it must be included in the rule-package. A rule-package can only reference one DSL definition.

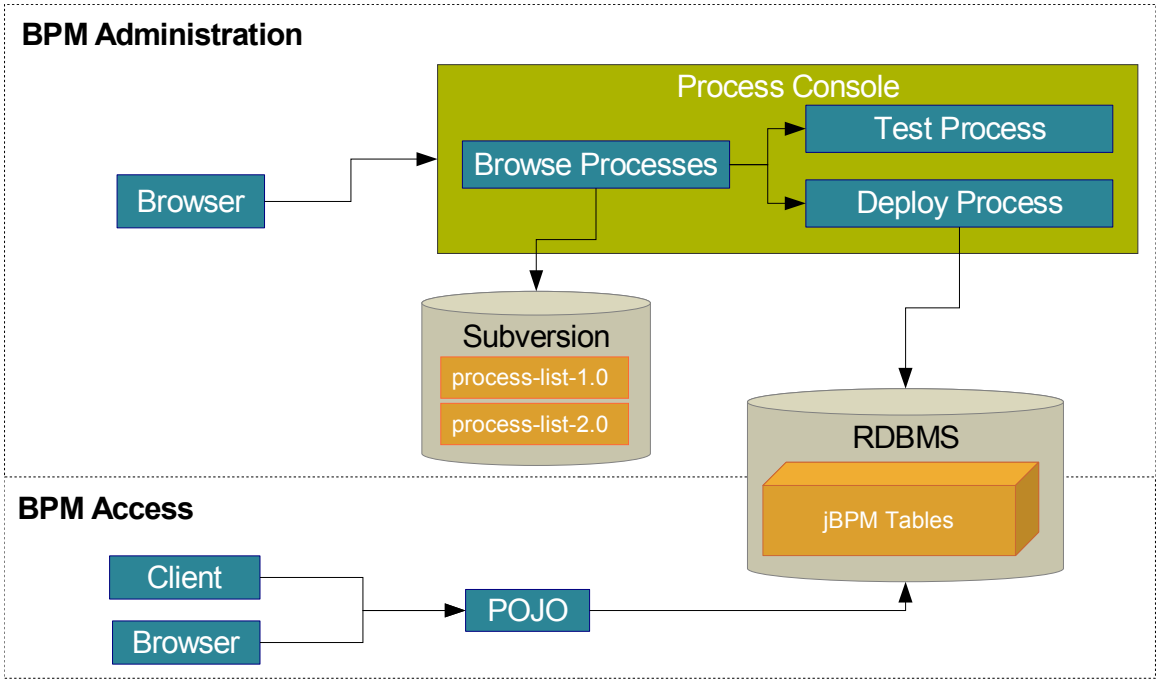
Rule Access

For rule access, the application simply calls a BPM action handler that gets a pre-loaded WorkingMemory instance from the RuleBase, asserts objects and fireAllRules().

```
public void execute(ExecutionContext ctx) throws Exception {

        //Get working memory from rules engine interface
        RulesInterface engine = new RulesEngine();
        WorkingMemory wm = engine.getWorkingMemory("online-quote");

        //get POJOs that are stored in this process instance
        //good for long running processes b/c this data lives in a database
        MyContextStore ctxStore = (MyContextStore) ctx.getVariable("context-objects");

        //assert objects into working memory
        wm.assertObject(ctxStore.getCarDetailBean());
        wm.assertObject(ctxStore.getMotorcycleDetailBean());
        wm.assertObject(ctxStore.getCoverageDetailBean());

        //fire off all rules. This might trigger changes in asserted objects
        wm.fireAllRules();
}
```

For all intensive purposes, the application should only require this level of interaction with the rules engine.

BPM Administration and Access



BPM Administration
For administration, a single page console is leveraged, just like rules management. This console provides process browsing from the Subversion “tags” repository location. Remember, a Subversion tag is just a group of process files. One process can be selected and viewed via the console. If desired, a process can be tested prior to deployment. Deploying a process will trigger inserts/updates to the jBPM tables. And, jBPM will handle process instance version control. In-flight process instances will complete with the same process definition that they started with even if a new process definition is deployed before existing processes have a chance to complete.

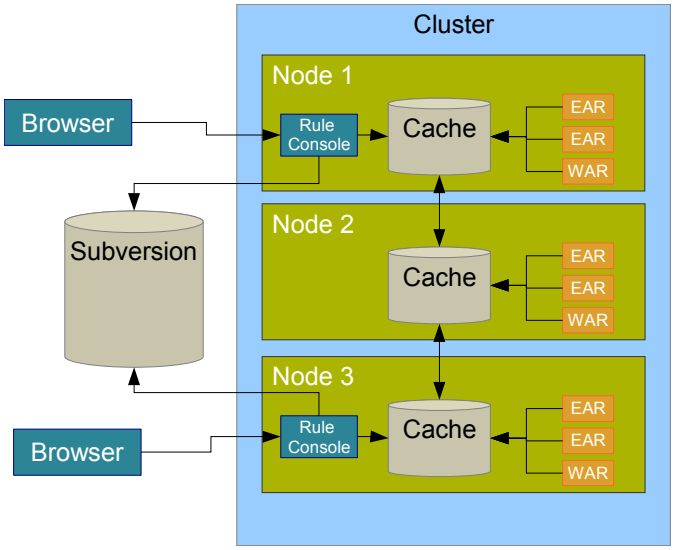
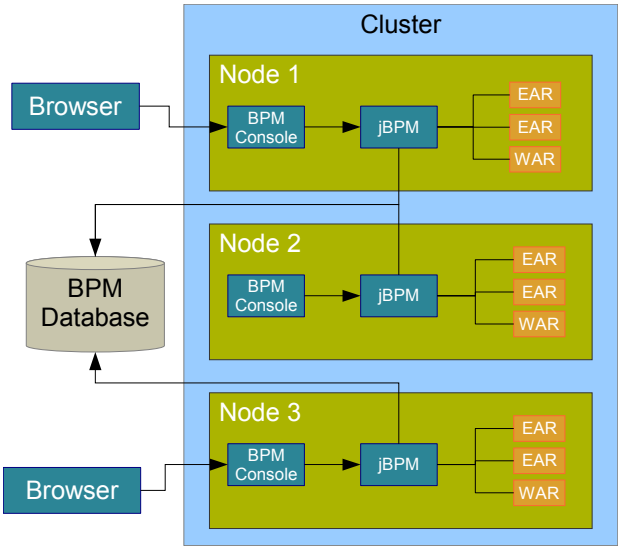
BPM Access

For BPM access, a POJO method will typically have logic that connects to jBPM, gets a process instance by ID, stores variables for that process instance (good in situations where processes are long lived), signals BPM to advance the process to the next node (might trigger BPM actions like the rules engine access listed above), then exits.

```
public void doSomethingWithBpm()
{
	JbpmContext jbpmContext = null;

	try{
		//connect to the BPM engine
		jbpmContext = JbpmConfiguration.getInstance().createJbpmContext();

		//get a handle to the process instance by passing the unique process ID
		ProcessInstance process = jbpmContext.loadProcessInstance(this.processId);

		//set a variable to the BPM database, so that it can be retrieved later
		process.getContextInstance().setVariable("context-objects", ctxStore);

		//trigger the process instance to advance to the next step
		//which may trigger BPM actions, like the rules example above
		process.getRootToken().signal();

		//save the state of this process instance
		jbpmContext.save(process);
	}

	catch (Exception e) {
		printStackTrace(e);
	}

	finally{
		//close the connection to BPM
		jbpmContext.close();
	}
}
```

Multi-Server Deployments

Both JBoss Rules and jBPM support clustered deployments without any additional configuration. For the rules engine. Here's what clustered deployments look like in the process server:

Rules	BPM
	
Multiple servers are supported by leveraging JBoss Cache, a distributed/transactional cache. Any changes to the deployed rules will be synchronized across all servers in a cluster at deploy time.	All processes are stored and managed in a database, so clustered JBoss application server instances should have the same datasource configuration.

Implementation Instructions

Okay, this is where the rubber meets the road. This implementation will require an HTTP accessible Subversion repository and the JBoss Process Server.

Configure Subversion with HTTP Access

An HTTP accessible Subversion repository is needed. The recommended approach is to:

1. Install Subversion 1.4
2. Install Apache 2.0.x
3. Add the *mod_dav.so* and *mod_dav_svn.so* libraries to *$APACHE_HOME/modules*
4. Add the appropriate entries to *$APACHE_HOME/conf/http.conf*, so clients can access Subversion via HTTP.

```
#Subversion modules
LoadModule dav_module modules/mod_dav.so
LoadModule dav_svn_module modules/mod_dav_svn.so


#Subversion configuration
<Location /svn>
       DAV svn
       SVNParentPath C:\<your-svn-path>
</Location>
```

5. Create a “trunk” and “tags” folder in Subversion. “trunk” is the development stream and “tags” represent the labeled rule/process packages that the process server console will use.

More info on how to install Subversion with HTTP support can be found at: http://svnbook.red-bean.com/nightly/en/svn-book.html.

JBoss Process Server Demo

The JBoss Process server demo can be used to:

1. Create a JBoss Application Server configuration that includes JBoss Cache and is jBPM + Rules friendly.
2. Deploy the jboss-process-server.sar, which includes all jBPM, Rules and administration console binaries.
3. Deploy an test Subversion repository that works with the example application (insurance-quote-web).
4. Deploy/run a test web application (insurance-quote-web.war) that provides a working example of how jBPM and JBoss Rules can be used.

Install Instructions

1. Make sure you have ant 1.6 or higher installed on your system
2. Download/unzip the latest GA version of JBoss Application Server (4.0.5)
3. Download/unzip the JBoss Process Server bundle from: http://wiki.jboss.org/wiki/attach?page=JBossProcessServerGuide/jboss-process-server-1.0.1.zip
4. cd to the $JBOSS_PROCESS_SERVER
5. edit *build.properties* (make sure *jboss.home* and *subversion.home* are correct)
6. Open a command shell and type one of the following options:
   - `setup default` (jbpm/drools as a shared service – non EJB3)
   - `setup default-ejb3` (jbpm/drools as a shared service - EJB3)
   - `setup default-with-demo` (jbpm/drools as a shared service with insurance demo and subversion based process admin console - non EJB3)
   - `setup default-ejb3-with-demo` (jbpm/drools as a shared service with insurance demo and subversion based process admin console - EJB3)
7. cd to $JBOSS_HOME/bin
8. Start the server with the process server configuration: *run.sh -c process*
9. [optional] Open the jmx-console and make sure the *rulesSvnURL* and *bpmSvnURL* point your Subversion “tags” folder: http://localhost:8080/jmx-console/HtmlAdaptor?action=inspectMBean&name=SvnAdmin%3Aservice%3DSvnAdminConsole%2Ctype%3DXMBean
10. [optional] The process server web console can be accesses at: *http://localhost:8080/process-admin-console*
11. [optional] Test and Deploy DRL files – The business rules (An example is included for you)
12. [optional] Test and Deploy Process Archives – The business processes
13. [optional] Go to the example application URL and request an online insurance quote: http://localhost:8080/insurance-quote-web/index.faces

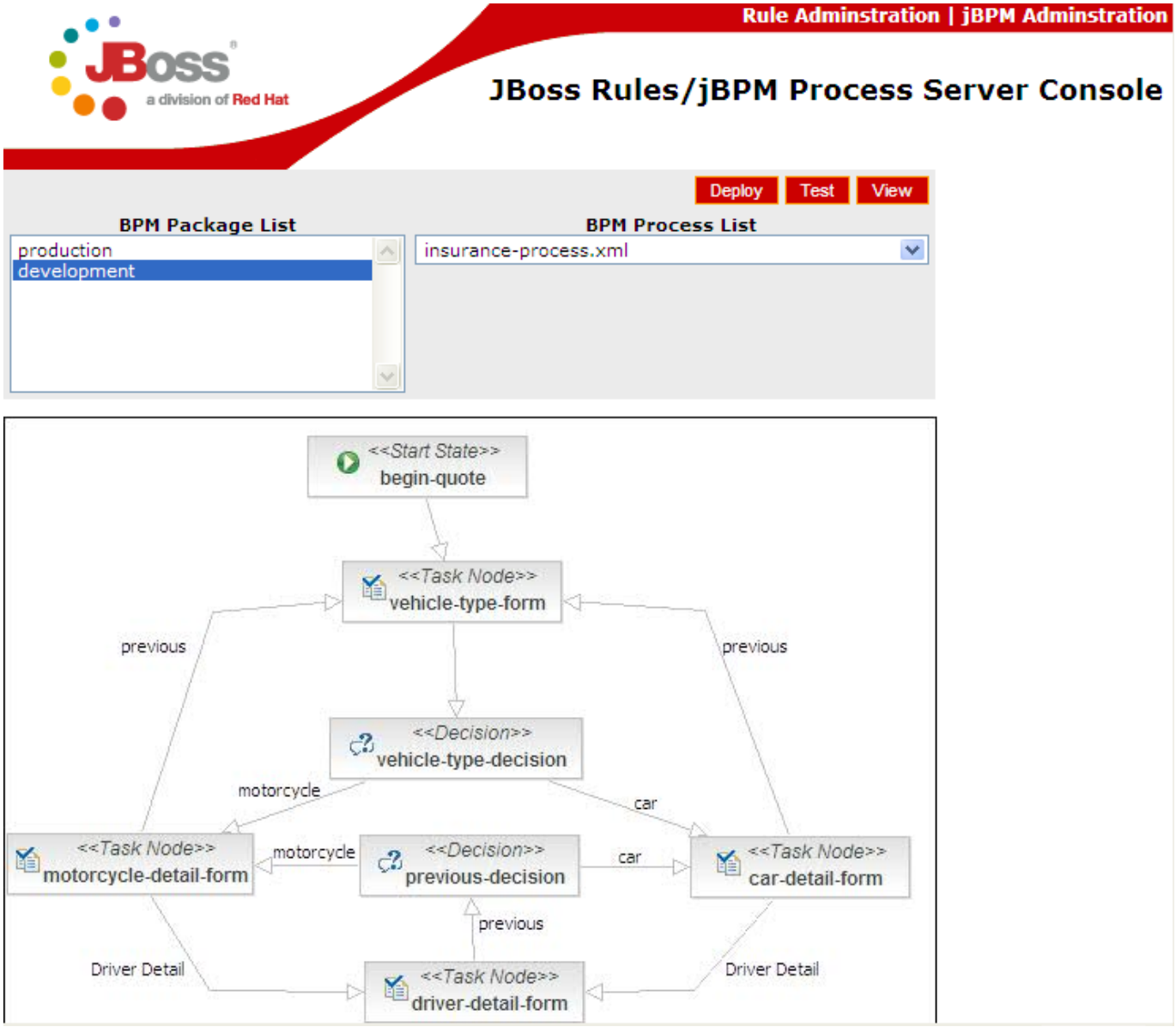
Rule Administration Screens

Good-Car-Driver.drl

```
package com.regressive.quote;

import com.regressive.model.*;

expander insurance.dsl

rule "Very Good Driver Discount"
        when
                Female driver older than "25";
        then
                Set insurance quote base price : "300";
                Add quote criteria label "Steller Driver Discount" and "For a female
        end

rule "OK Driver Discount"
        when
           Male driver older than "25";
        then
                Set insurance quote base price : "500";
                Add quote criteria label "OK Driver Discount" and "For a male that's
        end
```

This screen shot displays a DRL file that has been loaded into a cached RuleBase called “online-quote”. DRL files can also be tested to make sure that they compile before deployment. This console is actually interacting with the Subversion repository to display/load the DRL files into the RuleBase, so there's no way to deploy a DRL that's not checked into the version control system. The console also supports DSL(Domain Specific Language). If the DRL file references a DSL, that DSL must be included in the “rule-package”, otherwise DRL compilation will fail.

Process Administration Screen



This screen shot displays a jBPM process definition that can be deployed to the server. The process can also be tested to make prior to deployment. As with rule administration, the console is interacts with Subversion to display/load the available BPM processes. There's no way to deploy a business process that's not checked into the version control system.