The 6th

Israeli JBoss User Group

5.7.2007

JBPM

By : Zvika Markfeld, Consultant, Tikal

Israel JBUG

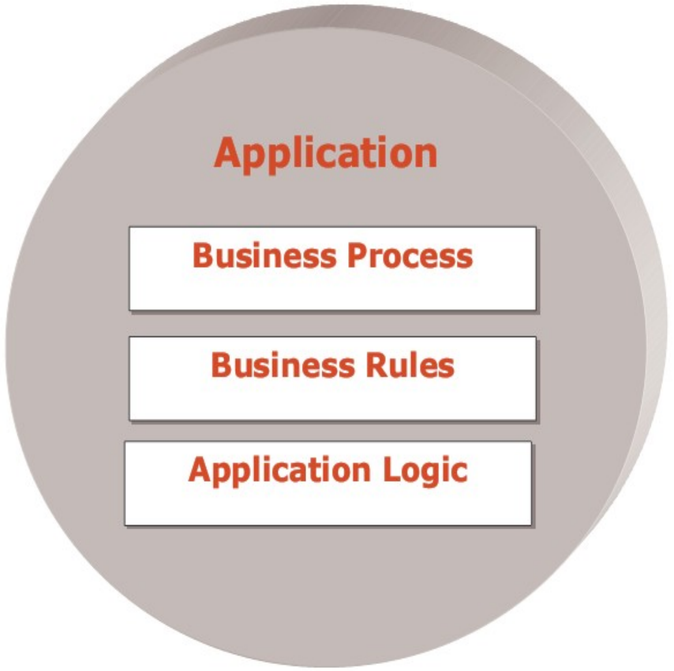
Introduction to Workflow Systems

Enterprise Application

- Software that exists in the context of an enterprise
- Consists of Application Logic
- Is part of the organization's Business Process
- Has to follow a number of organization's Business Rules

Traditional Approach

- Business Processes are embedded in an application
- Business Rules are also embedded in an application

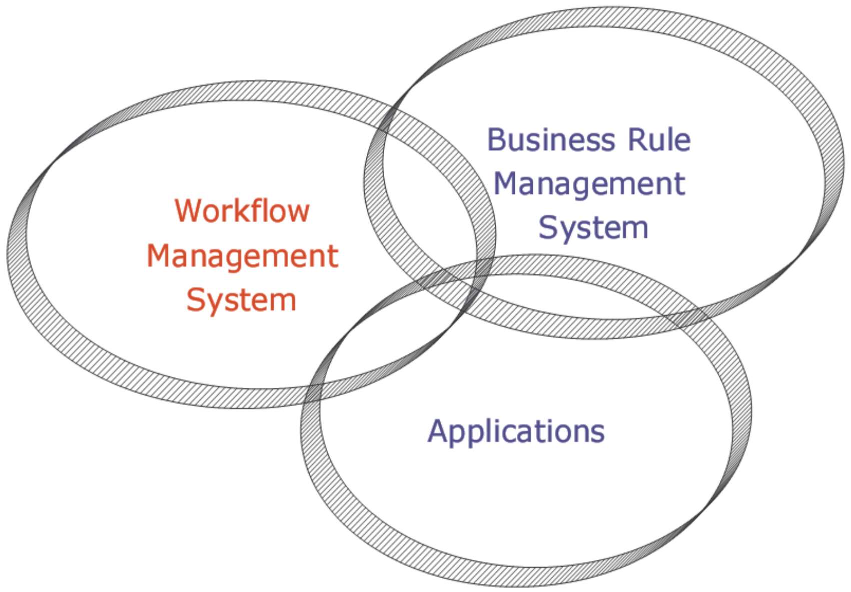


Traditional Approach – Cons

- ***Reuse:***
  Reuse of applications is low since business processes and rules are specific to organizations.
- ***Engineering:***
  Engineering of business processes is difficult as there is no independent representation of business processes.
- ***Communication:***
  Communication between business analysts and developers is difficult.
- ***Maintenance:***
  The cost of maintaining applications is high as changes to processes/rules require changes to the application

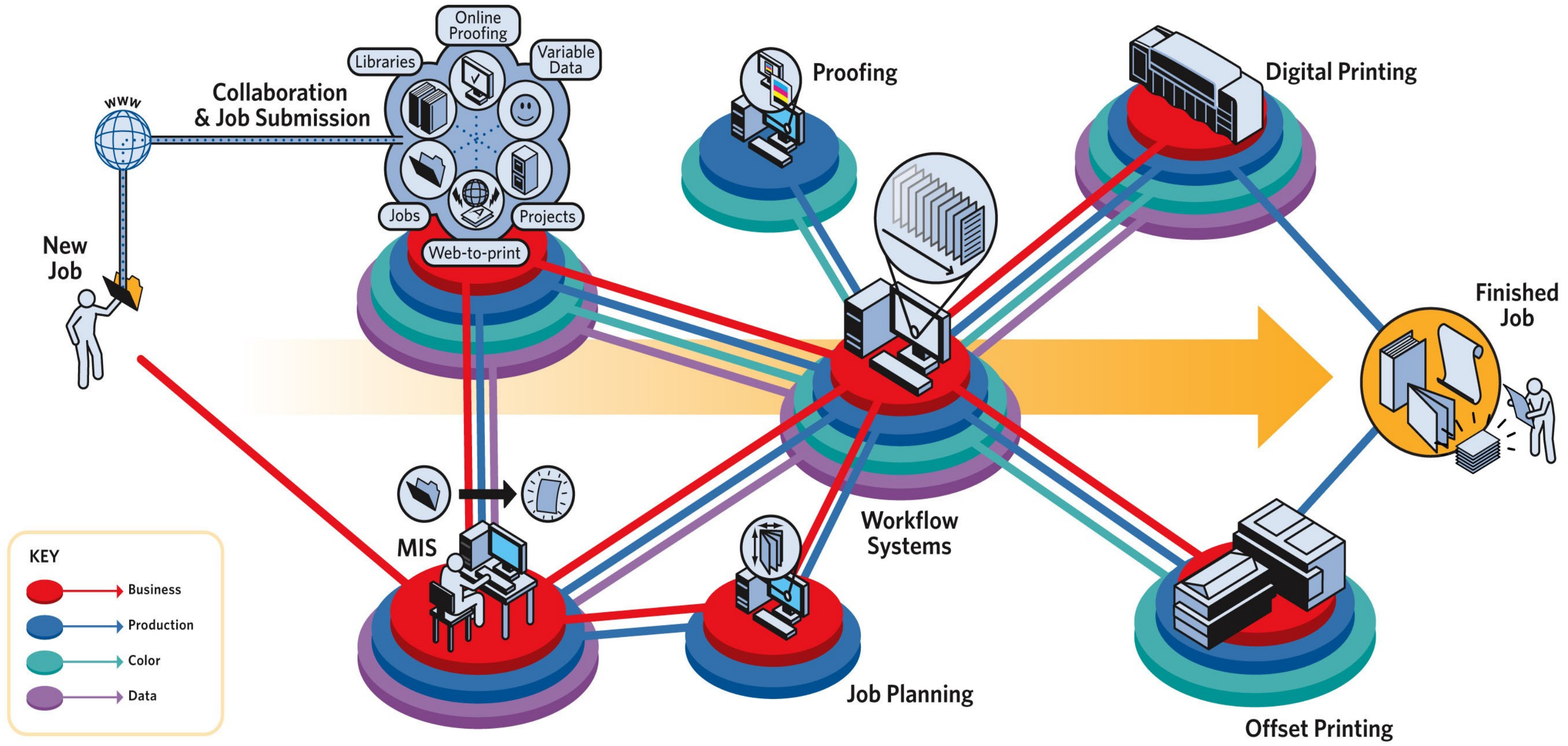
Solution

- Clear separation of concerns:
  - Business Processes are supported by an independent Business Process Management(BPM) System
  - Business Rules are maintained separately by a Rule Management System
  - Application Logic only implements specific steps of business processes or conditions associated with business rules



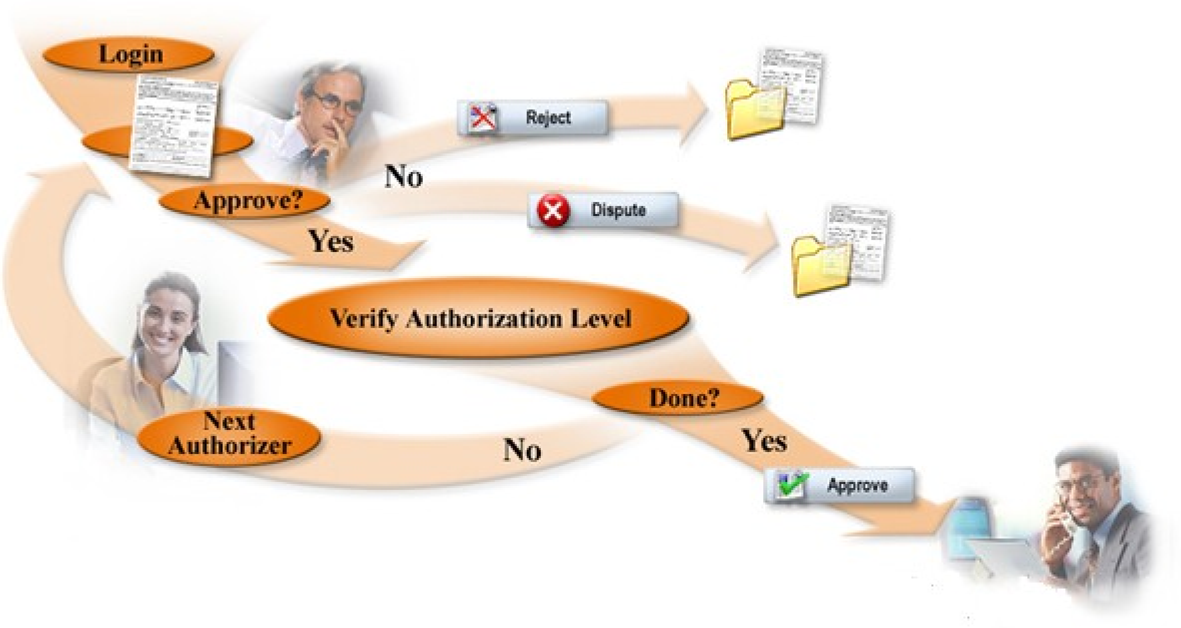
Terminology :: Workflow

- *The automation of procedures where documents,information or tasks are passed between participants according to a defined set of rules to achieve or contribute to an overall business goal.*



Terminology :: Business Process

- *A set of linked procedures or activities which collectively realize a business objective or policy goal,within the context of an organization structure defining functional roles and relationships*



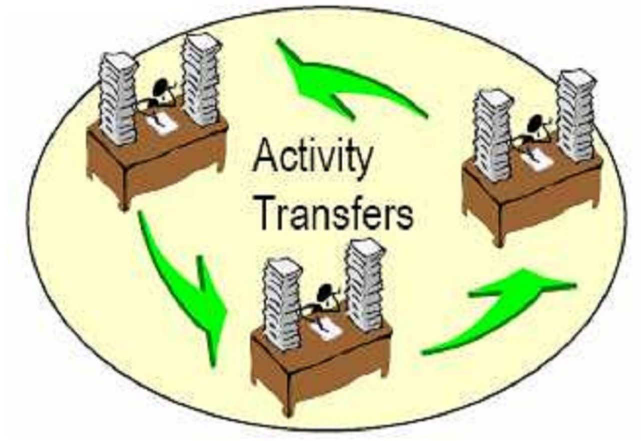
Workflow Management Systems(WMS)

1. Activities employed by enterprises to automate business processes in dynamic environments.
2. Software used to achieve such automation.

   For example, workflow engines

- Both offer a structured approach for designing and executing business actions / transactions
  - Preferably using automated processes
- Automation:
  - Interpreting process definitions
  - Interacting with workflow participants
  - Invoking IT tools and applications
- This approach includes concepts such as actions, tasks, and flows.

WMS History



- ***Motivations***
  - Office Automation
  - Document Management
  - “Paperless Office”
- ***1st Generation:***
  e-mail based, team collaboration via Lotus SmartSuite, MS-Office, etc.
- ***2nd generation:***
  Introduced notion of a process, executed by a dedicated system. However:
  - Too general
  - Detached from existing middleware
  - Lacked specific key functionality

Why Use WMS?

- Business-Level Benefits:
  - Business Analysts gain more control over actual process implementation
  - Improved efficiency – elimination of unnecessary steps.
  - Better process control – standard working methods, audit trails
  - Improved customer service – greater predictability in customer response levels.
  - Flexibility – easier re-design in line with changing business needs.
  - Business process improvement – streamlining and simplification.

Why Use WMS

- Software-Level Benefits
  - No need for translation between analyst & developer: reduced development risk
  - Centralized implementation - business processes may change without requiring major changes in the software application
  - RAD - composition of processes leads to faster and better maintainable development
  - **The implementation is no longer a fuzzy combination of software pieces scattered over various systems**

Workflow Solutions

- There are many commercial workflow solutions in the market, implementing the Workflow Reference model
  - COSA
  - FLOWer
  - Domino Workflow
  - Eastman
  - Visual Workflow
  - Forte Conductor
  - Meteor
  - Mobile
  - MQSeries
  - Staffware
  - Verve Workflow
  - I-Flow
  - InConcert
  - Changegine
  - SAP R/3 Workflow

Workflow Analysis

- ***WHO?***
  - Who is involved in the flow of the business process? What roles do they play?
  - Organizations? applications? employees? webservices? other workflows?
- ***WHAT?***
  - What should participants do and how?
  - Make decisions? approve others' decisions? perform transactions? create documents? track inventory? call external entities for information? transfer information to other participants?
- ***WHEN?***
  - How do they know when to start and when to finish?
  - In what order? sequentially or in parallel? do tasks time out? do they retry on fail?

BPM – Practical Break-Down

- BPM is embodied in three distinct practices:
  - *Process Design:*
    Modeling/designing existing and new processes
  - *Process Execution:*
    The execution of an automated sequence of related events involving software processes and/or human activities
  - *Process Monitoring:*
    Observance and auditing of the state of individual processes so that the statistics and performance of these processes can be recorded, reported, and optimized

BPM As a Intermediate Language

- BPM seeks to allow software engineers to ***share the same concepts*** and frameworks as business analysts;
- Consequentially, software vendors have attempted to create tools that will allow enterprises to capture, design, and optimize business processes
  - Graphical modeling tools
  - Domain-specific languages
  - Proprietary applications

JBoss jBPM

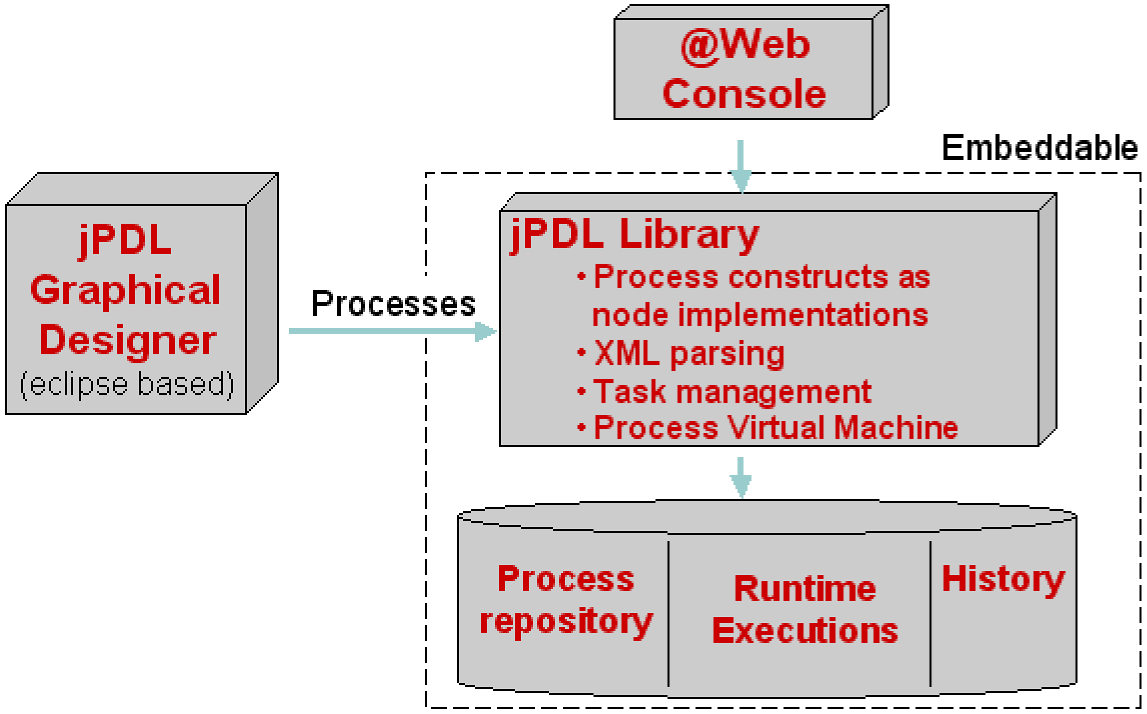
JBoss jBPM

- Flexible, extensible framework for process languages:
  - *jPDL:*
    A Java-based language with task management features
  - *BPEL(alpha stage):*
    Service orchestration language based on WSDL and XML
  - *Pageflow:*
    JBoss SEAM's simple graph based page-flow definition
- Allows developing automated business processes and workflows with standard orchestration
- Acts as an intermediary between analysts and developers for process definitions
- May be deployed as web/standalone application

JBoss jBPM Architecture

- JBoss jBPM is encapsulated within the following components:
  - *Process language:*
    The process definition language (jPDL) is based on Graphic Oriented Programming(GOP)
  - *Process engine:*
    Executes defined process actions, maintains process state, and logs process events.
  - *Process monitor:*
    This module tracks, audits, and reports the state of processes as they execute
  - *Interaction services:*
    These services expose "legacy applications" as functions or data to be used in process executions

JBoss jBPM Architecture



jBPM Process Definitions

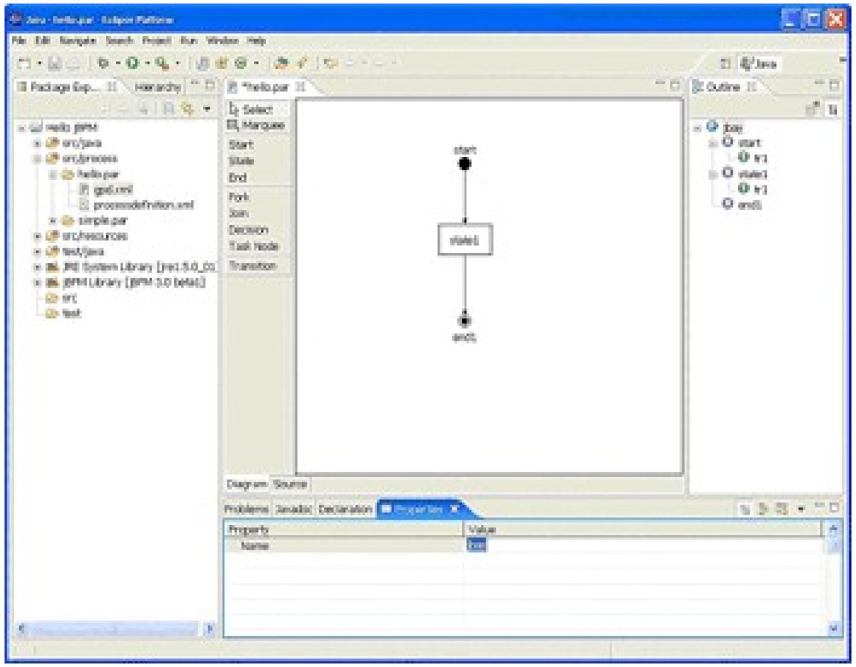
- Written in jPDL, GOP language based on a model of nodes, transitions, and actions
  - *Nodes:*
    Commands executed as they are encountered during the flow of a process definition.
  - *Transitions:*
    direct the flow of execution of a process definition
  - *Actions:*
    perform specific logic when a node or transition event occurs
- Packaged as process archives and passed to the jPDL process engine for execution.

The jPDL Process Engine

- Traverses a process graph, executes defined actions, maintains process state, and logs all process events. Composed of:
  - A request handler
  - A state manager
  - A log manager
  - A definition loader
  - An execution service

The jPDL Graphical Process Designer

- Tool for authoring business processes.
- Implemented as an Eclipse plug-in
- Includes support for business analysts as well as technical developers



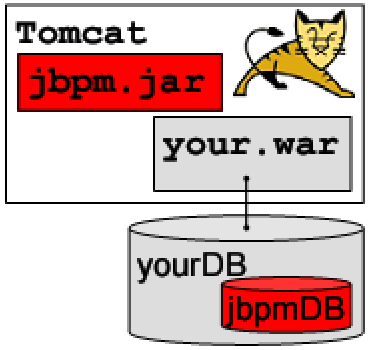
The jPDL Suite Contents

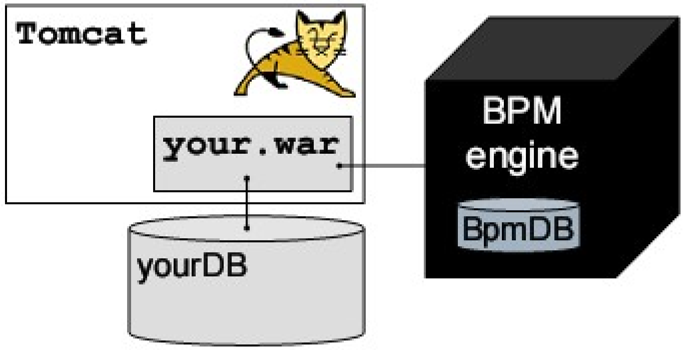
- Configuration files for a standard Java environment
- SQL scripts for DB creation and compatibility information
- jPDL graphical process designer
- Preconfigured JBoss application server, including:
  - jBPM web console, used by process participants as well as jBPM administrators
  - Job Executor, executing timers and asynchronous messages.
  - The jBPM tables, in the default hypersonic database
  - Example process, deployed into the jBPM database.
- Sources, examples, (java)docs, user-guide, ...

jBPM Deployment Models

- JBoss jBPM is a plain(J2SE) Java library for managing process definitions and executing process instances
- It can be accessed from any Java environment:
  - Web Application
  - Swing application
  - EJB
  - Web Service
- When packaged and exposed as a stateless session EJB, it can be deployed on a cluster, allowing scalability, fault tolerance, etc.

Embeddable vs. Blackbox BPM



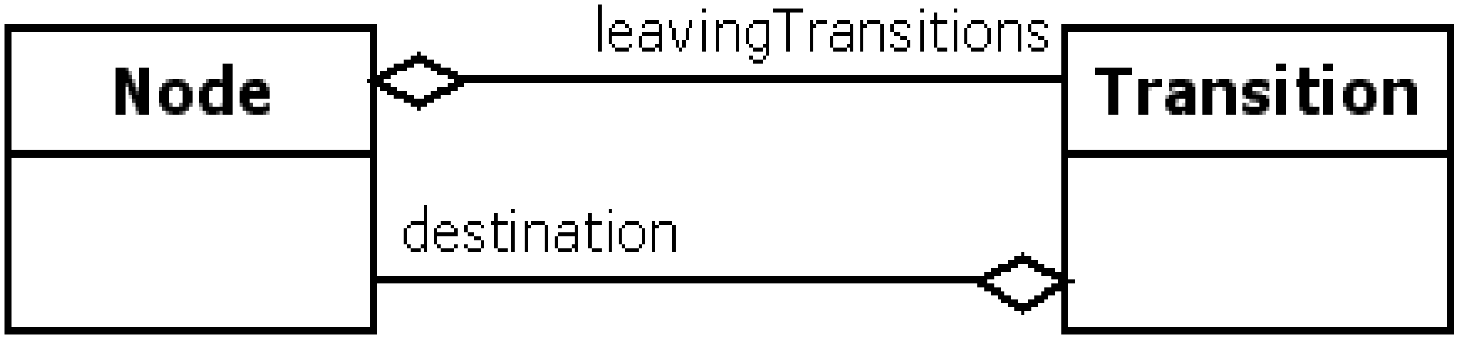


- Integrates into your software:
  - DB
  - Configuration
  - Tx
- Benefits from JEE Server:
  - JTA
  - MOM
  - Security
- TDD friendly
- Portable(Java/App' Server/DB)

- Traditional
- Productized
- Separate

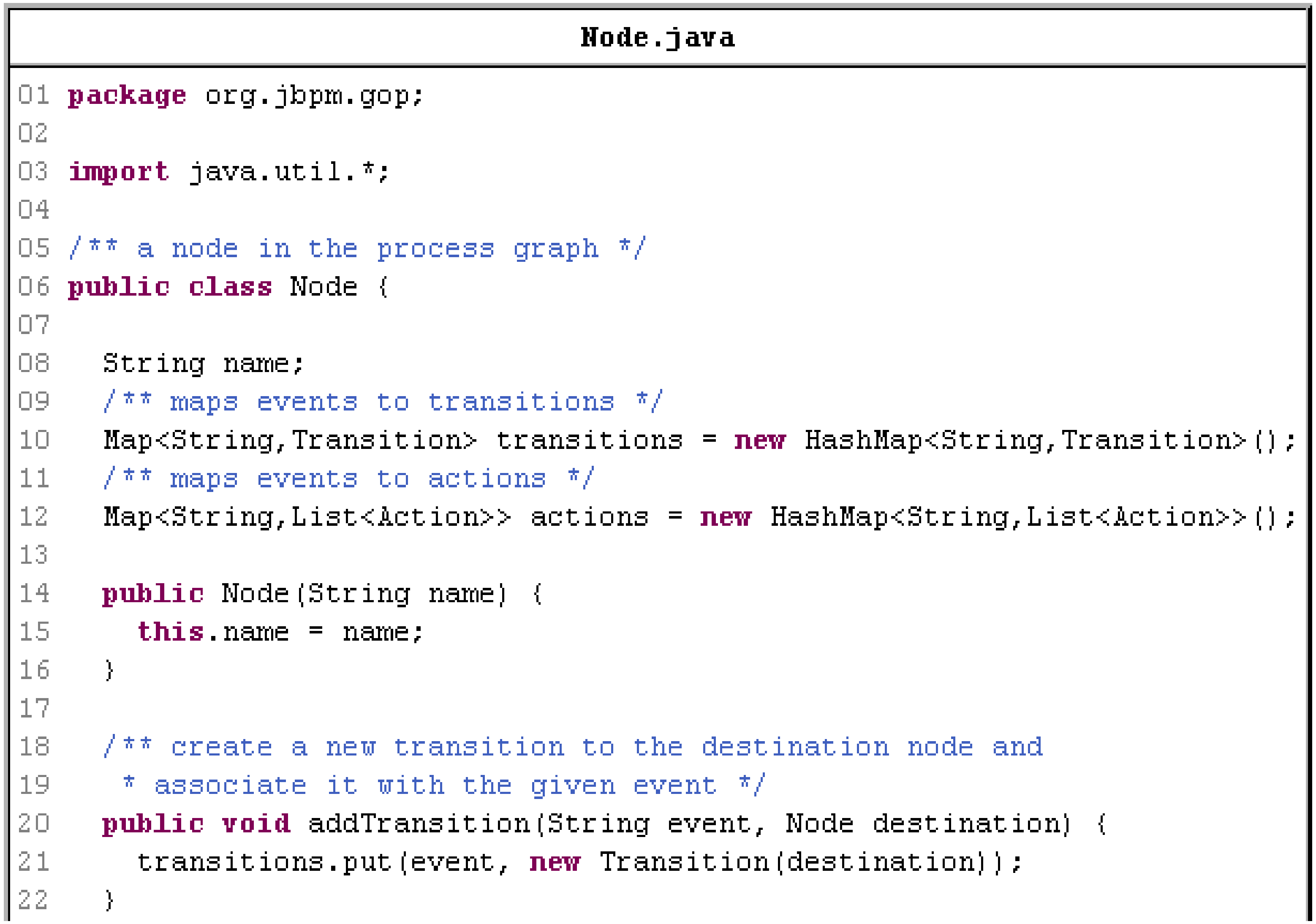
jPDL Constructs

Graph Structure



- The structure of the graph is represented by *Nodes* and *Transitions*
- A Node is a command and has an execute method. Subclasses of Node are supposed to implement some specific behavior
- A Transition has a direction so the nodes have leaving and arriving transitions.

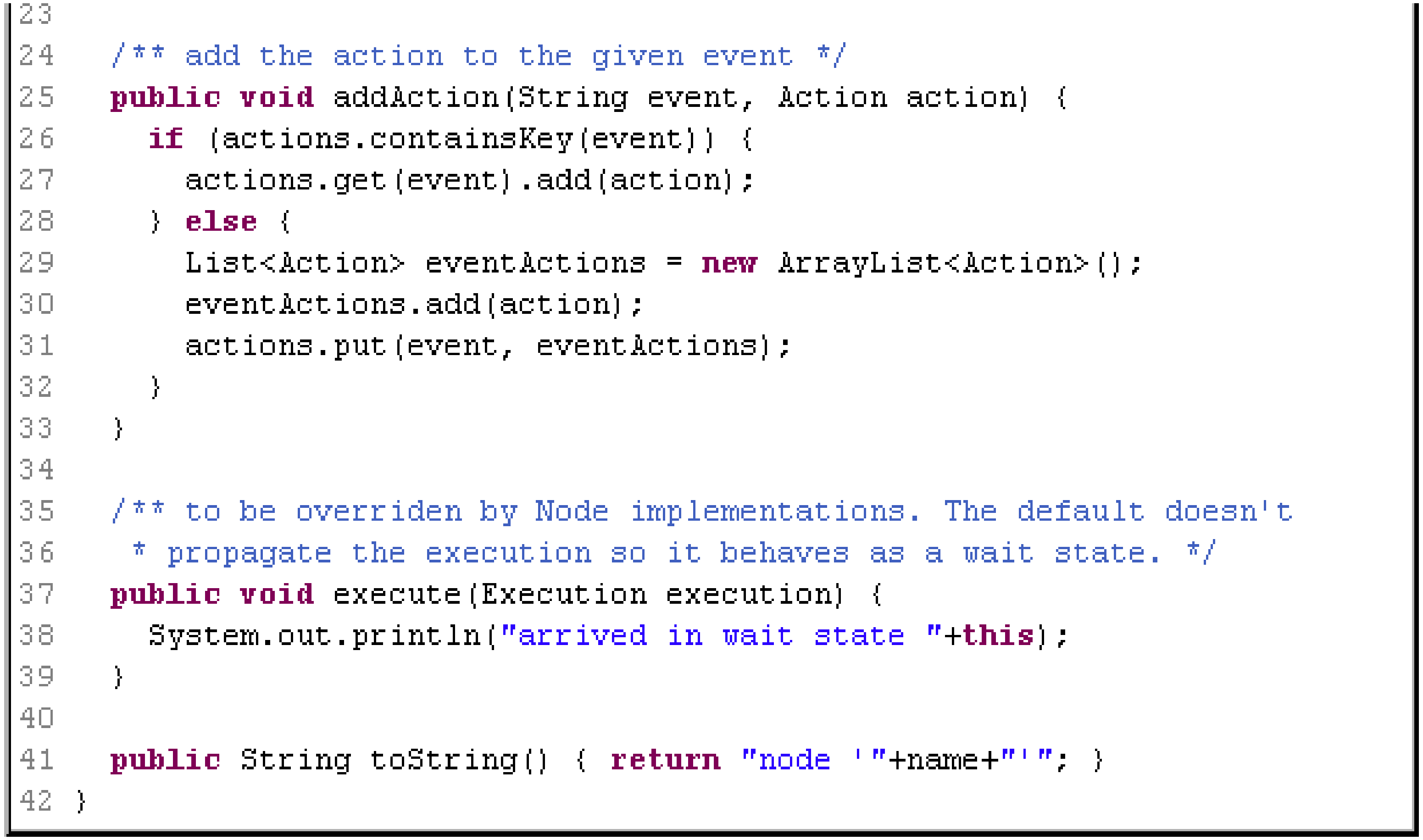
Node.java



Node.java

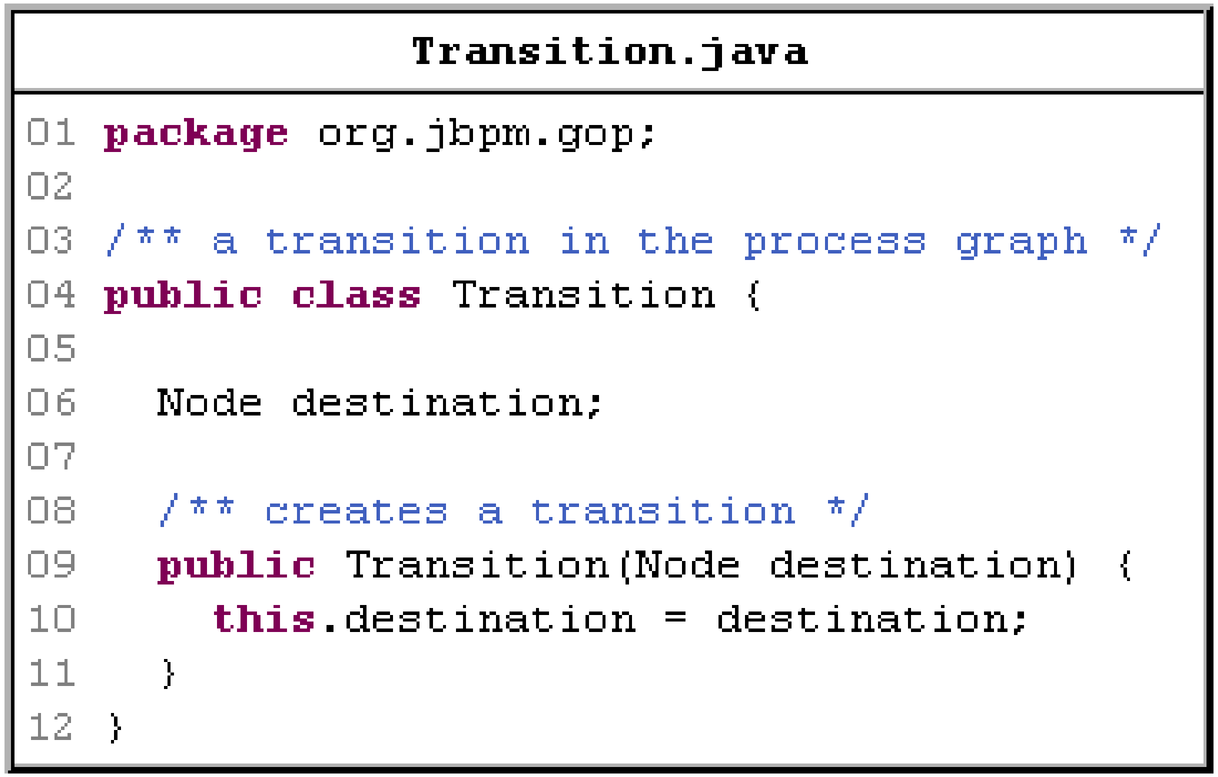
```
package org.jbpm.gop;

import java.util.*;

/** a node in the process graph */
public class Node {

  String name;
  /** maps events to transitions */
  Map<String,Transition> transitions = new HashMap<String,Transition>();
  /** maps events to actions */
  Map<String,List<Action>> actions = new HashMap<String,List<Action>>();

  public Node(String name) {
    this.name = name;
  }

  /** create a new transition to the destination node and
   * associate it with the given event */
  public void addTransition(String event, Node destination) {
    transitions.put(event, new Transition(destination));
  }
```

Node.java



```
  /** add the action to the given event */
  public void addAction(String event, Action action) {
    if (actions.containsKey(event)) {
      actions.get(event).add(action);
    } else {
      List<Action> eventActions = new ArrayList<Action>();
      eventActions.add(action);
      actions.put(event, eventActions);
    }
  }

  /** to be overriden by Node implementations. The default doesn't
   * propagate the execution so it behaves as a wait state. */
  public void execute(Execution execution) {
    System.out.println("arrived in wait state "+this);
  }

  public String toString() { return "node '"+name+"'"; }
}
```

Transition.java

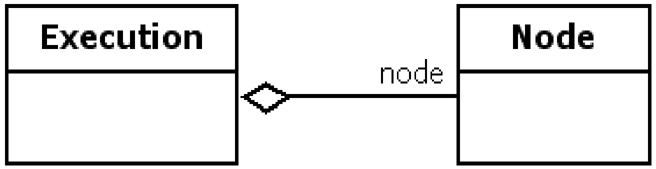


Transition.java

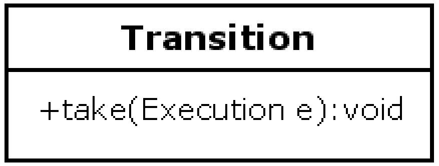
```
package org.jbpm.gop;

/** a transition in the process graph */
public class Transition {

  Node destination;

  /** creates a transition */
  public Transition(Node destination) {
    this.destination = destination;
  }
}
```

Execution(a.k.a. Token)

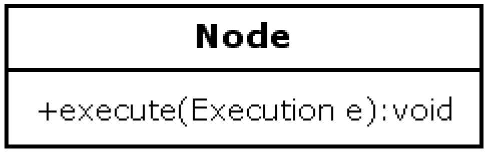
- An Execution has a reference to the current node.



- Transitions can pass the execution from a source node to a destination node with the method take().

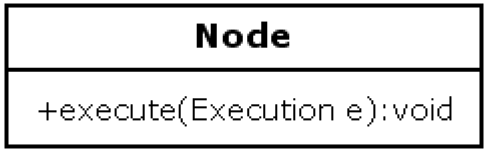


- When an execution arrives in a node, that node is executed.

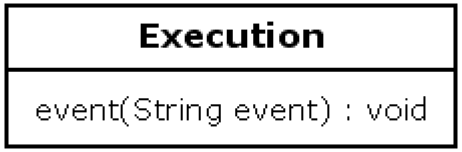


Execution

- The Node's is also responsible for propagating the execution over one of its leaving transitions



- Otherwise, it behaves as a wait state.
  - For example, start node
- An external event may then be passed to an execution via event() method



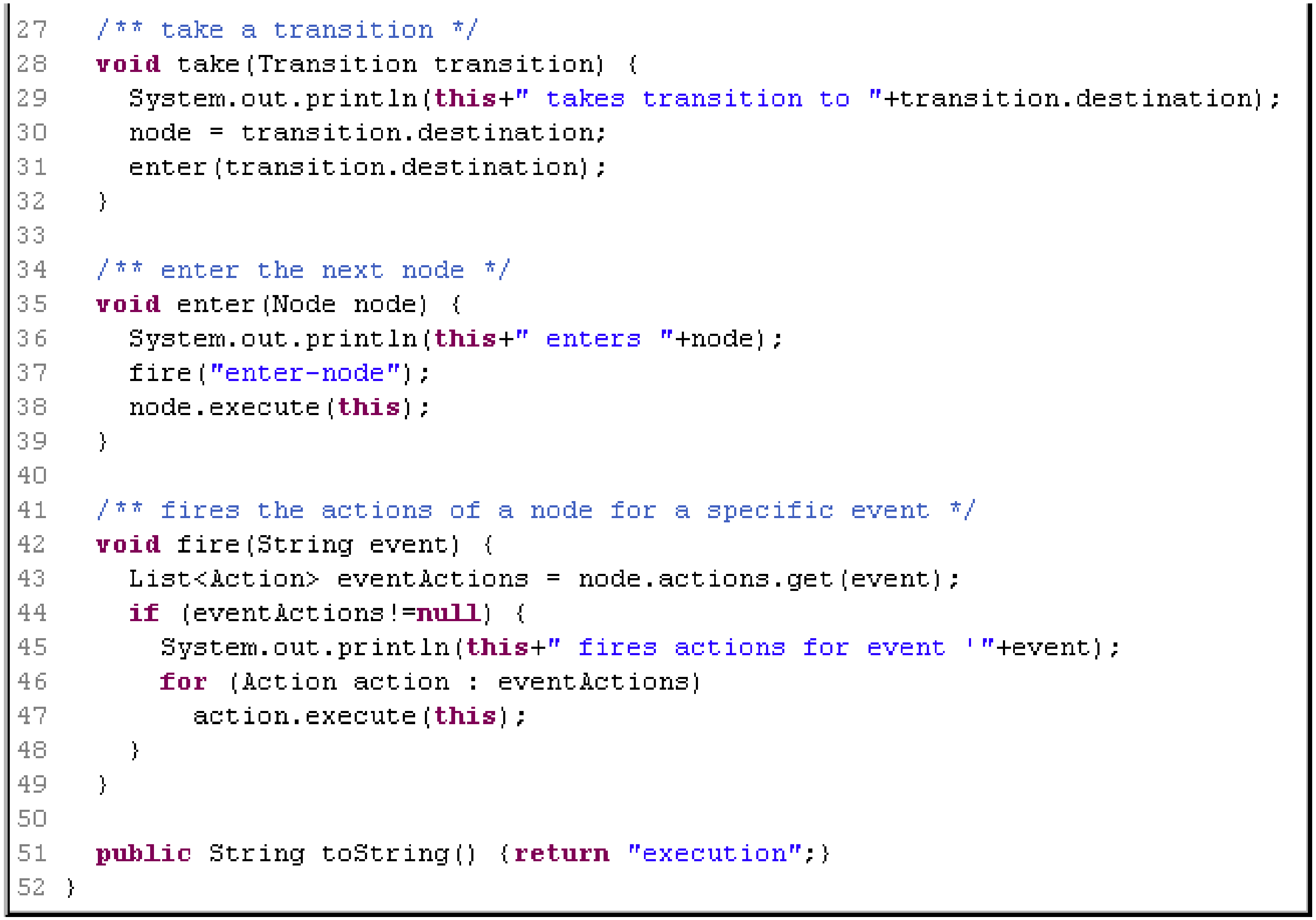
- If the event relates to a leaving transition, the execution takes that transition.
  - Continuing to propagate until next wait state

Execution.java

Execution.java

```java
package org.jbpm.gop;

import java.util.List;

/** one path of execution */
public class Execution {

  /** pointer to the current node */
  public Node node = null;

  /** an execution always starts in a given node */
  public Execution(Node node) {
    this.node = node;
  }

  /** executes the current node's actions and takes the event's transition */
  public void event(String event) {
    System.out.println(this+" received event '"+event+"' on "+node);
    fire(event);
    if (node.transitions.containsKey(event)) {
      System.out.println(this+" leaves "+node);
      fire("leave-node");
      take(node.transitions.get(event));
    }
  }

```

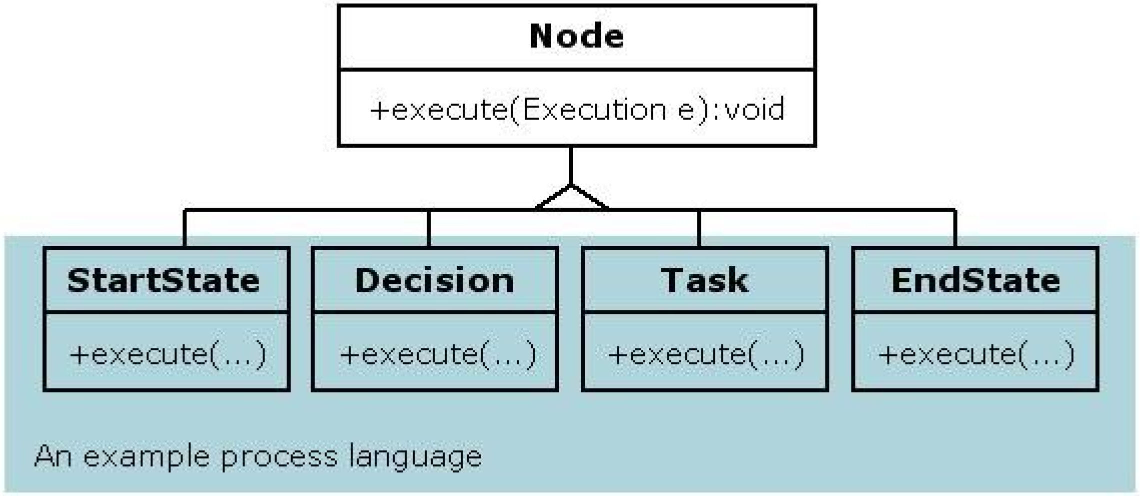
Execution.java



```
 /** take a transition */
 void take(Transition transition) {
  System.out.println(this+" takes transition to "+transition.destination);
  node = transition.destination;
  enter(transition.destination);
 }

 /** enter the next node */
 void enter(Node node) {
  System.out.println(this+" enters "+node);
  fire("enter-node");
  node.execute(this);
 }

 /** fires the actions of a node for a specific event */
 void fire(String event) {
  List<Action> eventActions = node.actions.get(event);
  if (eventActions!=null) {
   System.out.println(this+" fires actions for event '"+event);
   for (Action action : eventActions)
    action.execute(this);
  }
 }

 public String toString() {return "execution";}
}
```

A Process Language

- A process language is merely a set of Node implementations
- Each implementation corresponds to a process construct
- The exact behavior of the process construct is implemented by overriding the execute method



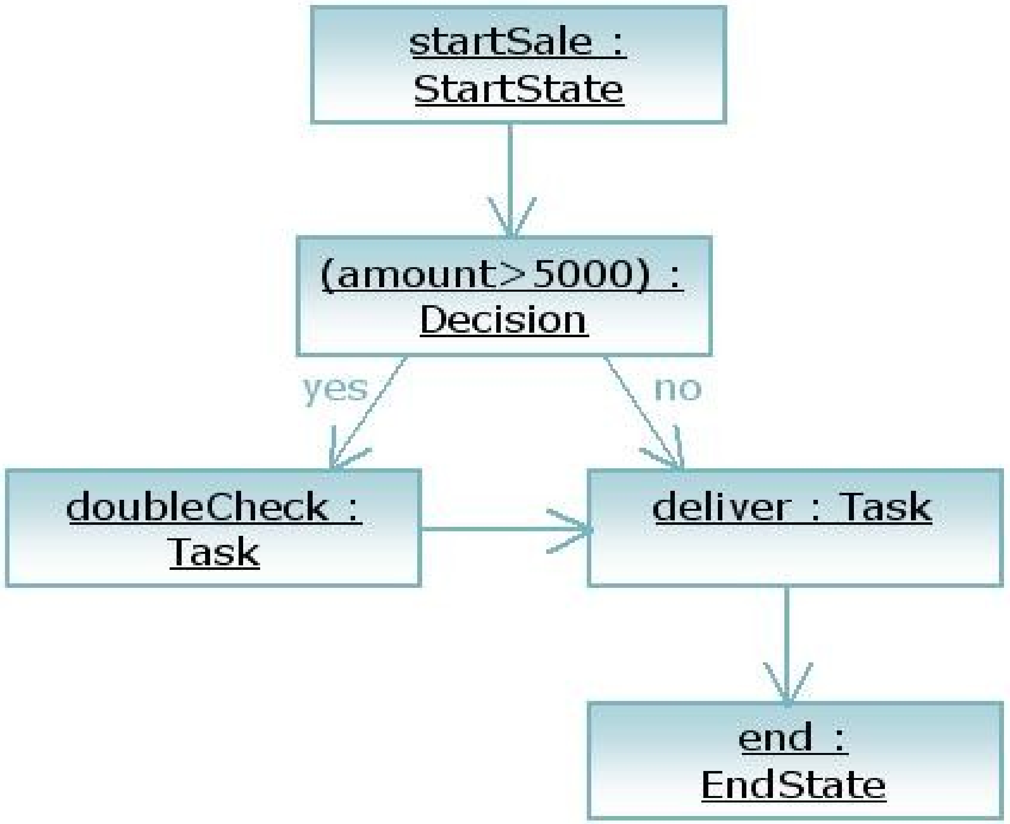
Israel JBUG

Node Types

- Task:
  - Represents task(s) to be performed by humans.
  - Creates task instances for participants and waits
- State:
  - Bare-bone wait state
  - Waiting for an external system
- Decision:
  - Node that takes decisions
  - Declarative (beanshell) or programatic
- Fork / Join:
  - Split and recap one path of execution into multiple ones
- Others...

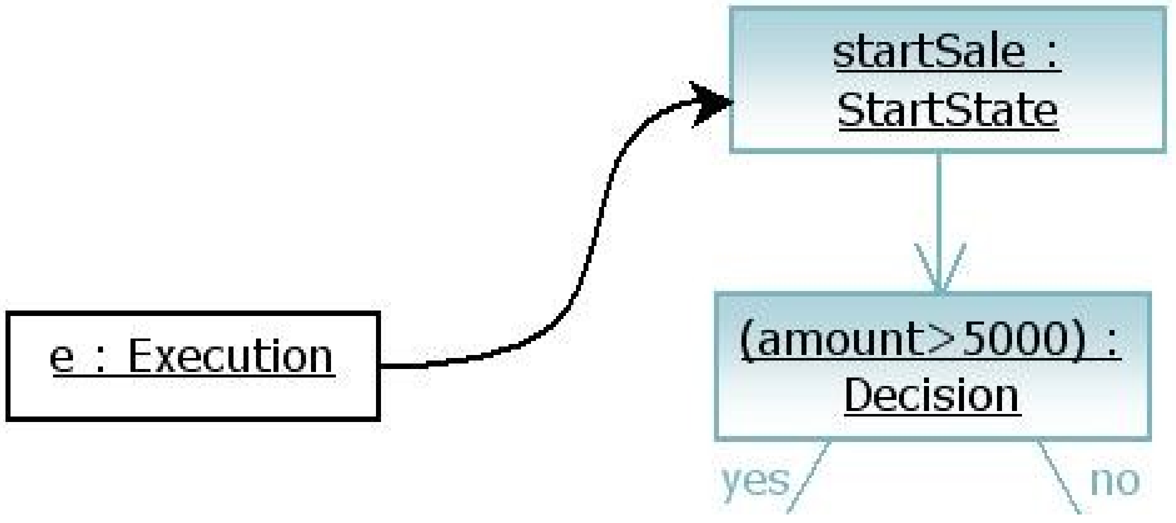
An Example Process Language

- Concrete node objects can now be used to create process graphs in our example process language

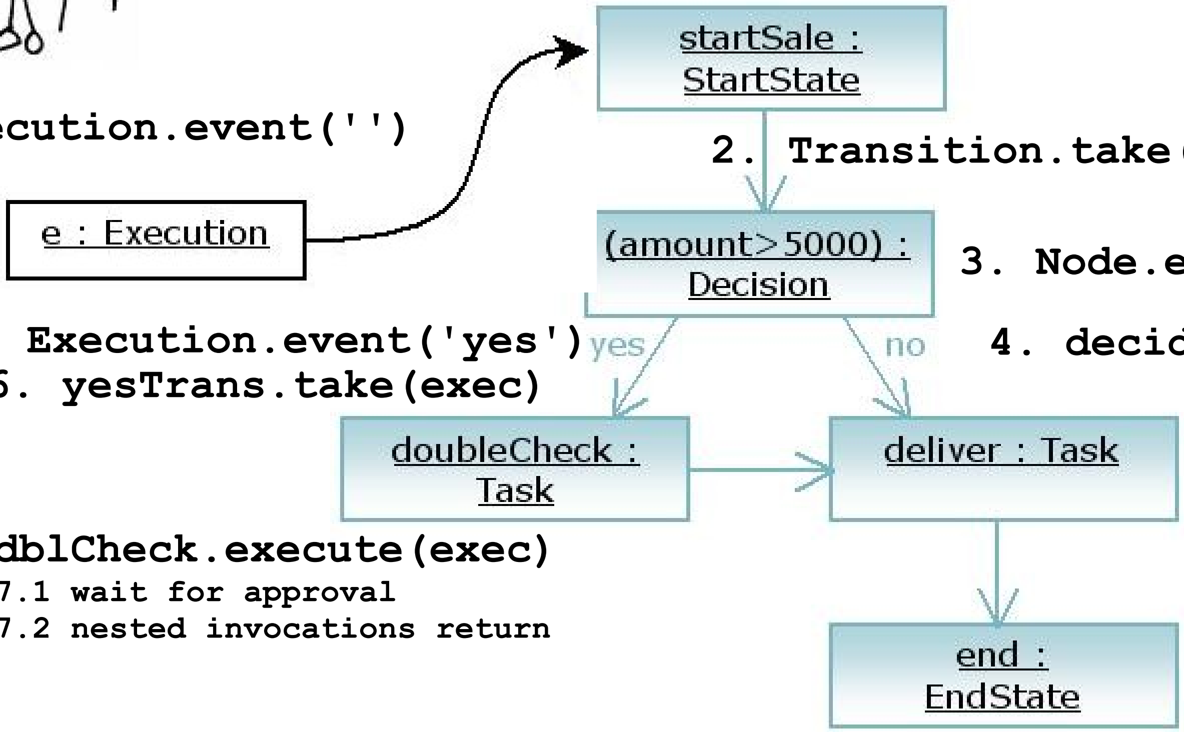


An Example Process

- When creating a new execution, we start by positioning the execution in the start node.
- Until the execution receives an event, it will remain in the start state.



Process Flow



1. Execution.event('')

2. Transition.take(exec)

3. Node.execute(exec)

4. deciding...

5. Execution.event('yes')

6. yesTrans.take(exec)

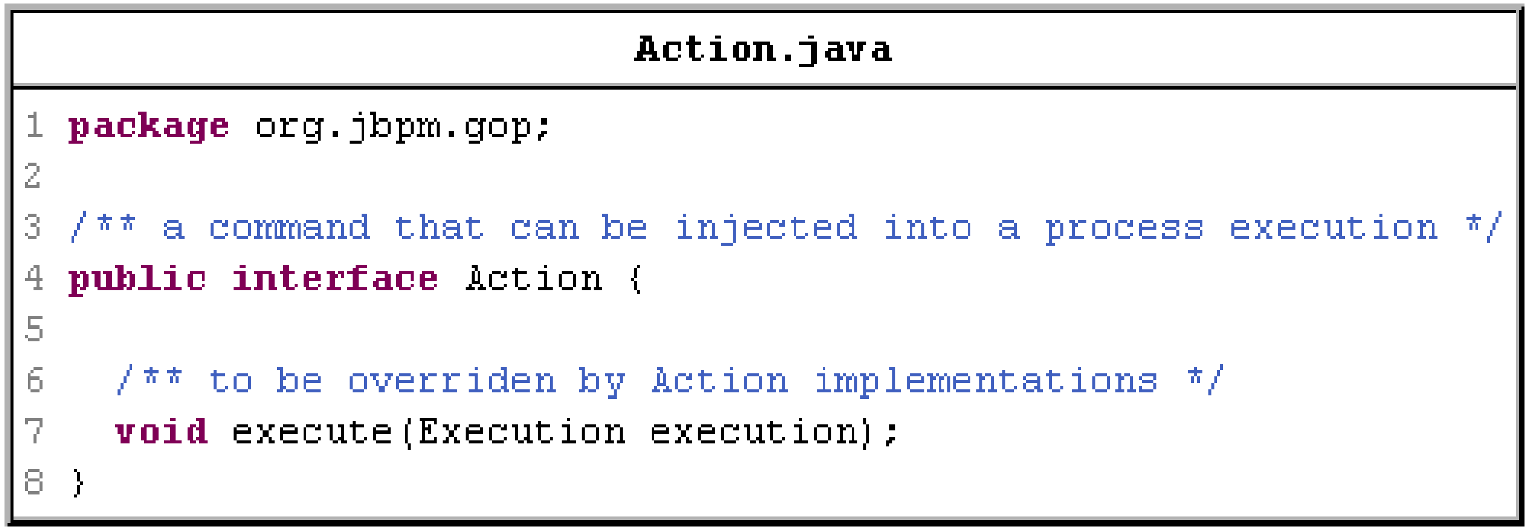
7. dblCheck.execute(exec)
   7.1 wait for approval
   7.2 nested invocations return

8. Waiting...

Actions

- A way to include the execution of programmatic logic without introducing new nodes
  - Business analyst is sole responsible for the graphical representation of the process
  - It is not acceptable that the developer would have to change the diagram just to accommodate technical detail
- This kind of logic is encapsulated within an ***Action***
- Actions can be associated with events
  - node-leave, node-enter, and all transition-bound events

Action.java

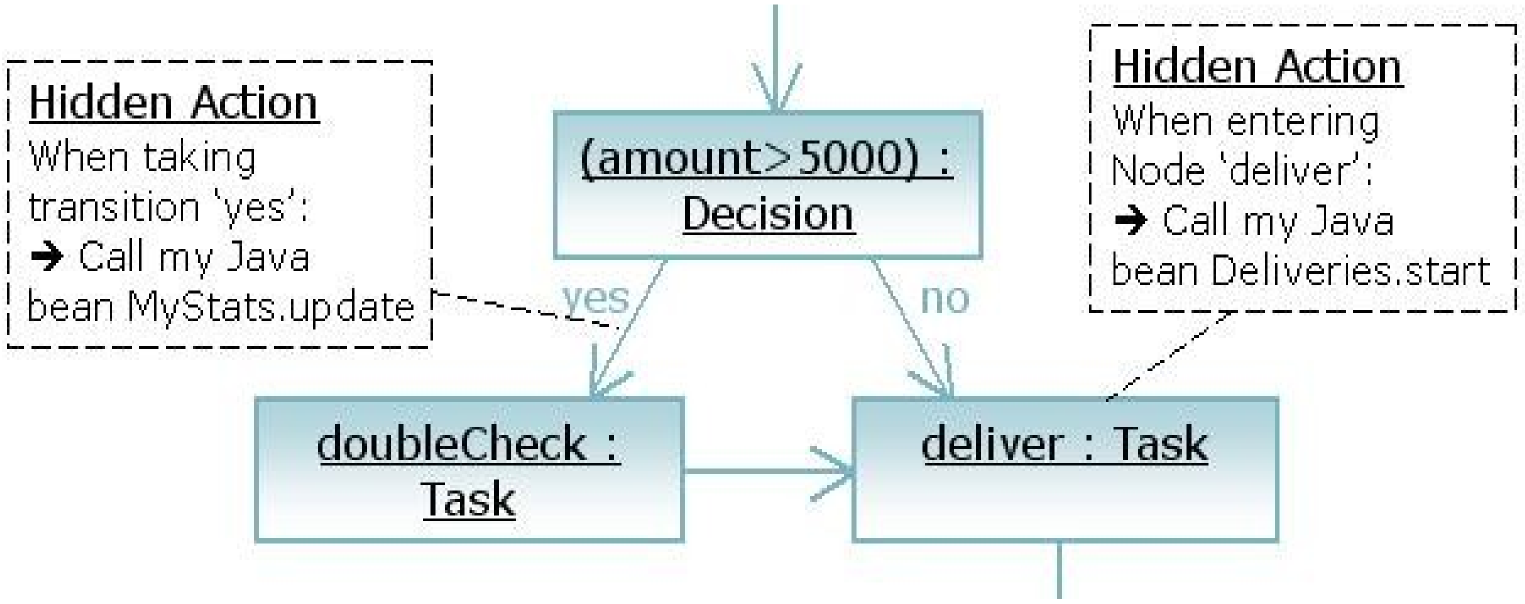


Action.java

```java
package org.jbpm.gop;

/** a command that can be injected into a process execution */
public interface Action {

  /** to be overriden by Action implementations */
  void execute(Execution execution);
}
```

Actions

- Back to the example...



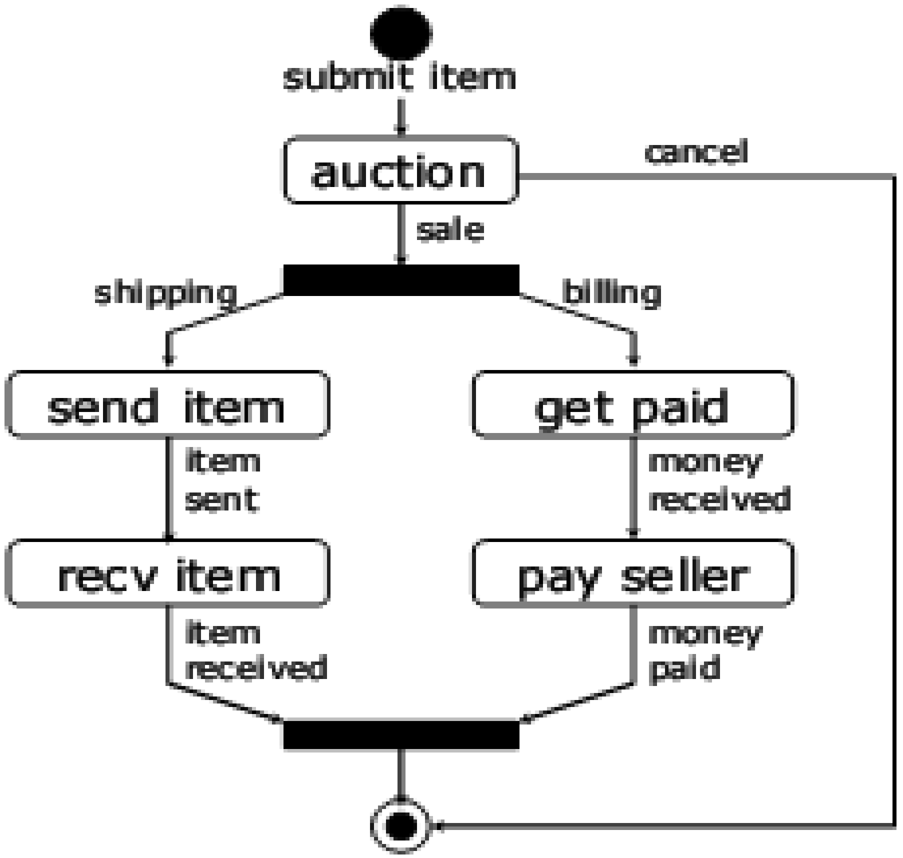
Other Constructs

- Process variables
  - Maintain the contextual data of a process execution
  - In an insurance claim process, the *claimed amount*, *approved amount* and *isPaid* might be process variables
- Concurrent executions
  - Think about a sale process, where the billing activities and shipping activities can be done ***in parallel***.
  - ***Process Execution*** *<1: N >* ***Execution Paths***
  - Paths can be modeled hierarchically

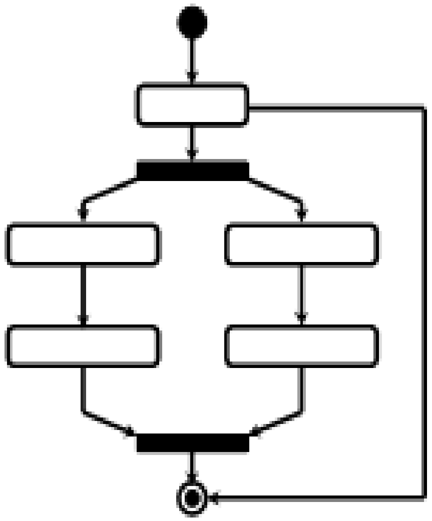
Concurrent Executions: The jBay Auction Process

- Users may submit items for auction
  - Duration
  - Reserved price
- When Auction finishes
  - A sale happens if a minimal price is met
  - Otherwise the auction is cancelled
- After a sale
  - Seller sends items, buyer receives
  - Buyer pays, seller gets paid
- Payment and shipping are done in parallel

Concurrent Executions: The jBay Auction Process



Concurrent Executions: The jBay Auction Process

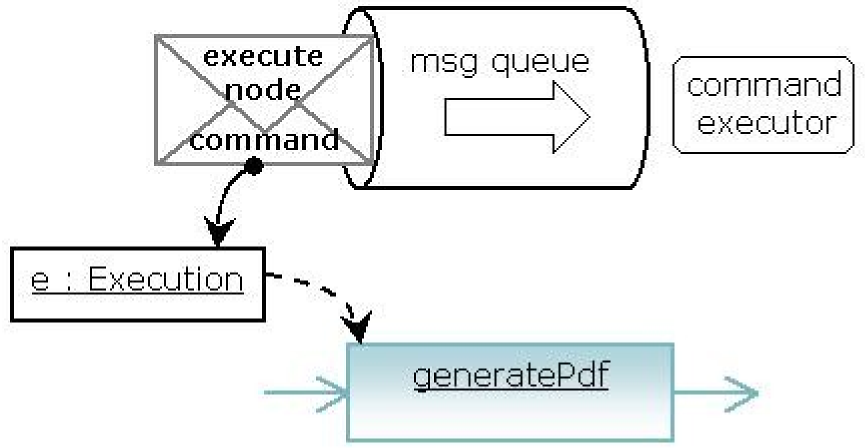


```
<process-definition name="jbay">

  <start-state name="submit item">
    <transition to="aution" />
  </start-state>

  <state name="auction">
    <transition to="fork" />
  </state>

  <fork name="fork">
    <transition to="send item" />
    <transition to="get paid" />
  </fork>

  ...
```

Other Constructs

- Process Composition
  - Including a sub process as part of a super process
  - Adds abstraction to process modeling
  - Breaks down large models in smaller blocks
  - Issues: process variables, one leaving transition
- Asynchronous Continuations
  - Demarcate transaction boundaries
  - Messaging system required

Asynchronous Continuations



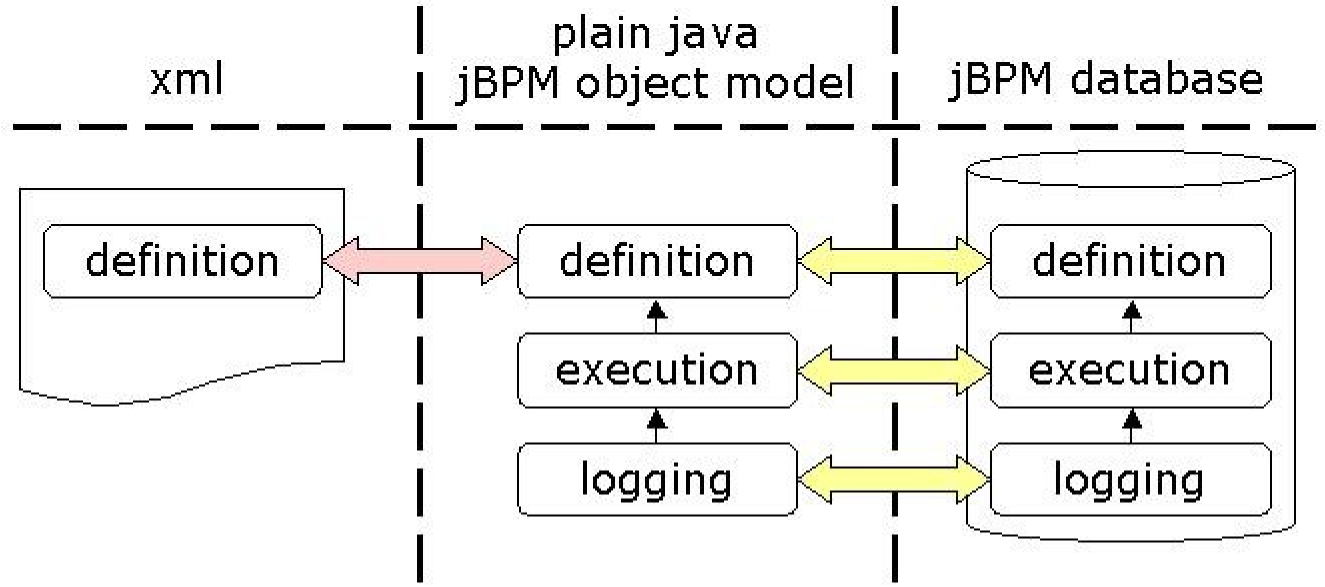
- Not invoking the generatePdf.execute() directly!
- New command *message* holds ref to the execution
- Message is sent to the command executor
  - Asynchronously, over the queue
- The executor invokes the execute method
- Two separate transactions
  - One that originated from the original event
  - Second where the command message was consumed

Israel JBUG

Persistence and Transactions

- Persistence:
  - Both process definition information (Nodes, Transitions, Actions) and runtime execution information can(read: should) be stored in a relational database
  - All process definition information is static and can be cached in memory
- Transactions:
  - Start when an event is being processed
  - Ends when a new wait state is reached, as **`Execution.event(...)`** returns

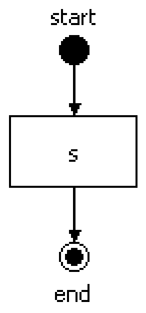
Persistence



Code Samples

Hello World

```
ProcessDefinition processDefinition = ProcessDefinition.parseXmlString(
    "<process-definition>" +
    "  <start-state>" +
    "    <transition to='s' />" +
    "  </start-state>" +
    "  <state name='s'>" +
    "    <transition to='end' />" +
    "  </state>" +
    "  <end-state name='end' />" +
    "</process-definition>"
  );
ProcessInstance processInstance = new ProcessInstance(processDefinition);

Token token = processInstance.getRootToken();
assertSame(processDefinition.getStartState(), token.getNode());

token.signal();
assertSame(processDefinition.getNode("s"), token.getNode());

token.signal();
assertSame(processDefinition.getNode("end"), token.getNode());
```



Storing Process Instance In DB

```
jbpmConfiguration = JbpmConfiguration.parseXmlString(
     "<jbpm-configuration>" +
     "  <jbpm-context>" +
     "    <service name='persistence' " +
     "             factory='org.jbpm.persistence.db.DbPersistenceServiceFactory' />" +
     "  </jbpm-context>" +
     "  <string name='resource.hibernate.cfg.xml' " +
     "          value='hibernate.cfg.xml' />" +
     "  <string name='resource.business.calendar' " +
     "          value='org/jbpm/calendar/jbpm.business.calendar.properties' />" +
     "  <string name='resource.default.modules' " +
     "          value='org/jbpm/graph/def/jbpm.default.modules.properties' />" +
     "  <string name='resource.converter' " +
     "          value='org/db/hibernate/jbpm.converter.properties' />" +
     "  <string name='resource.action.types' " +
     "          value='org/jbpm/graph/action/action.types.xml' />" +
     "  <string name='resource.node.types' " +
     "          value='org/jbpm/graph/node/node.types.xml' />" +
     "  <string name='resource.varmapping' " +
     "          value='org/jbpm/context/exe/jbpm.varmapping.xml' />" +
    "</jbpm-configuration>"
   );
jbpmConfiguration.createSchema();
```

Storing Process Instance In DB

```
ProcessDefinition processDefinition =
   ProcessDefinition.parseXmlString(...);

// 1. Standalone Application
JbpmContext jbpmContext = jbpmConfiguration.createJbpmContext();
// Deploy the process definition in the database
jbpmContext.deployProcessDefinition(processDefinition);
jbpmContext.close();

// 2. JSF Managed Bean
JbpmContext jbpmContext = jbpmConfiguration.createJbpmContext();
GraphSession graphSession = jbpmContext.getGraphSession();
ProcessDefinition processDefinition =
  graphSession.findLatestProcessDefinition("hello world");
ProcessInstance processInstance = new ProcessInstance(processDefinition);
Token token = processInstance.getRootToken();
token.signal();
jbpmContext.save(processInstance);
```

Storing Process Instance In DB

```
// 3. MDB

JbpmContext jbpmContext = jbpmConfiguration.createJbpmContext();
GraphSession graphSession = jbpmContext.getGraphSession();

ProcessDefinition processDefinition =
  graphSession.findLatestProcessDefinition("hello world");

// assuming only one process instance exists...
List processInstances =
  graphSession.findProcessInstances(processDefinition.getId());
ProcessInstance processInstance =
   (ProcessInstance) processInstances.get(0);

processInstance.signal();
assertTrue(processInstance.hasEnded());
jbpmContext.save(processInstance);
```

Seam PageFlow

```
<pageflow-definition [...]
               name="shopping">

   <start-state name="start">
      <transition to="browse"/>
   </start-state>

   <page name="browse" view-id="/browse.xhtml"
                          redirect="true">
      <transition name="browse" to="browse"/>
      <transition name="checkout" to="checkout"/>
   </page>

   <page name="checkout" view-id="/checkout.xhtml"
                            redirect="true">
      <transition name="checkout" to="checkout"/>
      <transition name="complete" to="complete"/>
   </page>

   <page name="complete" view-id="/complete.xhtml"
                            redirect="true">
      <end-conversation />
   </page>

</pageflow-definition>
```

Other Examples... Not Covered

- Process Variables
  - Contain the context information during process executions
  - java.util.Map that maps variable names to values
  - Variables are persisted as a part of the process instance
- Task Assignment
  - Assigning certain tasks to specific players
- Custom Actions
  - Binding custom java code into a jBPM process
  - Can be associated with their own nodes
  - Or be placed on events
    - Taking a transition
    - Entering / leaving a node

Israel JBUG

Demo

Creating – Testing – Deploying – Using – Monitoring

Conclusion

- JBoss jBPM provides a sophisticated platform for designing and developing workflows and business process management systems.
- It is Composed of APIs, domain-specific language, and graphical modeling tool
- It enables developers and business analysts to communicate and operate using a common semantics and platform.