NOVA WorkFlow

User Manual

(version 0.1)

NOVA WorkFlow User Documentation
Version 0.1 / 2010 August
Author: Fazle Rabbi
Reviewer: Hao Wang, Janet Norgrove

Centre for Logic and Information
St. Francis Xavier University
St Mary's Street
Antigonish, NS Canada
B2G 2A5

Home page: www.logic.stfx.ca

ACKNOWLEDGMENT

This work is sponsored by Natural Sciences and Engineering Research Council of Canada (NSERC), by an Atlantic Computational Excellence Network (ACEnet) Post Doctoral Research Fellowship and by the Atlantic Canada Opportunities Agency (ACOA) through the Atlantic Innovation Fund.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Product Overview

Developed with understanding of compensable transaction and formal verification, NOVA WorkFlow is an innovative workflow modeling framework based on the Compensable Workflow Modeling Language (CWML)[1], a formal graphical language proposed by CLI. The framework consists of a graphical editor, a translator and a workflow engine.

The graphical editor provides visual modeling of workflow which ensures correctness by construction. The editor is developed as an Eclipse RCP plug-in[2], so you can make use of many UI features provided by Eclipse and install the editor in different OS platform.

The translator can automatically translate a workflow model to a model in the input language of a model checker3. After building a model using our editor, you only needs to click on a action button and will obtain a translated model for simulation and verification in the model checker. As workflow models in reality can be rather huge and complicated, resulting in unbearable long verification time, the translator incorporates a model reduction algorithm accelerate the verification time while maintaining the equivalence of the original model and the reduced one.

The workflow engine let you execute the verified workflow model built using the editor. The engine is developed using popular Spring (http://www.springsource.org) and Hibernate (http://hibernate.org) framework with a good understanding of current J2EE framework. The workflow engine can run in different platform with various database and web application servers.

[1] For details on CWML, please refer to Fazle Rabbi, Hao Wang and Wendy MacCaull. "Compensable WorkFlow Net". The 12th International Conference on Formal Engineering Methods (ICFEM 2010).

[2] Eclipse, a popular and powerful Java IDE, is architected so that its components could be used to build just about any client application. The minimal set of plug-ins needed to build a rich client application is collectively known as the Rich Client Platform (RCP).

[3] Currently we use the model checker DiVinE (http://divine.fi.muni.cz/), The framework will provide support to other model checkers in the near future.

How to install

Product Requirements

Operating system (any one)

- ✓ SUN Solaris 2.6, 7, 8, 9 or 10[sparc]
- ✓ Linux- Red Hat Enterprise Linux/Fedora, Debian etc
- ✓ Windows 2000/2003 Server, Advanced Server
- ✓ Windows 2000/XP/Vista/2007

Application Server (any one)

- ✓ BEA Weblogic Server 8.1/9
- ✓ Resin 3.0.x
- ✓ Apache Tomcat 5.0.x

Database Server (any one)

- ✓ Oracle 9i Release 9.2
- ✓ MySQL 5
- ✓ Sybase 12.5 or higher
- ✓ PostgreSQL 8

Java Devleopment Kit

- ✓ SUN JDK 1.5

Model Checker

- ✓ DiVinE

Open source software's

- ✓ Spring Framework 1.2

- ✓ Hibernate 3.5.4
- ✓ Eclipse Galileo 3.5

Starting the installation

Download and install Sun JDK 1.5 from http://java.sun.com and Eclipse Galileo from http://www.eclipse.org . Download NOVA Workflow plugin ca.stfx.logic.novawf.jar from http://logic.stfx.ca/novaworkflow and paste under eclipse/plugins directory.

Create a project

Open Eclipse and create a ***Java Project***. Create a library folder named '*lib*' in your project. Download **cwf.jar** from http://logic.stfx.ca/novaworkflow and paste it into '*lib*' directory. Add **cwf.jar** into your ***Build Path***.

What you will see

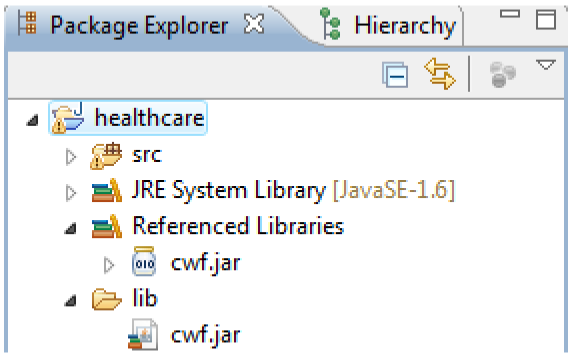


Fig1.1: Directory Structure of Java Project

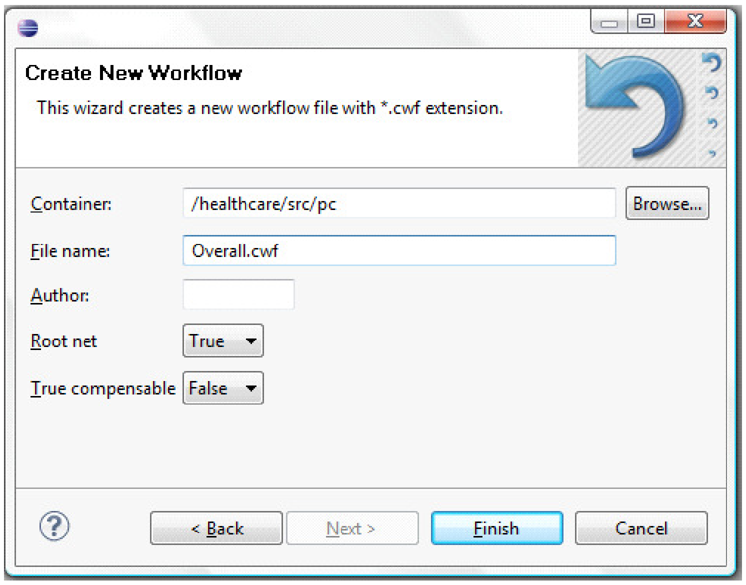
Create a Workflow model

Create a package in your ***source*** (src) directory where you want to store your workflow models. Right click on your ***package*** and select ***New -> Example***. Select ***'New Workflow'*** wizard and click next. You will see 'Create New Workflow' Wizard. Enter the name of the workflow (file extension .cwf), author name. Select additional attributes for the workflow from drop down list. Attributes are described in the following table:

Table1.1 Workflow attributes

Attribute	Value	Description
Root Net	True	During execution, a workflow with Root Net = True will start first. There can be only one workflow with Root Net = True in your workflow package.
	False	A workflow with Root Net = False is a subnet. A subnet can be decomposed by a composite task. During execution of a composite task, it is unfolded to a subnet
True Compensable	True	A True compensable workflow can hold only compensable tasks. A compensable task can only be decomposed to a True compensable workflow.
	False	A workflow with True Compensable = False can hold both compensable and uncompensable task. An uncompensable task can only be decomposed to this workflow.

What you will see


Create New Workflow

This wizard creates a new workflow file with *.cwf extension.

Container: /healthcare/src/pc

File name: Overall.cwf

Author:

Root net: True

True compensable: False

Fig1.2: Create New Workflow Wizard

Click ***Finish*** to create your first workflow model. An empty workflow model will open in the editor pane with an ***Input Condition*** and an ***Output Condition***.

What you will see

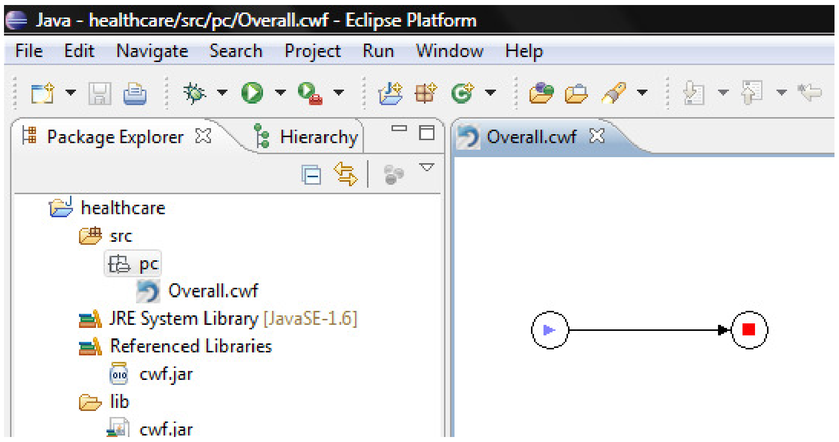

Java - healthcare/src/pc/Overall.cwf - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Package Explorer
Hierarchy
Overall.cwf
healthcare
src
pc
Overall.cwf
JRE System Library [JavaSE-1.6]
Referenced Libraries
cwf.jar
lib
cwf.jar

Fig1.3: Workflow Editor

Workflow Components View

To edit the workflow, open ***Workflow Components*** View. Workflow Components view can be found from ***Window->Show View-> Other -> CWML***. You can also use the ***Outline*** view to get an outline of your workflow components.

Where you will find

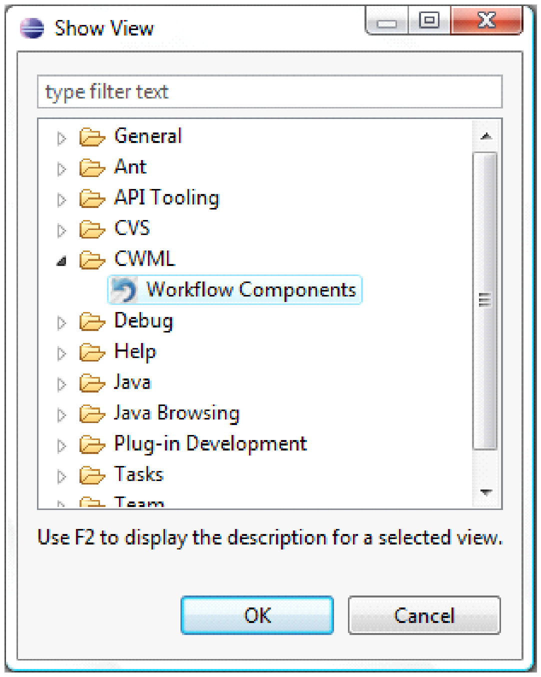


Fig1.4: Open Workflow Components View

Fig1.5 shows the ***Workflow Components*** View. Using the tools you can easily edit your workflow.

What you will see

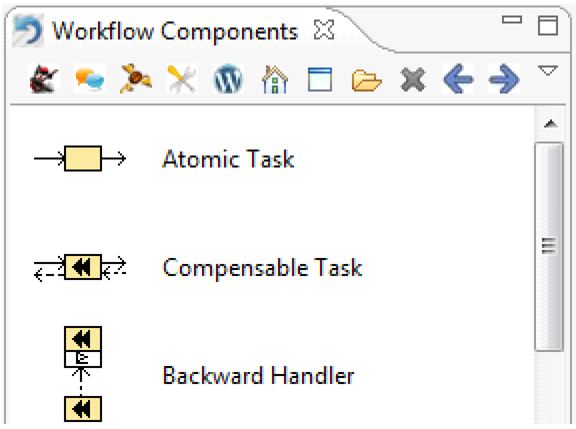


Fig1.5: Workflow Components View

CHAPTER 2

USING THE EDITOR

NOVA Workflow comes with a graphical editor for workflow modeling. The workflow you will make using this editor will be a structured workflow. The workflow model is stored in xml format.

Insert an atomic task

To insert an atomic task in your workflow model use ***Pre-Selection*** and ***Post-Selection*** tools. The tools are available in ***Workflow Components*** View.

Where you will find

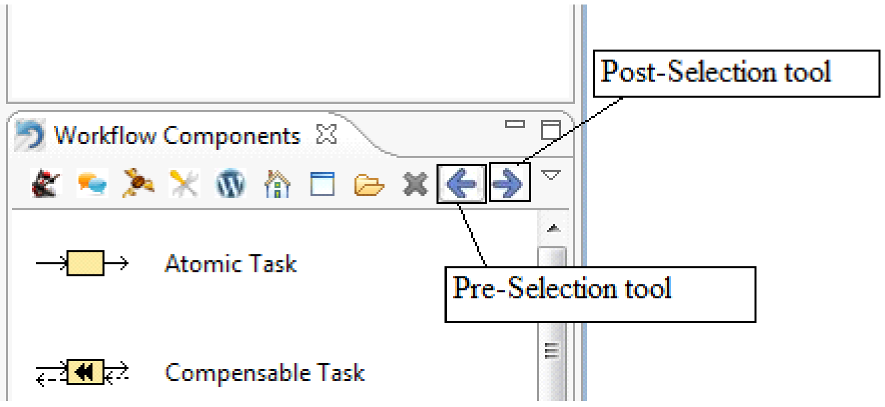


Fig2.1: Pre-Selection and Post-Selection tool

The ***Pre-Selection*** tool will change the color of a node to Green and ***Post-Selection*** tool will change the color of a node to Blue. After selecting two nodes by ***Pre-Selection*** and ***Post-Selection***, ***double click*** on ***Atomic Task*** from the Task list of ***Workflow Components***.

What you will see

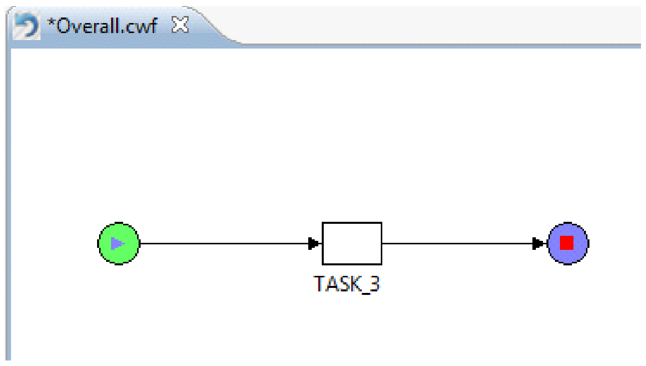


Fig2.2: Insert an Atomic Task

Insert a split-join block

To insert a split-join block use the ***Pre-Selection*** and ***Post-Selection*** tool as before. Select two nodes where you want to insert your block, and then ***double click*** on the split-join block from ***Workflow Components***.

What you will see

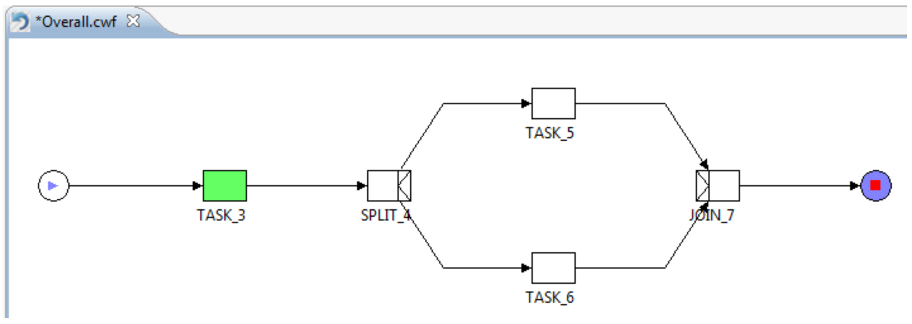


Fig2.3: Insert a Split-Join block

You can insert AND, XOR, OR, Parallel Composition, Internal Choice, Alternative Choice, Speculative Choice block in the same way.

You cannot insert an uncompensable task or block inside a compensable block. If you try to insert, you will get an *Error message* "Invalid Selection: You cannot insert uncompensable task inside Compensable block".

Increase number of branches of Split-Join block

You can insert a new branch to a split-join block with an atomic task or another split-join block. To do this select the Split task of the block by ***Pre-Selection*** and the Join task by ***Post-Selection*** and ***double click*** on the task that you want to insert in a new branch from Workflow Components View.

What you will see

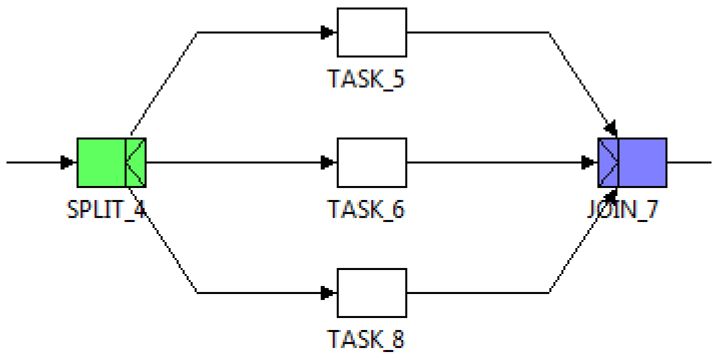


Fig2.4: Insert atomic task in a new branch

Insert a Loop

To insert a loop around some tasks select two nodes using ***Pre-Selection*** and ***Post-Selection***. ***Double click*** on ***Loop*** from Workflow Components list.

What you will see

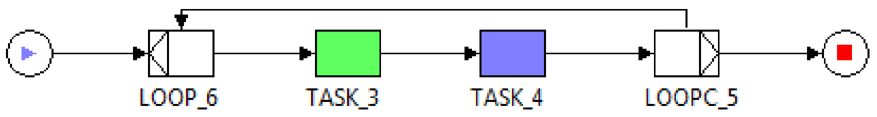


Fig2.5: Insert a Loop

To insert a loop around a single atomic task, select the atomic task twice-first by *Pre-Selection* tool and then again by *Post-Selection* tool.

Add an Error Handler

An error handler (Backward/Forward) can be added to a ***Compensable task***. Select the Compensable task by ***Pre-Selection*** tool and ***double click*** on the ***Error-Handler*** from Workflow Components view.

What you will see

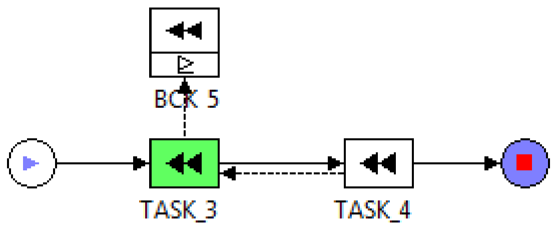


Fig2.6: Add an error handler

Add a Programmable Compensation

To add a ***Programmable Compensation*** to a ***Compensable task***, select the task by ***Pre-Selection*** tool, and ***double click*** on ***Programmable Compensation*** from Workflow Components.

What you will see

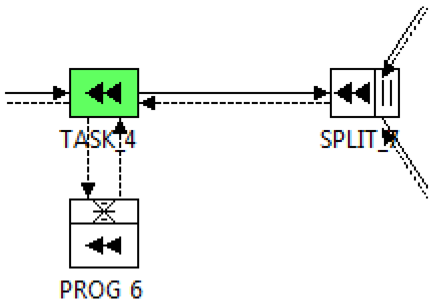


Fig2.6: Add a Programmable Compensation

Edit a task

To edit a task, select it and then click on the ***Task Property Settings*** tool from Workflow Components View.

Where you will find

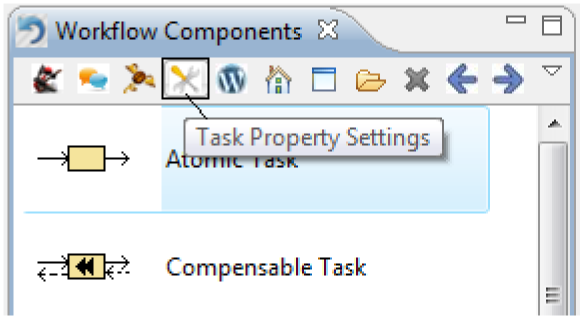


Fig2.7: Task Property Settings

This will open a Task Property dialog where you can edit task name, description, author name. There is a check box for creating Property file; this will be described in the next Chapter.

What you will see

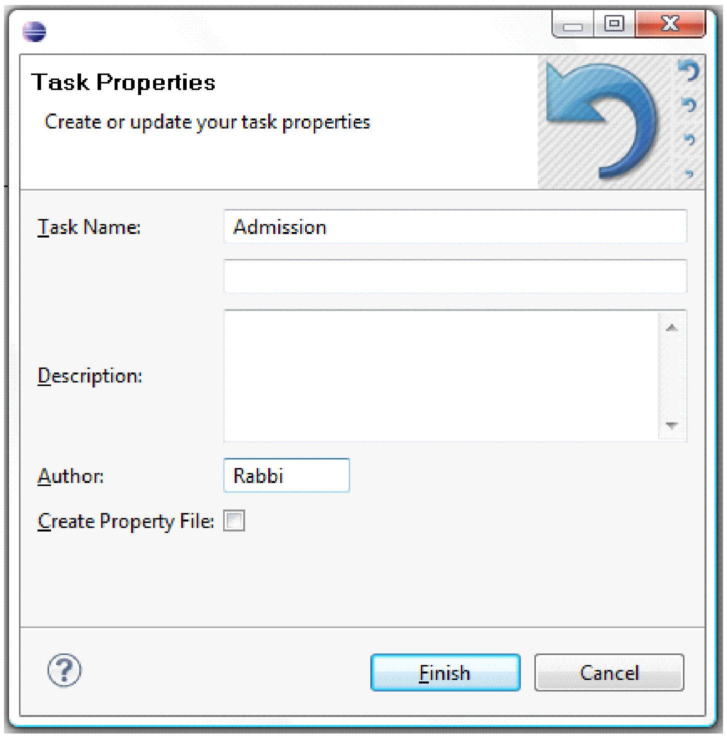

Task Properties
Create or update your task properties
Task Name: Admission
Description:
Author: Rabbi
Create Property File: ☐

Fig2.8: Task Property Settings

Delete a task

To delete a task or a split-join block, select the task by Pre-Selection tool, and then click on the delete tool from Workflow Components View. If you select a split task and click this tool, this will delete the whole block (split, join and all of its branches).

Where you will find

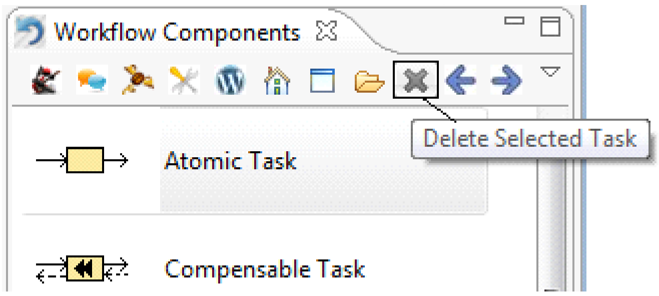


Fig2.9: Delete task tool

Make Composite Task

Select the task (***Atomic*** or ***Compensable***) you want to make composite, and then click on the ***Make Composite*** tool from Workflow Components view. A ***Subnet*** workflow selection dialog will open from where you can assign the subnet.

Where you will find

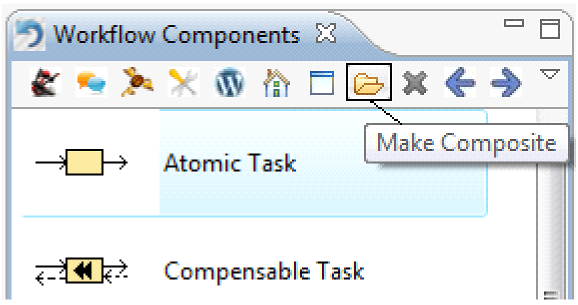
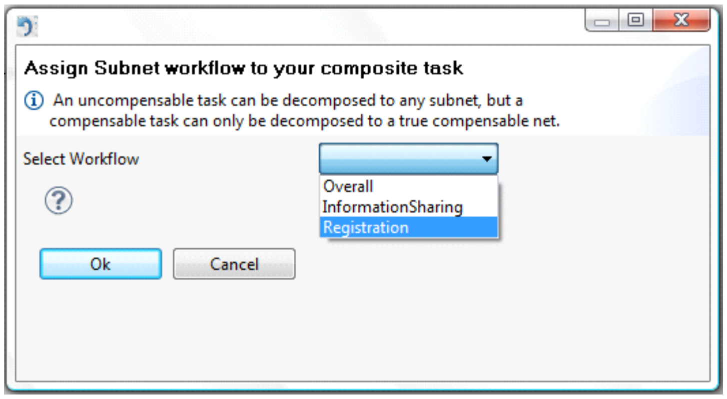


Fig2.10: Make Composite tool

What you will see



Assign Subnet workflow to your composite task

An uncompensable task can be decomposed to any subnet, but a compensable task can only be decomposed to a true compensable net.

Select Workflow

Overall
InformationSharing
Registration

Fig2.11: Subnet Workflow Selection Dialog

If you select a *Compensable* task to make it composite, you will only see Subnet workflows with *True Compensable = True*.

Chapter 3 USING THE TRANSLATOR FOR FORMAL VERIFICATION

NOVA WorkFlow incorporates a translator which translates the workflow model to a model in the input language for a model checker. Current version translates the workflow model to DiVinE model checker. One important feature of our translator is that it translates data fields in the model and provides data-abstraction capabilities for complex data types.

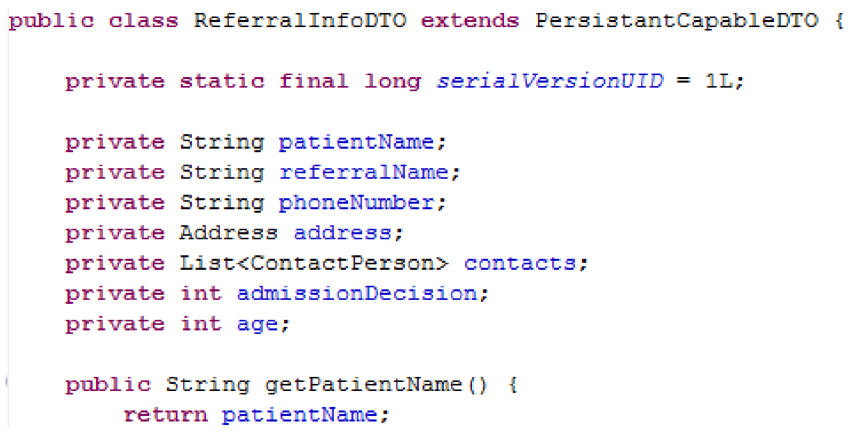
Supported Data types

For application development you can use any java data type. You can use Class, List, Vector, and Aggregate Class also. But for the verification you can only take byte, integer, long and boolean, although these data types can be specified inside a Class/List/Vector/Aggregate Class. String, Float and Double are not supported as they are not supported by the model checker. NOVA WorkFlow encourages you to make your entities object oriented.

All the entities/properties of your application might not be important for verification. For example *Patient's name* is not an interesting property that guides the flow, so you can simply ignore this property for verification.

Fig3.1 shows one simple entity bean that you can use in NOVA WorkFlow. If you are using Hibernate for your application development your entities will be a POJO (Plain Old Java Object).

Example



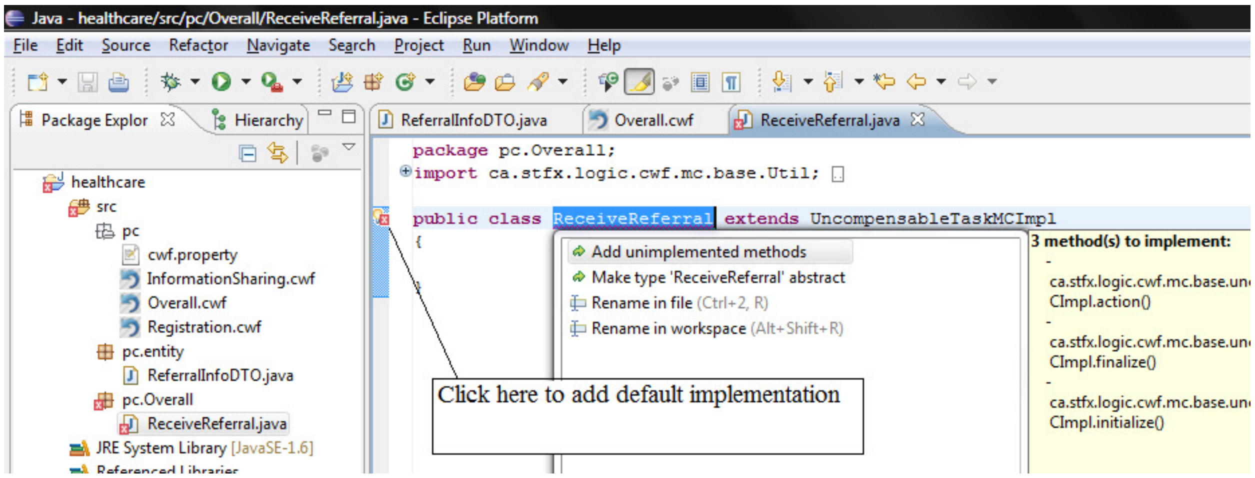
```java
public class ReferralInfoDTO extends PersistantCapableDTO {

    private static final long serialVersionUID = 1L;

    private String patientName;
    private String referralName;
    private String phoneNumber;
    private Address address;
    private List<ContactPerson> contacts;
    private int admissionDecision;
    private int age;

    public String getPatientName() {
        return patientName;
```

Fig3.1: Entity Bean

NOVA WorkFlow provides a base class named *PersistantCapableDTO* for entity beans and some abstract classes for Data Access Objects. There is *no restriction* to use *PersistantCapableDTO*, and you can ignore them.

Create Task Property File

When you have all your data-types defined as entity beans, you need to create task property files. In this property file, you can write statements that will be translated to the input language of model checker. To create a task property file, select a task and click on ***Task Property Settings*** tool from workflow components (*see Fig2.7*). Select the checkbox ***Create Property File*** and click ***Finish***. A Java class with the task name will be created under a package named with the workflow name. Depending on the type of the task you selected, the generated class will extend different abstract classes. The abstract class has some abstract method that you have to implement. Fig3.2 shows an example property file of an atomic task.


Java - healthcare/src/pc/Overall/ReceiveReferral.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explor
Hierarchy
ReferralInfoDTO.java
Overall.cwf
ReceiveReferral.java
healthcare
src
pc
cwf.property
InformationSharing.cwf
Overall.cwf
Registration.cwf
pc.entity
ReferralInfoDTO.java
pc.Overall
ReceiveReferral.java
JRE System Library [JavaSE-1.6]
Add unimplemented methods
Make type 'ReceiveReferral' abstract
Rename in file (Ctrl+2, R)
Rename in workspace (Alt+Shift+R)
Click here to add default implementation

```
package pc.Overall;
import ca.stfx.logic.cwf.mc.base.Util;

public class ReceiveReferral extends UncompensableTaskMCImpl
{
```

3 method(s) to implement:
- ca.stfx.logic.cwf.mc.base.un
CImpl.action()
- ca.stfx.logic.cwf.mc.base.un
CImpl.finalize()
- ca.stfx.logic.cwf.mc.base.un
CImpl.initialize()

Fig3.2: Task property file of an atomic task

Table3.1: Tasks, their abstract classes and interfaces for model checking implementation.

Task Type	Abstract Class and Interfaces	Abstract Methods
Atomic Task	UncompensableTaskMCImpl	initialize(), action(), finalize()
AndSplitTask	AndSplitMCImpl	initialize(), action(), finalize()
AndJoinTask	AndJoinMCImpl	initialize(), action(), finalize()
XorSplitTask	XorSplitMCImpl, IMCBranchCondition, IMCBranchOrder	initialize(), action(), finalize(), branchCondition(), getBranchOrder()
XorJoinTask	XorJoinMCImpl	initialize(), action(), finalize()
OrSplitTask	ORSplitMCImpl, IMCBranchCondition	initialize(), action(), finalize(), branchCondition()
OrJoinTask	ORJoinMCImpl	initialize(), action(), finalize()
LoopSplitTask	LoopSplitMCImpl, IMCBranchCondition, IMCBranchOrder	initialize(), action(), finalize(), branchCondition(), getBranchOrder()
LoopJoinTask	LoopJoinMCImpl	initialize(), action(), finalize()
CompensableTask	CompensableTaskMCImpl	initialize(), action(), finalize(),

		abortInitialize(), abort(), abortFinalize()
ParallelSplitTask	ParallelSplitMCImpl	initialize(), action(), finalize(), abortInitialize(), abort(), abortFinalize()
ParallelJoinTask	ParallelJoinMCImpl	initialize(), action(), finalize(), abortInitialize(), abort(), abortFinalize()
InternalChoieSplitTask	InternalChoiceSplitMCImpl, IMCBranchCondition	initialize(), action(), finalize(), abortInitialize(), abort(), abortFinalize(), branchCondition()
InternalChoiceJoinTask	InternalChoiceJoinMCImpl	initialize(), action(), finalize(), abortInitialize(), abort(), abortFinalize()
SpeculativeSplitTask	SpeculativeChoiceSplitMCImpl	initialize(), action(), finalize(), abortInitialize(), abort(), abortFinalize()
SpeculativeJoinTask	SpeculativeChoiceJoinMCImpl	initialize(), action(), finalize(), abortInitialize(), abort(), abortFinalize()
AlternativeSplitTask	AlternativeSplitMCImpl, IMCBranchOrder	initialize(), action(), finalize(), abortInitialize(), abort(), abortFinalize(), getBranchOrder()
AlternativeJoinTask	AlternativeJoinMCImpl	initialize(), action(), finalize(), abortInitialize(), abort(), abortFinalize()
BackwardHandlerTask	BackwardHandlerMCImpl	initialize(), action(), finalize()
ForwardHandlerTask	ForwardHandlerMCImpl	initialize(), action(), finalize(), abortInitialize(), abort(), abortFinalize()
Programmable Compensation Task	ProgrammableCompensationMCImpl	initialize(), action(), finalize()

More about task property file

Variable Declaration

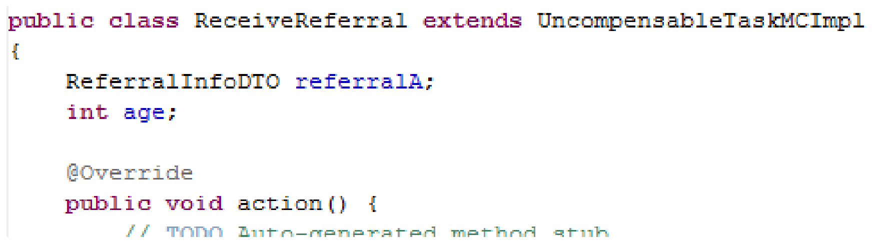
There are two types of variables in DiVinE in terms of visibility.

i) Global Variables
ii) Local Variables

If you want to use an entity bean in a task property file, you have to declare a variable for that as class attribute; this variable will be translated to DiVinE as global variable. If you want to use a local variable, you need to declare it in the task property class as class attribute with primitive data type.

Entity beans will be translated as Global Variable. On the other hand variables declared as primitive data type will be translated as Local Variable in a DiVinE process.

Example



```
public class ReceiveReferral extends UncompensableTaskMCImpl
{
    ReferralInfoDTO referralA;
    int age;

    @Override
    public void action() {
        // TODO Auto-generated method stub
```

Fig3.3: Variable declaration in task property file

Fig3.3 shows an example. In this example ***referralA*** will be declared as global variable in DiVinE and in any other ***task property file,*** if you use the same name they will refer to this variable. On the other hand ***age*** is declared as primitive data type, it will be translated to a local variable of DiVinE process.

NOVA Workflow translator will not translate the whole entity bean; it will read the statements of the property files and will translate only those properties of entity bean that is used. For example if patient's Name is not used in the property file, it will not be translated to DiVinE.

Table3.2: Different statements of java data access and corresponding DiVinE variables.

Java Data Access	DiVinE variable
referralA.setAge(30); /*age is an attribute of class ReferralInfoDTO, and referralA is a variable of type ReferralInfoDTO */	Int referralA_age;
referralA.getAddress().setRoadNumber(55); /* address is a class attribute of ReferralInfoDTO. The data type of address is Address class. roadNumber is a property of Address class */	int referralA_address_roadNumber;
referralA.getContacts(1).getAddress().getLocation(); /* The data type of contacts is List, and this is a property of class ReferralInfoDTO */	int referralA_contacts_Element_1_address_location;

Data abstraction

NOVA Workflow ships with a ***Util*** class which has a method ***getNonDeterministicData()***. Use this method whenever you need to mention some non-deterministic values for any variable. The signature of the method is given below:

```
public static Object getNonDeterministicData(Object[] values)
```

Example

```
public class ReceiveReferral extends UncompensableTaskMCImpl
{
    ReferralInfoDTO referralA;
    int age;

    @Override
    public void initialize() {
        age = (Integer)Util.getNonDeterministicData(new Integer[]{20,30,40,50,60});

    }
```

Fig3.4: Syntax for assigning non-deterministic data

Tips

For an integer the value range is -32768 to +32767. For model checking if we allow having each value, there will be a huge state explosion which will make impossible to check a model. Using sample values for each of the class will solve this problem.

Syntax

Limited number of Java syntax is allowed in task property file. As this file will be translated to the input language of a model checker, not all java syntax is supported. Below is a list of syntax allowed for different methods.

Method Name	Allowed Syntax

initialize(), abortInitialize()	localVar = (Integer)Util.getNonDeterministicData(new Integer[]{1,2,..});
	
	localVar = (Long)Util.getNonDeterministicData(new Long[]{1,2,..});
	
	localVar = (Byte)Util.getNonDeterministicData(new byte[]{1,2,..});
	
	localVar = (Boolean)Util.getNonDeterministicData(new Booelan[]{true,false});
	
	localVar = globalVar.getAttribute();
	
	localVar = globalVar.getAggregateProperty().getAttribute();
	
	localVar = globalVar.getListAttribute().get(index);
	
	localVar = globalVar.getListAttribute().get(index).getAttribute();

action(), abort()	Assignment statements using local variables and numbers. Assignment statements can contain: ▪ Numbers, true, false ▪ Parenthesis: (,) ▪ Variable identifiers ▪ Unary operators () ▪ Binary operators (|, ^, &, ==, !=, <, <=, >, >=, >>, <<, -, +, /, *, %)

finalize(),
abortFinalize()
	globalVar.setAttribute(localVar);
	
	globalVar.getAggregateProperty().setAttribute(localVar);
	
	globalVar.getListAttribute().set(index, localVar);
	
	globalVar.getListAttribute().get(index).setAttribute(localVar);

branchCondition
 	
(int branchNumber)
	if(branchNumber == 1)
	
	    return Boolean_Expression;
	
	else if(branchNumber == 2)
	
	    return Boolean_Expression;
	
	else
	
	    return Boolean_Expression;
	
	Boolean expressions can be written using local variables and numbers. The statements can contain:
	▪ Numbers, true, false
	▪ Parenthesis: (,)
	▪ Variable identifiers
	▪ Unary operators ()
	▪ Binary operators (|, ^, &, ==, !=, <, <=, >, >=, >>, <<, -, +, /, *, %)

getBranchOrder
 	
(int branchNumber)
	if(branchNumber == 1)
	
	    return 2;
	
	else if(branchNumber == 2)
	
	    return 1;

	else
	
	    return 3;

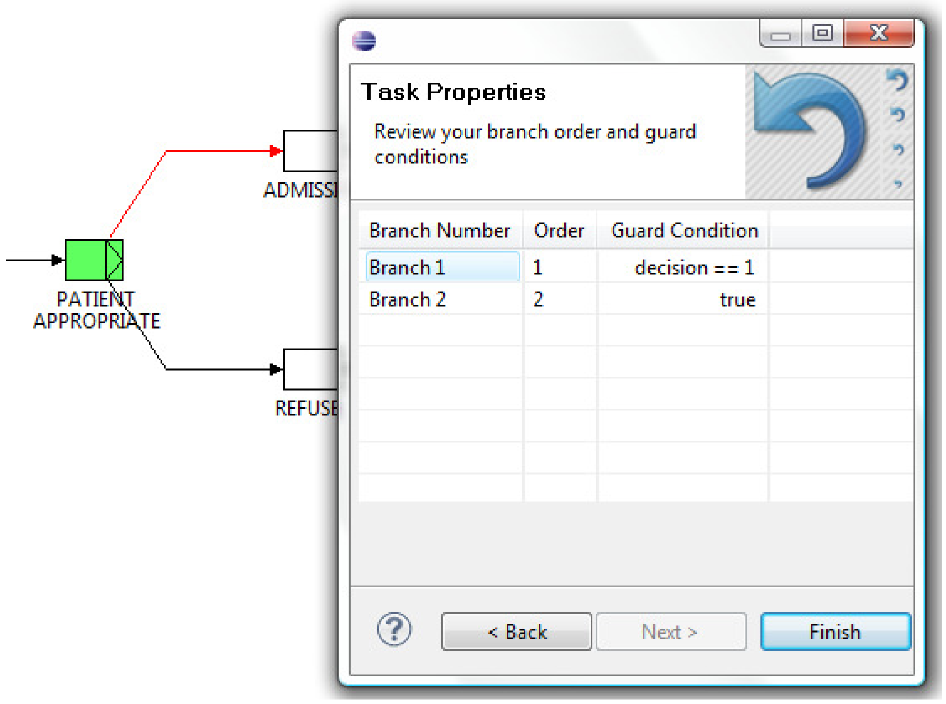
Translation Principle

NOVA WorkFlow translator will translate each of the task by reading the workflow model and its ***properties file***. If a task does not have any property file, only its flow will be translated to DiVinE. Each task will be translated to a DiVinE process. In the translation, ***initialize()*** method will be translated first in the process transition, then ***action()*** and at last ***finalize()*** method. The conditions specified in the ***branchCondition()*** will be translated as guard statement (pre-condition of a transition) of DiVinE process transition. ***getBranchOrder()*** method will be used to correctly translate the order of execution of the branches.

Review Branch Order and Condition

To review the branch condition mentioned in the task property file, select the task, and open ***Task Property Settings*** *(see Fig2.7)*. Click Next to view the dialog.

What you will see



Task Properties

Review your branch order and guard conditions

Branch Number	Order	Guard Condition
Branch 1	1	decision == 1
Branch 2	2	true

Fig3.5: Review branch condition and order

Only the task that implement either IMCBranchCondition or IMCBranchOrder interface or both will have this dialog to review. See Table3.1 for details.

Translate the model

After writing all necessary task property files for your workflow click on ***Translate to DiVinE*** tool from Workflow Components view. A file named ***translate.dve*** will be generated and stored in your workflow package. Use this file to verify your properties in DiVinE.

Where you will find

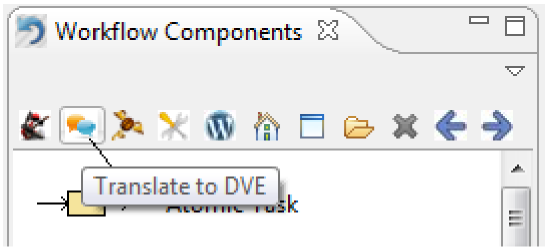


Fig3.6: Translate to DiVinE tool

If you don't see any code in the *translate.dve* file double check that you have a Workflow present with *Root Net = True*. NOVA WorkFlow translator will start the translation from Root net.

How to do the Reduction

NOVA WorkFlow translator ships with a reduction algorithm which can read LTL (Linear Temporal Logic) property and reduce the workflow model before doing the translation. To do this reduction write your property file and name it ***ltl.property***, store it in your workflow package. Now if you run the translation tool, the translation will be done after applying the reduction algorithm.

Syntax for Writing LTL property File

NOVA WorkFlow translator will read the ***#define*** statements of your LTL property file. If can mention any task state as a property. To do this use the following syntax:

_WorkflowName_TaskName_State

State can be either Successful, or Abort, or Failed. To mention about the task state use SUC, ABT, FAIL for Successful, Abort and Failed accordingly.

NOTE

When you mention task state as a property in your LTL property file, be careful about the *underscore* characters. There is a *leading underscore* before the workflow name.

Example

```
#define patient_appropriate (referralA_admissionDecision == 1)

#define registration_done (_Overall_Registration_SUC > 0)

#property write some property here
```

Chapter 4

USING THE WORKFLOW ENGINE

NOVA WorkFlow engine is a flexible workflow engine developed in the Spring & Hibernate framework. You can use the engine simply as a library. You can also get a custom made workflow management system by creating a client UI communicating with the engine, which can run with any web application server.

Configure your project

You have to include spring and hibernate jar files to the project. If you are a J2EE expert, you can safely jump to the next section. Below is a list of jar files you have to add in your project build path. Download required jar files from http://www.springsource.org/download and http://www.hibernate.org/downloads.html

Required jars

- activation.jar
- antlr-2.7.6.jar
- aopalliance.jar
- cglib-nodep-2.1.jar
- commons-collections-3.1.jar
- commons-discovery-0.2.jar
- commons-logging-1.0.4.jar
- dom4j-1.6.1.jar
- hibernate2.jar
- javassist-3.9.0.GA.jar
- jaxrpc.jar
- jstl.jar
- jta-1.1.jar
- jta.jar
- log4j-1.2.9.jar
- mysql-connector-java-3.0.15-ga-bin.jar
- saaj.jar

- slf4j-api-1.5.8.jar
- spring-aop.jar
- spring-beans.dtd
- spring-beans.jar
- spring-context.jar
- spring-core.jar
- spring-dao.jar
- spring-hibernate.jar
- spring-jdbc.jar
- spring-mock.jar
- spring-orm.jar
- spring-remoting.jar
- spring-support.jar
- spring-web.jar
- spring-webmvc.jar
- spring.jar
- spring.vm
- standard.jar
- wsdl4j-1.5.1.jar

To add the jar files in your build path, paste them into your ***'lib'*** directory. Select the jars from eclipse editor, right click on them and select ***Build Path -> Add to Build Path***.

Generate service classes

Click on ***Create Service Class*** tool from your Workflow Components view.

Where you will find

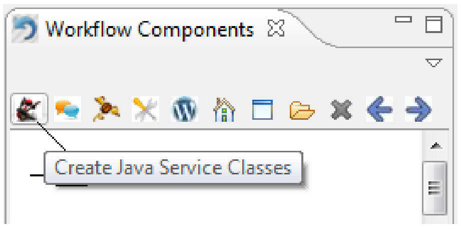
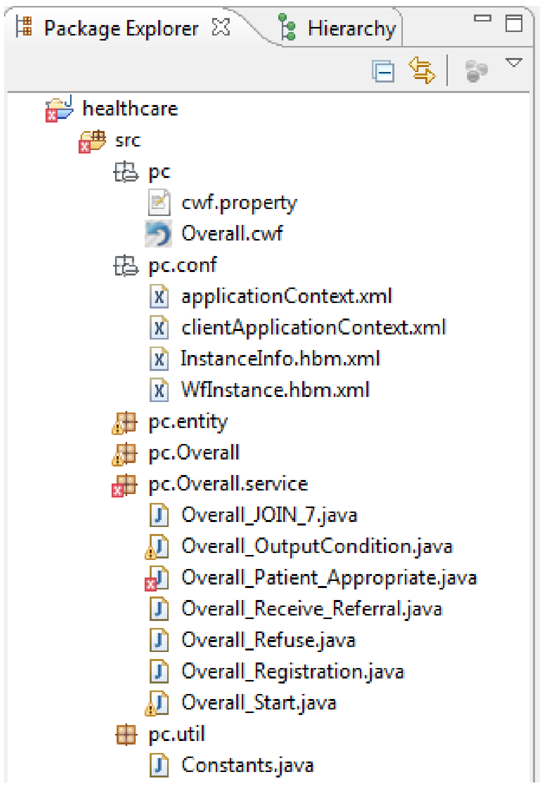


Fig4.1: Tool for generating Service Class

For each of the task of your workflow a service class will be generated. Some hibernate mapping files; an applicationContext.xml file will be generated that is required for the execution of workflow engine. A clientApplicationContext.xml will be generated that you can use for your client application development. Most of the Service classes have default implementation; some service classes will need one or two methods to be implemented by the application developer (i.e., ***getBranchCondition(), isManual()***).

What you will see



```
healthcare
  src
    pc
      cwf.property
      Overall.cwf
    pc.conf
      applicationContext.xml
      clientApplicationContext.xml
      InstanceInfo.hbm.xml
      WfInstance.hbm.xml
    pc.entity
    pc.Overall
    pc.Overall.service
      Overall_JOIN_7.java
      Overall_OutputCondition.java
      Overall_Patient_Appropriate.java
      Overall_Receive_Referral.java
      Overall_Refuse.java
      Overall_Registration.java
      Overall_Start.java
    pc.util
      Constants.java
```

Fig4.2: An Example scenario

Every Service class extends ***abstractTask*** and depending on the type of task, generated service classes will implement some Interfaces.

Table4.1 Task types and their Implemented Interfaces

Task Type	Interfaces	Abstract Methods
Atomic Task	ActionInterface	action()

AndSplitTask	ActionInterface	action()
AndJoinTask	ActionInterface	action()
XorSplitTask	ActionInterface, IBranchCondition	action(), getBranchCondition()
XorJoinTask	ActionInterface	action()
OrSplitTask	ActionInterface, IBranchCondition	action(), getBranchCondition()
OrJoinTask	ActionInterface	action()
LoopSplitTask	ActionInterface, IBranchCondition	action(), getBranchCondition()
LoopJoinTask	ActionInterface	action()
CompensableTask	ActionInterface, AbortInterface, FailInterface	action(), abort(), fail()
ParallelSplitTask	ActionInterface, AbortInterface, FailInterface	action(), abort(), fail()
ParallelJoinTask	ActionInterface, AbortInterface, FailInterface	action(), abort(), fail()
InternalChoieSplitTask	ActionInterface, AbortInterface, FailInterface, IBranchCondition	action(), abort(), fail(), getBranchCondition()
InternalChoiceJoinTask	ActionInterface, AbortInterface, FailInterface	action(), abort(), fail()
SpeculativeSplitTask	ActionInterface, AbortInterface, FailInterface	action(), abort(), fail()
SpeculativeJoinTask	ActionInterface, AbortInterface, FailInterface	action(), abort(), fail()
AlternativeSplitTask	ActionInterface, AbortInterface, FailInterface	action(), abort(), fail()
AlternativeJoinTask	ActionInterface, AbortInterface, FailInterface	action(), abort(), fail()
BackwardHandlerTask	ActionInterface, AbortInterface, FailInterface	Initialize(), action(), finalize()
ForwardHandlerTask	ActionInterface, AbortInterface, FailInterface	action(), abort(), fail()
Programmable Compensation Task	ActionInterface, AbortInterface, FailInterface	action(), abort(), fail()

How to work with the service classes

There are two ways to work with the workflow service classes.

i) Extend service classes
ii) Invoke services from outside

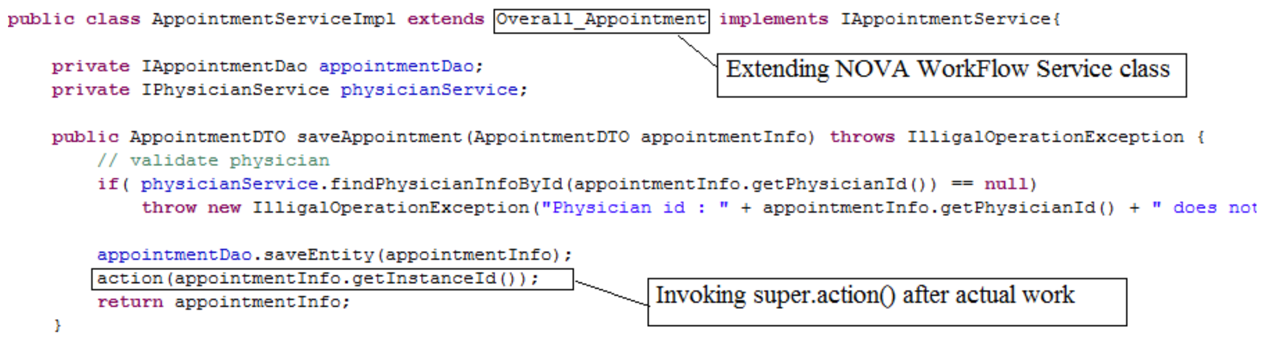
Option (i) is suitable for you if you want to use ***spring*** for your application development and option (ii) is suitable for you if you just don't want to use ***spring*** for application development.

Extend service classes

Extend a service class ***(generated by NOVA WorkFlow)*** and implement your business logic. Just after completing your actual work, invoke ***super.action()*** method to inform the engine that the work is accomplished. NOVA WorkFlow engine will update the task status. When you are invoking ***super.action()*** method you have to supply the ***instanceId*** as parameter. To know about ***instanceId*** see ***Workflow Engine Service*** section.

You have to change the service bean tag in applicationContext.xml. Configure your bean by replacing the Service Class name in the applicationContext.xml file. You need to replace the class name and add your bean references for your service bean *(see Fig4.4)*.

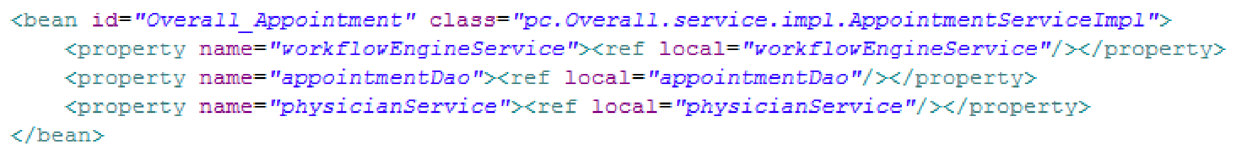
Example



```
public class AppointmentServiceImpl extends Overall_Appointment implements IAppointmentService{

    private IAppointmentDao appointmentDao;
    private IPhysicianService physicianService;

    public AppointmentDTO saveAppointment(AppointmentDTO appointmentInfo) throws IlligalOperationException {
        // validate physician
        if( physicianService.findPhysicianInfoById(appointmentInfo.getPhysicianId()) == null)
            throw new IlligalOperationException("Physician id : " + appointmentInfo.getPhysicianId() + " does not

        appointmentDao.saveEntity(appointmentInfo);
        action(appointmentInfo.getInstanceId());
        return appointmentInfo;
    }
```

Extending NOVA WorkFlow Service class
Invoking super.action() after actual work

Fig4.3: An example of Service class extension



```
<bean id="Overall_Appointment" class="pc.Overall.service.impl.AppointmentServiceImpl">
    <property name="workflowEngineService"><ref local="workflowEngineService"/></property>
    <property name="appointmentDao"><ref local="appointmentDao"/></property>
    <property name="physicianService"><ref local="physicianService"/></property>
</bean>
```

Fig4.4 Service Bean tag of applicationContext.xml

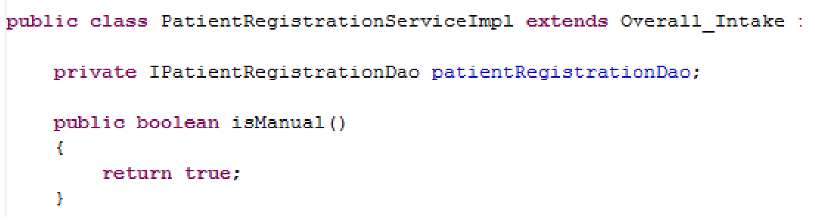
Invoke services from outside

To invoke a NOVA WorkFlow service bean from outside expose the bean interfaces by ***RMI/HttpInvoker/WebService***. After completing the actual work of a task, invoke the ***action()*** or ***abort()*** or ***fail()*** method from outside through the interface. NOVA WorkFlow engine will update the task accordingly.

Configure a service for Automatic or Manual execution

It is possible to configure a NOVA WorkFlow service for ***automatic/manual*** execution. If a task is configured for automatic execution, the engine will invoke the ***action()*** method automatically instead of waiting for its ***action()*** method to be invoked from the application. On the other hand if a service is configured for ***manual execution***, the engine will wait for the invocation of its ***action()*** method. A method ***isManual()*** is declared in ***abstractTask*** class and the default implementation of ***isManual()*** method returns false (means automatic execution). You can override this method and configure it for manual execution.

Example



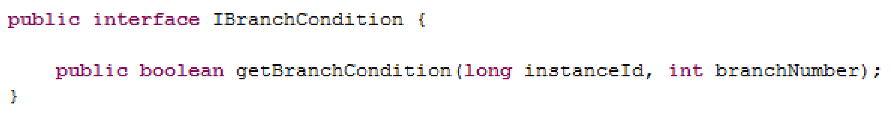
```
public class PatientRegistrationServiceImpl extends Overall_Intake :

    private IPatientRegistrationDao patientRegistrationDao;

    public boolean isManual()
    {
        return true;
    }
```

Fig4.5: Configure service for manual execution

The automatic/manual configuration for service execution is same for *action(), abort() and fail()* methods.

How to implement IBranchCondition Interface

NOVA WorkFlow engine needs to know about the branch Condition of some split tasks (see Table4.1). It does not provide any default implementation as it is totally depends on the application. The interface has only one method and the signature is given below:

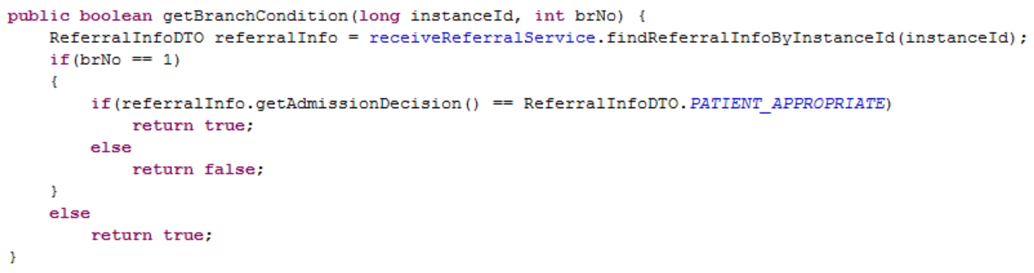


```
public interface IBranchCondition {

    public boolean getBranchCondition(long instanceId, int branchNumber);
}
```

Fig4.6: IBranchCondition Interface

You can write any java statement in the method while implementing. Typically you will need to do some database searching or invoking some service to know about the conditions. The workflow engine will invoke this method with ***instanceId*** and ***branchNumber*** to know which branch to execute.

Example



```
public boolean getBranchCondition(long instanceId, int brNo) {
    ReferralInfoDTO referralInfo = receiveReferralService.findReferralInfoByInstanceId(instanceId);
    if(brNo == 1)
    {
        if(referralInfo.getAdmissionDecision() == ReferralInfoDTO.PATIENT_APPROPRIATE)
            return true;
        else
            return false;
    }
    else
        return true;
}
```

Fig4.7: Example of getBranchCondition() implementation

NOVA WorkFlow Engine will get the branch orders from the task properties file that you configured for property verification. See ‘Review Branch Order and Condition’ section in previous chapter.

WorkFlow Engine Service

NOVA WorkFlow engine provides an interface ***IWorkFlowEngineService*** using which an application can be easily build with workflow support. The methods are described in Table 4.2

```
public interface IWorkflowEngineService {

	public WfInstance createNewWorkflowInstance(WfInstance newInstance);
	public WfInstance getInstance(long id);
	public List<WfInstance> getAllActiveInstances();

	public List<InstanceInfo> getAvailableMethods(WfInstance theInstance);
	public List<InstanceInfo> getAvailableMethods(Long instanceId, String taskId);
}
```

Table4.2 Description of the methods of IWorkflowEngineService

Method Name	**Functionality**
createNewWorkflowInstance	To create a new workflow instance use this method. This method inserts a new record in table ***WfInstance*** and generates a unique id for the newly created instance.
getInstance	To know details about an instance this method can be used.
getAllActiveInstances	This method returns all Active instances
getAvailableMethods	There are two overload methods: i) Takes a workflow instance and returns all ***InstanceInfo*** containing ***taskId*** and available methods of the tasks ii) Takes an ***instanceId*** and ***taskId*** as parameter and returns all available methods for the task

Attributes of class ***InstanceInfo*** is shown here:

```
	private Long id;
	private Long instanceId;
	private String taskId;
	private String workflowName;
	private String varName;
	private Integer value;
	private String availableMethod;
	private String actor;
```

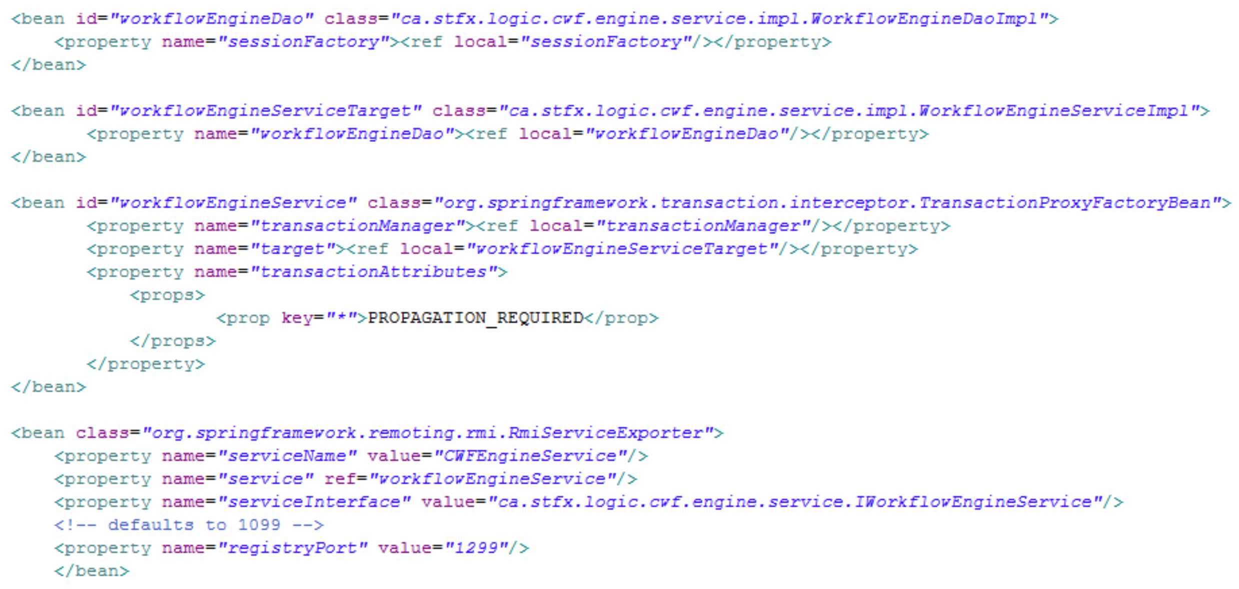
Table4.3 shows all possible values of ***availableMethod***. Your application will determine the available operation by these constants:

Table4.3 Possible values of availableMethod and their meaning

Value	**Meaning**
ACTION	Task is active for execution
ABORT	Task is enable for abort operation
FAIL	Task need to perform fail operation
CMP_ACTION	The composite task is active for execution
CMP_ABORT	The subnet tasks aborted, the composite task needs to perform abort operation

How to get the WorkFlow Engine Service

The workflow engine is deployed in applicationContext.xml as a service. You can expose the interface using RMI/HttpInvoker/WebService or any other methods. An example of exposing the service using RMI is shown here:



```xml
<bean id="workflowEngineDao" class="ca.stfx.logic.cwf.engine.service.impl.WorkflowEngineDaoImpl">
    <property name="sessionFactory"><ref local="sessionFactory"/></property>
</bean>

<bean id="workflowEngineServiceTarget" class="ca.stfx.logic.cwf.engine.service.impl.WorkflowEngineServiceImpl">
       <property name="workflowEngineDao"><ref local="workflowEngineDao"/></property>
</bean>

<bean id="workflowEngineService" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
       <property name="transactionManager"><ref local="transactionManager"/></property>
       <property name="target"><ref local="workflowEngineServiceTarget"/></property>
       <property name="transactionAttributes">
           <props>
                   <prop key="*">PROPAGATION_REQUIRED</prop>
           </props>
       </property>
</bean>

<bean class="org.springframework.remoting.rmi.RmiServiceExporter">
    <property name="serviceName" value="CWFEngineService"/>
    <property name="service" ref="workflowEngineService"/>
    <property name="serviceInterface" value="ca.stfx.logic.cwf.engine.service.IWorkflowEngineService"/>
    <!-- defaults to 1099 -->
    <property name="registryPort" value="1299"/>
    </bean>
```

Fig4.8: Exposing WorkFlow Engine Service using RMI

How to Deploy

Deploy the service classes, entity beans, hibernate mapping files and application context in your application server or web container. You have to include spring, hibernate jar files in the web application and additionally ***cwf.jar*** file.

The default applicationContext.xml that is generated by NOVA WorkFlow Service class generation tool includes default database information. You have to edit the information to configure your database.

Chapter 5

DEMO

A demo application is available in CLI website which shows the functionalities of NOVA WorkFlow. You can download it from http://logic.stfx.ca/novaflow/pc_demo.html. The demo contains two projects- i) Server side application including workflows, ii) client side application which is an eclipse RCP application. The webapp directory contains all required files that you will need to deploy in your web container. One could use other technologies like JSP/Struts/JSF/Others for the client side application.

Use Case Scenario

PC_Demo application is a small workflow of healthcare system. A patient is referred by community care to palliative care program, where s/he consults with a physician and get registered to the program. Once registered a team consisting of formal and informal caregiver is built for the patient. For this example our use-case is as follows

- A patient will be referred to Palliative-Care program.
- An appointment will be set with a physician for the patient.
- Physician will consult with the patient.
- Physician will decide if the patient is appropriate for the program or not
- If the patient is not appropriate s/he will be Refused with an explanation
- Otherwise the patient will be registered.
- A team will be made with different caregivers for the registered patient.

Workflows in pc_demo

There are two workflows in pc_demo named Ovreall and TeamBuilding. Overall is the Root net and TeamBuilding is a true compensable net. Fig5.1 and 5.2 shows the workflow models.

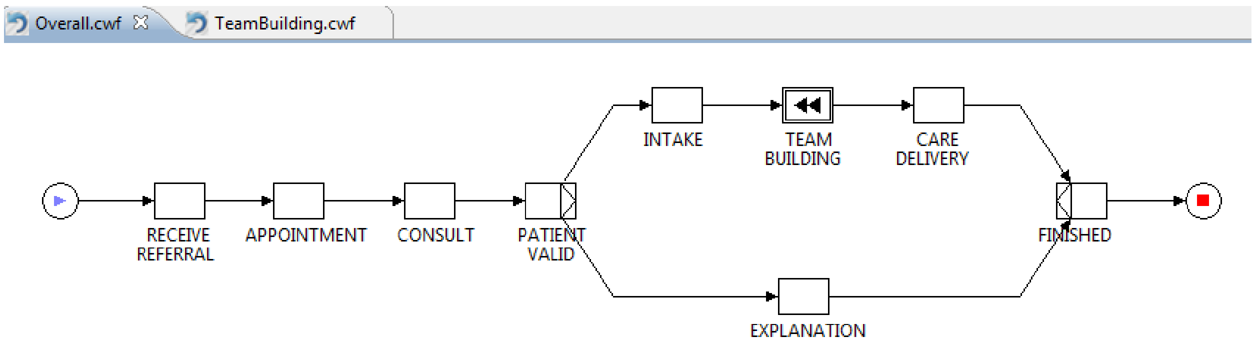


Fig5.1 Overall workflow of pc_demo

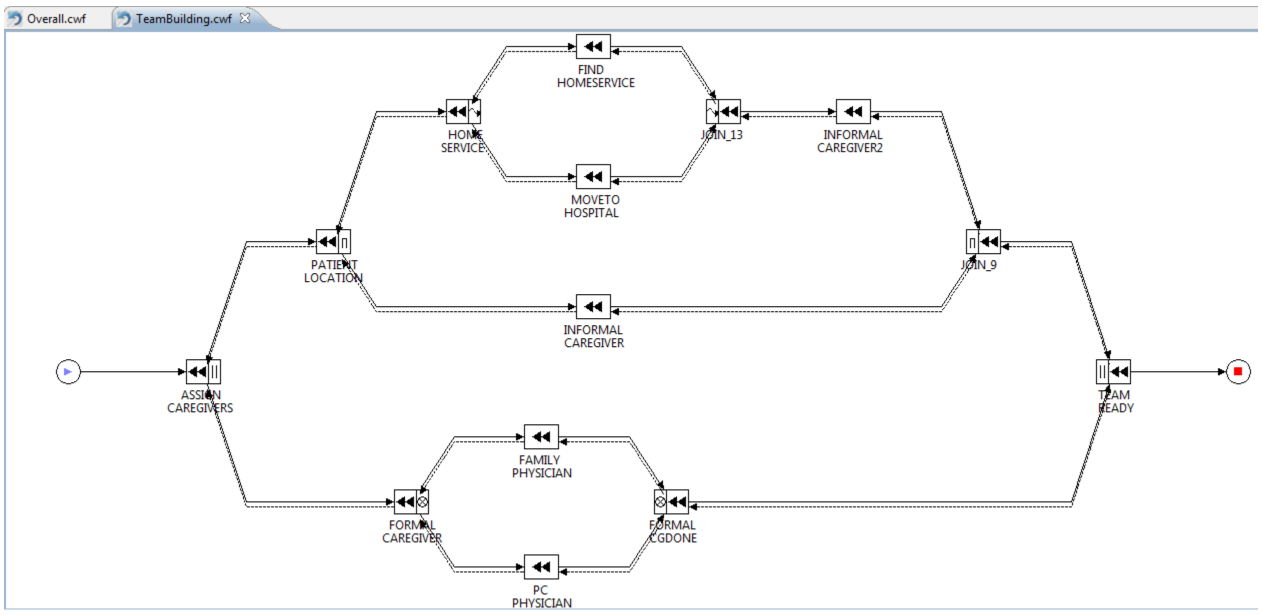


Fig5.2 TeamBuilding workflow of pc_demo

How to run

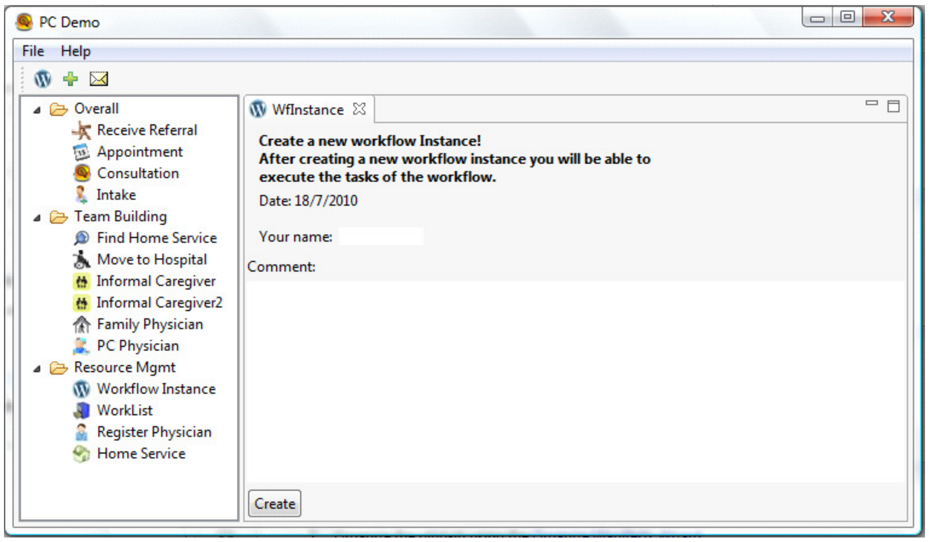
Create a database from the script provided in the ***demo_app***, or you can create tables using ***ant-script*** and the ***hibernate mapping files***. Run your database server and provide the information to the applicationContext.xml. Deploy the ***webapp*** to your web container.

To start your client application, import the project into eclipse. You can either launch the application from eclipse or you can build it as RCP product then run.

NOTE

To build/run the pc_demo client application, make sure that your eclipse installation has RCP development environment.

What you will see



Create a new workflow Instance!
After creating a new workflow instance you will be able to execute the tasks of the workflow.

Date: 18/7/2010

Your name:

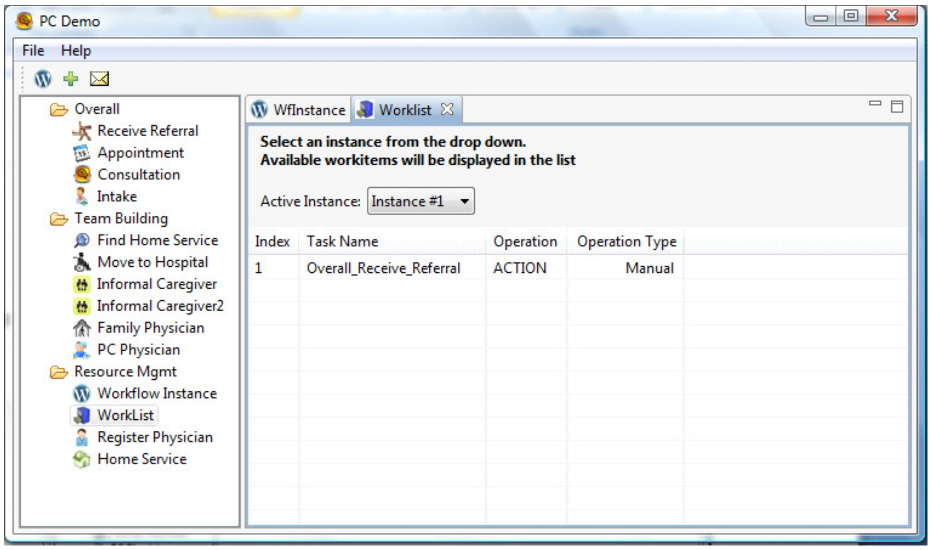
Comment:

Fig5.3: pc_demo client application

Play with the application

Create a workflow instance. The engine will persist a new workflow instance to database. To view the available tasks of an instance open WorkList view in your client application. It will show all active instances in a drop down. Select an instance from the drop down to see the active tasks of the instance.

What you will see



Select an instance from the drop down.
Available workitems will be displayed in the list

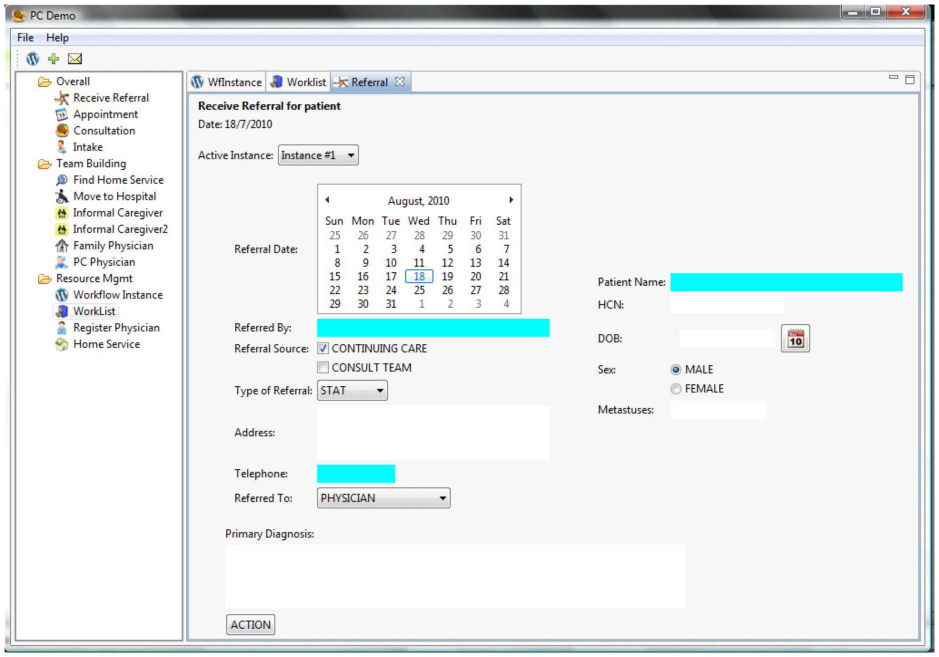
Active Instance: Instance #1

Index	Task Name	Operation	Operation Type
1	Overall_Receive_Referral	ACTION	Manual

Fig5.4: WorkList view of pc_demo client application

If you select a task from the worklist, its view (form) will be opened where you can insert information and execute the task.

What you will see



Receive Referral for patient
Date: 18/7/2010

Active Instance: Instance #1

Referral Date:

August, 2010

Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Referred By:
Referral Source: [✓ CONTINUING CARE] CONSULT TEAM
Type of Referral: STAT
Address:
Telephone:
Referred To: PHYSICIAN
Primary Diagnosis:

Patient Name:
HCN:
DOB:
Sex: [✓ MALE] FEMALE
Metastuses:

Fig5.5: Referral form of pc_demo client application

Some tasks are ***not part of*** this patient's workflow, for example registering a physician or home service. There can be a different workflow for managing the resources, but in this ***demo_app*** its simply done by accessing ***PhysicianService*** service bean.