Documentation Application VIF

Workflow framework

Benno Luthiger
Bachmattstr. 39
CH-8048 Zürich

benno.luthiger@id.ethz.ch

Version 1.0

Zurich, 26. July 2003

Table of Content

Abstract

This document describes content and use of the workflow framework.

Credits

The workflow package `org.hip.kernel.workflow` is the Java implementation of the Python workflow module *itools* created by Juan David Ibáñez Palomar *(jdavid@itaapy.com)*.

Licensing

1. Content of the workflow package

The VIF workflow package consists of the interfaces `State`, `Transition` and `WorkflowAware` as well as the classes `Workflow`, `WorkflowAwareImpl` and `WorkflowException`. The interfaces `State` and `Transition` are implemented internally. Instances implementing the `State` interface can be obtained calling `WorkflowAwareImpl.getState()`.Instances implementing the `Transition` interface can be obtained calling the `State.getTransition()`. Classes intended for taking part in a workflow have to be made workflow aware, i.e. they have to implement the `WorkflowAware` interface. However, because the functionality of a `WorkflowAware` object is implemented by `WorkflowAwareImpl`, such an object simply can create an instance of `WorkflowAwareImpl` and then delegate all calls to its `WorkflowAware` interface to this `WorkflowAwareImpl` instance.

1.1. Interface State

The `State` interface is used to describe a state. A `State` has transitions to other `State`s.

Methods

addTransition

```
void addTransition(String inTransitionName, Transition inTransition)
```

Adds the transition with the specified name to the state instance.

Parameters: `inTransitionName` Name of the Transition added.
`inTransition` Transition added to the State.

getTransition

```
Transition getTransition(String inTransitionName)
```

Returns the transition with the specified name.

Parameters: `inTransitionName` Name of the transition to be returned.

Returns: the transition identified with the specified name.

1.2. Interface Transition

The `Transition` interface is used to describe transitions. Transitions come from one `State` and go to another.

Methods

getStateFrom

```
String getStateFrom()
```

Returns the state the transition starts from.

Returns: the state the transition starts from.

getStateTo

```
String getStateTo()
```

Returns the state the transition is going to.

Returns: the state the transition is going to.

1.3. Interface WorkflowAware

The interface `WorkflowAware` has to be implemented by workflow aware objects.

Specific application semantics for states and transitions can be implemented as methods of the class implementing this interface.

These methods get associated with the individual states and transitions by a simple naming scheme. For example, if a Workflow has two states *Private* and *Public*, and a transition *Publish* that goes from *Private* to *Public*, the following happens when the transition is executed:

1. if implemented, the method `onLeave_Private` is called (it is called each time the object leaves the *Private* state)
2. if implemented, the method `onTransition_Publish` is called (it is called whenever this transition is executed)
3. if implemented, the method `onEnter_Public` is called (it is called each time the object enters the *Public* state)

Methods

enterWorkflow

```
void enterWorkflow(Workflow inWorkflow, Object[] inArgs)
throws WorkflowException
```

(Re-)Bind this object to the specified workflow. The `Workflow` must provide a default initial `State`.

Parameters: `inWorkflow` The Workflow the workflow aware object has to be bound to.
`inArgs` Arguments passed down to all handlers called.

Throws: `WorkflowException`

enterWorkflow

```
void enterWorkflow( Workflow inWorkflow,
                    String inInitialStateName, Object[] inArgs)
                    throws WorkflowException
```

(Re-)Bind this object to the specified workflow. The `inInitialStateName` parameter is the workflow `State` that should be taken on initially.

Parameters: `inWorkflow` The Workflow the workflow aware object has to be bound to.
`inInitialStateName` The name of the initial State.
`inArgs` Arguments passed down to all handlers called.

Throws: `WorkflowException`

doTransition

```
void doTransition(String inTransitionName, Object[] inArgs)
throws WorkflowException
```

Performs the transition with the specified name, changes the state of the object and runs any defined state/transition handlers. Extra arguments are passed down to all handlers called.

Parameters: `inTransitionName` The transition to perform.
`inArgs` Arguments passed down to all handlers called.

Throws: `WorkflowException`

getStateName

```
String getStateName()
```

Returns the name of the current State.

Returns: the name of the current State

getState

```
State getState() throws WorkflowException
```

Returns the current `State` instance.

Returns: `State` the current `State` instance

Throws: `WorkflowException`

1.4. Class Workflow

The class workflow is used to describe a workflow. A workflow has states (one of them is the initial state), and states have transitions that go to another state.

Constructors

```
public Workflow()
```

Construct a Workflow object.

Methods

setInitialStateName

```
public void setInitialStateName(String inInitialStateName)
throws WorkflowException
```

Sets the state with the specified name as default initial state.

Parameters: `inInitialStateName` The name of the initial state.

Throws: `WorkflowException`

getInitialStateName

```
public String getInitialStateName()
```

Returns the name of the initial state.

Returns: the name of the initial state

getState

```
public State getState(String inStateName) throws WorkflowException
```

Returns the State with the specified name.

Parameters: `inStateName` the name of the `State` to be returned

Returns: the State with the specified name

Throws: `WorkflowException`

addState

```
public void addState(String inStateName)
```

Adds a new State with the specified name.

Parameters: `inStateName` The name of the `State` to be added.

addTransition

```
public void addTransition( String inTransitionName,
                           String inStateFrom, String inStateTo)
                           throws WorkflowException
```

Adds a new Transition. The specified state names from and to respectively are the origin and destination states of the transition.

Parameters: `inTransitionName` Name of the `Transition` to be added
`inStateFrom` Name of the `State` the `Transition` comes from
`inStateTo` Name of the `State` the `Transition` goes to

Throws: `WorkflowException`

1.5. Class WorkflowAwareImpl

The class `WorkflowAwareImpl` implements generic functionality which can be use by workflow aware objects, i.e. objects implementing the interface `WorkflowAware`. These objects can delegate calls to `WorkflowAware` methods to instances of this class.

Constructors

```
public WorkflowAwareImpl(Workflow inWorkflow,
                         Object[] inArgs,
                         WorkflowAware inCaller)
                         throws WorkflowException
```

Constructs a WorkflowAwareImpl object initialized with the specified Workflow object.

Parameters: `inWorkflow` The relevant Workflow.
`inArgs` Arguments passed down to all handlers called.
`inCaller` The object delegating the workflow aware behaviour.

Throws: `WorkflowException`

```
public WorkflowAwareImpl(Workflow inWorkflow,
                         String inInitialStateName,
                         Object[] inArgs,
                         WorkflowAware inCaller)
                         throws WorkflowException
```

Constructs a WorkflowAwareImpl object initialized with the specified Workflow object and the specified initial state.

Parameters: `inWorkflow` The relevant Workflow.
`inInitialStateName` The name of the initial State.
`inArgs` Arguments passed down to all handlers called.
`inCaller` The object delegating the workflow aware behaviour.

Throws: `WorkflowException`

Methods

enterWorkflow

```
public void enterWorkflow(Workflow inWorkflow, Object[] inArgs,
                          inCaller WorkflowAware)
                          throws WorkflowException
```

(Re-)Bind this object to the specified workflow. The `Workflow` must provide a default initial `State`.

Parameters: `inWorkflow` The Workflow the workflow aware object has to be bound to.
`inArgs` Arguments passed down to all handlers called.
`inCaller` The object delegating the workflow aware behaviour.

Throws: `WorkflowException`

enterWorkflow

```
public void enterWorkflow(Workflow inWorkflow,
                          String inInitialStateName, Object[] inArgs,
                          inCaller WorkflowAware)
                          throws WorkflowException
```

(Re-)Bind this object to the specified workflow. The `inInitialStateName` parameter is the workflow `State` that should be taken on initially.

Parameters: `inWorkflow` The Workflow the workflow aware object has to be bound to.
`inInitialStateName` The name of the initial State.
`inArgs` Arguments passed down to all handlers called.
`inCaller` The object delegating the workflow aware behaviour.

Throws: `WorkflowException`

doTransition

```
public void doTransition( String inTransitionName, Object[] inArgs,
                          inCaller WorkflowAware)
                          throws WorkflowException
```

Performs the transition with the specified name, changes the state of the object and runs any defined state/transition handlers. Extra arguments are passed down to all handlers called.

Parameters: `inTransitionName` The transition to perform.
`inArgs` Arguments passed down to all handlers called.
`inCaller` The object delegating the workflow aware behaviour.

Throws: `WorkflowException`

getStateName

`public String getStateName()`

Returns the name of the current State.

Returns: the name of the current State

getState

`public State getState() throws WorkflowException`

Returns the current `State` instance.

Returns: `State` the current `State` instance

Throws: `WorkflowException`

1.6. Class WorkflowException

Exception to signal problems thrown during workflow handling.

Constructors

`public WorkflowException()`

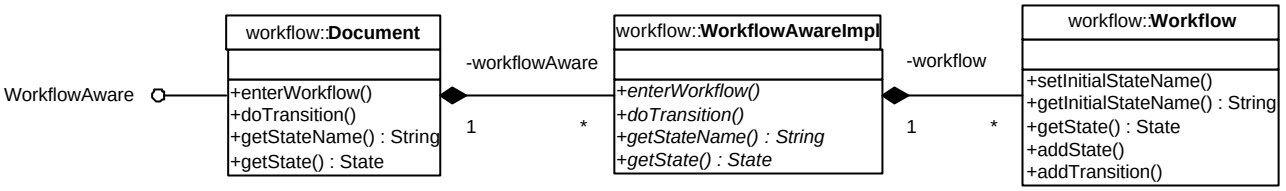
Constructs a WorkflowException object.

`public WorkflowException(String inSimpleMessage)`

Constructs a WorkflowException object with the specified message.

Parameters: `inSimpleMessage` The exception message.

2. Using the workflow packag



A document that is intended to take part in a workflow has to implement the interface `WorkflowAware`. This means, this class has to implement the workflow methods `enterWorkflow()`, `doTransition()`, `getStateName()` and `getState()`. However, the work is simplified by using the class `WorkflowAwareImpl`. The workflow aware document can create an instance of `WorkflowAwareImpl` and then delegate all calls to the workflow methods to this instance incorporated.

To initialize an instance of `WorkflowAwareImpl`, the workflow aware document has to create an instance of `Workflow`, which then can be entered as initializing parameter to the constructor of `WorkflowAwareImpl`.

This is the moment the workflow aware document can define the workflow it takes part. A `Workflow` is defined by adding various states and transitions to get from one state to the other. After defining the `Workflow` and handing it over to `WorkflowAwareImpl`, all calls of the workflow methods can safely delegated to `WorkflowAwareImpl`.

The only thing left to do is to implement the functionality that has to be carried out when state transition occurs. This is done by implementing methods using a specific naming scheme. For example, if a Workflow has two states *Private* and *Public*, and a transition *Publish* that goes from *Private* to *Public*, the following happens when the transition is executed:

1. if implemented, the method `onLeave_Private()` is called (it is called each time the object leaves the *Private* state)
2. if implemented, the method `onTransition_Publish()` is called (it is called whenever this transition is executed)
3. if implemented, the method `onEnter_Public()` is called (it is called each time the object enters the *Public* state)

Therefore, it the application has something to do if the document leaves the *Private* state, you can implement this behaviour in a method of the document named `onLeave_Private()`.

3. Example

The example shows the code of a `Document` with a `Workflow` consisting of two `State`s *Private* and *Public* and one `Transition` *Publish*.

```
import org.hip.kernel.workflow.*

public class Document implements WorkflowAware {
     //constants
1)   public final static String STATE_PRIVATE =       "Private";
     public final static String STATE_PUBLIC =        "Public";
     public final static String TRANS_PUBLISH =       "Publish";

     //instance variables
2)   WorkflowAwareImpl workflowAware;

     public Document() throws WorkflowException {
          super();
3)        workflowAware = new WorkflowAwareImpl(createWorkflow(),
               new Object[] {"Just created."}, this);
     }

4)   private Workflow createWorkflow() throws WorkflowException {
          Workflow lWorkflow = new Workflow();
          lWorkflow.addState(STATE_PRIVATE);
          lWorkflow.addState(STATE_PUBLIC);
          lWorkflow.addTransition(TRANS_PUBLISH, STATE_PRIVATE, STATE_PUBLIC);
          lWorkflow.setInitialStateName(STATE_PRIVATE);
          return lWorkflow;
     }


5)   public void doTransition(String inTransitionName, Object[] inArgs)
               throws WorkflowException {
          workflowAware.doTransition(inTransitionName, inArgs, this);
     }

     public void enterWorkflow(Workflow inWorkflow, Object[] inArgs)
               throws WorkflowException {
          workflowAware.enterWorkflow(inWorkflow, inArgs, this);
     }

     public void enterWorkflow(Workflow inWorkflow, String inInitialStateName,
               Object[] inArgs) throws WorkflowException {
          workflowAware.enterWorkflow(inWorkflow, inInitialStateName,
               inArgs, this);
     }

     public String getStateName() {
          return workflowAware.getStateName();
     }

     public State getState() throws WorkflowException {
          return workflowAware.getState();
```

```
    }

6)  public void onEnter_Private(String inMessage) {
        System.out.println("onEnter_Private: " + inMessage);
    }

    public void onLeave_Private(String inMessage) {
        System.out.println("onLeave_Private: " + inMessage);
    }

    public void onEnter_Public(String inMessage) {
        System.out.println("onEnter_Public: " + inMessage);
    }

    public void onLeave_Public(String inMessage) {
        System.out.println("onLeave_Public: " + inMessage);
    }

    public void onTransition_Publish(String inMessage) {
        System.out.println("onTransition_Publish: " + inMessage);
    }
```

1) Define the constants for the two states and the transition.

2) Define the `Document`'s instance variable for the instance of `WorkflowAwareImpl` the workflow aware object is delegating its workflow behaviour.

3) Initialize the document's `WorkflowAwareImpl` by creating the document's workflow.

4) The document's workflow is set up in a private method. The states and transition are added to the `Workflow` and the initial state is set.

5) All workflow aware methods are implemented simply by passing the call to `WorkflowAwareImpl`.

6) The functionality carried out when state transition occurs is implemented. Be aware that the method names correspond to the values of the state and transition constants. Be sure that the method signature is the same for all methods.

```
import org.hip.kernel.workflow.*

   WorkflowAware lDocument = new Document();
   lDocument.doTransition(Document.TRANS_PUBLISH,
     new Object[] {"Publish this document."});
   lDocument.getStateName();
```

The code above shows how to use the workflow. First the workflow aware document is created. Then the document is published by calling the transition with the name of the *Publish* transition as parameter. In the last step the name of the document's actual state is queried.