Using the new fault tolerance features in Taverna Beta8

Tom Oinn, tmo@ebi.ac.uk, 13th Feb 2004

Introduction

The latest version of Taverna incorporates new features to help make your workflows more robust in the event of service failure. These features can be summarized as follows:

- Retry processor instances in the event of failures
- Configurable back off of delays between retries
- Flexible specification of alternate processors to use when retries have been exceeded or are not applicable

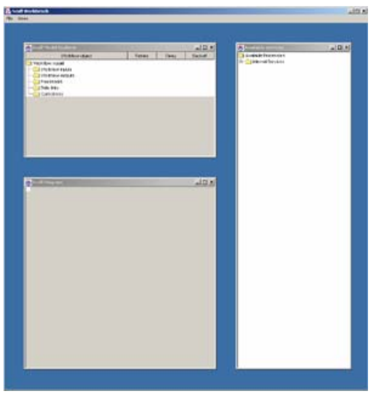
In order to make these new features visible from the Taverna Workbench, we have also enhanced several of the graphical components. This document gives an overview of these changes and walks you through the process of creating a fault tolerant workflow using a simple example using local service implementations for simplicity.

Screenshots in this document are taken from the latest workbench build running with the MS Windows look and feel, the precise appearance of your version may differ depending on the system you're running it on.

Example

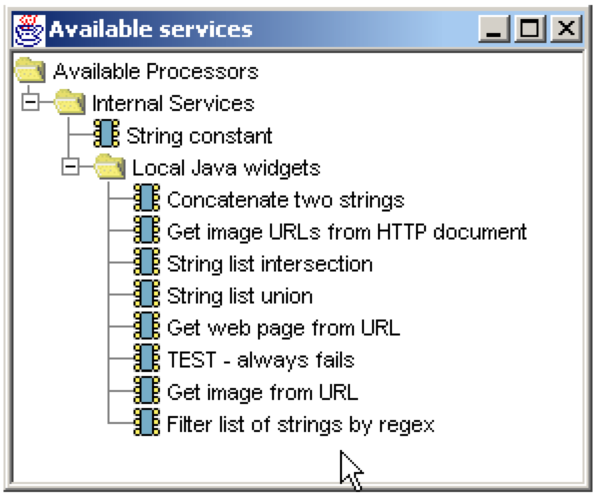
1) Launch the Taverna Workbench

The workbench should appear with three active windows. The top left contains a new version of the Scufl Model Explorer component; the Service Selection Panel and Scufl Diagram are unchanged from the previous version, at least in the current view (there are couple of extra features in both these components that will become evident shortly).



2) Expand the Service Selection Panel

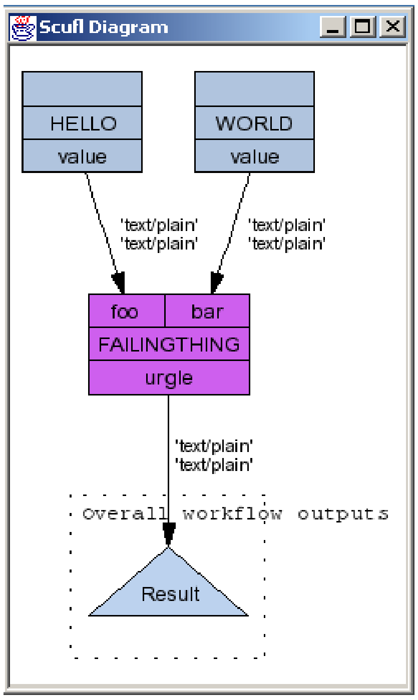
You will need three components from the service selection panel, all of which are already present when the application is started. To make these visible, click on any unexpanded items in the tree to show the entire list.



Available Processors
- Internal Services
  - String constant
  - Local Java widgets
    - Concatenate two strings
    - Get image URLs from HTTP document
    - String list intersection
    - String list union
    - Get web page from URL
    - TEST - always fails
    - Get image from URL
    - Filter list of strings by regex

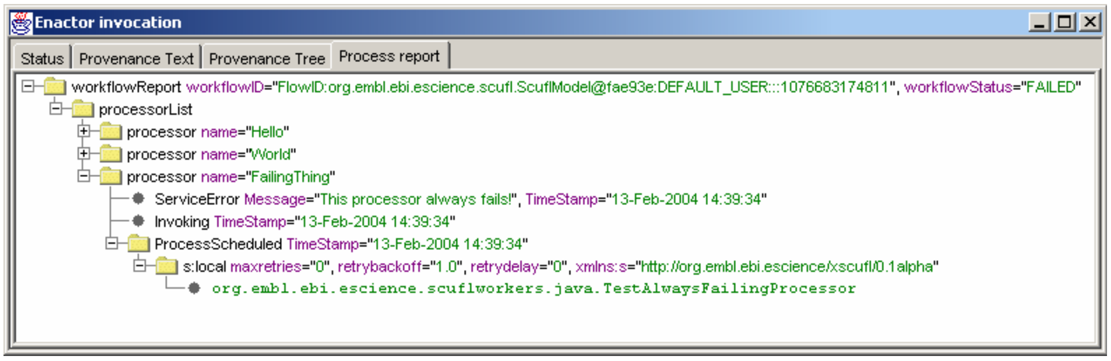
3) Create the first version of the workflow

For this workflow you will need to create two 'String constant' processors, an instance of the 'TEST - always fails' processor and a single workflow output, linking them together as shown in the diagram below. As you might deduce from the name, the test processor is one that fails in an immediate and fatal fashion whenever it is invoked, and is therefore obviously not a sensible choice for a real workflow! It is, however, a good demonstration of how to handle failures so we'll use it in this example.



4) Invoke the workflow

Once you've constructed the workflow in step 3 you can attempt to run it. It will come as no surprise (hopefully!) that this workflow fails; an inspection of the 'Process report' tab in the enactor invocation panel should give you more information as to why:

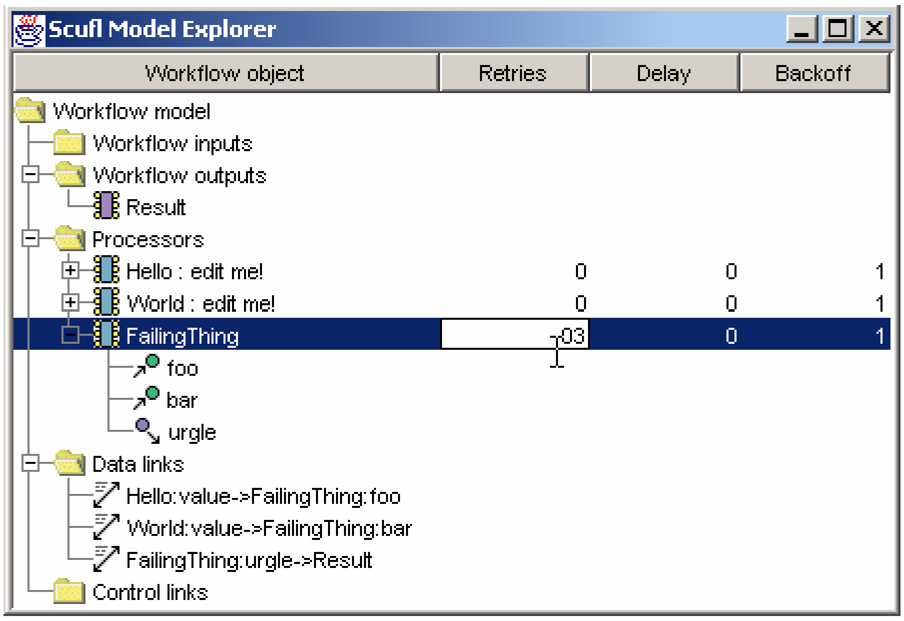


```
workflowReport workflowID="FlowID:org.embl.ebi.escience.scufl.ScuflModel@fae93e:DEFAULT_USER:::1076683174811", workflowStatus="FAILED"
  processorList
    processor name="Hello"
    processor name="World"
    processor name="FailingThing"
      ServiceError Message="This processor always fails!", TimeStamp="13-Feb-2004 14:39:34"
      Invoking TimeStamp="13-Feb-2004 14:39:34"
      ProcessScheduled TimeStamp="13-Feb-2004 14:39:34"
        s:local maxretries="0", retrybackoff="1.0", retrydelay="0", xmlns:s="http://org.embl.ebi.escience/xscufl/0.1alpha"
          org.embl.ebi.escience.scuflworkers.java.TestAlwaysFailingProcessor
```

Clearly, the test processor that always fails has failed (well really...) In case it isn't obvious, the events for each processor should be read upwards, the most recent being at the head of the list.

5) Modifying the workflow to include retries

Feigning gross stupidity, you're going to modify the workflow to retry the failing operation in complete oblivion to the 'This processor always fails' message. In order to do this you need to go to the Scufl Model Explorer window. You should be able to see the three processors in the workflow displayed in the tree, if not you'll have to expand the 'Processors' node:



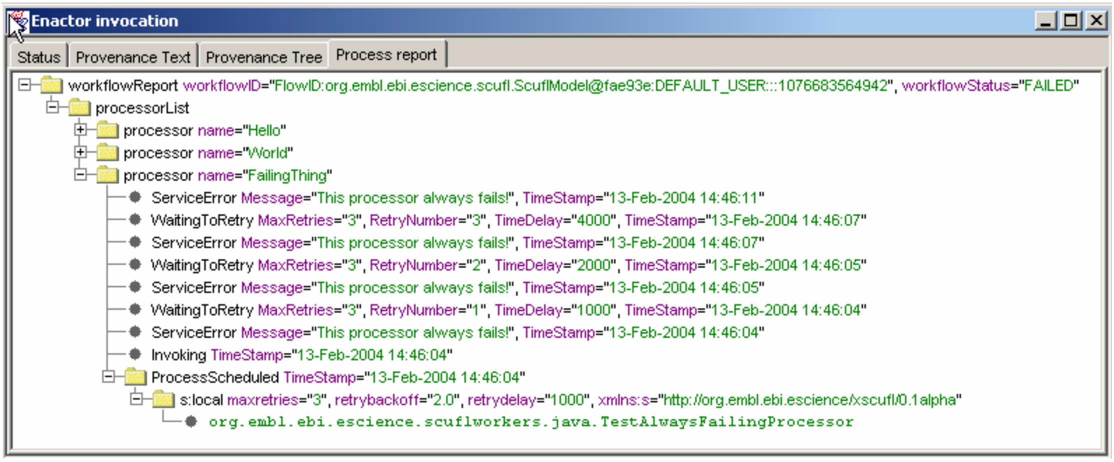
```
Workflow object                      Retries    Delay    Backoff
Workflow model
  Workflow inputs
  Workflow outputs
    Result
  Processors
    Hello : edit me!                       0        0          1
    World : edit me!                       0        0          1
    FailingThing                          03        0          1
      foo
      bar
      urgle
  Data links
    Hello:value->FailingThing:foo
    World:value->FailingThing:bar
    FailingThing:urgle->Result
  Control links
```

To the right of each processor in the tree display you should see three numbers, you can edit these simply enough by clicking the mouse on them and typing. Depending on the operating system you are running on the arrow keys may or may not allow you to move between the numbers, have a go and see.

These three numbers define the retry behaviour of the processor they are adjacent to. The first number is an integer number of times that the process will be retried. By default this is zero corresponding to the behaviour of no retries. In your case you have decided (you know this because I'm telling you) that three would be a sensible enough number of times to retry the obviously non functional process, so go to the retry number for the always failing processor and enter 3. This by itself is not adequate to define the behaviour, there are two more numbers. The first of these is the number of milliseconds to wait between retries, I suggest setting it to 1000, or one second in this case. The last number is slightly more complex - this is a back off factor for the retry delay. The delay between the first failure and first retry is always the number you entered for the delay; however, the delay between subsequent retries can be extended using this factor. The rule is that the delay time is multiplied by the back off factor for each additional retry delay beyond the first. Setting this value to two, for example, would mean that the first retry is 1 second, the second is 2 seconds and the third and final retry four seconds. The default value of 1.0 here indicates that there is no back off; the delay is uniform between all retries.

6) Reinvoking the workflow with retry behaviour enabled

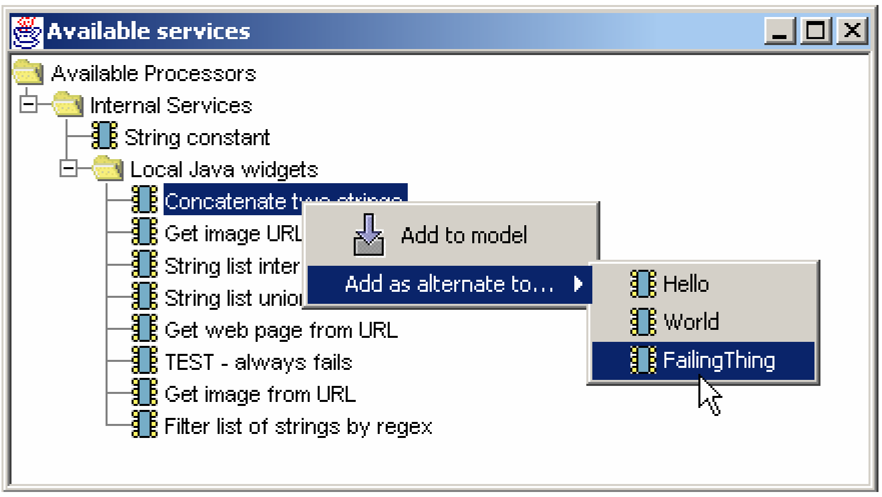
As you would expect, the workflow still doesn't actually complete successfully, after all, trying something that never works four times is no more useful than running it once, this is just to show you what appears in the process report, showing the events corresponding to retries - this assumes that you set the retry count to 3, delay to 1000 and back off to 2.0:



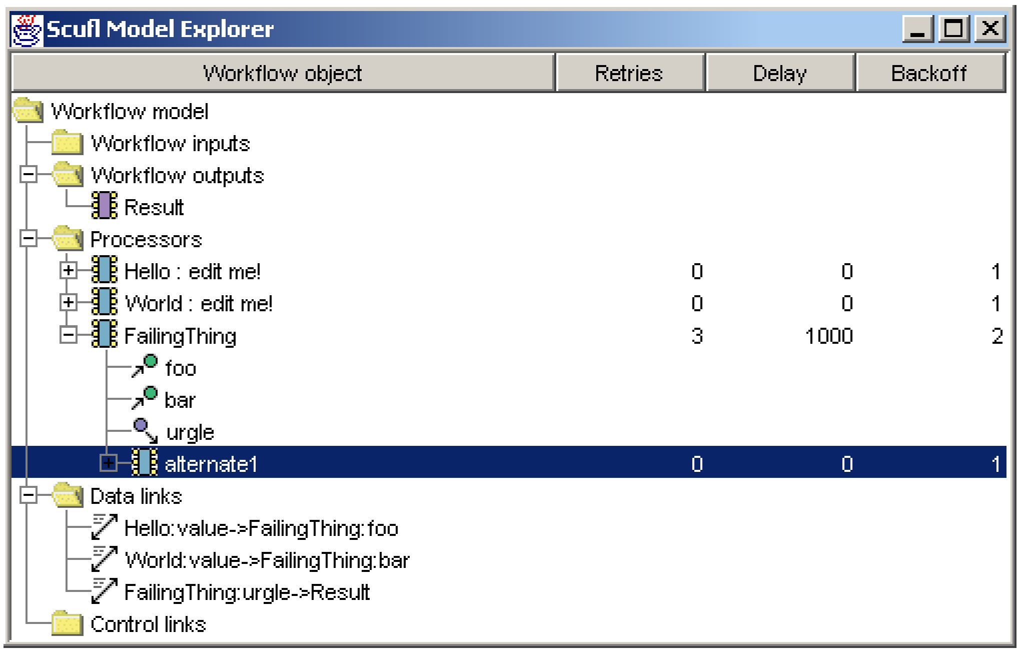
```
workflowReport workflowID="FlowID:org.embl.ebi.escience.scufl.ScuflModel@fae93e:DEFAULT_USER:::1076683564942", workflowStatus="FAILED"
  processorList
    processor name="Hello"
    processor name="World"
    processor name="FailingThing"
      ServiceError Message="This processor always fails!", TimeStamp="13-Feb-2004 14:46:11"
      WaitingToRetry MaxRetries="3", RetryNumber="3", TimeDelay="4000", TimeStamp="13-Feb-2004 14:46:07"
      ServiceError Message="This processor always fails!", TimeStamp="13-Feb-2004 14:46:07"
      WaitingToRetry MaxRetries="3", RetryNumber="2", TimeDelay="2000", TimeStamp="13-Feb-2004 14:46:05"
      ServiceError Message="This processor always fails!", TimeStamp="13-Feb-2004 14:46:05"
      WaitingToRetry MaxRetries="3", RetryNumber="1", TimeDelay="1000", TimeStamp="13-Feb-2004 14:46:04"
      ServiceError Message="This processor always fails!", TimeStamp="13-Feb-2004 14:46:04"
      Invoking TimeStamp="13-Feb-2004 14:46:04"
      ProcessScheduled TimeStamp="13-Feb-2004 14:46:04"
        s:local maxretries="3", retrybackoff="2.0", retrydelay="1000", xmlns:s="http://org.embl.ebi.escience/xscufl/0.1alpha"
          org.embl.ebi.escience.scuflworkers.java.TestAlwaysFailingProcessor
```

7) Modifying the workflow to include an alternate processor

In order to actually make the workflow do something, you can make use of the other new feature, that of alternative processors. In this case, you're going to provide an alternative to the always failing process in the form of a string concatenation operation. To do this, right click on the 'Concatenate two strings' processor in the service selection panel and open the 'Add as alternate to...' submenu, selecting the always failing processor:

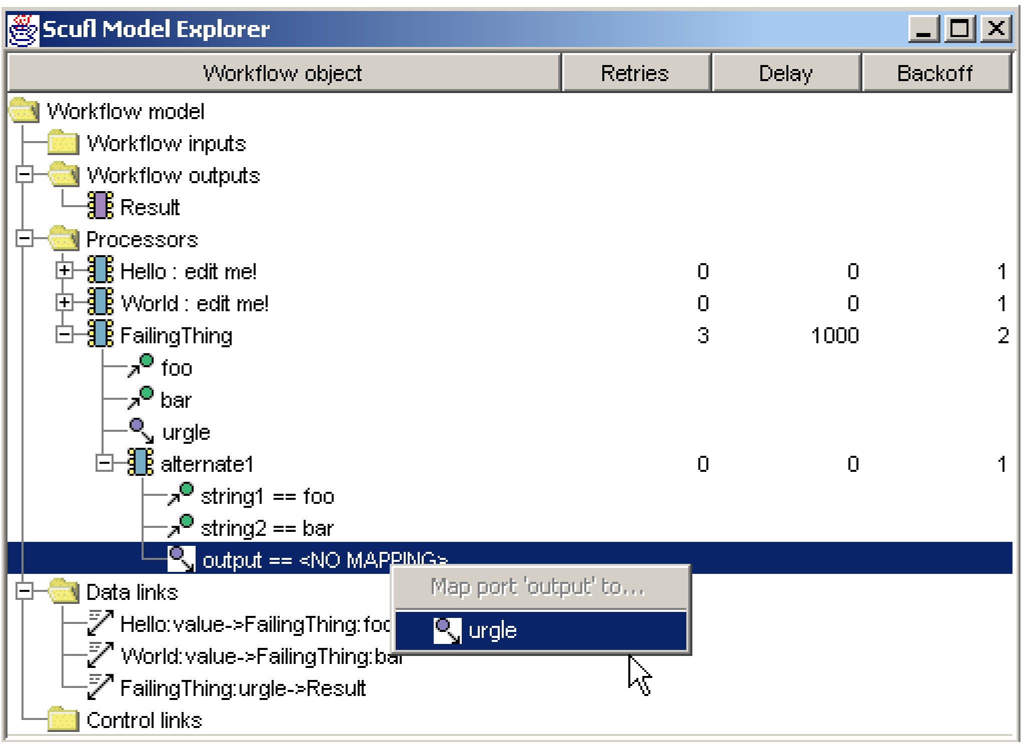


You should be able to see the alternate displayed as a child of the processor in the explorer panel:



Workflow object	Retries	Delay	Backoff
Workflow model			
Workflow inputs			
Workflow outputs			
Result			
Processors			
Hello : edit me!	0	0	1
World : edit me!	0	0	1
FailingThing	3	1000	2
foo			
bar			
urgle			
alternate1	0	0	1
Data links			
Hello:value->FailingThing:foo			
World:value->FailingThing:bar			
FailingThing:urgle->Result			
Control links			

One problem with using this processor as an alternative is that the names of the input and output ports are different to those of the processor it is substituting for. In order to fully specify the alternate you must tell it which ports in the alternate are equivalent to the ports in the original. You do this by right clicking on the ports within the alternate processor (you may have to further expand the tree to see them) and selecting the original port name you wish to bind the port to. Bind 'string1' to 'foo' and 'string2' to 'bar', with the 'output' port being bound to the only available output, ‘urgle’:



Scufl Model Explorer

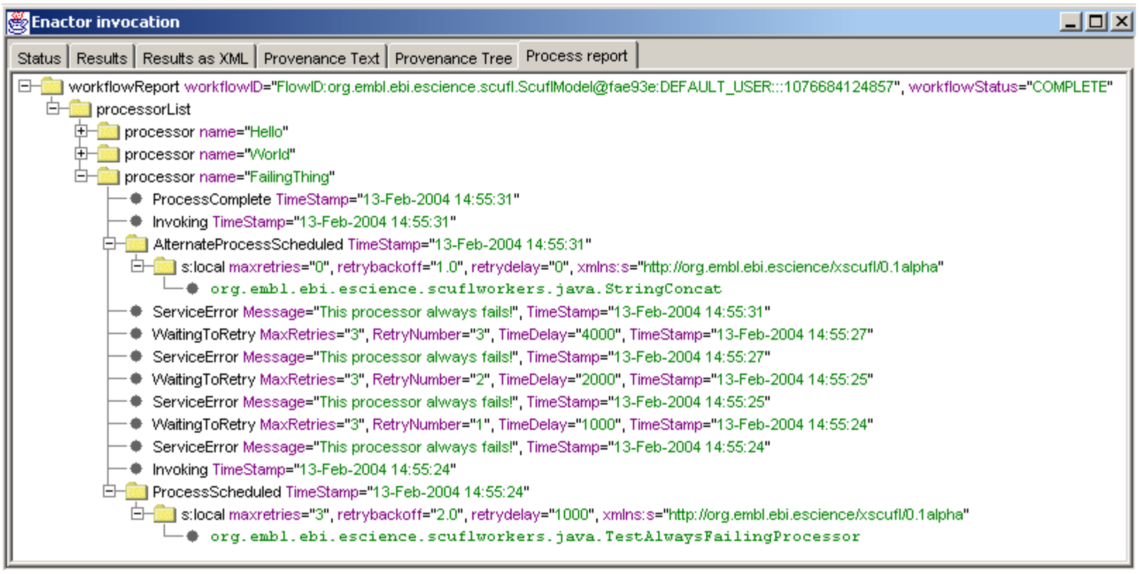
Workflow object	Retries	Delay	Backoff
Workflow model			
Workflow inputs			
Workflow outputs			
Result			
Processors			
Hello : edit me!	0	0	1
World : edit me!	0	0	1
FailingThing	3	1000	2
foo			
bar			
urgle			
alternate1	0	0	1
string1 == foo			
string2 == bar			
output == <NO MAPPING>			
Data links			
Hello:value->FailingThing:fo			
World:value->FailingThing:ba			
FailingThing:urgle->Result			
Control links			

You could also at this point modify the retry behaviour of the alternate itself, this works in exactly the same way as doing so for the main processor.

8) Invoking the workflow with retry and alternate behaviour enabled

Invoking the workflow this time should result in success. The enactment engine will try the processor that always fails several times as before, but this time after the final retry it will reschedule itself using the first alternate in the list (you only have one in this case, you can add further alternates by repeatedly selecting the 'add as alternate to...' option). As the alternate does work, the workflow should complete and return you a 'hello world' string.

Examining the process report should show what has happened. The service failure is the same as before, but note the new event indicating that an alternate process has been scheduled, the subsequent invocation of which succeeds:



```
workflowReport workflowID="FlowID:org.embl.ebi.escience.scufl.ScuflModel@fae93e:DEFAULT_USER:::1076684124857", workflowStatus="COMPLETE"
  processorList
    processor name="Hello"
    processor name="World"
    processor name="FailingThing"
      ProcessComplete TimeStamp="13-Feb-2004 14:55:31"
      Invoking TimeStamp="13-Feb-2004 14:55:31"
      AlternateProcessScheduled TimeStamp="13-Feb-2004 14:55:31"
        s:local maxretries="0", retrybackoff="1.0", retrydelay="0", xmlns:s="http://org.embl.ebi.escience/xscufl/0.1alpha"
          org.embl.ebi.escience.scuflworkers.java.StringConcat
      ServiceError Message="This processor always fails!", TimeStamp="13-Feb-2004 14:55:31"
      WaitingToRetry MaxRetries="3", RetryNumber="3", TimeDelay="4000", TimeStamp="13-Feb-2004 14:55:27"
      ServiceError Message="This processor always fails!", TimeStamp="13-Feb-2004 14:55:27"
      WaitingToRetry MaxRetries="3", RetryNumber="2", TimeDelay="2000", TimeStamp="13-Feb-2004 14:55:25"
      ServiceError Message="This processor always fails!", TimeStamp="13-Feb-2004 14:55:25"
      WaitingToRetry MaxRetries="3", RetryNumber="1", TimeDelay="1000", TimeStamp="13-Feb-2004 14:55:24"
      ServiceError Message="This processor always fails!", TimeStamp="13-Feb-2004 14:55:24"
      Invoking TimeStamp="13-Feb-2004 14:55:24"
      ProcessScheduled TimeStamp="13-Feb-2004 14:55:24"
        s:local maxretries="3", retrybackoff="2.0", retrydelay="1000", xmlns:s="http://org.embl.ebi.escience/xscufl/0.1alpha"
          org.embl.ebi.escience.scuflworkers.java.TestAlwaysFailingProcessor
```

9) Providing further alternates and manipulating their order

In order to create more than one alternate, simply repeat the process you used in step 7. In this case there is no advantage to doing this, as the alternate you have will always work, but in the real world you may well have more than one service available as an alternative, and this mechanism allows you to specify this. Removing an alternate is as simple as right clicking on the alternate processor in the explorer view and selecting delete. In addition to deleting processors using this menu, if there is more than one alternate you will be presented with options to move the selected alternate up or down in the list (promote or demote). The enactor will use the alternates in the order they appear in this view, only moving onto subsequent alternates if the current one fails and exceeds any retries.

Conclusion

You should now be aware of how to use the new features in Taverna Beta8 regarding reliability, hopefully these will help make your workflows more robust. We have further plans in the pipeline, in particular configurable 'on fail' behaviour, timeout handling and checkpointing, but this current set should be useful in its own right.