MANCHESTER 1824
The University of Manchester

myGrid

Advanced Taverna

Stian Soiland-Reyes and Christian Brenninkmeijer
University of Manchester

materials by Katy Wolstencroft, Aleksandra Pawlik, Alan Williams

http://orcid.org/0000-0001-9842-9718
http://orcid.org/0000-0002-2937-7819
http://orcid.org/0000-0002-1279-5133
http://orcid.org/0000-0001-8418-6735
http://orcid.org/0000-0003-3156-2105

Bonn University, 2014-09-01 / 2014-09-03
http://www.taverna.org.uk/

Advanced Exercises

- The Taverna engine can also help you control the data flow through your workflows. It allows you to manage iterations and loops, add your own scripts and tools, and make your workflows more robust
- The following exercises give you a brief introduction to some of these features
- Like in the previous tutorial workflows in this practical use small data-sets and are designed to run in a few minutes. In the real world, you would be using larger data sets and workflows would typically run for longer

List handling - cross or dot product

As you may have already seen, Taverna can automatically iterate over sets of data, calling a service multiple times for each value in the input list.

When 2 sets of iterated data are combined (one to each input port), Taverna needs extra information about how they should be combined. You can have:

- **A cross product** – combining every item from port A with every item from port B - ***all against all***
- **A dot product** – only combining item 1 from port A with item 1 from port B, item 2 with item 2, and so on – ***line against line***

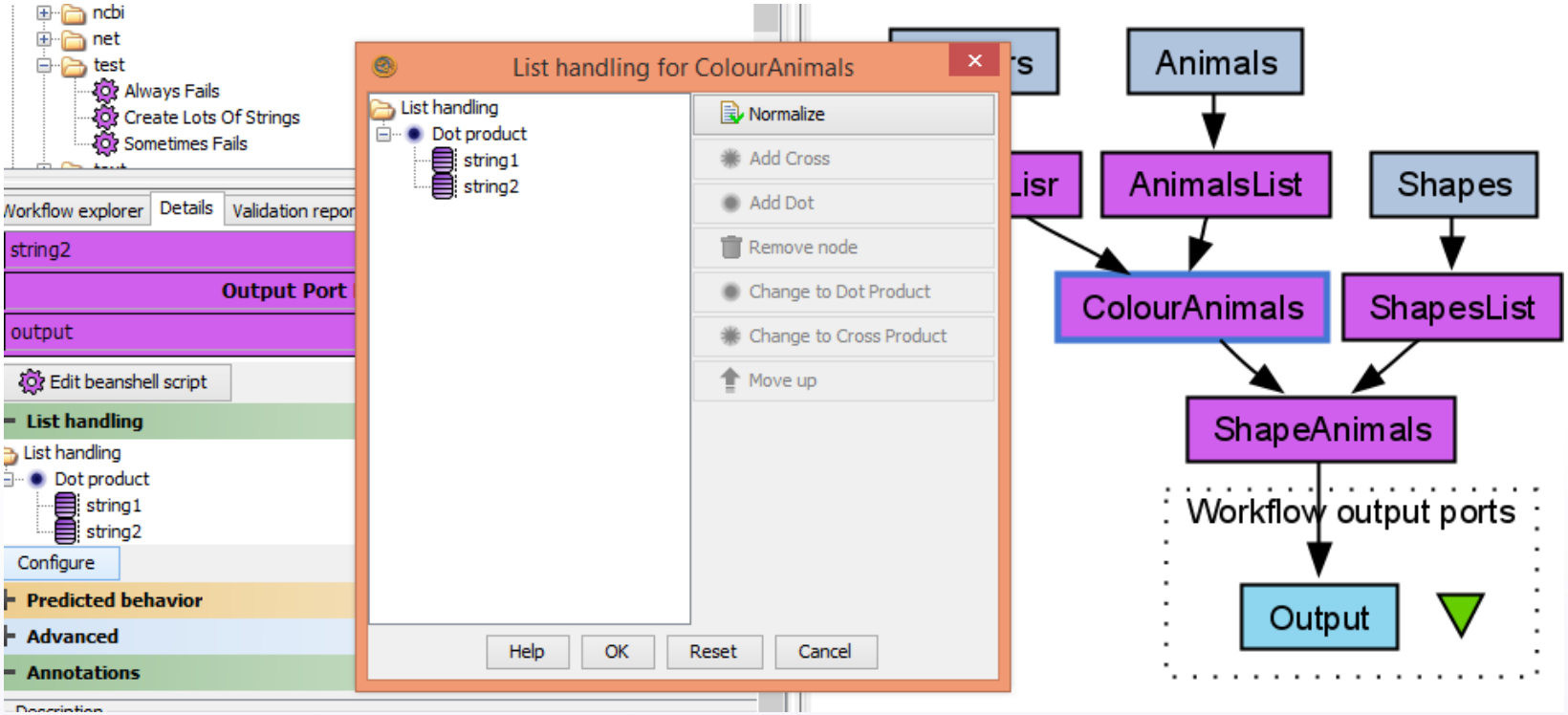
List handling – example workflow

- Download and open the workflow "*Demonstration of configurable iteration*" from http://www.myexperiment.org/workflows/4332
  - Or see "Run this workflow in Taverna" on myExperiment, and copy the link into File -> Open Workflow Location
- Read the workflow metadata to find out what the workflow does (by looking at the 'Details')
- Run the workflow and look at the **results**
- Click on individual services to inspect the *intermediate values* and multiple invocations for:
  - AnimalsList, ColourAnimals, ShapeAnimals
  - Alternatively, add additional workflow output ports from AnimalsList and ColourAnimals, and rerun.

List handling - configuration

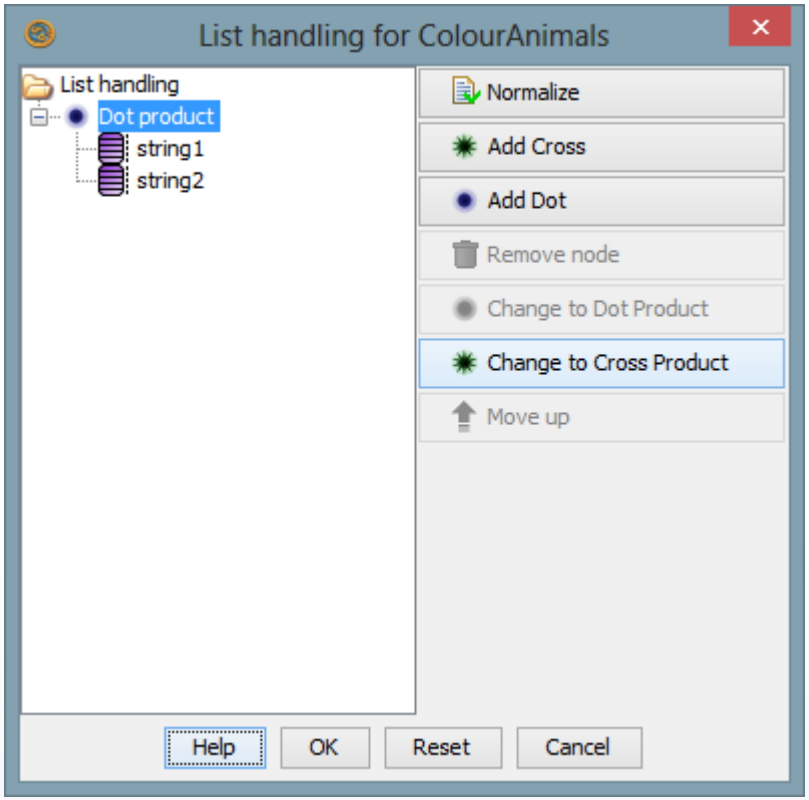
- Go back to the **Design** view
- Select the *ColourAnimals* service by clicking on it
- Select the **Details** tab in the workflow explorer, open **List handling** and click on **Configure**,
- or right-click on *ColourAnimals*, select **Configure running…** then **List handling…**
- Click on **Dot product** in the pop-up window. This allows you to switch to cross product (see the next slide)

List handling– configuring - 1



List handling – configuring - 2

- Click on **Dot Product**
- Click **Change to Cross Product** on the right
- Click **OK**
- Run the workflow again



List handling - difference

- What is the difference between the results of the two runs? What does it mean to specify dot or cross product?

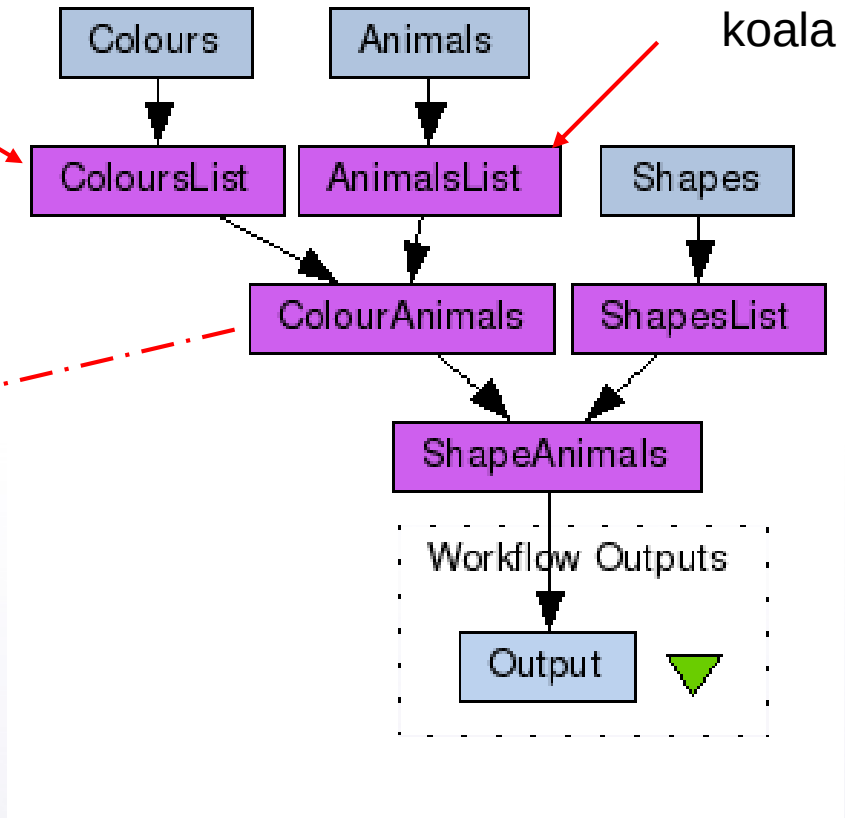
  NOTE: The iteration strategies are very important. Setting cross product instead of dot when you have 2000x2000 data items can cause large and unnecessary increases in computation!

List handling - workflow

e.g. red, green, blue, yellow

e.g. cat, donkey, koala

How does Taverna combine them?



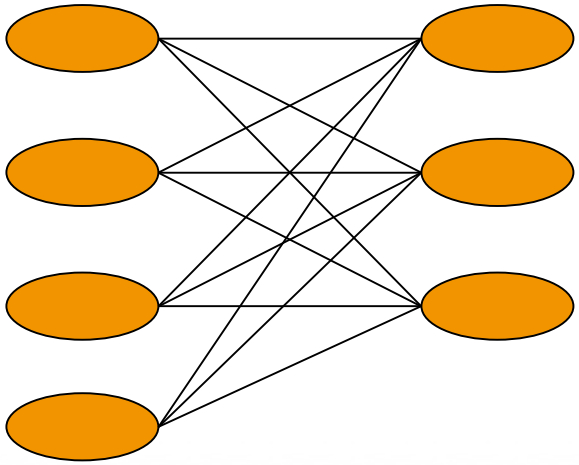
List handling - Cross product

Red

Green

Blue

Yellow



Cat

Donkey

Koala

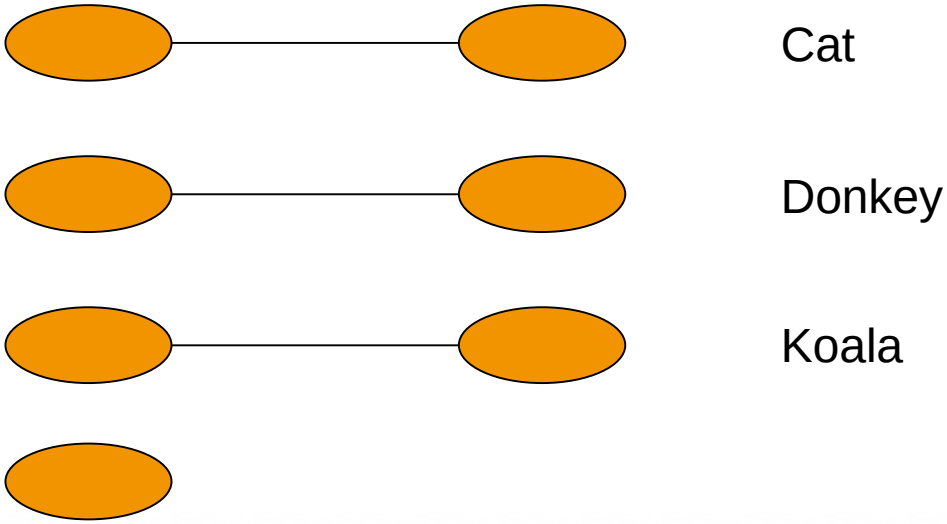
Red cat, red donkey, red koala

Green cat, green donkey, green koala

Blue cat, blue donkey, blue koala

Yellow cat, yellow donkey, yellow koala

List handling - Dot product



Red cat

Green donkey

Blue koala

There is no yellow animal because the list lengths don't match!

List handling - summary

- The default in Taverna is cross product
- Be careful! All against all in large iterations give very big numbers!
- For more complex list handling, e.g. combination of 3 or more ports, see http://dev.mygrid.org.uk/wiki/display/tav250/List+handling

Looping asynchronous services

- Find the workflow "*EBI_InterproScan_broken*" in the workshop pack on myExperiment
- InterproScan analyses a given protein sequence (or set of sequences) for functional motifs and domains
- This workflow is **asynchronous**. This means that when you submit data to the 'runInterproScan' service, it will return a jobID and place your job in a queue (this is very useful if your job will take a long time!)
- The 'Status' nested workflow will query your job ID to find out if it is complete

Looping

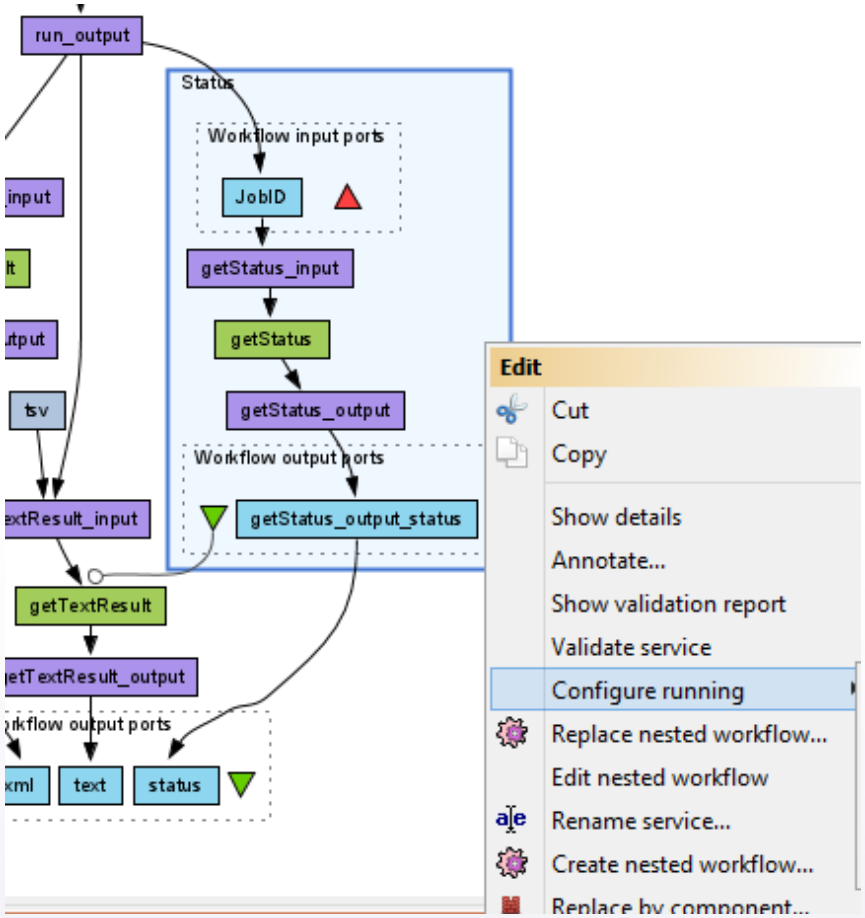
The default behaviour in a workflow is to call each service only once for each item of data – so what if your job has not finished when 'Status' workflow asks?

- Download and run the workflow, using the default protein sequence and your own email address
- Almost every time, the workflow will fail because the results are not available before the workflow reaches the 'get_results' service – the '*status*' output is still `RUNNING`

Looping

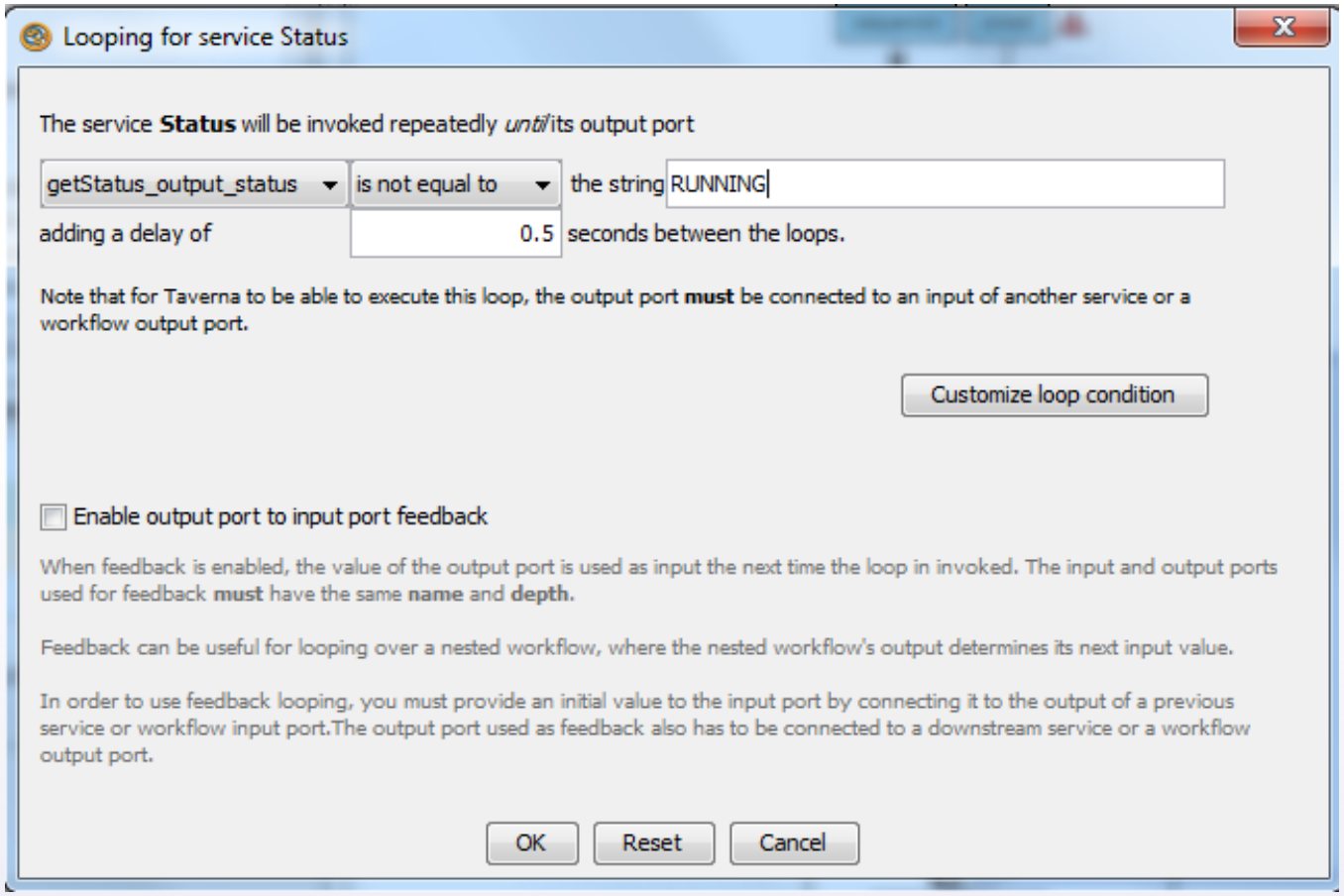
- This is where looping is useful. Taverna can keep running the *Status* service *until* it reports that the job is done.
- Go back to the **Design** view
- Select the *Status* nested workflow
- Select the **Details** tab in the workflow explorer, open **Advanced** and click on **Add looping**,
- or right-click on *Status*, select **Configure running…** then **Looping…**
  - **(Example on next slide)**

Looping


run_output
Status
Workflow input ports
JobID
getStatus_input
getStatus
getStatus_output
Workflow output ports
getStatus_output_status
getTextResult
text
status
Edit
Cut
Copy
Show details
Annotate...
Show validation report
Validate service
Configure running
Replace nested workflow...
Edit nested workflow
Rename service...
Create nested workflow...
Replace by component...

Looping

- Use the drop-down boxes in the looping window to set *getStatus_output_status* **is not equal to** *RUNNING*

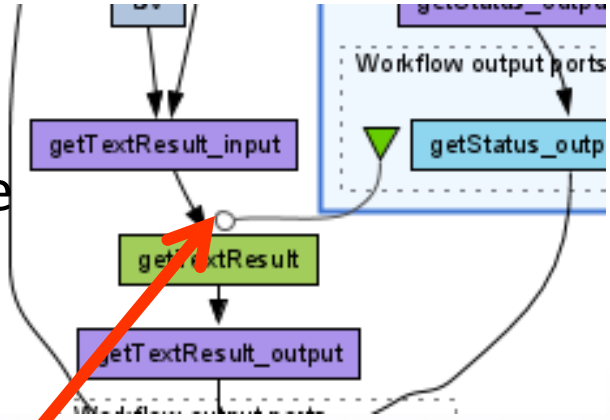


Looping

- Save the workflow and run it again
- This time, the workflow will run until the ‘Status’ nested workflow reports that it is either DONE, or it has an ERROR.
- You will see results for *text*, but you will still get an error for ‘*xml*’. This is because there is one more configuration to change – we also need **Control Links** to delay the exectution of *getXmlResult*.

Control Links

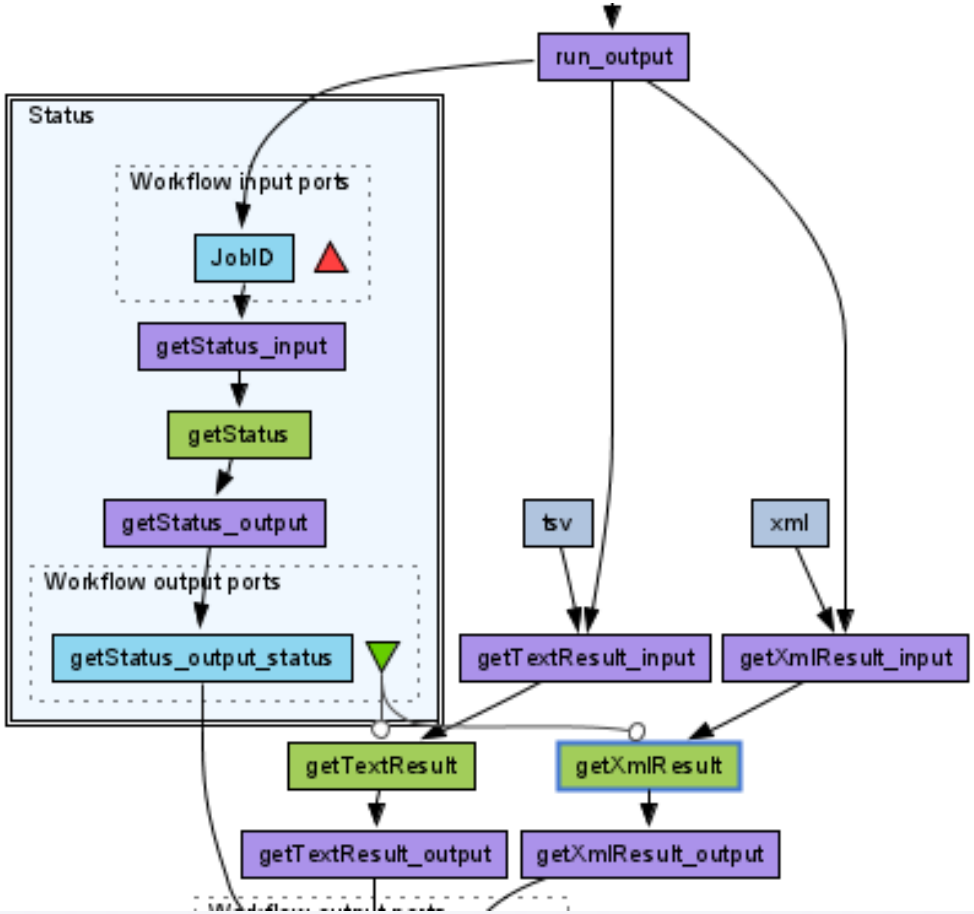
- Normally a service in a workflow will run as soon as all its input ports are available – even if graphically it may be "further down"
- A **control link** specifies that there is a dependency on another service even if there is no direct or indirect data flowing between them.
  - In a way the data still flows, but internally on the called service, outside the workflow
- A control link is shown as a line with a white circle at the end. In our workflow this means that *getTextResult* will not run until the *Status* nested workflow is finished



Control Links

- We will add control links to fix the '*xml*' output
- Switch to the **Design** view
- Right-click on *getXmlResult* and select **Run after** from the drop down menu.
- *getXmlResults* is moved down in the diagram, showing the new control link
- Set it to **Run after** -> *Status*
- Save and run the workflow
- Now you will see that *getXmlResults* and *getTextResults* take a bit longer before they run
- This time, results are available for both *xml* and *text*

Control Links

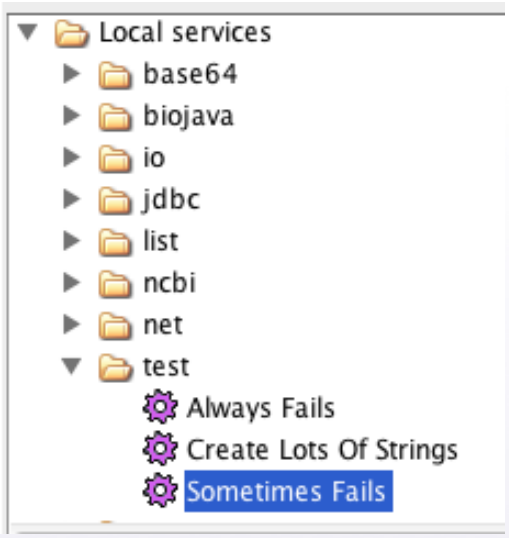


Retries: Making your Workflow Robust

- Web services can sometimes fail due to network connectivity
- If you are iterating over lots of data items, this is more likely to cause problems because Taverna will be making lots of network connections.
- You can guard against these temporary interruptions by adding **retries** to your workflow
- As an example, we'll use two local services to emulate iteration and occasional failures.
- Click a *File* -> **New workflow**

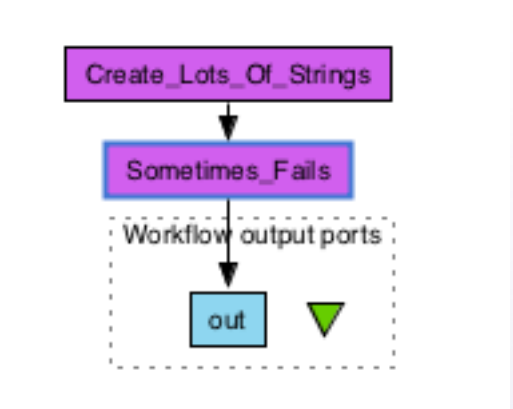
Retries: Making your Workflow Robust

- In the **Service panel**,
- Select the service *Create Lots Of Strings* under **Available Services** -> **Local services** -> **test**
- Add it to the workflow by dragging it into the workflow diagram
- Also add *Sometimes Fails*



Retries: Making your Workflow Robust

- Add an output port and connect the service as on the picture below
- Run the workflow as it is and count the number of failed iterations. (Tip: Change **view values** to **view errors**)
- Run the workflow again. Is the number the same?
- Inspect the **intermediate values** at *Sometimes_fails*.

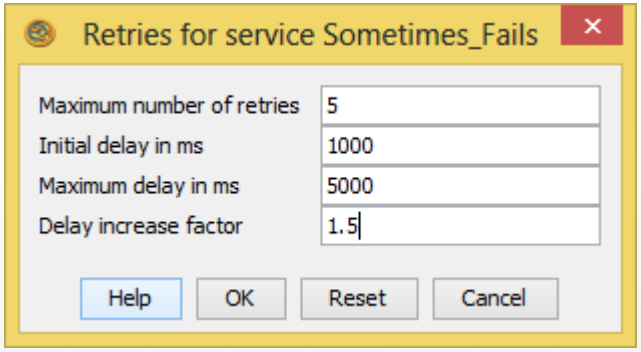


Retries: Making your Workflow Robust

- Now, select the *Sometimes_Fails* service and select the **Details** tab in the workflow explorer panel
- Click on **Advanced** and **Configure** for **Retry**
- In the pop-up box, change it so that it retries each service iteration 2 times
- Run the workflow again – how many failures do you get this time? Did you notice the slow down due to retries?
- Change the workflow to retry 5 times – does it work every time now?

Retries: Making your Workflow Robust

- In network communication, a common strategy for handling errors is to incrementally wait longer and longer before a retry – improving chance of recovery.
- In Taverna Retries this can be set by modifying "Delay increase factor" and "Maximum delay2.
- The settings on the right would retry after delays of:



1. 1.0 s
2. 1.5 s (1.0 s * 1.5)
3. 2.3 s (1.5 s * 1.5)
4. 3.4 s (2.3 s * 1.5)
5. 5.0 s (3.4 s * 1.5 = 5.1s) – above max 5.0 s

Parallel Service Invocation

- If Taverna is iterating over lots of independent input data, you can often improve the efficiency of the workflow by running those iterated jobs in parallel
- Run the Retry workflow again and time how long it takes
- Go back to the **Design** window, right-click on the 'sometimes_fails' service, and select '*configure running*'
- This time select '*Parallel jobs*' and change the maximum number to 20
- Run the workflow again
- Does it run faster?

Parallel Service Invocation : Use with Caution

- Setting parallel jobs usually makes your workflows run faster (at a cost of more memory/cpu usage)
  - Be careful if you are using **remote services**. Sometimes they have policies for the number of concurrent jobs individuals should run (e.g. The EBI ask that you do not submit more than 25 at once).
  - If you exceed the limits, your service invocations may be blocked by the provider. In extreme cases, the provider may block your whole institution!
  - Some remote services don't handle parallel calls well, as it could cause concurrency issues server side – e.g. overwriting internal files.
- A good number of concurrent jobs can be anything between 3 and 20 – **trial and error** is as important as checking the **service documentation**.