Web Services Demos Guide

Version 1.2

July, 2008

Contents

1. Introduction

The volume of information is growing at an exponential rate. One of the greatest scientific and engineering challenges of the 21st century is to effectively understand and leverage this growing wealth of data. To analyze and understand large volumes of data, complex computational processes need to be assembled, be it to mine the data or to create insightful visualizations. Recently, workflows have emerged as a paradigm for representing and managing complex computations. Workflows can capture complex analyses processes at various levels of detail and provide the provenance information necessary for reproducibility, result publication and sharing among collaborators.

VisTrails, is a provenance management system that introduces a set of new technologies to support and streamline exploratory processes through workflows. Whereas workflows have been traditionally used to automate repetitive tasks, for applications that are exploratory in nature, change is the norm. As a scientist generates and evaluates hypotheses about data under study, a series of different, albeit related, workflows are created while the computational task is adjusted in an interactive process. VisTrails was designed to manage rapidly-evolving, exploratory computational tasks.

By automatically capturing detailed history information about the exploration process and explicitly maintaining the relationships among the workflows created, VisTrails not only allows results to be reproduced, but it also enables users to efficiently and effectively navigate through the space of workflows used in an exploration task. In addition, this provenance information is used to simplify the creation and maintenance of workflows; to optimize their execution; to provide scalable mechanisms for collaborative exploration of large parameter spaces in a distributed setting; and it serves as the basis of an infrastructure for knowledge sharing and re-use. An important goal of the VisTrails project is to produce tools that domain scientists who are not expert programmers can use. VisTrails provides intuitive, point-and-click interfaces that allow users to interact with and query the provenance information, including the ability to visually compare different workflows and their results.

The Web services interface provided by VisTrails can be used to quickly construct workflows by weaving together bioinformatics workflows. This will enable scientists to access more information distributed in different repositories and improve the data discovery process. In this guide, the user will learn how to invoke web services from within VisTrails workflows.

2. Enabling and configuring the webServices Package

In order to use web services in VisTrails, you need to ensure that the the webServices package is enabled. Within the Module Packages tab of the Preferences dialog, click the Configure button to open the configuration dialog for this package. Select the wsdlList and click on the Value field. This is where you will enter the URL(s) of the web service(s) you wish to access. If there is more than one URL, place a semicolon (;) between each URL. In other words, the URLs must be semicolon-delimited. You can type the web services URLs you want and then click on close. After closing the dialog, you need to disable then re-enable the webServices package in order to load the changes. Then, close the Preferences dialog.

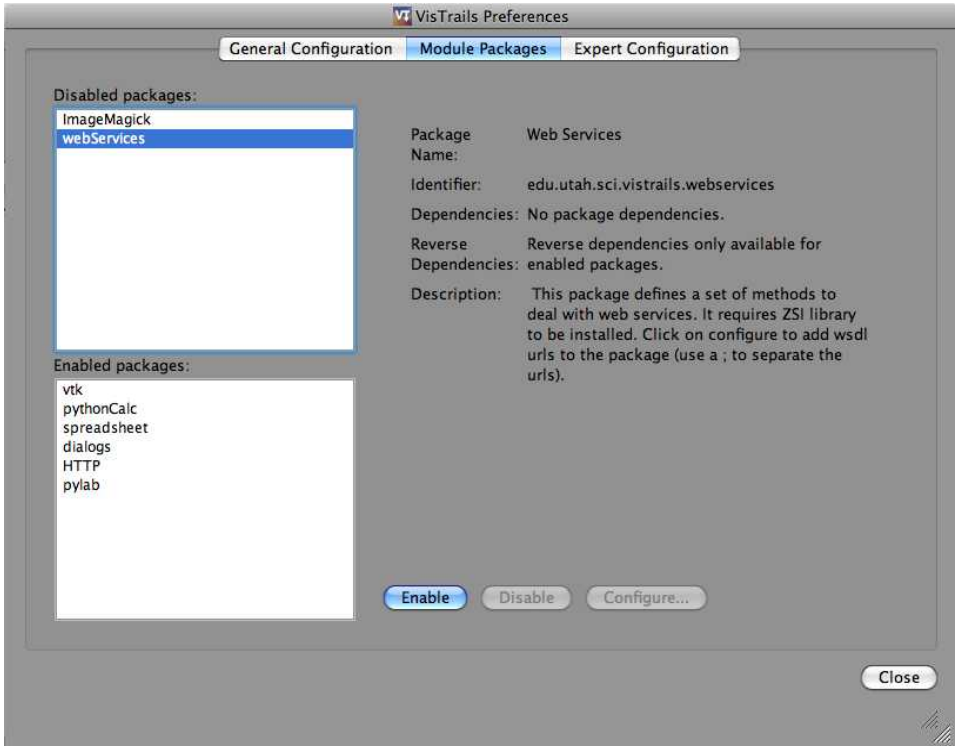


Figure 1: webServices package information is shown on VisTrails Preferences Dialog

After configuring the webServices package properly, you will see that there will be a tab "Web Services" in the Modules panel (see Figure 2). In this tab you will see the list of available web services specified by the user in VisTrails Preferences. A web service can have methods and complex types and in VisTrails each of them is a module. The modules that belong to a web service can be classified in two groups: Methods and Types. The Types group contains the complex types if there is any in the web service. The Methods group contains the published methods of the web service.

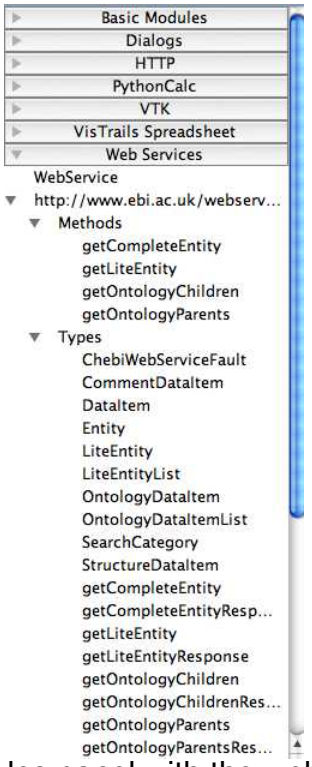


Figure 2: A view of the Modules panel with the webServices package enabled.

3. Web services Demos

In the following subsections we will explain how to build some of the web services demos that come with VisTrails. We will first begin with a simple example to familiarize the user with the web services interface and then construct more detailed ones.

3.1 DDBJ web service

In this demo a genetic sequence is retrieved from the DDBJ[1] database and is displayed in three different ways: XML, HTML and as a sequence element that has been extracted from the XML file. It is based on the 06-WebServicesAndDataTransformation.xml example provided by the Kepler[2] workflow system. You will need to add the DDBJ web service to VisTrails. The following is the URL: http://xml.nig.ac.jp/wsdl/DDBJ.wsdl.

3.1.1 XML Output

In this workflow we will invoke the getXMLEntry method of the DDBJ web service. An accession number will be sent as input parameter and an XML file that contains the equivalent genetic sequence will be returned.

Start with a new empty workflow in the Pipeline view, and drag the following modules to the canvas.

1 DDBJ is an acronym for DNA Data Bank of Japan.

2 Kepler is an open-source scientific workflow system that allows scientists to design scientific workflows and execute them efficiently using emerging Grid-based approaches to distributed computation.

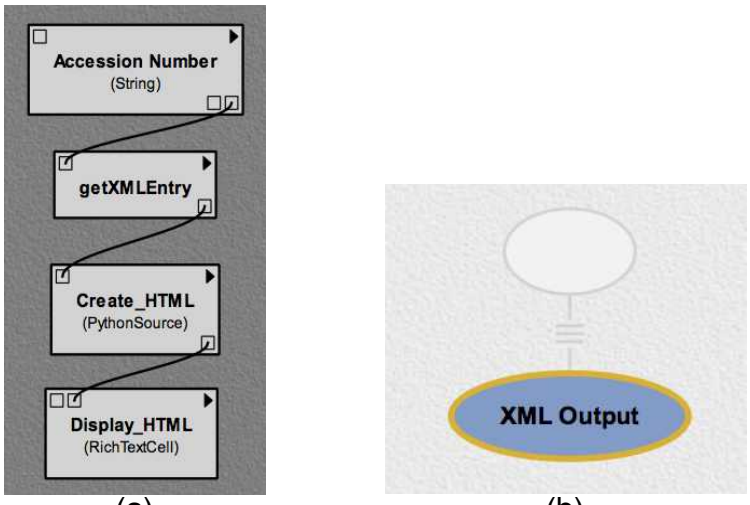
- String (under “Basic Modules”)
- getXMLEntry (under “Methods” for the current web service)
- PythonSource (under “Basic Modules”)
- RichTextCell (under “Spreadsheet”)

Give descriptive names to the modules. Let’s start with the PythonSource module that will be used to create the HTML file that will contain the results from executing the XML Output workflow. Click on the arrow that is on the top-right corner of the module box and select Set Module Label menu option and type the following into the text field: “Create_HTML” and click on the OK button. Set the label of modules RichTextCell to “Display_HTML” and String to “Accession Number”.

PythonSource modules has no input and output ports by default; we need to create them. Open the configuration dialog for PythonSource by selecting this module in the pipeline canvas and typing ‘Ctrl-E’. Add a new input port named “sequencexml” of type String, and a new output port named “outfile” of type File. We will now add some Python code to this module. This code generates a HTML document based on the information retrieved from the web service query. Type or paste the following source code into the PythonSource configuration dialog:

```
outseqinput = self.getInputFromPort('sequencexml')
output1 = self.interpreter.filePool.create_file()
f1 = open(str(output1.name), 'w')
text = '<HTML><TITLE>Data Transformation Webservice</TITLE><BODY BGCOLOR="#FFFFFF">'
f1.write(text)
text = '<H2>XML output</H2><BR>'
f1.write(text)
#Replace symbols to be able to display the xml content in the html file
outseqinput = outseqinput.replace("<","&lt;")
outseqinput = outseqinput.replace(">","&gt;")
splitnewline = outseqinput.split('\n')
for line in splitnewline:
    f1.write(line)
    f1.write('<BR>')
text = '</BODY></HTML>'
f1.write(text)
self.setResult("outfile",output1)
f1.close()
```

Click OK to close the dialog, and connect the modules together as shown in Figure 3(a).



(a) (b)

Figure 3: (a) XML Output pipeline. (b) Corresponding version tree.

The getXMLEntry method receives an accession number as input. In Bioinformatics an accession number is a unique identifier given to a DNA or protein sequence. In the String module labeled as Accession Number, set the parameter value with the following DNA sequence accession number: “AA045112”.

We will name the pipeline as “XML Output”. To do that, we need to switch to the Version Tree View by pressing the History button on the VisTrails toolbar. In the version tag field located in the right side of the view, type “XML Output” and then press 'Enter'. Your version tree should look similar to the one in Figure 3(b). Give a name to your file, such as “DataTransformation_webservice.vt” and save your work.

The workflow is now ready to be visualized. As we have a RichTextCell module, pressing the Execute button of the VisTrails toolbar will send the current pipeline with the current parameters to the VisTrails Spreadsheet. The result of executing this pipeline is an XML file with a genetic sequence (see Figure 4).

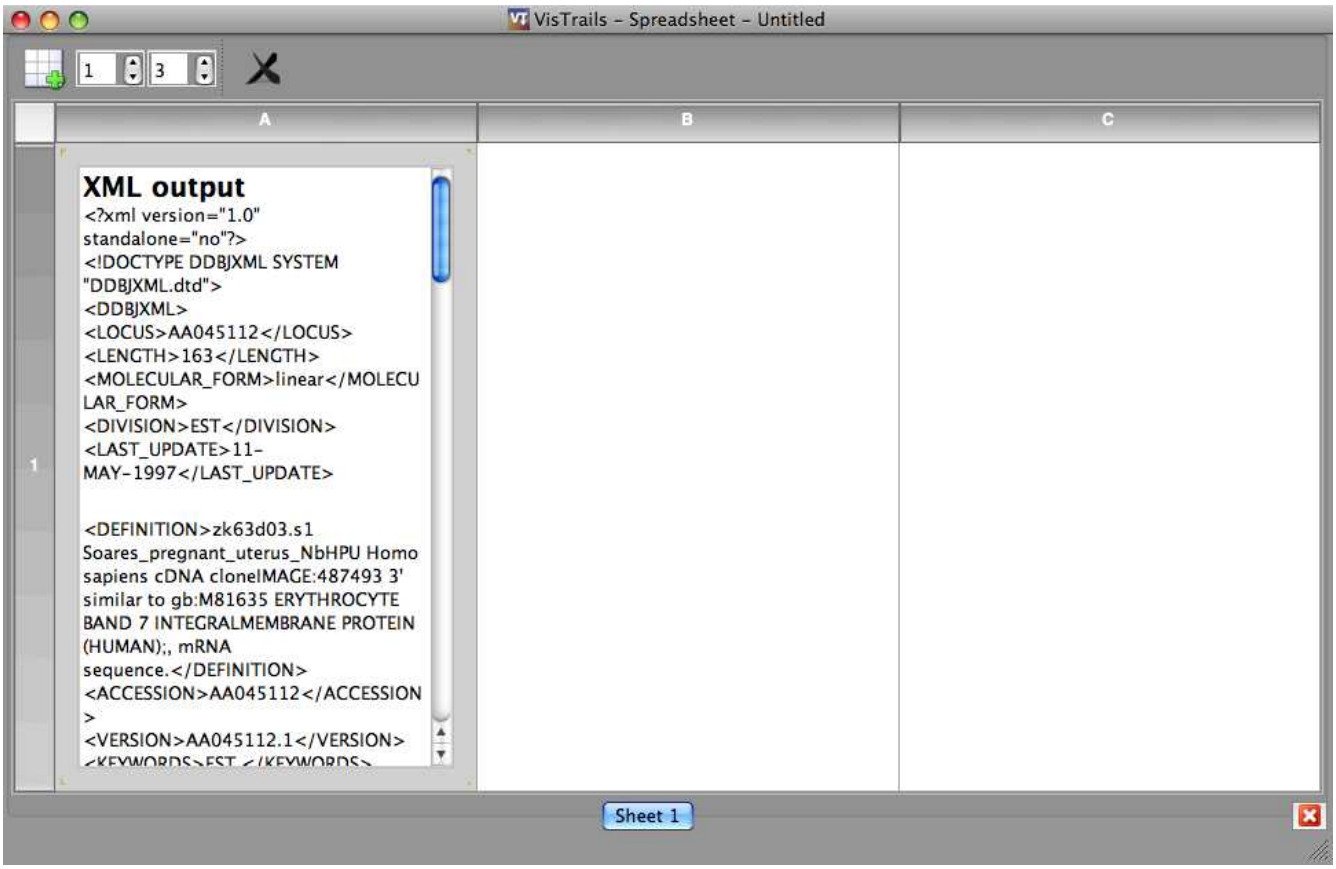


Figure 4: The display of the execution of the pipeline.

3.1.2 XML and HTML Output

In this section we will add more functionality to our demo. An HTML file will be generated from the XML file obtained in the previous section and a XSL[3] file.

Add the following modules to the “XML Output” pipeline. A new node in the version tree will

[3] XSL: Is a style-sheet that can be used to transform XML documents into other document types.

be created after you add these modules. The number in parenthesis indicates the number of modules that have to be added.

- HTTPFile (under “HTTP”)
- PythonSource (3) (under “Basic Modules”)
- RichTextCell (under “Spreadsheet”)

Label HTTPFile module as “XSL file”, the first PythonSource module as “Create_HTML”, the second PythonSource module as “Create_XML_File”, the third PhythonSource module as “XML_TO_HTML_FROM_XSL” and the RichTextCell module as “Display_HTML”.

Configure the python source modules. In the “Create_XML_File” PythonSource module add one input port, “strxml" of type String and one output port called “xmlfile" of type File and type or paste the following code in the text area:

```
#Receives a String with the XML content and returns the XML file
outseqinput = self.getInputFromPort('strxml')
xmlcontent = outseqinput.replace('<!DOCTYPE DDBJXML SYSTEM "DDBJXML.dtd">','')
output1 = self.interpreter.filePool.create_file()
f1 = open(str(output1.name), 'w')
f1.write(xmlcontent)
self.setResult("xmlfile",output1)
f1.close()
```

This PythonSource module gets the XML content returned by getXMLEntry module as a String and saves its content in a file of type XML.

In the “XML_TO_HTML_FROM_XSL” PythonSource module. Add the following input ports, “xmlfile” of type File and “xslfile” of type File and an output port called “outfile" of type File and type or paste the following code in the text area:

```
import libxml2
import libxslt
import sys
#This module transforms a XML content file in an HTML file
#with the format specified in a XSL file
output1 = self.interpreter.filePool.create_file()
f1 = open(str(output1.name), 'w')
namefile1 = f1.name
try:
f2 = open(str(xslfile.name), 'r')
f2content = f2.read()
output2 = self.interpreter.filePool.create_file()
f3 = open(str(output2.name), 'w')
f3.write(f2content)
f3.close()
styledoc = libxml2.parseFile(f3.name)
style = libxslt.parseStylesheetDoc(styledoc)
doc = libxml2.parseFile(xmlfile.name)
result = style.applyStylesheet(doc, None)
#Save the HTML file in the namefile1
style.saveResultToFilename(namefile1, result, 0)
style.freeStylesheet()
doc.freeDoc()
result.freeDoc()
except:
print sys.exc_type
print sys.exc_value
f1.close()
f1 = open(str(namefile1), 'r')
filecontent = f1.read()
index = filecontent.find("<html>")
htmlcontent = filecontent[index:]
```

```
#This replacements are done to be able to display the HTML tags of the
#transformed file in the HTML file that will be displayed in the Spreadsheet
htmlcontent = htmlcontent.replace("<","&lt;")
htmlcontent = htmlcontent.replace(">","&gt;")
output1 = self.interpreter.filePool.create_file()
f2 = open(str(output1.name), 'w')
f2.write(htmlcontent)
self.setResult("outfile",output1)
f1.close()
f2.close()
```

This PythonSource module transforms an XML file in an HTML file with the format specified in a XSL file.

In the “Create_HTML” PythonSource module. Add the input port “htmlfile” of type File and the output port “outfile" of type File and type or paste the following code in the text area:

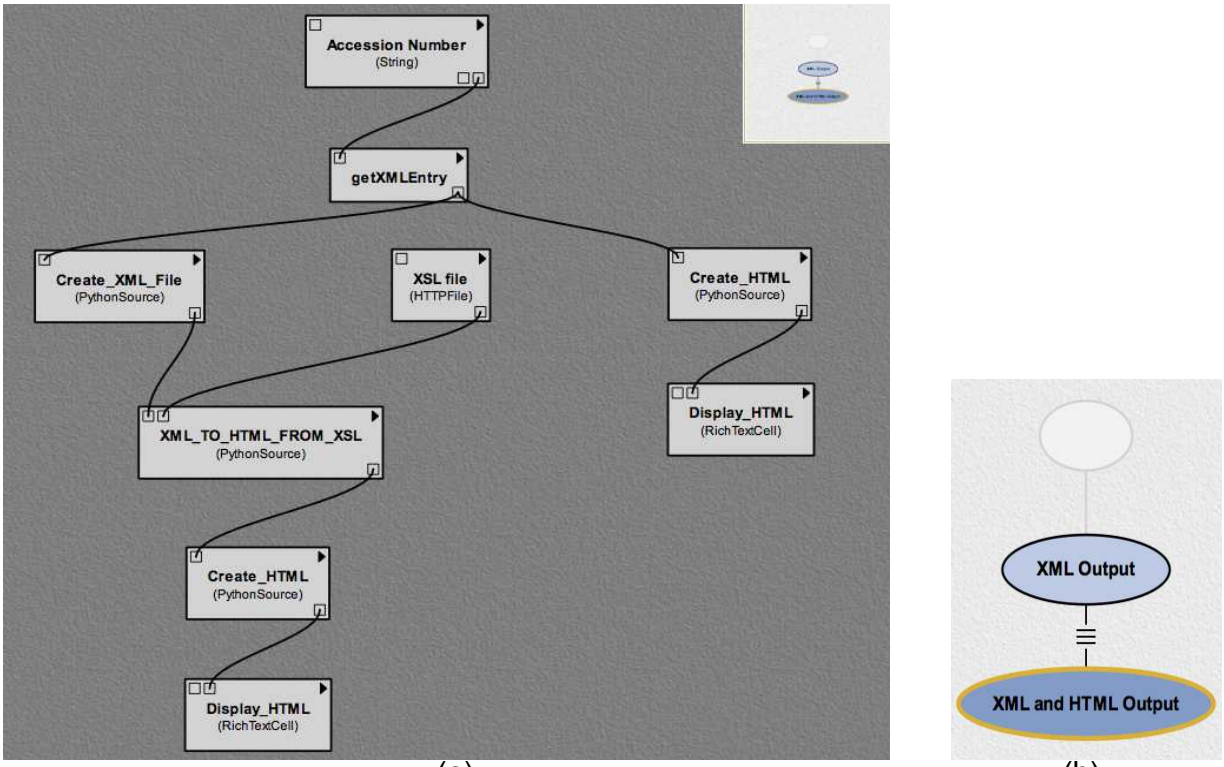
```
#Create HTML file to be displayed in the Spreadsheet
f1 = open(htmlfile.name, 'r')
htmlcontent = f1.read()
f1.close()
output1 = self.interpreter.filePool.create_file()
f1 = open(str(output1.name), 'w')
text = '<HTML><TITLE>Data Transformation Webservice</TITLE><BODY BGCOLOR="#FFFFFF">'
f1.write(text)
text = '<H2>HTML output</H2><BR>'
f1.write(text)
f1.write(htmlcontent)
text = '</BODY></HTML>'
f1.write(text)
self.setResult("outfile",output1)
f1.close()
```

This PythonSource module creates an HTML file that will be displayed in the Spreadsheet, with the HTML content obtained from executing the previous PythonSource module.

In the HTTPFile module, set the value of the url parameter of type String to: http://www.vistrails.org/images/XSLTSample.xsl.

Connect the modules together as shown in Figure 5(a).

Label the corresponding version tree node as “XML and HTML Output”. Save your work. Your version tree should look similar to the one in Figure 5(b).



(a) (b)

Figure 5: (a) XML and HTML Output pipeline (b) Corresponding version tree.

An HTML file generated by the combination of a XSL file and an XML file and the XML output are displayed in the Spreadsheet (see Figure 6).

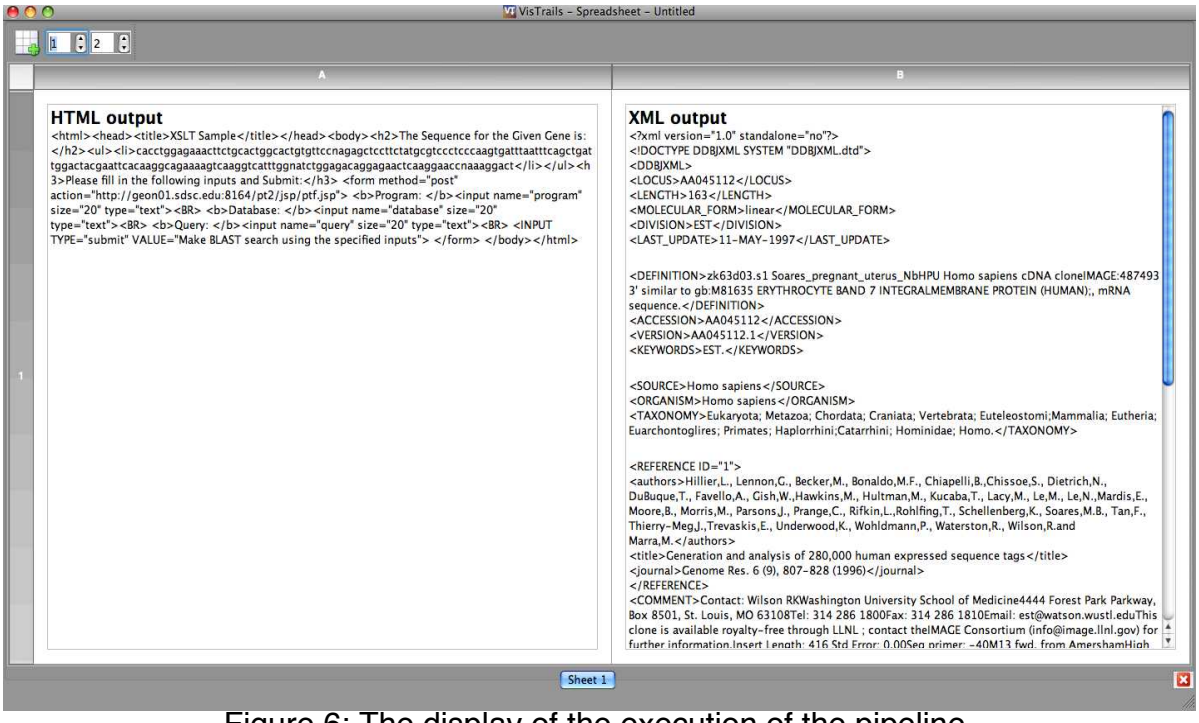


Figure 6: The display of the execution of the pipeline.

3.1.3 XML, HTML and Sequences

In this workflow, a sequence in the XML file will be extracted using xpath evaluation.

Add the following modules to the “XML and HTML Output” pipeline:

- PythonSource (2) (under “Basic Modules”)
- RichTextCell (under “Spreadsheet”)

Label the first PythonSource module as “Get_Sequence_From_XML”, the second PythonSource module as “Create_HTML” and the RichTextCell as “Display_HTML”.

Configure the PythonSource modules. Open the configuration window of the “Get_Sequence_From_XML” PythonSource module. Add one input port: “xmlfile" of type File and one output port: “sequences" of type List and type or paste the following code in the text area:

```
import libxml2
import libxslt
import sys
sequenceslist = []
try:
doc = libxml2.parseFile(xmlfile.name)
#Get the sequences from the XML file
for url in doc.xpathEval("//SEQUENCE"):
sequenceslist.append(url.content)
except:
print sys.exc_type
print sys.exc_value
self.setResult("sequences",sequenceslist)
```

This PythonSource module gets the sequences from the XML file using xpathEval function that belongs to the libxml2 python library.

Open the configuration window of the “Create_HTML” PythonSource module. Add one input port: “inputsequences” of type List and one output port: “outfile" of type File and type or paste the following code in the text area:

```
import libxml2
import libxslt
import sys
output1 = self.interpreter.filePool.create_file()
f1 = open(str(output1.name), 'w')
text = '<HTML><TITLE>Data Transformation Webservice</TITLE><BODY BGCOLOR="#FFFFFF">'
f1.write(text)
text = '<H2>Sequences</H2><BR>'
f1.write(text)
text = '<CENTER><table border = "1">'
f1.write(text)
for element in inputsequences:
line = "<tr><td>" + str(element) + "</td></tr>"
f1.write(line)
text = '</table></CENTER></BODY></HTML>'
f1.write(text)
self.setResult("outfile",output1)
f1.close()
```

This PythonSource module receives a list of sequences and creates an HTML file that contains a table of these sequences.

Connect the modules together as shown in Figure 7.

Name the pipeline as “XML, HTML and Sequences (xpath evaluation)” and save your work.

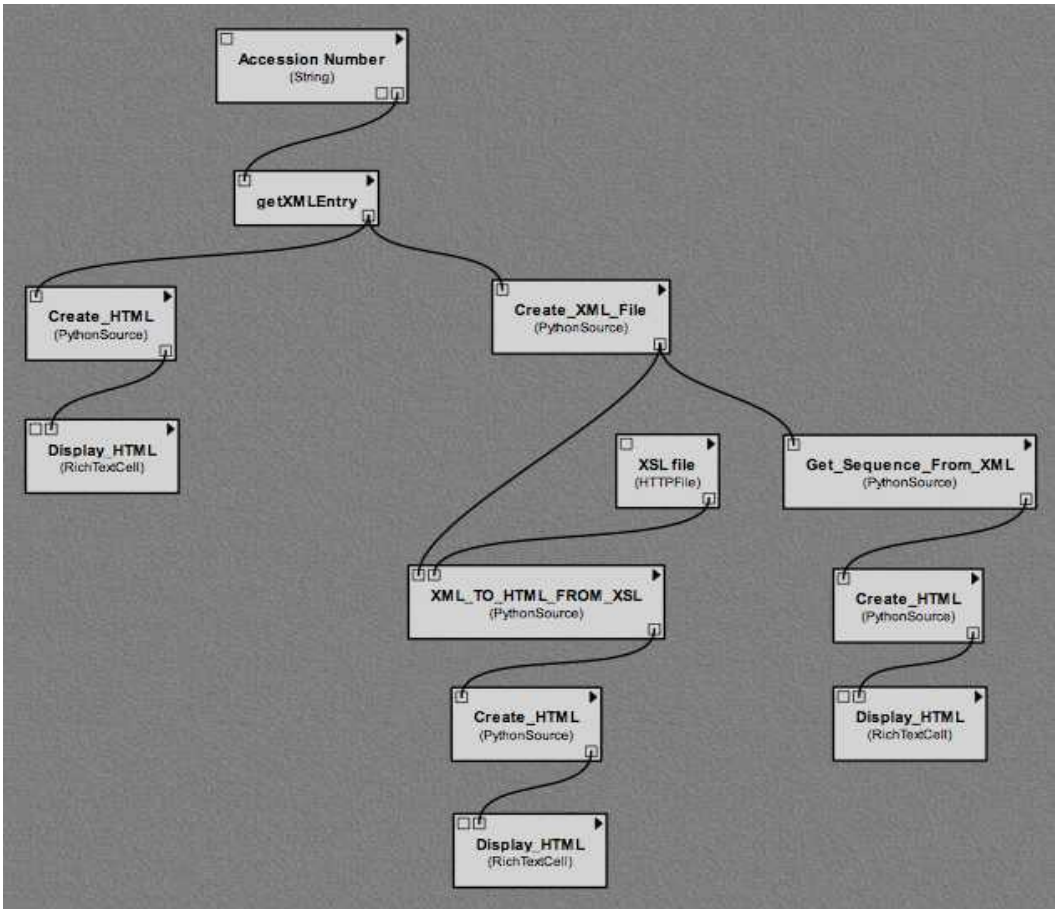


Figure 7: XML, HTML and Sequences pipeline

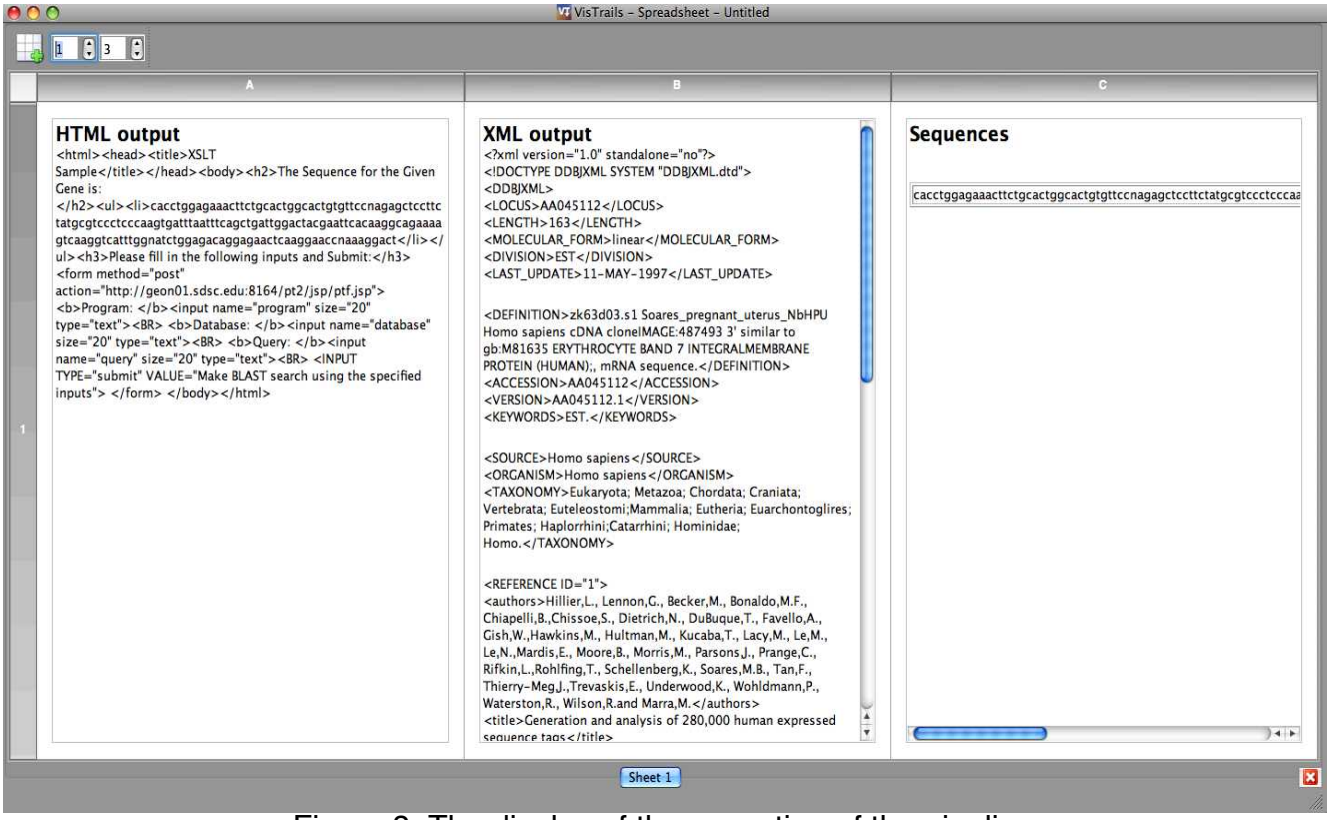


Figure 8: The display of the execution of the pipeline.

The sequence extracted from the XML document is displayed along with the HTML output and XML Output (see Figure 8).

3.1.4 Adding more nodes to the version tree

In this subsection we will take advantage of VisTrails provenance platform and reuse the workflow built in the previous section and edit it, changing the accession number.

In the “XML, HTML and Sequences (xpath evaluation)” pipeline, select the String module “Accession Number” and change the parameter value to “AB033606”. Notice that a new node is created in the version tree; tag the node with a descriptive name like: “Accession AB033606”. Do the same for the accession number “AY542895”. Your version tree should look similar to the one in Figure 9. Save your work.

VisTrails gives the user the capability to do multiple visualizations in the SpreadSheet. Side by side comparisons can be easily done. Execute the following three pipelines: “XML, HTML and Sequences (xpath evaluation)”, “Accession AB033606” and “Accession AY542895”. The execution results of these pipelines can be seen in Figure 10.

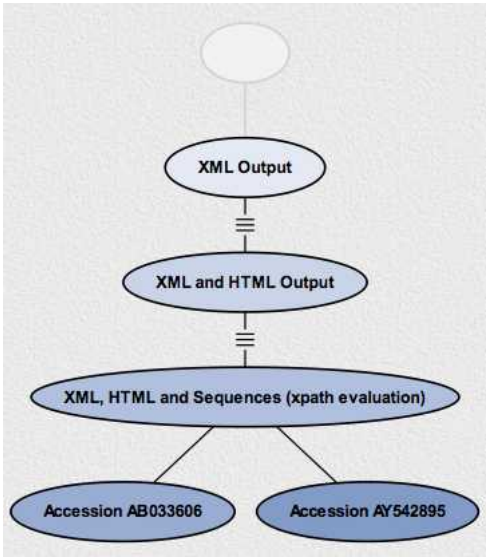


Figure 9: Version Tree view

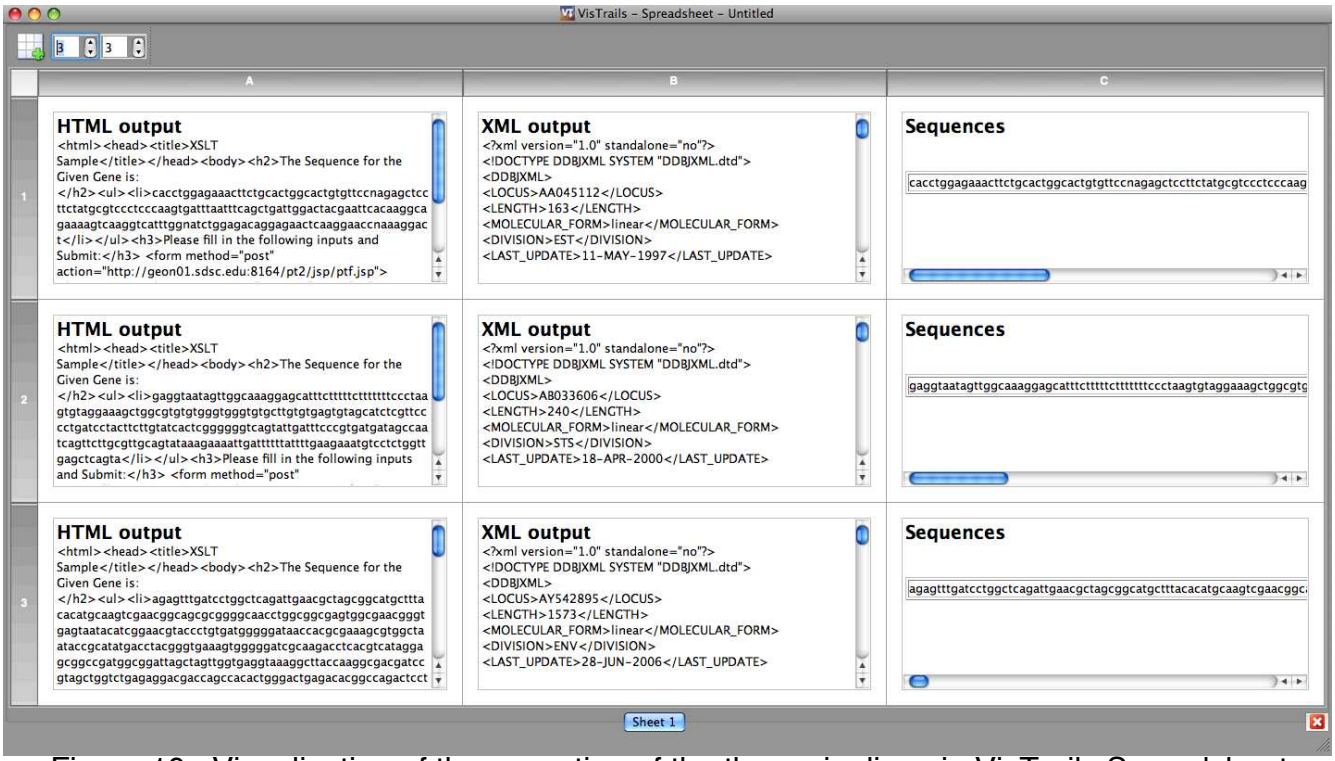


Figure 10: Visualization of the execution of the three pipelines in VisTrails Spreadsheet

3.2 NOAA web services

This example is based on some of the NOAA (National Oceanic and Atmospheric Administration) web services demos, which can be found in the following link: http://opendap.co-ops.nos.noaa.gov/axis/.

The following web services URLs need to be added:

- WaterLevelVerifiedHourly:
http://opendap.co-ops.nos.noaa.gov/axis/services/WaterLevelVerifiedHourly?wsdl
- WaterLevelVerifiedSixMin:
http://opendap.co-ops.nos.noaa.gov/axis/services/WaterLevelVerifiedSixMin?wsdl
- Wind: http://opendap.co-ops.nos.noaa.gov/axis/services/Wind?wsdl

These web services have complex types elements. A complex type element is an XML element that contains other elements and/or attributes. In VisTrails, the elements that a complex type contains are available as input and output ports in the module and they have the same names as the equivalent complex elements in the web service wsdl[4] file, with the first letter in uppercase.

3.2.1 Water Level Hourly Heights Data

We will create a workflow in which we will get the verified hourly heights water level data for a specific station by executing the web service WaterLevelVerifiedHourly.

Start with a new empty workflow in the Pipeline view, and drag the following modules to the canvas.

- Parameters (under “Types” for the current web service)
- getWaterLevelVerifiedHourly (under “Methods” for the current web service)
- PythonSource (under “Basic Modules”)
- RichTextCell (under “Spreadsheet”)

The getWaterLevelVerifiedHourly method, receives an object of type Parameters and returns an object of type WaterLevelVerifiedHourlyMeasurements. The module Parameters represents the complex type Parameters in the web service and contains all of the elements and attributes that the getWaterLevelVerifiedHourly method needs to be executed. The module WaterLevelVerifiedHourlyMeasurements represents a complex type in the web service and encapsulates the hourly heights water level data.

Label the PythonSource module as “Create_HTML” and the RichTextCell as “Display_HTML”.

Add one input port: “result" of type WaterLevelVerifiedHourlyMeasurements and one output port: “output1" of type File and type or paste the following code in the text area of the PythonSource:

```
response = self.getInputFromPort('result')
data = response.Data
itemlist  = data.Item
#Notice that the attributes of a Type module have the same name of the primitive
#type fields in the complex type (see the wsdl file) but the first letter is
#uppercase. If you want to see all the attributes of an object use dir(object)
#as the following sentences:
#print "dir(response): ", dir(response)
#print "dir(response.Data): ", dir(response.Data)
output1 = self.interpreter.filePool.create_file()
f1 = open(str(output1.name), 'w')
text = '<HTML><TITLE>NOAA Web services</TITLE><BODY BGCOLOR="#FFFFFF">'
f1.write(text)
text = '<H2> Water Level Hourly Heights Data</H2><BR>'
f1.write(text)
```

[4] WSDL: is a XML based protocol for information exchange in decentralized and distributed environments.

```
text = '<table border = "0" cellpadding= "2"><tr><td>Date Time</td><td>Date and time of the data</td></tr>'
f1.write(text)
text = '<tr><td>WL</td><td>Water level height</td></tr>'
f1.write(text)
text = '<tr><td>Sigma</td><td>Standard deviation of 1 second samples used to compute the water level height</td></tr>'
f1.write(text)
text = '<tr><td>I</td><td>A flag that indicates that the water level value has been inferred</td></tr>'
f1.write(text)
text = '<tr><td>L</td><td>A flag that when set to 1 indicates that either the maximum or minimum expected water level height
limit was exceeded</td></tr></table><BR>'
f1.write(text)
text = '<CENTER><table border = "1" cellpadding= "5"><tr bgcolor="yellow"><TH>Time Stamp</TH> <TH>WL</TH>
<TH>Sigma</TH> <TH>I</TH> <TH>L</TH></tr>'
f1.write(text)
for element in itemlist:
line = "<tr><td>" + str(element.TimeStamp) + "</td><td>" + str(element.WL) + "</td><td>" + str(element.Sigma) + "</td><td>"
+ str(element.I) + "</td><td>" + str(element.L) + "</td></tr>"
f1.write(line)
text = '</table></CENTER></BODY></HTML>'
f1.write(text)
self.setResult("outfile",output1)
f1.close()
```

In the first line of the PythonSource module code, the value of the input port result, is assigned to the variable response. The variable response will hold an object of type WaterLevelVerifiedHourlyMeasurements. In the second line, the attribute Data of the object response is assigned to the variable data. The variable data will hold and object of type Item. The third line access the Item attribute of the data object and assigns it to the itemlist variable. The variable itemlist will hold a list of objects of type Data. The data structure of the response object is similar to the one in the web service wsdl document. You can either use the built-in Python function dir()[5] to check the attributes of the objects or check the wsdl. In the next lines of the code, a HTML document that contains a table with the information of the verified hourly heights water level data is created.

We also need to set a few parameters. In module Parameters, set the following parameters: stationId: "8454000"; beginDate: "20080301"; endDate: "20080323"; datum: "MLLW"; unit: 0; timeZone: 0. The dates have the format YYYYMMDD or YYYYMMDD HH:MM. Datum can take the following values: MLLW[6], MSL[7], MHW[8], Station Datum, IGLD[9], NGVD[10], NAVD[11]. Unit can have the following values: Meters, Feet. TimeZone can take the following values: 0 for GMT and 1 for LST.

Connect the modules together as shown in Figure 11(a).

Label the corresponding version tree node as "Water Level Hourly Heights Data, StationID: 8454000". Save your work, give a name to your file, such as "NOAA_webservices.vt". Your version tree should look similar to the one in Figure 11(b).

5 dir(): The built-in function dir() is used to find out which names a module defines.

6 MLLW: Mean Lower Low Water

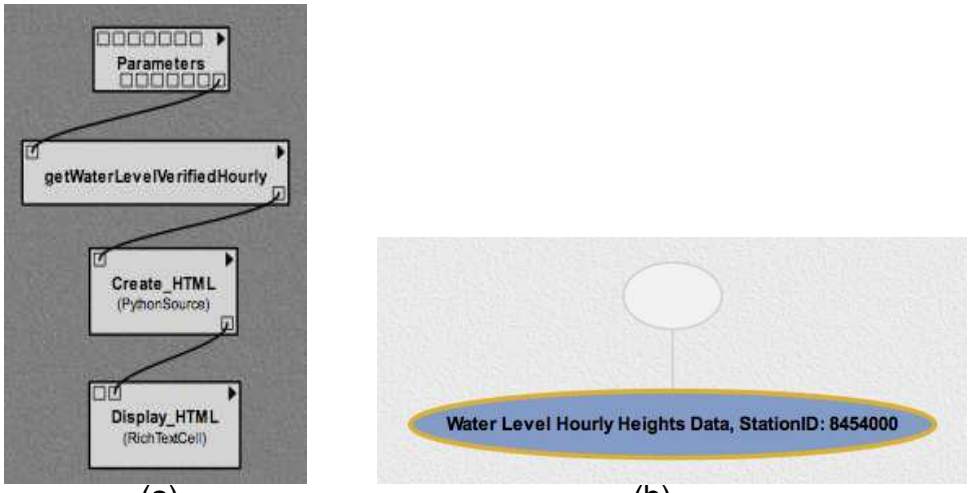
7 MSL: Mean Sea Level

8 MHW: Mean High Water

9 IGLD: International Great Lakes Datum

10 NGVD: National Geodetic Vertical Datum

11 NAVD: North American Vertical Datum



(a) (b)

Figure 11: (a) Water Level Hourly Heights Data pipeline. (b) Corresponding version tree.

The workflow is now ready to be visualized. As we have a RichTextCell module, pressing the Execute current pipeline button will send the current pipeline with the current parameters to the VisTrails Spreadsheet, resulting on an image similar to Figure 12. A table that contains the water level data per hour, is displayed. The first column contains the date and time of the data. The second column has the water level height. The third column contains the standard deviation of one second samples used to compute the water level height. The forth column has a flag that indicates that the water level value has been inferred. The fifth column contains a flag that when set to 1 indicates that either the maximum or minimum expected water level height limit was exceeded.

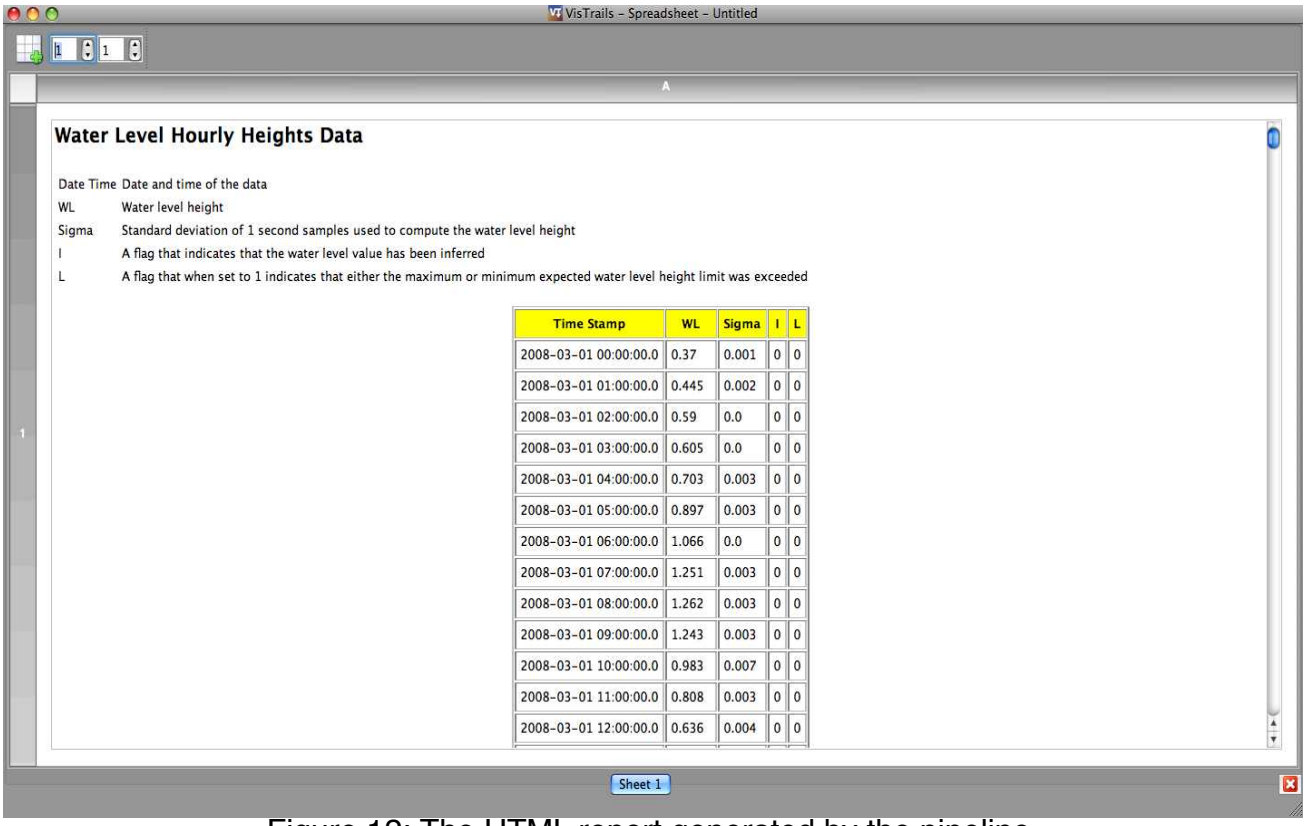


Figure 12: The HTML report generated by the pipeline.

3.2.2 Water Level Verified Six Minutes

In this subsection we will create a pipeline that will get six minute verified water level data for a specific station.

Add the following modules:

- Parameters (under “Types” for the current web service)
- getWaterLevelVerifiedSixMin (under “Methods” for the current web service)
- PythonSource (under “Basic Modules”)
- RichTextCell (under “Spreadsheet”)

Label PythonSource module as “Create_HTML” and the RichTextCell module as “Display_HTML”.

In the PythonSource module add the input port: “result" of type WaterLevelVerifiedSixMinMeasurements and one output port: “output1" of type File and type or paste the following code in the text area:

```
response = self.getInputFromPort('result')
itemlist = response.Data.Item
#Notice that the attributes of a Type module have the same name of the primitive
#type fields in the complex type (see the wsdl file) but the first letter is
#uppercase. If you want to see all the attributes of an object use dir(object)
#as the following sentences:
#print "dir(response): ", dir(response)
#print "dir(response.Data): ", dir(response.Data)
output1 = self.interpreter.filePool.create_file()
f1 = open(str(output1.name), 'w')
text = '<HTML><TITLE>NOAA Web services</TITLE><BODY BGCOLOR="#FFFFFF">'
f1.write(text)
text = '<H2>Water Level 6 Minute Verified Data</H2><BR>'
f1.write(text)
text = '<table border = "0" cellpadding= "2"><tr><td>Date Time</td><td>Date and time of the data</td></tr>'
f1.write(text)
text = '<tr><td>WL</td><td>Water level height</td></tr>'
f1.write(text)
text = '<tr><td>Sigma</td><td>Standard deviation of 1 second samples used to compute the water level height</td></tr>'
f1.write(text)
text = '<tr><td>I</td><td>A flag that indicates that the water level value has been inferred</td></tr>'
f1.write(text)
text = '<tr><td>F</td><td>A flag that when set to 1 indicates that the flat tolerance limit was exceeded</td></tr>'
f1.write(text)
text = '<tr><td>R</td><td>A flag that when set to 1 indicates that the rate of change tolerance limit was exceeded</td></tr>'
f1.write(text)
text = '<tr><td>T</td><td>A flag that when set to 1 indicates that the temperature difference tolerance limit was
exceeded</td></tr></table><BR>'
f1.write(text)
text = '<CENTER><table border = "1" cellpadding= "5"><tr bgcolor="yellow"><TH>Time Stamp</TH> <TH>WL</TH>
<TH>Sigma</TH> <TH>I</TH> <TH>F</TH><TH>R</TH><TH>T</TH></tr>'
f1.write(text)
for element in itemlist:
line = "<tr><td>" + str(element.TimeStamp) + "</td><td>" + str(element.WL) + "</td><td>" + str(element.Sigma) + "</td><td>"
+ str(element.I) + "</td><td>" + str(element.F) + "</td><td>" + str(element.R) + "</td><td>" + str(element.T) + "</td></tr>"
f1.write(line)
text = '</table></CENTER></BODY></HTML>'
f1.write(text)
self.setResult("outfile",output1)
f1.close()
```

In this PythonSource module an HTML file with a table that contains water level 6 minute verified data, is created.

We also need to set a few parameters in order to getWaterLevelVerifiedSixMin to work properly. Set the following parameters in module Parameters: stationId: “8454000”; beginDate: “20080301”; endDate: “20080301”; datum: “MLLW”; unit: 0; timeZone: 0. The dates have the format YYYYMMDD or YYYYMMDD HH:MM. Datum can take the following values: MLLW, MSL, MHW, Station Datum, IGLD, NGVD, NAVD. Unit can have the following values: Meters, Feet. TimeZone can take the following values: 0 for GMT and 1 for LST.

Connect the modules together as shown in Figure 13. Name your pipeline as “WaterLevelVerifiedSixMin, StationID: 8454000”, and save your work.

The workflow is now ready to be visualized. As we have a RichTextCell module, pressing the Execute current pipeline button will send the current pipeline with the current parameters to the VisTrails Spreadsheet (See Figure 14). A table that contains the water level data per six minutes, is displayed. The first column contains the date and time of the data. The second column has the water level height. The third column contains the standard deviation of one second samples used to compute the water level height. The fourth column has a flag that indicates that the water level value has been inferred. The fifth column contains a flag that when set to 1 indicates that the flat tolerance limit was exceeded. The sixth column has a flag that when set to 1 indicates that the rate of change tolerance limit was exceeded. The seventh column contains a flag that when set to 1 indicates that the temperature difference tolerance limit was exceeded.

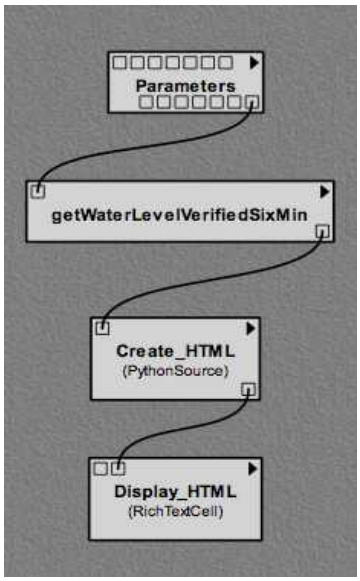


Figure 13: Water Level Verified Six Minutes pipeline.

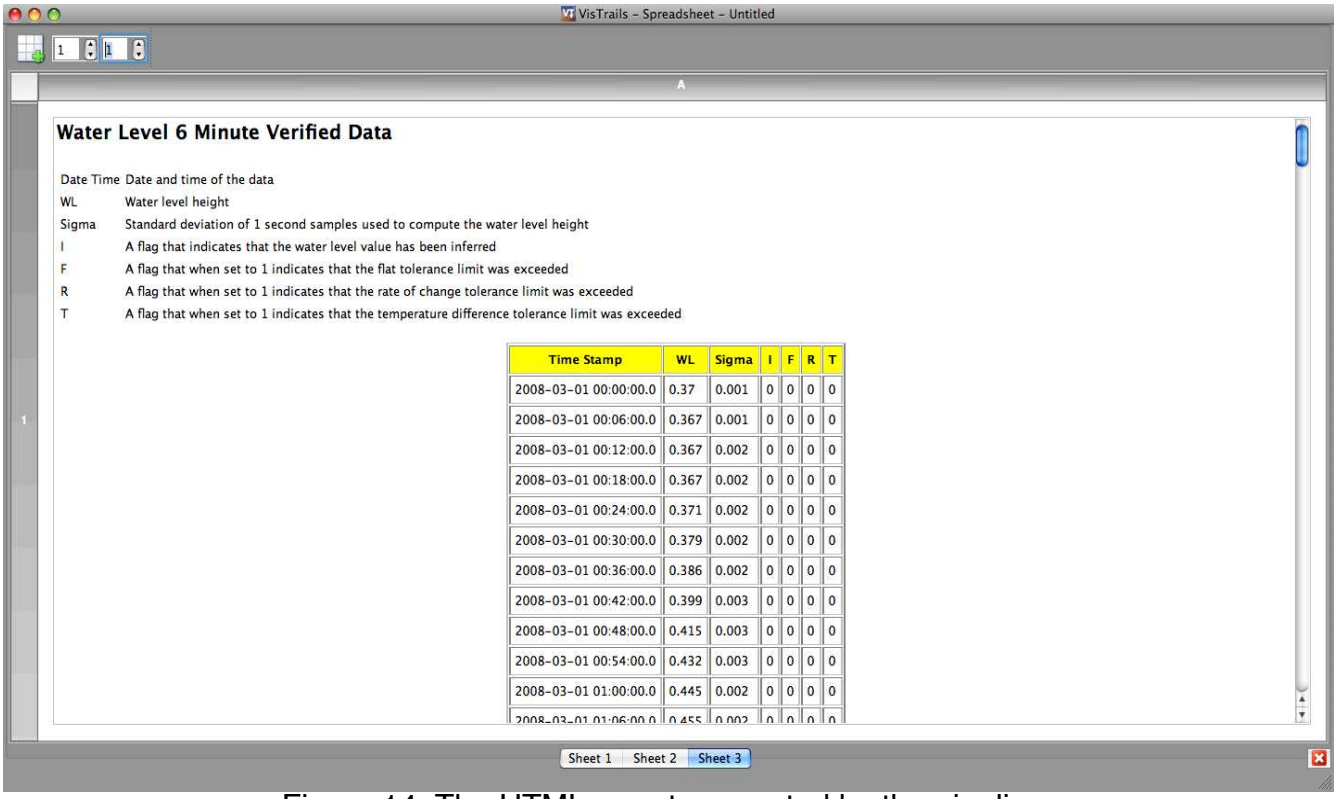


Figure 14: The HTML report generated by the pipeline.

3.2.3 Wind Data

In this subsection we will create a workflow in which we will get the wind data for a specific station.

Also add the following modules:

- Parameters (under “Types” for the current web service)
- getWind (under “Methods” for the current web service)
- PythonSource (under “Basic Modules”)
- RichTextCell (under “Spreadsheet”)

Label PythonSource module as “Create_HTML” and the RichTextCell module as “Display_HTML”.

In the PythonSource module add one input port, “result" of type WindMeasurements and add an output port called “output1" of type File and type the following code in the text area:

```
response = self.getInputFromPort('result')
itemlist = response.Data.Item
#Notice that the attributes of a Type module have the same name of the primitive
#type fields in the complex type (see the wsdl file) but the first letter is
#uppercase. If you want to see all the attributes of an object use dir(object)
#as the following sentences:
#print "dir(response): ", dir(response)
#print "dir(response.Data): ", dir(response.Data)
output1 = self.interpreter.filePool.create_file()
f1 = open(str(output1.name), 'w')
text = '<HTML><TITLE>NOAA Web services</TITLE><BODY BGCOLOR="#FFFFFF">'
```

```
f1.write(text)
text = '<H2>Wind Data</H2><BR>'
f1.write(text)
text = '<table border = "0" cellpadding= "2"><tr><td>Date Time</td><td>Date and time of the data</td></tr>'
f1.write(text)
text = '<tr><td>WS</td><td>Wind speed is in meters per second</td></tr>'
f1.write(text)
text = '<tr><td>WD</td><td>Wind direction in degrees</td></tr>'
f1.write(text)
text = '<tr><td>WG</td><td>Wind gust is in meters per second</td></tr>'
f1.write(text)
text = '<tr><td>X</td><td>A flag that when set to 1 indicates that the maximum wind speed was exceeded</td></tr>'
f1.write(text)
text = '<tr><td>R</td><td>A flag that when set to 1 indicates that the rate of change tolerance limit was
exceeded</td></tr></table><BR>'
f1.write(text)
text = '<CENTER><table border = "1" cellpadding= "5"><tr bgcolor="yellow"><TH>Time Stamp</TH> <TH>WS</TH>
<TH>WD</TH> <TH>WG</TH> <TH>X</TH><TH>R</TH></tr>'
f1.write(text)
for element in itemlist:
line = "<tr><td>" + str(element.TimeStamp) + "</td><td>" + str(element.WS) + "</td><td>" + str(element.WD) + "</td><td>" +
str(element.WG) + "</td><td>" + str(element.X) + "</td><td>" + str(element.R) + "</td></tr>"
f1.write(line)
text = '</table></CENTER></BODY></HTML>'
f1.write(text)
self.setResult("outfile",output1)
f1.close()
```

In this PythonSource an HTML file that contains a table with wind data, is created.

We also need to set a few parameters in order to getWind to work properly. Set the following parameters in module Parameters: stationId: “8454000”; beginDate: “20080523”; endDate: “20080523”; timeZone: 0. The dates have the format YYYYMMDD or YYYYMMDD HH:MM. Datum can take the following values: MLLW, MSL, MHW, Station Datum, IGLD, NGVD, NAVD. TimeZone can take the following values: 0 for GMT and 1 for LST.

Connect the modules together as shown in Figure 15. Name the pipeline as “Wind Data, StationID: 8454000”.

The workflow is now ready to be visualized. As we have a RichTextCell module, pressing the Execute current pipeline button will send the current pipeline with the current parameters to the VisTrails Spreadsheet (See Figure 16). A table that contains the wind data is displayed. The first column contains the date and time of the data. The second column has the wind speed is in meters per second. The third column contains the wind direction in degrees. The fourth column has the wind gust in meters per second. The fifth column contains a flag that when set to 1 indicates that the maximum wind speed was exceeded. The sixth column has a flag that when set to 1 indicates that the rate of change tolerance limit was exceeded.

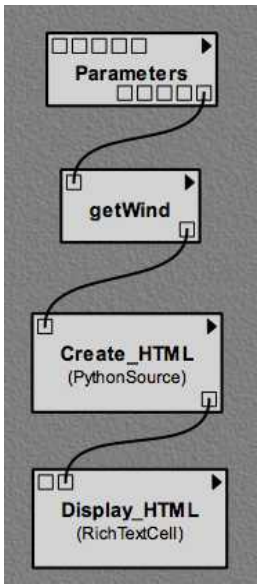


Figure 15: Wind Data pipeline.

VisTrails – Spreadsheet – Untitled

Wind Data

Date Time Date and time of the data
WS Wind speed is in meters per second
WD Wind direction in degrees
WG Wind gust is in meters per second
X A flag that when set to 1 indicates that the maximum wind speed was exceeded
R A flag that when set to 1 indicates that the rate of change tolerance limit was exceeded

Time Stamp	WS	WD	WG	X	R
2008-05-23 00:00:00.0	3.278	286	4.299	0	0
2008-05-23 00:06:00.0	3.527	297	4.664	0	0
2008-05-23 00:12:00.0	4.48	298	8.324	0	0
2008-05-23 00:18:00.0	4.725	315	7.361	0	0
2008-05-23 00:24:00.0	4.536	309	6.597	0	0
2008-05-23 00:30:00.0	4.167	318	5.326	0	0
2008-05-23 00:36:00.0	4.348	312	6.188	0	0
2008-05-23 00:42:00.0	4.546	313	6.586	0	0
2008-05-23 00:48:00.0	5.787	313	8.096	0	0
2008-05-23 00:54:00.0	4.9	316	6.5	0	0
2008-05-23 01:00:00.0	3.239	314	4.116	0	0
2008-05-23 01:06:00.0	3.213	295	4.928	0	0

Sheet 1 Sheet 2 Sheet 3

Figure 16: The HTML report generated by the pipeline.

3.3 EMBOSS web services

The EMBOSS web services demo that we will build in this section is based on the EMBOSS tutorial and it uses the following web services:

- seqret: http://www.ebi.ac.uk/soaplab/emboss4/services/edit.seqret.derived?wsdl
- emma: http://www.ebi.ac.uk/soaplab/emboss4/services/alignment_multiple.emma.derived?wsdl
- prettyplot: http://www.ebi.ac.uk/soaplab/emboss4/services/alignment_multiple.prettyplot.derived?wsdl
- tmap: http://www.ebi.ac.uk/soaplab/emboss4/services/protein_2d_structure.tmap.derived?wsdl
- prophecy: http://www.ebi.ac.uk/soaplab/emboss4/services/nucleic_profiles.prophecy.derived?wsdl
- transeq: http://www.ebi.ac.uk/soaplab/emboss4/services/nucleic_translation.transeq.derived?wsdl
- prophet: http://www.ebi.ac.uk/soaplab/emboss4/services/nucleic_profiles.prophet.derived?wsdl

3.3.1 Seqret Workflow

In this workflow we will invoke EMBOSS[12] seqret web service, which extracts sequences from databases. Start with a new empty workflow in the Pipeline view, and drag the following modules to the canvas.

- Sequence (under “Types” for the seqret web service)
- Tseqret (under “Types” for the seqret web service)
- runAndWaitFor (under “Types” for the seqret web service)
- runAndWaitForResponse (under “Types” for the seqret web service)
- TseqretResult (under “Types” for the seqret web service)
- runAndWaitFor (under “Methods” for the seqret web service)
- PythonSource (under “Basic Modules”)
- RichTextCell (under “Spreadsheet”)

Set the labels of PythonSource module to “Create_HTML” and of RichTextCell to “Display_HTML”.

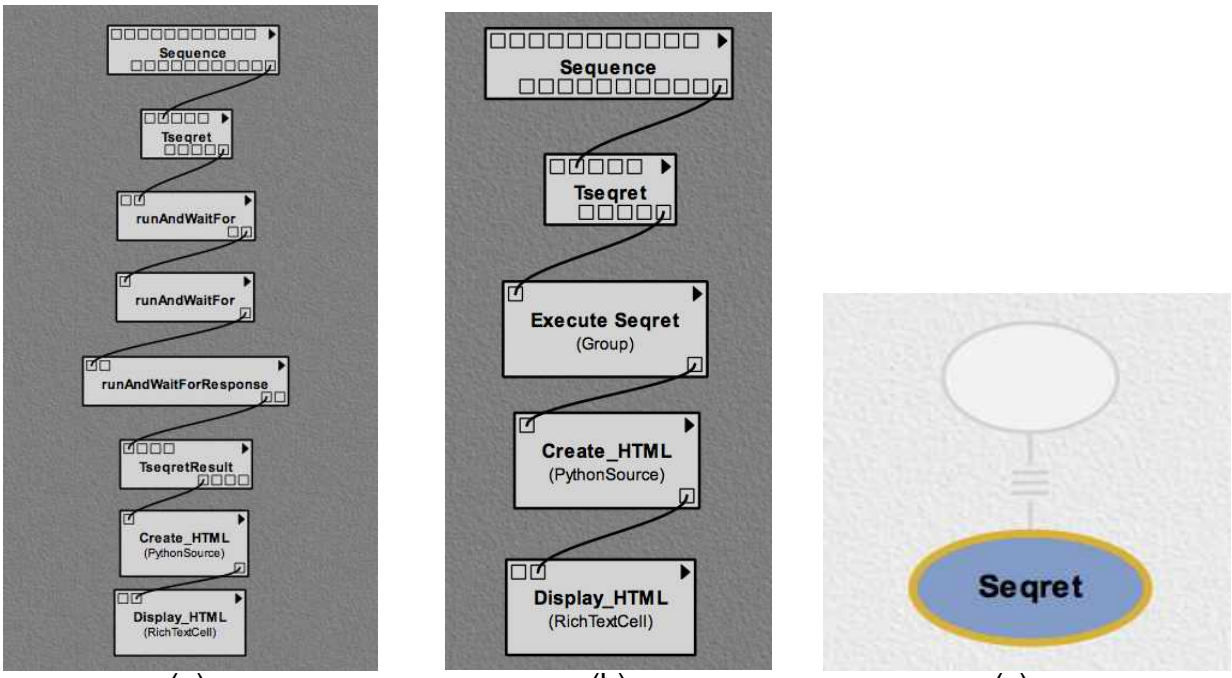
In the PythonSource module, add one input port, “outseq" of type String and one output port called “filename1" of type File and type or paste the following code in the text area:

```
outseqinput = self.getInputFromPort('outseq')
output1 = self.interpreter.filePool.create_file()
f1 = open(str(output1.name), 'w')
text = '<HTML><TITLE>EMBOSS Webservice</TITLE><BODY BGCOLOR="#FFFFFF">'
f1.write(text)
text = '<H2>Seqret Out Sequence</H2><BR>'
f1.write(text)
splitnewline = outseqinput.split('\n')
for element in splitnewline:
f1.write(element)
f1.write('<BR>')
text = '</BODY></HTML>'
f1.write(text)
self.setResult("filename1",output1)
f1.close()
```

[12] EMBOSS is an acronym for The European Molecular Biology Open Software Suite

This PythonSource creates an HTML file with the results of invoking the seqret web service.

Connect the modules together as shown in Figure 17(a).



(a) (b) (c)

Figure 17: (a) Seqret Workflow before grouping (b) Seqret Workflow after grouping (c) Corresponding Version Tree.

In the module Sequence, set the parameter sequence_usa with the value “uniprot:ops2_*”. This parameter receives a string with a USA[13] format: “database:identifier”. In this example, we will query the UniProt (Universal Protein Resource) database and get all sequences whose identifier begins with “ops2_”.

Now we are going to create a group to make our workflow easier to read and understand. Select the following modules: runAndWaitFor (Type module), runAndWaitFor (Method module), runAndWaitForResponse, TseqretResult and then select the option Group from the Edit menu or press Ctrl+G ('Command+G' on Mac). A module Group is created. We will set its label by clicking on the arrow that is on the top-right corner of the module box and then select “Set Module Label” option and write the following in the text field: “Execute Seqret”, see Figure 17(b).

Name the pipeline as “Seqret”. Your version tree should look similar to the one in Figure 17(c). Save your work. Give a name to your file, such as “EMBOSS_webservices.vt”.

The workflow is now ready to be visualized. As we have a RichTextCell module, pressing the Execute current pipeline button will send the current pipeline with the current parameters to the VisTrails Spreadsheet, resulting on an image similar to Figure 18. The data received from the seqret web service is in FASTA format. In this particular format, the first line of each sequence contains a brief description of the organism. There are three ops2 sequences, two of them represent fruit fly species and the other one a desert locust.

13 USA: Uniform Sequence Address, is a string defining how the software accesses a biological sequence.

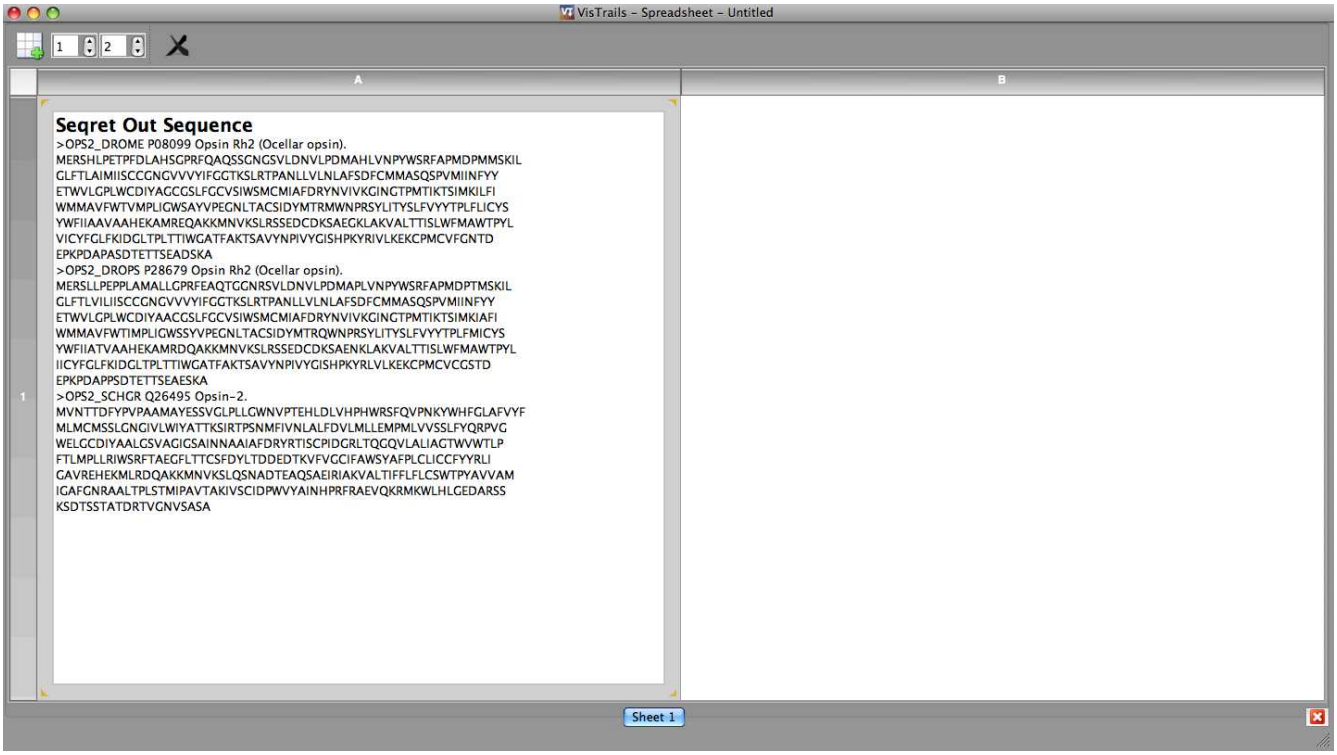


Figure 18: The ops2 sequences obtained from querying the UniProt database.

3.3.2 Emma Workflow

In this section we will use the emma web service to align a set of given sequences. We will reuse the workflow that we built in the previous section to get the set of ops2 sequences and pass it to the emma web service in order to align them.

Add the following modules to the “Seqret” pipeline.

- Sequence (under “Types” for emma web service)
- Temma (under “Types” for the emma web service)
- runAndWaitFor (under “Types” for the emma web service)
- runAndWaitForResponse (under “Types” for the emma web service)
- TemmaResult (under “Types” for the emma web service)
- runAndWaitFor (under “Methods” for the emma web service)
- PythonSource (under “Basic Modules”)
- RichTextCell (under “Spreadsheet”)

Give names to the modules. Set the label “Create_HTML” to the PythonSource module and the label “Display_HTML” to the RichTextCell module.

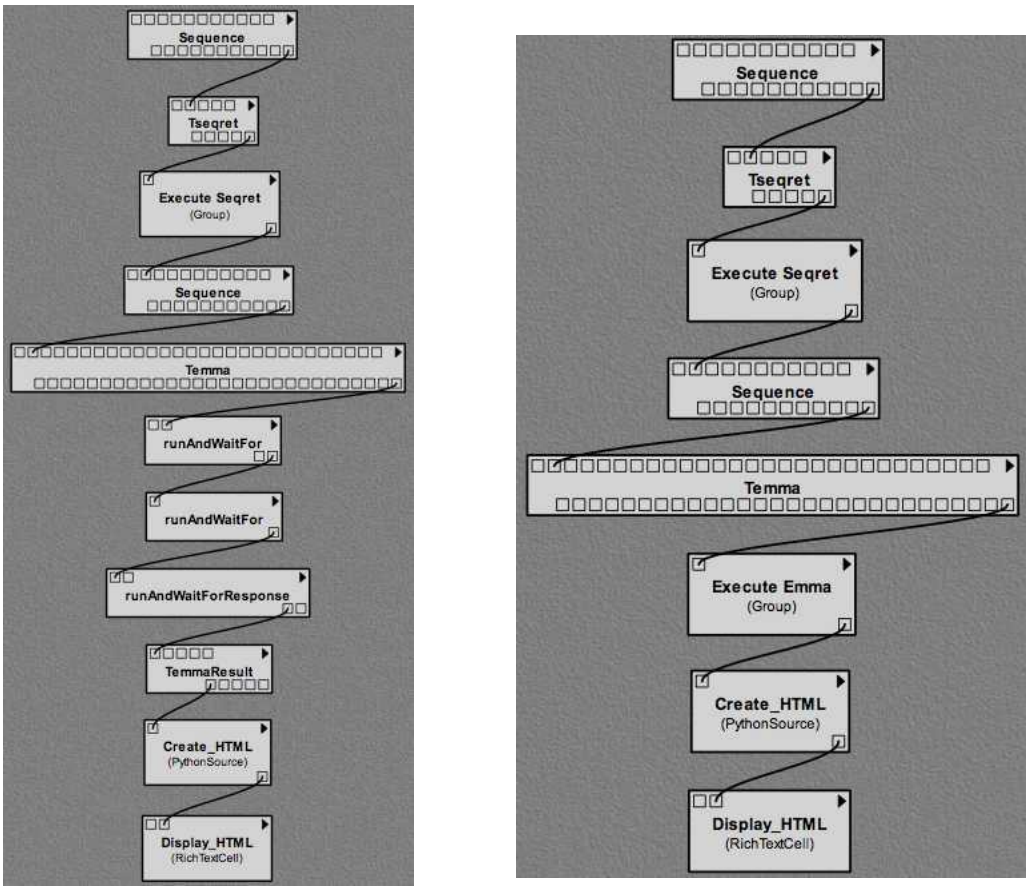
In the PythonSource module add one input port, “outseq" of type String and one output port called “filename1" of type File and type or paste the following code in the text area:

```
outseqinput = self.getInputFromPort('outseq')
output1 = self.interpreter.filePool.create_file()
f1 = open(str(output1.name), 'w')
text = '<HTML><TITLE>EMBOSS Webservice</TITLE><BODY BGCOLOR="#FFFFFF">'
f1.write(text)
```

```
text = '<H2>Emma Out Sequence</H2><BR>'
f1.write(text)
splitnewline = outseqinput.split('\n')
for element in splitnewline:
f1.write(element)
f1.write('<BR>')
text = '</BODY></HTML>'
f1.write(text)
self.setResult("filename1",output1)
f1.close()
```

This PythonSource module, creates an HTML file with the results of executing the emma web service.

Connect the modules together as shown in Figure 19(a).



(a) (b)

Figure 19: (a) Emma Workflow before grouping (b) Emma Workflow after grouping.

Create the group “Execute Emma” with the following modules: runAndWaitFor (Type module), runAndWaitFor (Method module), runAndWaitForResponse, TemmaResult, see Figure 19(b). Name the pipeline as “Emma” and save our work.
The workflow is now ready to be visualized. As we have a RichTextCell module, pressing the Execute current pipeline button will send the current pipeline with the current parameters to the VisTrails Spreadsheet, resulting on an image similar to Figure 20. The three ops2 sequences were aligned using emma. Notice that some gaps were introduced to make all the sequences the same length.

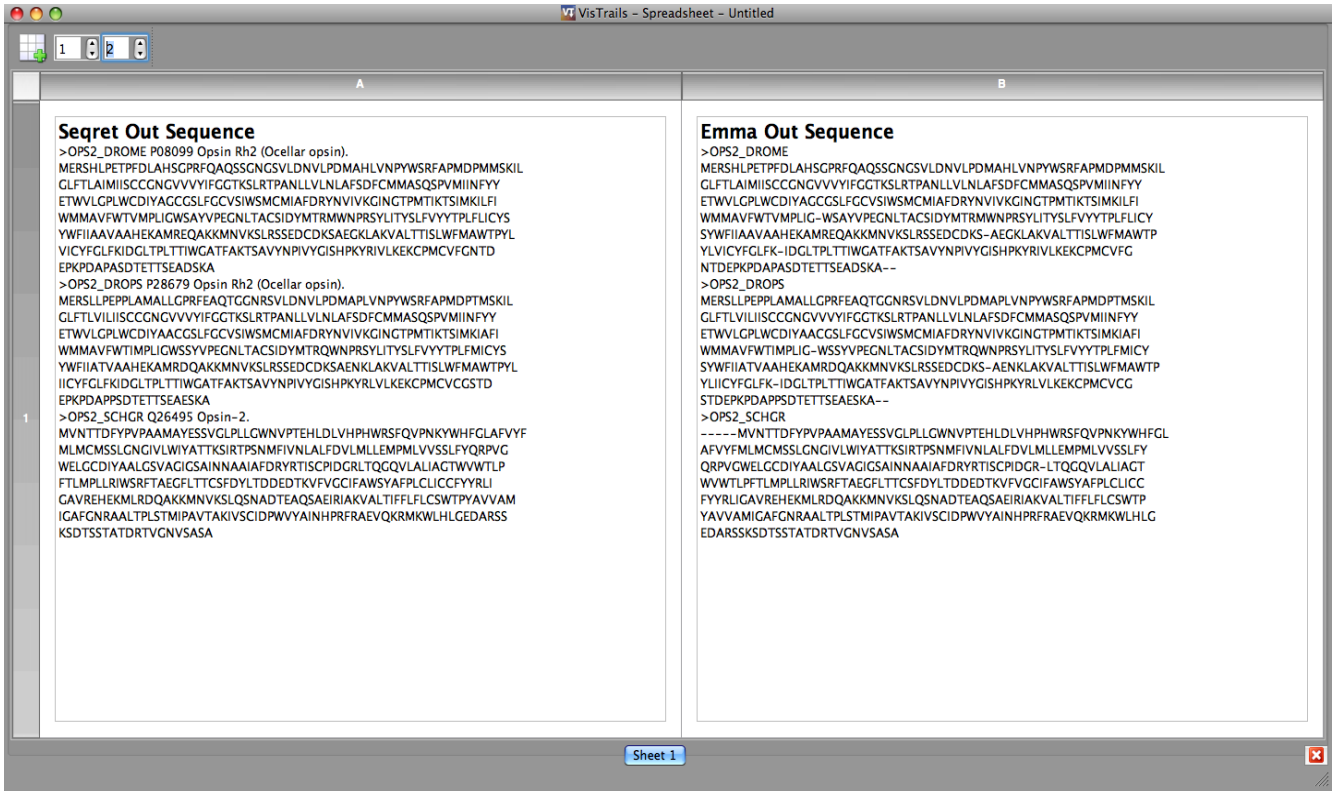


Figure 20: The ops2 sequences were aligned using the emma web service.

3.3.3 Prettyplot Workflow

We have successfully aligned the sequences with the emma web service, now we will take advantage of the visual cues that prettyplot web service provides to have a better visualization of the alignment.

We will reuse the Emma workflow created in the previous section, to get the alignment of the ops2 sequences and pass this data to the prettyplot web service. Add the following modules to the “Emma” pipeline.

- Sequence (under “Types” for prettyplot web service)
- Tprettyplot (under “Types” for the prettyplot web service)
- runAndWaitFor (under “Types” for the prettyplot web service)
- runAndWaitForResponse (under “Types” for the prettyplot web service)
- TprettyplotResult (under “Types” for the prettyplot web service)
- TprettyplotResultGraphics_container (under “Types” for the prettyplot web service)
- runAndWaitFor (under “Methods” for the prettyplot web service)
- PythonSource (under “Basic Modules”)
- RichTextCell (under “Spreadsheet”)

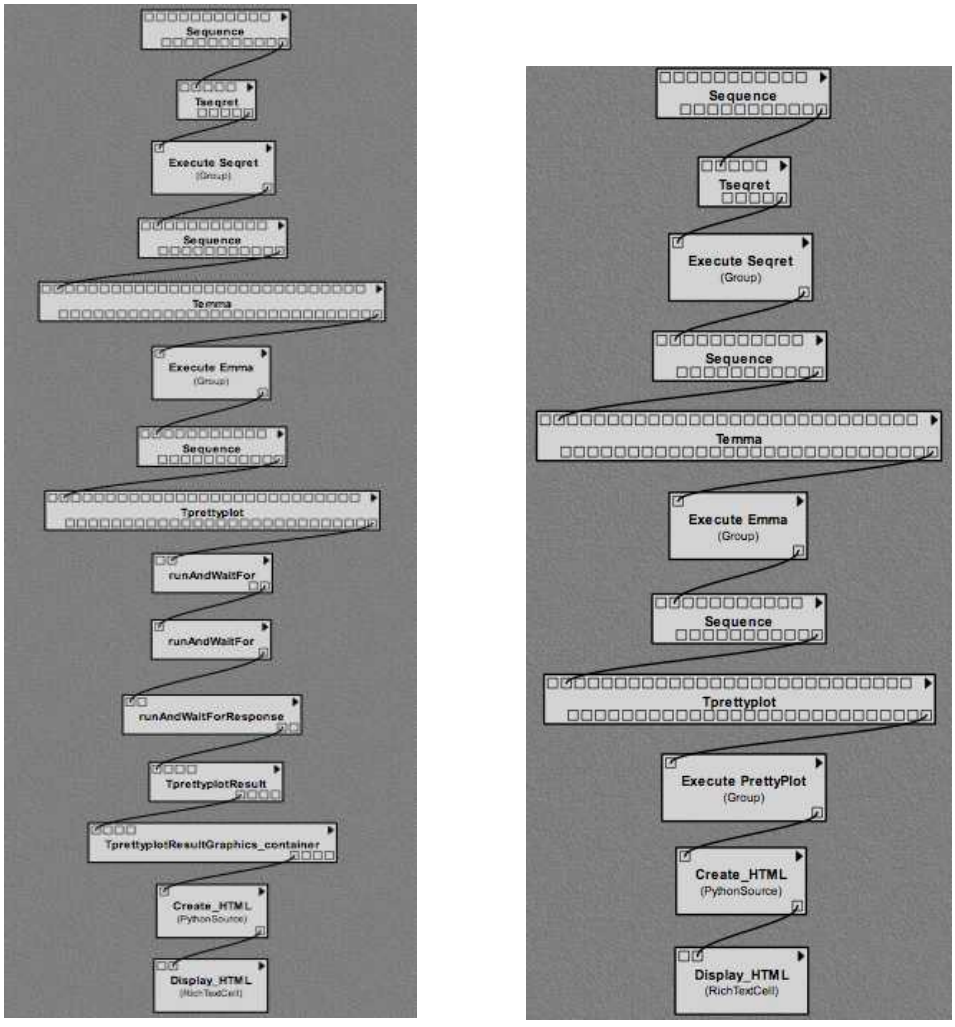
Set the label “Create_HTML” to the PythonSource module and the label “Display_HTML” to the RichTextCell module.

In the PythonSource module, add one input port: “graph" of type String and one output port: “htmlfile" of type File and type or paste the following code in the text area:

```
f = self.interpreter.filePool.create_file('.png')
my_file = open(str(f.name), 'wb')
my_file.write(graph[0])
my_file.close()
text = '<HTML><TITLE>Pretty Plot Graph</TITLE><BODY BGCOLOR="#FFFFFF">'
text += '<TABLE WIDTH="100%" BORDER="1" BGCOLOR="#FFFFFF" CELLPADDING="4">'
text += '<TR><TD VALIGN="TOP"><P><IMG SRC="'
text += f.name + '"></TD></TR></TABLE></BODY></HTML>'
output = self.interpreter.filePool.create_file()
my_file = open(str(output.name), 'w')
my_file.write(text)
my_file.close()
self.setResult("htmlfile",output)
```

This PythonSource module creates an HTML file that contains the graphs of the alignments of the sequences.

Connect the modules together as shown in Figure 21(a).



(a) (b)

Figure 21: (a) Prettyplot Workflow before grouping (b) Prettyplot Workflow after grouping.

Group the following modules: runAndWaitFor (Type module), runAndWaitFor (Method module), runAndWaitForResponse, TprettyplotResult, TprettyplotResultGraphics_container

and then name the group as “Execute PrettyPlot” (See Figure 21(b)). Set the pipeline name as “Prettyplot” and save your work.

The workflow is now ready to be visualized. As we have a RichTextCell module, pressing the Execute current pipeline button will send the current pipeline with the current parameters to the VisTrails Spreadsheet, resulting on an image similar to Figure 22. The alignment of the ops2 sequences is visualized using prettyplot web service. Identical residues are shown in red, and similar residues in green.

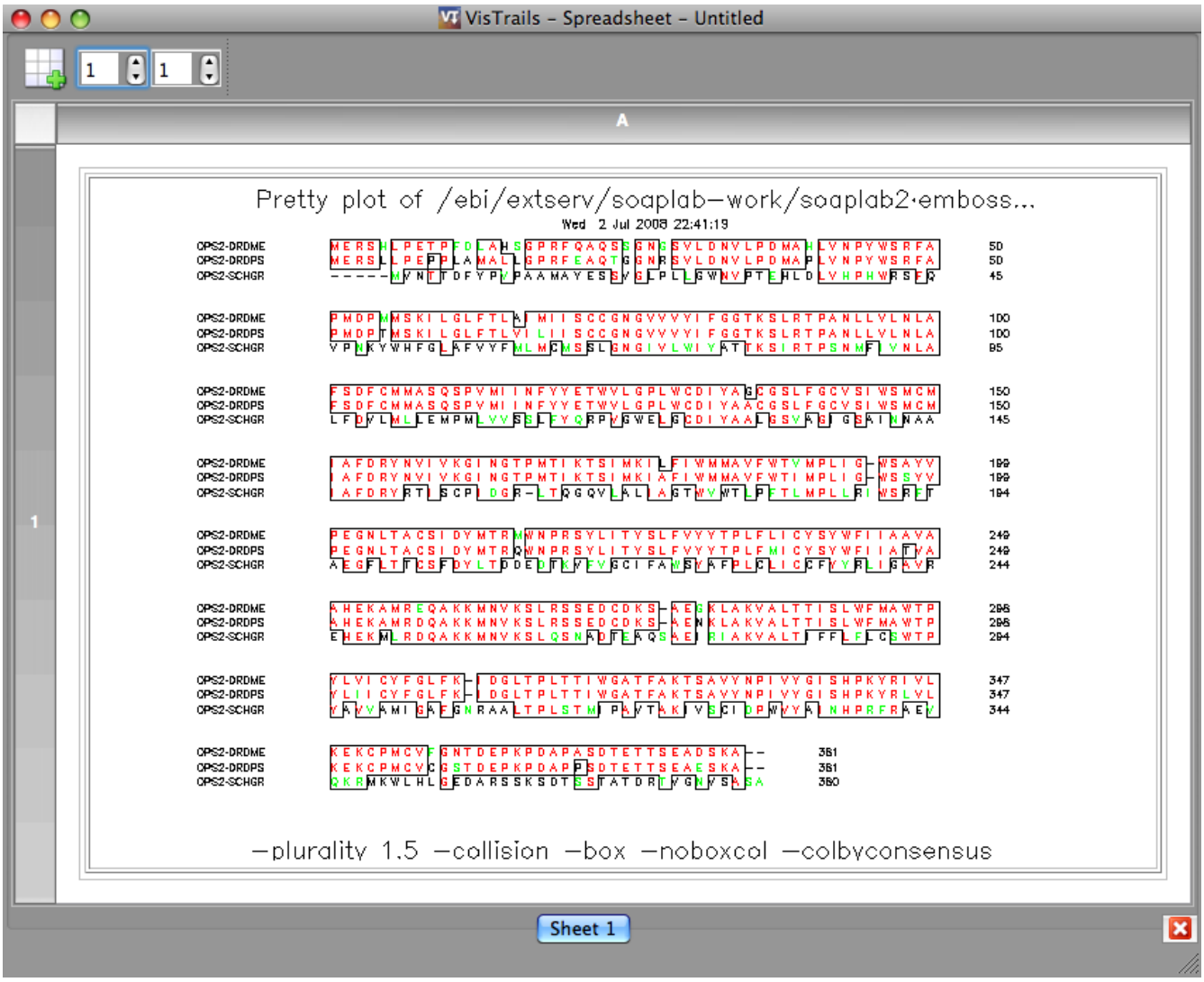


Figure 22: Visualization of the alignments of the ops2 sequences usign prettyplot web service.

3.3.4 Tmap Workflow

In this section of the tutorial we will create a workflow to display membrane spanning regions. We will reuse the Seqret workflow, to get the list of sequences from the database and then pass this data to the tmap web service. Add the following modules to the “Seqret” pipeline.

- Sequence (under “Types” for tmap web service)
- Sequence (under “Types” for seqret web service)

- Ttmap (under “Types” for the tmap web service)
- Tseqret (under “Types” for the seqret web service)
- runAndWaitFor (under “Types” for the tmap web service)
- runAndWaitForResponse (under “Types” for the tmap web service)
- TtmapResult (under “Types” for the tmap web service)
- TtmapResultGraphics_container (under “Types” for the tmap web service)
- runAndWaitFor (under “Methods” for the tmap web service)
- PythonSource (under “Basic Modules”)
- RichTextCell (under “Spreadsheet”)

Copy and paste the “Execute Seqret” Group.

Set the label “Create_HTML” to the PythonSource module and the label “Display_HTML” to the RichTextCell module.

In the PythonSource module add one input port: “graph" of type String and one output port: “htmlfile" of type File and type or paste the following code in the text area:

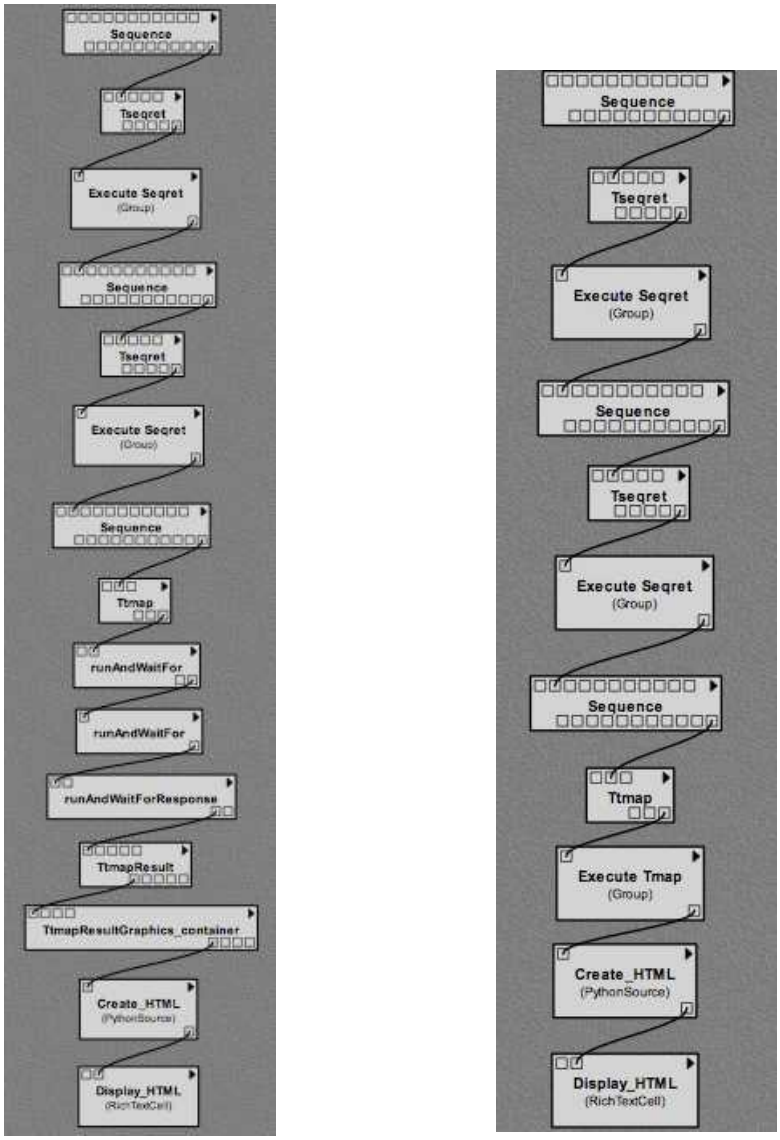
```
f = self.interpreter.filePool.create_file('.png')
my_file = open(str(f.name), 'wb')
my_file.write(graph[0])
my_file.close()
text = '<HTML><TITLE>Tmap Plot Graph</TITLE><BODY BGCOLOR="#FFFFFF">'
text += '<TABLE WIDTH="100%" BORDER="1" BGCOLOR="#FFFFFF" CELLPADDING="4">'
text += '<TR><TD VALIGN="TOP"><P><CENTER><IMG SRC="'
text += f.name + '"></CENTER></TD></TR></TABLE></BODY></HTML>'
output = self.interpreter.filePool.create_file()
my_file = open(str(output.name), 'w')
my_file.write(text)
my_file.close()
self.setResult("htmlfile",output)
```

This PythonSource module will create a HTML document with an image generated from executing the tmap web service.

Connect the modules together as shown in Figure 23(a).

Group the following modules: runAndWaitFor (Type module), runAndWaitFor (Method module), runAndWaitForResponse, TtmaptResult, TtmapResultGraphics_container and name the group as “Execute Tmap”, see Figure 23(b).

Name the pipeline as “Tmap” and save your work.



(a) (b)

Figure 23: (a) Tmap Workflow before grouping (b) Tmap Workflow after grouping.

The workflow is now ready to be visualized. As we have a RichTextCell module, pressing the Execute current pipeline button will send the current pipeline with the current parameters to the VisTrails Spreadsheet, resulting on an image similar to Figure 24. The bars across the top represent areas where transmembrane segments are predicted.

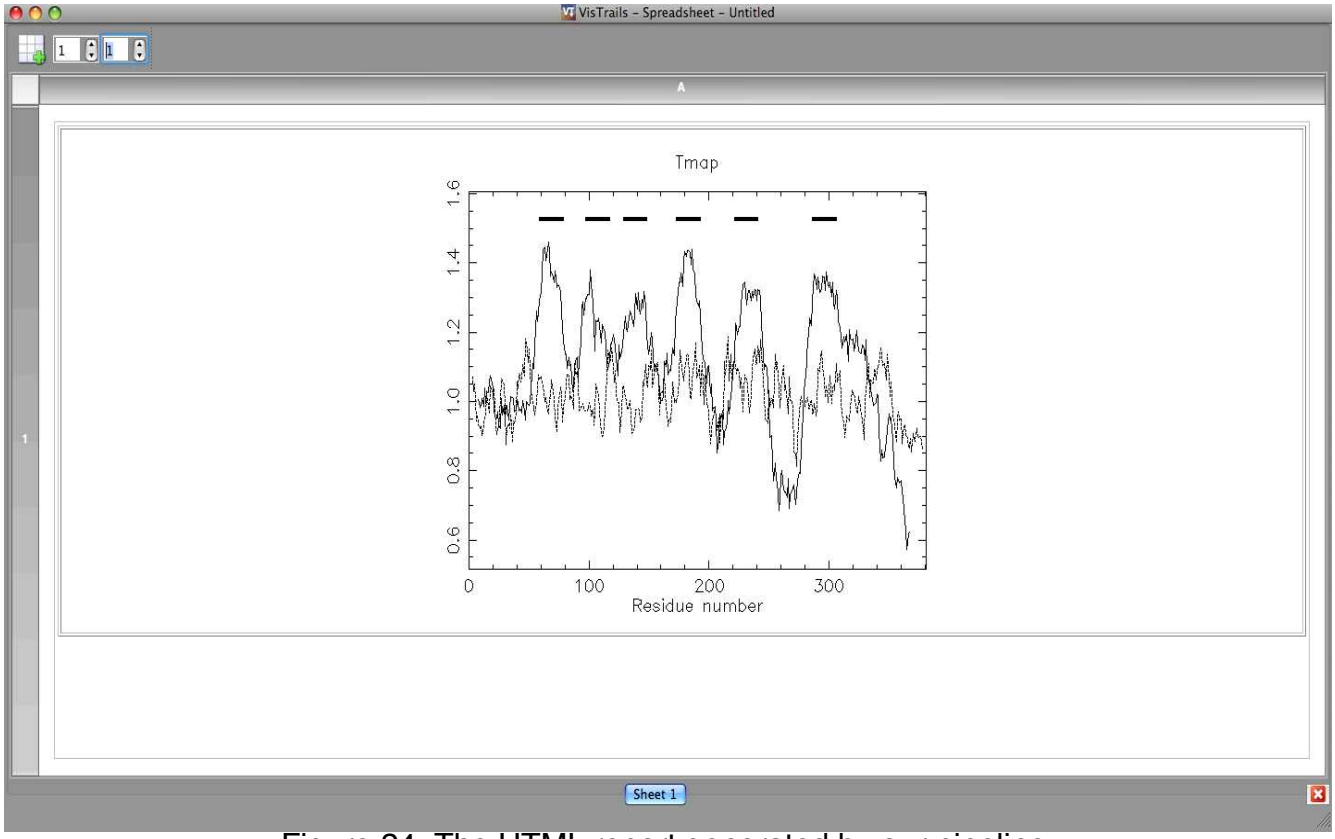


Figure 24: The HTML report generated by our pipeline.

3.3.5 Prophet Workflow

In the Prophet workflow a profile/matrix is created from a multiple alignment using prophecy program. The region from 110 to 1171 in the cDNA sequence, obtained from querying the database with the following entry embl:xlrhodop, is translated with the transeq program. The translated sequence and the profile/matrix are sent to prophet program to create a profile analysis. A profile analysis is a sequence comparison method for finding and aligning distantly related sequences.

Add the following modules to the “Emma” pipeline:

- Sequence (under “Types” for the prophecy web service)
- Tprophecy (under “Types” for the prophecy web service)
- runAndWaitFor (under “Types” for the prophecy web service)
- runAndWaitForResponse (under “Types” for the prophecy web service)
- TprophecyResult (under “Types” for the prophecy web service)
- runAndWaitFor (under “Methods” for the prophecy web service)
- Sequence (under “Types” for the transeq web service)
- Ttranseq (under “Types” for the transeq web service)
- runAndWaitFor (under “Types” for the transeq web service)
- runAndWaitForResponse (under “Types” for the transeq web service)
- TtranseqResult (under “Types” for the transeq web service)
- runAndWaitFor (under “Methods” for the transeq web service)
- TprophetInfile (under “Types” for the prophet web service)
- Tprophet (under “Types” for the prophet web service)
- runAndWaitFor (under “Types” for the prophet web service)

- runAndWaitForResponse (under “Types” for the prophet web service)
- TprophetResult (under “Types” for the prophet web service)
- runAndWaitFor (under “Methods” for the prophet web service)
- PythonSource (under “Basic Modules”)
- RichTextCell (under “Spreadsheet”)

Set the label “Create_HTML” to the PythonSource module and the label “Display_HTML” to the RichTextCell module.

In the PythonSource module add one input port: “outseq" of type String and one output port: “filename1" of type File and type or paste the following code in the text area:

```
outseqinput = self.getInputFromPort('outseq')
output1 = self.interpreter.filePool.create_file()
f1 = open(str(output1.name), 'w')
text = '<HTML><TITLE>EMBOSS Webservice</TITLE><BODY BGCOLOR="#FFFFFF">'
f1.write(text)
text = '<H2>Prophet Out Sequence</H2><BR>'
f1.write(text)
splitnewline = outseqinput.split('\n')
for element in splitnewline:
f1.write(element)
f1.write('<BR>')
text = '</BODY></HTML>'
f1.write(text)
self.setResult("filename1",output1)
f1.close()
```

This PythonSource module creates an HTML file with the profile analysis.

Connect the modules together as shown in Figure 25.

Select the following modules: runAndWaitFor (Type module), runAndWaitFor (Method module), runAndWaitForResponse, TprophecyResult and create the group “Execute Prophecy”. Create the following groups also: “Execute Transeq” and “Execute Prophet” (see Figure 26).

Name the pipeline as “ProphetOutput” and save your work.

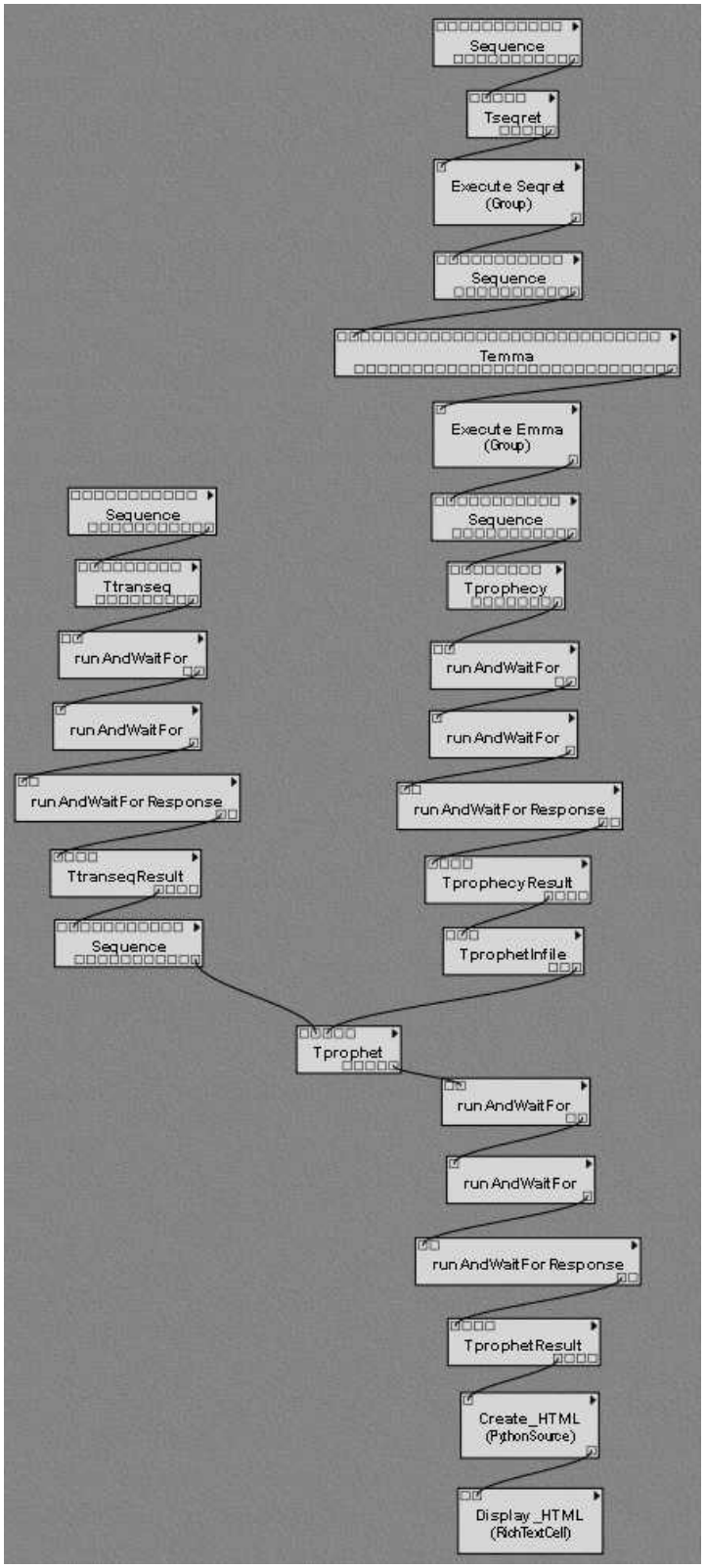


Figure 25: Prophet workflow before grouping modules.

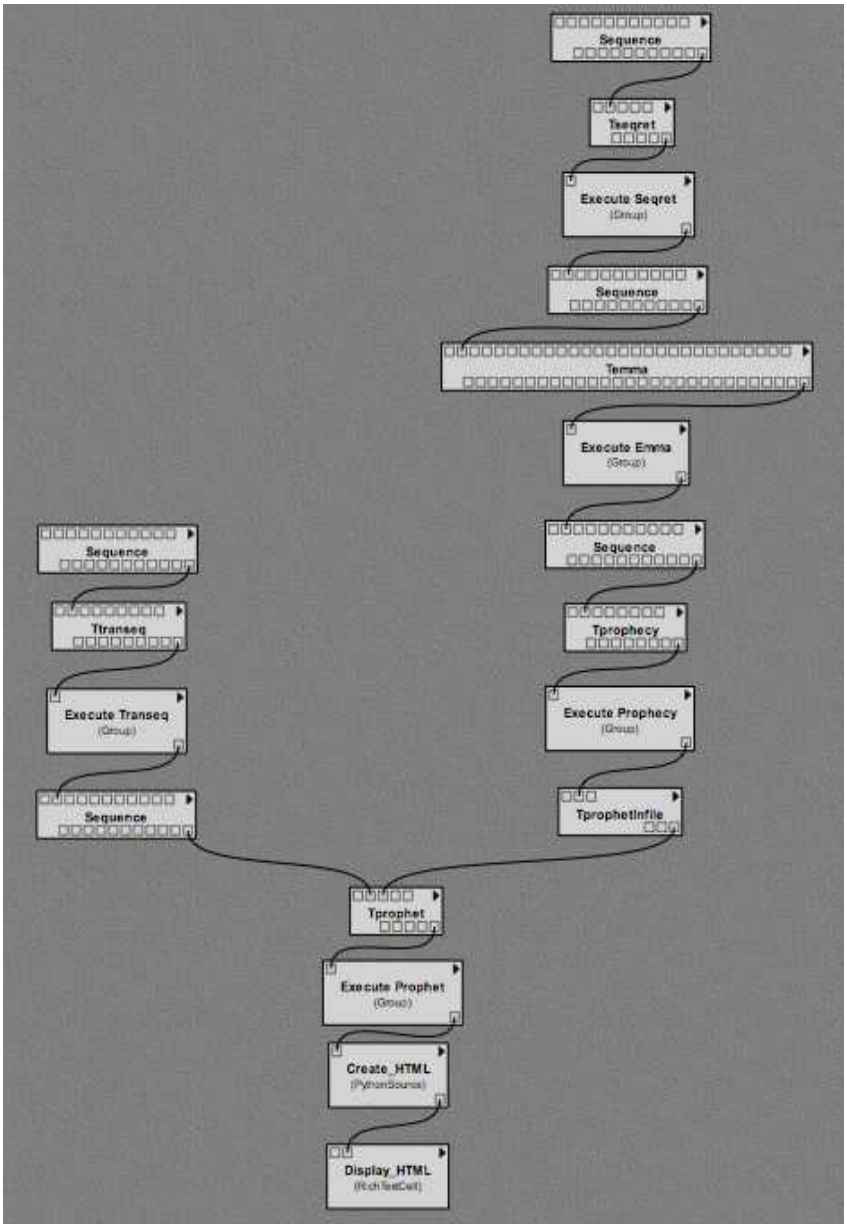


Figure 26: Prophet workflow after grouping modules.

The workflow is now ready to be visualized. As we have a RichTextCell module, pressing the Execute current pipeline button will send the current pipeline with the current parameters to the VisTrails Spreadsheet, resulting on an image similar to Figure 27. The vertical bars represent residues that are identical between the ops2 aligned sequences and the rhodopsin, and the colons represent conservative substitutions.

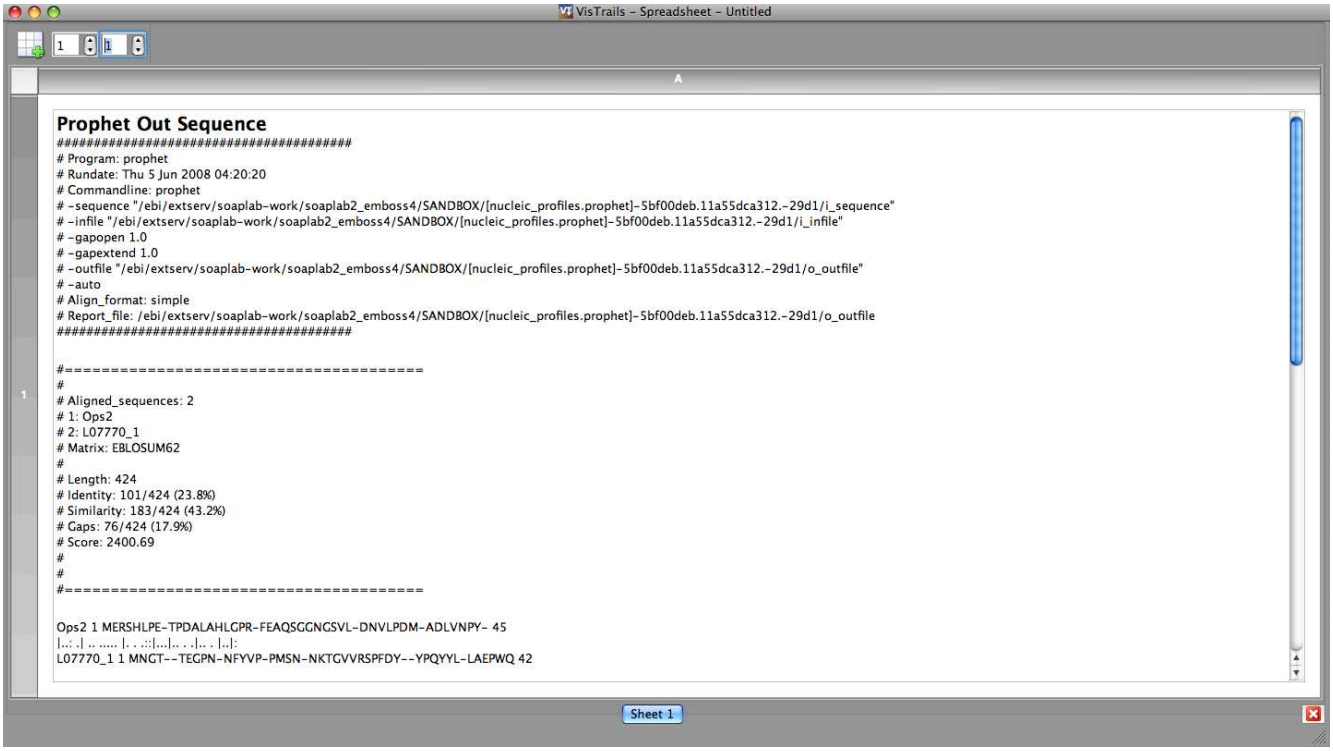


Figure 27: The HTML report generated by the execution of the pipeline.

3.3.6 Adding more nodes to the version tree

In this section we will send two different sequenceIDs to the seqret program and display the different sequences in the VisTrails spreadsheet. We will also pick different sequences align and visualize them with prettyplot program.

In the “Seqret” workflow select the module Sequence and change the parameter sequence_usa with the following value: embl:X13776. A new node will be created, set the tag to: “Seqret embl:X13776”. Repeat this procedure but change sequence_usa parameter value to “Seqret uniprot:p12345”. Save your work.

Execute the following three pipelines: “Seqret”, “Seqret embl:X13776” and “Seqret uniprot:p12345”. You can visualize the different results obtained from the execution of the Seqret workflow with different sequences IDs in the same screen with VisTrails Spreadsheet (See Figure 28).

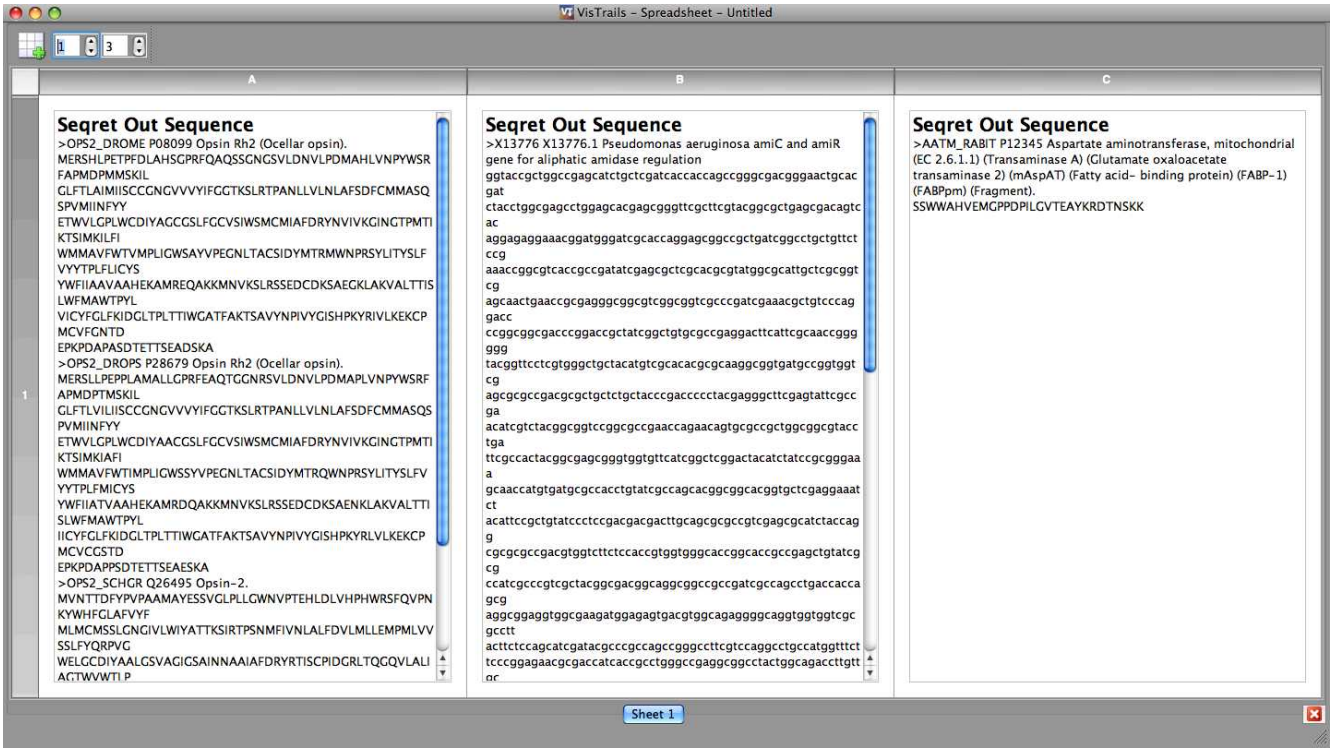


Figure 28: Output of running the different Seqret workflow versions.

Now let's compare the alignments of different sequences using Prettyplot workflow. Go to Prettyplot workflow, select the module Sequence and change the value of the sequence_usa parameter to "uniprot:op1*". A new node in the Version Tree will be created, label it with a descriptive name like: "Prettyplot uniprot:op1*". Save your work. Execute the following two pipelines: "Prettyplot" and "Prettyplot uniprot:op1*" (See Figure 29).

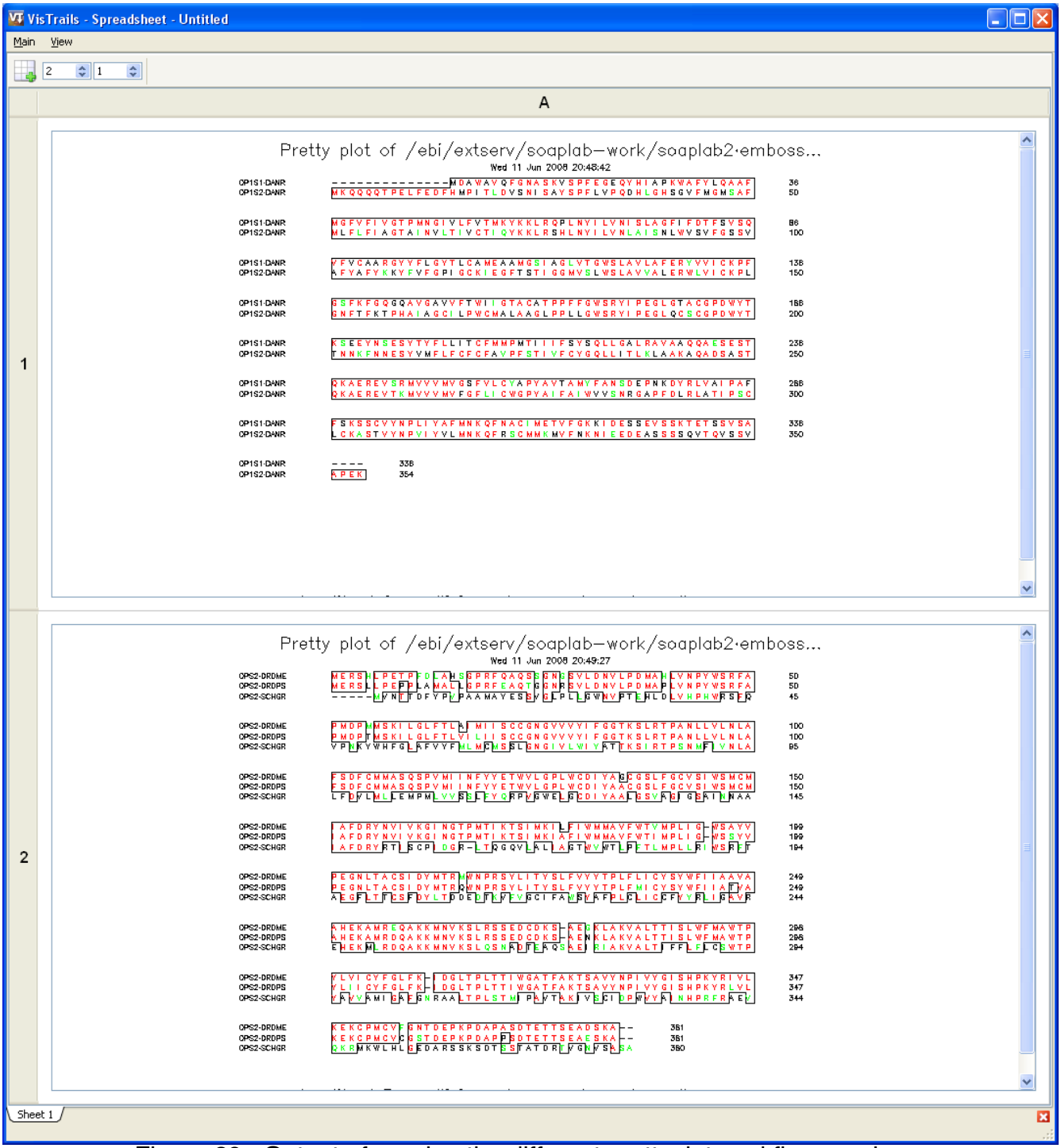


Figure 29: Output of running the different prettyplot workflow versions.

Figure 29 shows the alignment of different sequences. In the first cell of the Spreadsheet we have aligned ops1 sequences from two zebrafish species and in the second cell ops2 sequences from two Fruit fly and a desert locust species are aligned. The red colored residues are conserved residues in the alignment. The green colored residues are where similar residues occur in a particular position in the alignment.