20th International Congress on Modelling and Simulation, Adelaide, Australia, 1–6 December 2013
www.mssanz.org.au/modsim2013

Creating workflows that execute external code bases that are under development

T. Smith[a], **N.J. Car**[a] **and D. Smith**[b]

[a] CSIRO Land & Water, Environmental Information Systems
[b] Bureau of Meteorology Climate & Water, Water Data Management
Email: timothy.smith@csiro.au

Abstract: There is an increasing interest in the use of scientific workflows as a way to automate data management and model execution without requiring deep computing technical knowledge. Scientific workflows allow users to re-use previously developed code in multiple languages while providing repeatability and flexibility.

Additional complexity is introduced when scientific workflows use code that is still in development, especially when the code development is not linked to workflow development. In this paper, within the context of a particular workflow engine, we look at existing tools for code re-use and techniques to manage the complexity of working with rapidly changing code.

Hydrologist's Workbench (HWB) is a suite of tools, activities and recommendations built to support Microsoft's Project Trident, a scientific workflow engine. HWB contains several tools that assist in turning existing code into Trident Activities – atomic, composable, executable modules – via a process known as code "wrapping". The existing tools within HWB are designed to wrap code that is reasonably stable and unchanging and therefore these tools are insufficient for code that is subject to changes over time.

We have investigated techniques to minimise the effort required to turn code under active development into Activities and workflows. The techniques include: an agile methodology for workflow and code co-development; treating the code as a dataset itself; using an agreed interface; dynamically generating then executing scripted code; a simplified, template-based Activity generation tool and manual transliteration. Issues around testing, versioning, integration and communication are also discussed

Keywords: *scientific workflows, versioning, software design methodology*

1. INTRODUCTION

Scientific workflows have many advantages over applications, but it is problematic to turn an application's code base into a workflows when the code base is still changing. Scientific workflows are used to automate data management, for program and model execution and to carry out other data-related tasks. They do this without requiring the operator to have deep computing technical knowledge which makes them very useful for professionals such as scientists who may know well their research domain but may lack skills such as programming. Scientific workflows also often display an interface for operators to use to in order to monitor task execution and track the outcomes of multiple workflow runs.

This paper discusses the problem faced implementing Microsoft Trident, a Scientific Workflow Engine[1] workflows on top of a changing code base. This issue will occur whenever workflows are applied to code bodies that are not complete, which is somewhat likely to be the case for large projects where scientific workflows are expected to be employed, such as the Bioregional Assessments project (SEWPaC, 2012)[2].

Our team consisted of two sets of developers; part in CSIRO, part in the Bureau of Meteorology (BoM), and each with a different focus. The CSIRO sub-team used Trident as a workflow tool and applied it to geoprocessing code written by the BoM sub-team. The geoprocessing code produces hydrological catchment boundaries and a node-link network from a series of geospatial inputs derived from digital elevation models and topographical mapping data for the BoM's Geofabric project (BoM, 2013). Using Trident added provenance and metadata capture for, multi-user access to, and non-technical use of the geoprocessing code. Until Trident's introduction, running the geoprocessing tasks involved command line scripting, the interpretation of error logging and lots of manual file management for inputs and outputs. While implementing Trident, the BoM's geoprocessing code was itself still undergoing development.

We state a series of approaches we tried to solving this problem and then characterise and present techniques we distilled from testing those approaches. We make recommendations based on the merits of these techniques for those facing similar problems.

2. PROBLEM SPECIFICS

The BoM's geoprocessing code was a series of custom, complex Python[3] modules that undertook processing using Python's code libraries, extension modules and ESRI's ArcGIS[4] program libraries. To run the code, a user executed a Windows batch file, which called a Python main program, which called Python code in a series of other files, which called ArcGIS as needed. This is illustrated in Figure 2 "BoM Geofabric code".

Trident is a scientific workflow system that takes re-usable software modules known as Activities and allows users to graphically compose these into workflows. Trident Activities are .NET classes and were written in C#. HWB (Fitch *et. al.*, 2011) contained pre-built Trident Activities that were used in this project, and extensive use was made of certain HWB utilities.

Four characteristics were desired for a solution to the problem:

Provenance: That provenance at least sufficient for reproducibility was preserved.

Identical: That the Trident workflow and the Python code performed identically. This was hard to measure at certain scales due to there being few code test routines written.

Updatable: That should the BoM Python code be changed in the future, the workflow could be updated without requiring significant effort or Trident knowledge. Changes to the BoM code could range from simple changes to the code through to the addition (or removal) of entire conceptual steps. Post handover of the Trident workflow system to the BoM the BoM would be entirely responsible for updating the workflow.

Understandable: That the Trident workflow be reasonably easy to understand, especially by those already familiar with the BoM Python code.

The problems faced were primarily in three areas: communication between sub-teams, compatibility between modules and compatibility between abstraction levels. Changes to code had to be communicated to the other sub-team, which introduced extra communication overhead and thus reduced coding efficiency. In this project, the BoM and CSIRO team were severely time constrained, working on multiple projects

[1] http://tridentworkflow.codeplex.com
[2] CSIRO investigation for this project proposes Trident for use within it.
[3] http://www.python.org
[4] http://www.esri.com/software/arcgis

simultaneously and located in separate cities, Layers of abstraction were used to help minimise the necessity and frequency of communication, but it could be difficult to know which changes would break compatibility between abstraction levels. Finally changes in modules could introduce compatibility problems, meaning modules could not be isolated from development.

Technique: Agile Development

The two sub-teams investigated the use of the Agile software development methodology. Agile development is predicated on rapid, iterative and incremental development. While Agile was followed within sub-teams, it was not used between sub-teams – though many tools and techniques of Agile were adopted. Our joint team's code development lifecycle consisted of the steps presented in Figure 1.

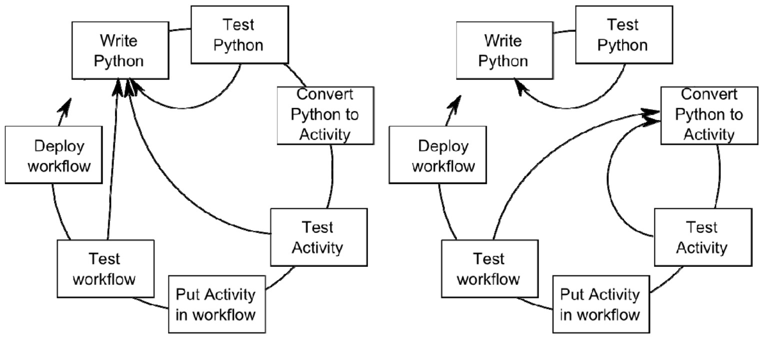


Figure 1: Agile methodology for Geofabric project (left) and methodology as applied (right). Note weaker connection between *Test Python* and *Convert Python to Activity*.

In order to increase iteration frequency, Agile strongly suggests increasing automation, but it was difficult to automate all steps, especially "Convert Python to Activity". Without automation, the manual effort involved reduces the utility derived from rapid iteration and therefore Agile generally. Over time, techniques were developed to increase automation and this increased responsiveness to changes and confidence in deployment (benefits of Agile manifested within the CSIRO team).

3. APPROACHES

Four approaches were tried and its value assessed against the four desired characteristics (see *Introduction*). Figure 2 shows flow charts for the original BoM approach and the 4 new approaches for comparison and will be referred to in the following subsections. Note that Approach 4 is quite similar to the BoM code.

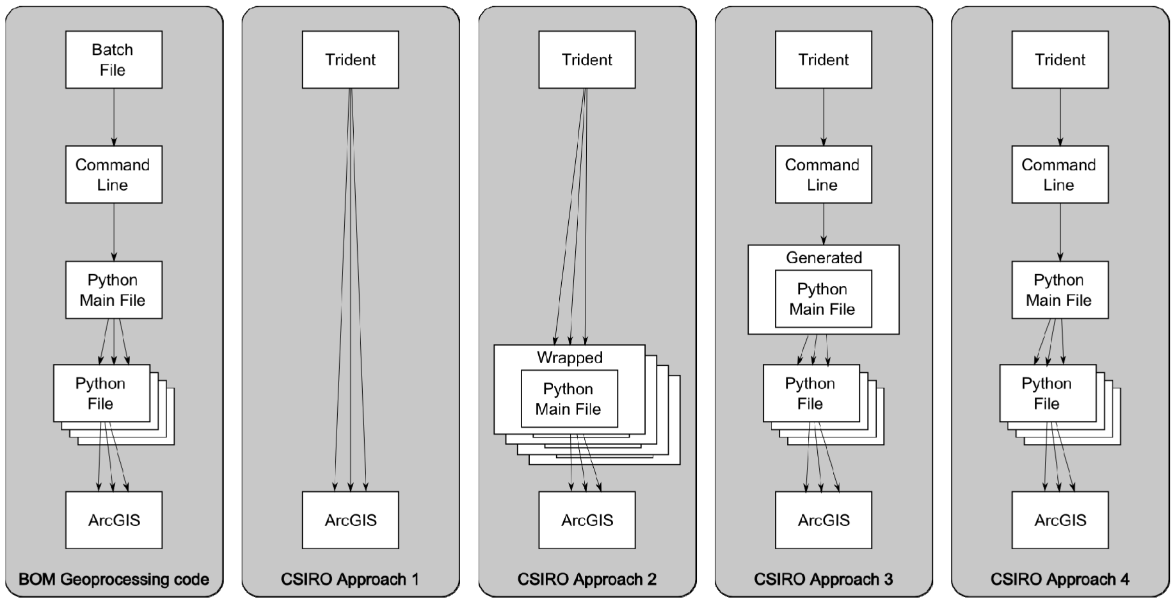


Figure 2: Different approaches to interacting with an ArcGIS geospatial processing library through a custom, complex, Python code base under active development. Arrows indicate hierarchy of execution (calling).

3.1. Manual duplication of Python behaviours

The first approach was to use Trident Activities to mimic the behaviours of the Python code. This approach required reverse-engineering the Python code into specifications, then building and assembling Trident

Activities that match those specifications. Most of the required Activities were simple data manipulations (that already existed) or calls to ArcGIS.

Tools within the ArcGIS toolbox were converted by a semi-automated process into a set of Trident Activities, with one Activity for each tool. Figure 3 gives an example tool list, and Figure 4 gives an example of how an ArcGIS tool was represented in Trident as an Activity. The approach is expressed as "CSIRO Approach 1" in Figure 2.

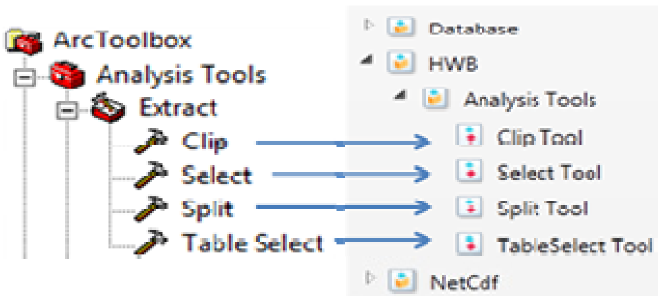


Figure 3: A list of geospatial processing tools as listed in ArcGIS (left) and as Activities in Trident (right). This 1:1 mapping of geoprocessing tools to Activities is the 'CSIRO Approach 1' as per Figure 2.

This approach was initially proposed before the start of the project mentioned here and was expected to result in a very large Trident workflow of over four hundred Activities (Car and Box, 2012). While this was inconveniently large, there is a subsection feature in Trident that can help with such large workflows so it may be applicable elsewhere.

Provenance: Good. The version of Python code being duplicated was not captured.

Identical: Poor. Turning Python code into specifications, then specifications into Activities, then assembling Activities into workflows left a great deal of room for human error and for code behaviour to diverge.

Updatable: Poor. All updating required manual work. Changes to the Python code must also be duplicated in Trident, potentially requiring the creation of new Activities.

Understandable: Good. The Trident workflow was easy to understand as it was more or less a translation of the Python code and thus familiar to the BoM team. The size of the workflow meant that behaviour was surfaced to the user at a very low level.

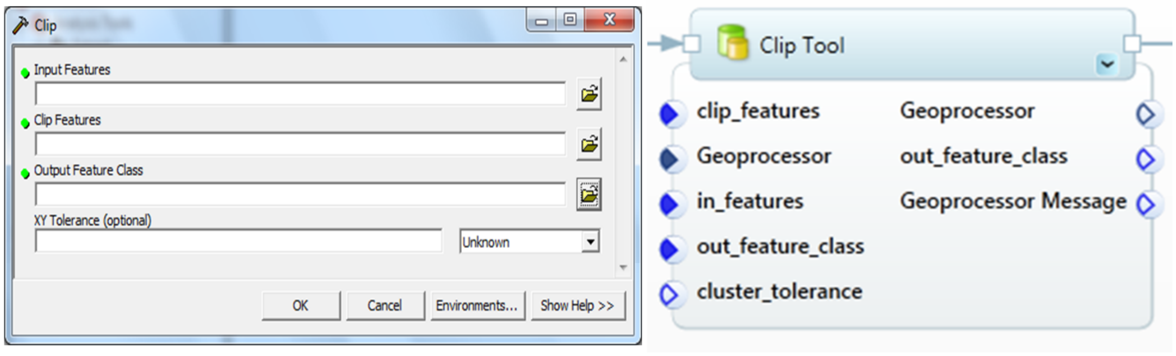


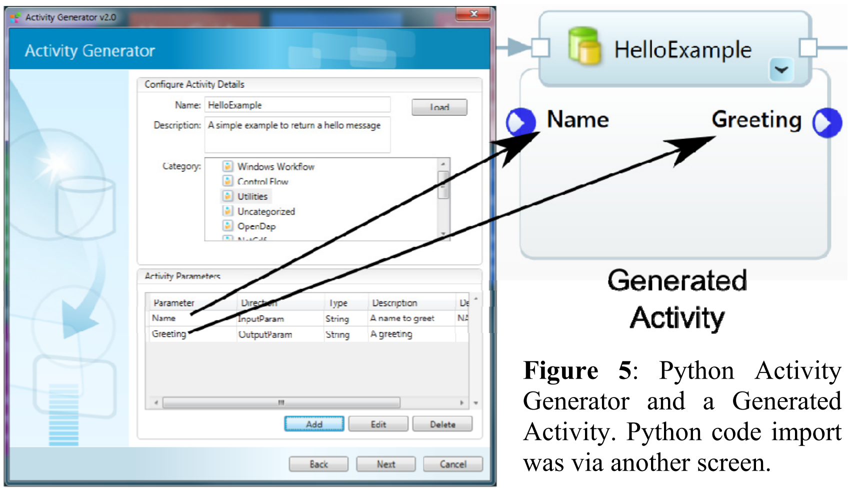
Figure 4: The ArcGIS 'Clip' tool interface as seen in ArcGIS (left) and as an Activity in Trident (right) showing the interface inputs as Activity inputs (circular icons on the left). The Trident Activity also shows outcomes from the tool's use as Activity outputs (icons on the right).

3.2. Wrapping Python code into Trident Activities using Activity Generator

This approach used an Activity Generator tool that is part of HWB to turn Python scripts into Trident Activities – these Activities are referred to hereafter as Wrapped Python Files. Figure 5 shows a screenshot of the Activity Generator Tool and a resultant generated Activity. Not shown is the screen responsible for specifying the file to be imported.

The Activity Generator was easy to use but limited in three ways; complex Python types were not supported, only one file could be imported at a time and the process of using Activity Generator was a manual one.

This approach is summarised as "CSIRO Approach 2" in Figure 2 (Car and Box, 2012), where Trident uses Wrapped Python Files that each use Python code to manipulate data and call ArcGIS.



Configure Activity Details

Name: HelloExample

Description: A simple example to return a hello message

Category: Windows Workflow, Control Flow, Utilities, Uncategorized, OpenDap

Activity Parameters

Parameter	Direction	Type	Description	De
Name	InputParam	String	A name to greet	NA
Greeting	OutputParam	String	A greeting	

Figure 5: Python Activity Generator and a Generated Activity. Python code import was via another screen.

Provenance: Excellent. Even the Python code used could be retrieved with some effort.

Identical: Good. The Activity Generator tool removed any chance of human error in Python script wrapping.

Updatable: Poor. Any changes to the Python code required rewrapping and replacing the appropriate Wrapped Python File Activities.

Understandable: Excellent. Existing Python code modules were duplicated largely 1:1 with Activities.

3.3. Trident downloading and modifying Python code

This approach was to create custom Activities that download the actual Python code used by the BoM team. It then rewrote the Python main method as each Activity was executed, passing Trident parameter information into the Python code via the main method. After execution Python outputs were harvested and translated into Trident output parameters.

For a summary see "CSIRO Approach 3" in Figure 2. Trident generated a Python Main file. Trident used a Command line interface to call the Python Main File. The Python Main File called the normal BoM Python files, which in turn called ArcGIS.

Provenance: Excellent. The BoM Python code was available or specified precisely.

Identical: Good. The use of BoM Python code meant almost all code used by Trident was the same as that used by the BoM (and therefore had the same behaviour). Only the generated file could differ.

Updatable: Good. Only changes in the Python Main File required any work at all. Changes to underlying Python code were immediately and invisibly incorporated.

Understandable: Excellent. Python modules were duplicated and Python warning, error and debug messages were propagated into Trident.

Technique: Code as Data

A technique used to very good effect during this approach was that of Code as Data – treating the code (in this case Python code) that controlled the behaviour of Activities as if it were simply another form of data that could be downloaded at execution time.

This added a level of indirection which complicated testing of Activities (behaviour at execution time was indeterminate without knowing in advance the code that would be downloaded) but had the advantage of allowing the BoM team to update the behaviour of the Trident Activities automatically.

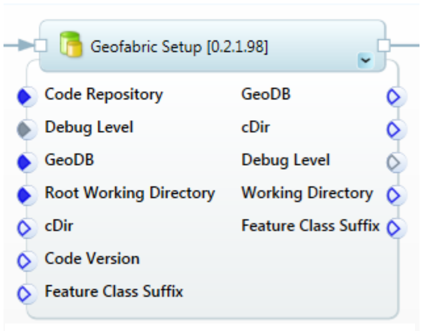


Figure 6: Trident Activity to check out subversion code.

The Python code was stored in a repository so definitive versions of it could be specified. Figure 6 shows the Trident Activity that downloaded code from the repository. All the Python code was downloaded at the start of the workflow rather than on a step-by-step basis. This had the advantages that all Activities within a workflow would use the same version of the code by default and that code could be modified or generated by one Activity and used by a later Activity.

3.4. Agreed command line interface

The final approach investigated was for the BoM and CSIRO teams to agree to a standard and stable command line interface for each step of the Python code. This happened as it was recognised that the last major potential difference between the Trident behaviour and the BoM Python code behaviour was now in the generated Python main file.

The Activity Builder tool was developed to make it easier to create and modify Trident Activities by modifying a simple configuration file. See Technique: Activity Builder Tool (below) for details.

Trident downloaded the Python files used by the BoM (that had the agreed interface) and then used the Command Line to call the Python code, passing parameters in and out via the agreed interface. The Python code called ArcGIS as normal. This approach "CSIRO Approach 4" can be seen in Figure 2.

Provenance: Excellent. Subversion reference to all Python code meant version was precisely recorded. Command Line parameters were recorded, allowing Python code to be manually and identically run outside Trident if necessary.

Identical: Excellent. Errors were reduced in data communication between Trident Activities and system configurations.

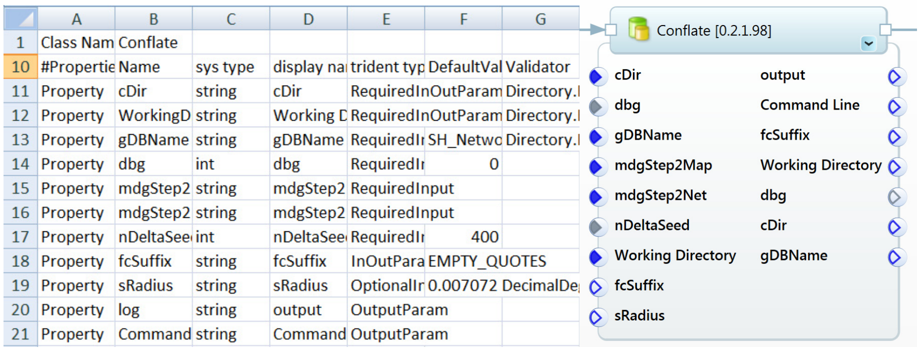
Updatable: Excellent. The Activity Builder tool and a standard template for the Activities reduced the complexity of creating and modifying Activities. Knowledge of C# and Visual Studio was still required.

Understandable: Excellent. Python modules were duplicated and Python warning, error and debug messages were propagated into Trident.

Technique: Activity Builder Tool

In order to make it easier for the BoM team to update Trident Activities in the future, and in an effort to increase build automation of the software development process (see Technique: Agile Development) the Activity Builder tool was developed.

The Activity Builder tool was a code generator that converted configuration files into Activities. The configuration file was a simple format readable by excel or a text editor. It specified properties for an Activity. When run, the tool would create C# files representing an Activity. One file (the Activity file) stored information on Activity metadata (such as description) and parameters and was updated each time the tool was run. The other file (the Activity Core file) contained the code that controlled the Activity's behaviour on execution and was created only once. A Trident Activity was created from both files at compile time.



	A	B	C	D	E	F	G
1	Class Nam	Conflate					
10	#Propertie	Name	sys type	display nai	trident typ	DefaultVal	Validator
11	Property	cDir	string	cDir	RequiredInOutParam		Directory.I
12	Property	WorkingD	string	Working D	RequiredInOutParam		Directory.I
13	Property	gDBName	string	gDBName	RequiredI	SH_Netwo	Directory.I
14	Property	dbg	int	dbg	RequiredI	0	
15	Property	mdgStep2	string	mdgStep2	RequiredInput		
16	Property	mdgStep2	string	mdgStep2	RequiredInput		
17	Property	nDeltaSee	int	nDeltaSee	RequiredI	400	
18	Property	fcSuffix	string	fcSuffix	InOutPara	EMPTY_QUOTES	
19	Property	sRadius	string	sRadius	OptionalI	0.007072	DecimalDe
20	Property	log	string	output	OutputParam		
21	Property	Command	string	Command	OutputParam		

Figure 7: The Excel interface (left) and the resultant Trident Activity (right) of the Activity Builder Tool

Technique: Agreed Command line interface

The code in question was called via the command line in all situations – either through a batch file (BoM), or via Trident making a command line call. This commonality was then exploited in order to create an agreed interface. The CSIRO team programmed Trident to assume the interface, and the BoM team programmed their Python code to conform to it.

The Agreed interface was very simple. Variable inputs were entered sequentially at the command line. All inputs were simple types (strings). Optional inputs could be left off, but were all at the end. New inputs could be added later, but only as optional inputs and only at the end[5].

[5] This was for backwards compatibility purposes. The intent was that an old version of the Trident workflow would be able to call new versions of the code. In practice, the code was still insufficiently mature to require backwards compatibility.

In order to conform to the interface the BoM team modified the Python code so that all outputs from the major steps of the Geofabric code were via file. File outputs could be turned into Trident outputs by the workflow. This was the least elegant part of using an agreed command line interface.

4. RESULTS

The most successful approaches were those that required the least human intervention to transform Python code into Activities. Table 1 summarises the performance of each approach against our four desirable criteria. This culminated in use of the *agreed command line interface*, an approach that minimises communication overheads by using a standard configuration.

Table 1: Summary of characteristics of approaches

	Provenance	**Identical**	**Updatable**	**Understandable**
Manual Duplication of Python behaviours	Good	Poor	Poor	Good
Wrapping Python code into Trident Activities	Excellent	Good	Poor	Excellent
Trident downloading and modifying Python code	Excellent	Good	Good	Excellent
Agreed command line interface	Excellent	Excellent	Excellent	Excellent

5. DISCUSSION

Which approach to use when this problem is encountered is dependent on many context-specific variables, however four techniques trialled by the Geofabric project are likely to be more widely applicable. These techniques can be implemented in multiple ways and are not workflow engine or application domain specific.

The most broadly applicable technique is **Agile Development**: The effort involved in increasing iteration speed was a good investment, increasing development speed and ability to respond to change. Agile development is strongly recommended for any multi-team project.

The **Code as data** approach was critically important to ensuring that the behaviour of the Python code and the workflow was identical. If working with code that is subject to change, viewing that code as data (which is mutable) rather than code (which generally is not mutable and ideally deterministic) can be very valuable.

An **Agreed command line interface** is highly useful, but was only possible due to the BoM team's effort in refactoring code. In situations where this is possible it is very useful to the workflow team. If in a similar situation and if at all possible the workflow team should attempt to come to an agreement on an interface with other teams, and a command line interface is compatible with an enormous number of tools.

The **Activity Builder tool** provided a standardised method for communicating interface changes, and reduced iteration time. While recommended for any multi-team project with communication constraints it is by far the most workflow engine specific of all the techniques.

REFERENCES

Barga, R., Simmhan, Y., Withana, E.C., Sahoo, S. and Jackson, J. (2010) Provenance for Scientific Workflows: Towards Reproducible Research. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering. Online at http://ceng.usc.edu/~simmhan/pubs/barga-deb-2010.pdf.

Bureau of Meteorology (BoM) (2013), The Australian Hydrological Geospatial Framework (Geofabric). Web page by the BoM. Online at http://www.bom.gov.au/water/geofabric. Accessed 7th July 2013.

Car, N.J. & Box, P. (2012). WIRADA 1.5 – SWIM HWB workflows for priority aspects of Geofabric production process. Water for a Healthy Country Flagship, WIRADA Client Report. CSIRO

Fitch, P., Perraud, J. M., Cuddy, S., Seaton, S., Bai, Q., & Hehir, D. (2011). The Hydrologists Workbench: more than a scientific workflow tool. In Proceedings, *Water Information Research and Development Alliance Science Symposium*.

SEWPaC (2012) The framework for bioregional assessments of coal seam gas and coal mining development. A report of the Independent Expert Scientific Committee on Coal Seam Gas and Coal Mining through the Department of Sustainability, Environment, Water, Population and Communities.