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

Characterisation of different integration strategies in scientific workflows

R.J. Bridgart [a] and T. Smith [a]

[a] *Commonwealth Scientific and Industrial Research Organisation, Department of Land & Water*
Email: robert.bridgart@csiro.au

Abstract: Scientific workflow engines are powerful tools that allow for the composition of activities (computational components) into a reproducible, adaptable and well orchestrated whole – but the utility of any workflow is critically dependent on the way in which the activities are created. In this case the workflow engine being used is called Trident (a Microsoft Research application built on .NET WF). Trident compatible activities are able to be created in different ways, each method having its own trade-offs in areas such as performance, physical disk size, network bandwidth use, reproducibility, documentation and provenance tracking.

The Hydrologists Workbench (HWB) project investigated and created tools to support several different methods to wrap stand-alone code / programs into activities and this paper details and characterises those methods. Using a common data interface, design conventions and support tools, HWB is able to facilitate the construction of activities which exhibit the desirable characteristics of reproducibility and provenance tracking whilst at the same time not sacrificing workflow flexibility. By being able to create different activities using specific methods the workflow as a whole can be at the same time both flexible (allowing for model addition, subtraction or substitution) and reliable (easily able to verify correctness and provenance).

Four methods of wrapping code / programs are discussed and evaluated against a common set of criteria. These methods are programming the Activity manually, using a tool to wrap code / scripts, using a tool to wrap an entire application and using a tool to wrap a reference to an external application. Each of the four techniques is applicable under different circumstances and their strengths and weaknesses are discussed in regards to reproducibility, provenance, efficiency and usability.

Although requiring more initial effort, encapsulating a program in an Activity and placing it within a workflow can provide improvements to reproducibility, provenance and reuse. By making an informed decision around which technique to use when wrapping functionality it is possible to optimise reproducibility, provenance and usability in the workflow.

Keywords: *Workflows, Reproducibility, Provenance, Hydrologists Workbench (HWB), Project Trident*

1. INTRODUCTION

Advances in science are increasingly being facilitated by the ability of researchers to automate the linking of complex processing steps, with workflows being used as a means to represent and manage these linkages (Gil *et al* 2007). A scientific workflow describes how data flows across these linkages and manages the running of the individual processing steps. In the context of the scientific workflow engine discussed in this paper, each of these processing steps will be referred to as an Activity. An Activity is a construct that defines a standard interface which exposes some input parameters, encapsulates some program logic and exposes some output parameters. A workflow (shown in Figure 1) describes the linkages between the Activities by defining which outputs from one Activity are connected to which inputs of another.

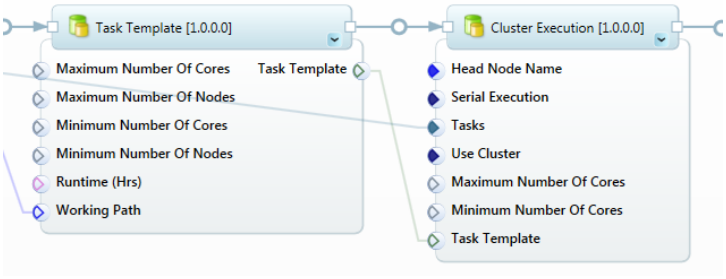

Figure 1. Example of two Activities in a Workflow

The creation of Activities is a trade-off between flexibility, usability, efficiency, reproducibility and recording, and it is possible to have multiple methods of Activity creation to allow different trade-offs to be used.

While this paper is heavily focused on a specific workflow system and existing tools the methods are applicable to similar tools for other workflow systems.

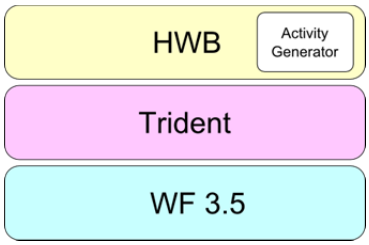

Figure 2. HWB technology stack

The workflow system we use consists of multiple layers of software as shown in Figure 2. Microsoft Windows Workflow Foundation (WF) 3.5 is a Microsoft .NET software framework which contains definitions for workflows and Activities including functionality to execute user defined workflows. Project Trident (Barga *et al*, 2008) is built on the .NET framework (written in .NET C#) and uses WF3.5. It allows users to graphically create new workflows, run them and to view provenance information, which is captured and saved each time a workflow is executed. Trident is one of many competing workflow tools; others like Kepler (Altintas *et al*, 2004) and Taverna (Oinn *et al*, 2004) perform similar functions but are built on alternate technology stacks (Java instead of .NET). The Hydrologists Workbench (HWB) (Cuddy & Fitch, 2010) is a suite of tools created to facilitate the development of the hydrologic and spatial analysis Activities in Trident. The primary tool that aids in the generation of Activities in HWB is Activity Generator.

The Activity Generator application provides a graphical user interface to facilitate the easy ‘wrapping’ of a program into a Trident compatible Activity. Wrapping, in this context, is the process of taking an existing, independent program and encapsulating it with the necessary software infrastructure to allow it to be compatible with WF3.5. A program can be expressed in several different forms; as code which needs compiling, as a script which needs interpreting, or as an executable file. There are multiple approaches which can be used to wrap programs, with different approaches being more suited to different program forms.

The different approaches to wrapping programs will each possess different strengths and weaknesses in areas such as reproducibility, provenance, efficiency and usability. The various approaches can be characterised by these factors:

Reproducibility refers to how reliably the Activity can be re-run and the same results[1] obtained given the same set of inputs. Factors that affect reproducibility are dependencies on data, programs and resources external to the scope of the Activity.

Provenance refers to the amount and quality of information captured by an Activity each time it is run (typically the inputs, program data, program logic and outputs), as well as other metadata where available. Factors that affect provenance are dependencies on external data, programs and resources, as well as the quantity and quality of metadata available.

Efficiency refers to how much using an Activity increases the costs (storage space, execution time, network usage, etc) compared to running the program directly. Factors that affect efficiency include the amount of overhead the Activity introduces, trade-offs made by the Activity between costs, and decreased ability to share common resources over multiple Activities.

Usability refers to how easily the Activity can be created by a user whether they are using the Activity Generator or constructing the Activity themselves from source code. Factors that affect usability are those which pertain to all user interfaces, as well as the number of steps in the wrapping process, the level of knowledge required and the amount of automated assistance provided to the user.

2. WRITING A NATIVE ACTIVITY IN .NET C#

The most direct way to capture program logic as an Activity is to write the code in the same programming language as the workflow framework is written in, .NET C#. This is achieved by implementing the WF3.5 Activity software interface and explicitly defining the Activities' inputs, outputs and the code to execute when the Activity is run. Doing so requires no special HWB tools, just the ability to create and compile C# source code and a reference to the required software libraries. Once compiled, the assemblies and any dependencies can be imported into Trident and used in a workflow.

2.1. Reproducibility

The reproducibility of an Activity of this type is determined by the contents of the main function which is called when the Activity is run. The function code may require the presence of external dependencies in the form of files or web services. These dependencies must be present and unchanging in order for the Activity to produce the same output for a given input. If these Activity dependencies are outside of the scope of the workflow itself (i.e. they are not created by other Activities in the workflow), then the potential reproducibility of the Activity is adversely affected. If the code is deterministic and does not rely on external dependencies, then the reproducibility of the Activity is excellent. Under these circumstances a user can be confident that the Activity will re-run and produce the expected outputs without the need to confirm the existence and integrity of any external dependencies.

2.2. Provenance

The program logic for this type of Activity is preserved in the compiled Activity assembly itself. In order to extract this logic the assembly needs to be decompiled; fortunately .NET (when not obfuscated) is readily able to be decompiled with the right tools. Doing so reveals the precise instruction set used to perform the actions of the Activity and is a very strong piece of evidence when tracking the provenance of the Activities output data. The inputs and outputs of the Activity can also be captured and saved by Trident. When no interactions with external sources take place in the code, an unambiguous record of what an Activity has done is recorded. When the code does interact with external sources (e.g. files), the design of the Activity determines what is captured and recorded at runtime (e.g. filenames, hash of file contents).

[1] This paper treats reproducibility as deterministic reproducibility, where the same inputs give exactly the same outputs. Reproducibility can also be stochastic, where results that are not identical can be thought of as 'the same'. Things that cause issues with reproducibility will do so for both deterministic and stochastic reproducibility.

2.3. Efficiency

The size of the Activity is directly related to its use of non-core libraries. Although compiled code is relatively small, all non-core libraries referenced by the code must also be imported into the Trident database so that the code can be executed when the Activity is run. These libraries may be large in size and their presence may increase the size of the Trident database considerably.

The performance of a directly coded Activity is typically very efficient relative to the other methods discussed in the paper.

2.4. Usability

The correct implementation of the Activity software interface requires careful attention to detail from the developer. Many details about the Activity must be set in a precise manner and failure to do so typically results in an Activity which will either fail to import into Trident or fail to operate as expected. This method requires reasonable technical abilities in order to create Activities as the user must not only be comfortable programming in C#, but also in implementing the Activity interface and compiling / deploying the final assemblies.

3. ASSISTED CODE WRAPPING IN ACTIVITY GENERATOR

Code written in some computer programming languages can be wrapped into Activities with the help of Activity Generator. Fitch *et al* (2011) describes how Activity Generator currently supports the use of two languages: Python and R. In order to wrap one of these languages using Activity Generator, the code is loaded into the application and the desired inputs and outputs are specified (which variables are to be exposed by the activity). The code is then embedded into the Activity as an encoded byte array. When the Activity is run, the embedded code and Activity inputs are fed into the local version of the language interpreter and executed. The desired results are then extracted and set to the output properties of the Activity.

3.1. Reproducibility

All of the program logic has been retained using this process and a high level of confidence is observed regarding the ability of the Activity to reproduce its results on subsequent runs. The leading cause of uncertainty regarding the reproducibility of the Activity is found to be in the reliance on the installed code interpreter which can vary in version and configuration. Interpreted code can rely on external dependencies just like the native C# discussed in the native code section and consequently has the same risks regarding reproducibility. Of greater concern is the possible presence of dependencies in the code on non-core code packages, the existence of these packages at runtime is an assumption beyond the Activities' control and so too is the packages' adherence to backwards compatibility in the event of newer versions being installed.

3.2. Provenance

For provenance this method has the advantage that the actual raw source code is embedded within the Activity and can be extracted and viewed if desired. At this stage the code is not easily extractable from the assembly and a helper tool to facilitate this is yet to be built, although the development of such a tool is planned. Improvements to provenance recording could be made by at least capturing the version of the interpreter used at runtime as well as the versions of relevant libraries / packages installed. The inputs, the code and the outputs can all be captured inside the Trident database, creating a good provenance trail for this type of Activity.

3.3. Efficiency

The size of the source code byte array is relatively small and so the overall size of the Trident database is only marginally affected by the presence of these Activities. Runtime performance is only slowed by the

process of extracting the code in order to prepare it for running by the interpreter. Unless the code is extremely fast executing, the extraction time overhead is relatively insignificant.

3.4. Usability

This option is very easy for most users as they only have to be familiar with the language they are wrapping. The Activity Generator wrapping process is simple and requires no special knowledge of C# or Trident beyond a basic understanding of what an Activity is in the context of a Trident workflow.

4. WRAPPING AN ENTIRE APPLICATION IN ACTIVITY GENERATOR

Executable files and their dependencies which can be invoked via the command-line can be wrapped in Activities. Activity Generator currently supports three such applications; IQQM, Realm and MODFLOW. Two are surface water models and one is a ground water model. Their common feature is that they are all physically small, self-contained command-line enabled applications. The wrapping approach taken was to embed the entire application (executables, dependent assemblies and model data) as a compressed embedded resource file in the Activity assembly itself. When the Activity is executed, this embedded data is be extracted to a temporary location and the application invoked with the input data.

4.1. Reproducibility

By preserving the entire application and its inputs an extremely high level of reproducibility is achieved as there is minimal reliance on external components. The only noteworthy external dependency in this instance would be any runtime libraries required by the application. For example a wrapped application written in C++ may use libraries which have not been included in the solution at compile-time and therefore require a C++ redistributable package to be installed on the machine. Such libraries typically adhere to strict backwards compatibility regimes and the application will fail to execute should a compatible version of the libraries not be present when the activity is run.

4.2. Provenance

The ability to recall how the Activity works is made simple because the entire application and data set is stored within the Activity assembly which itself is stored within the Trident database. The application and input data can be manually extracted and viewed when required, providing an excellent record of the Activities inner workings.

4.3. Efficiency

The most significant downside to using this approach is the physical size of the model/data payload that is embedded in the Activity. Even a small model with a modest data set will increase the size of the Activity assembly considerably. The model/data sets currently supported range in size from under one megabyte to ten megabytes when compressed. This size was considered acceptable considering the significant gains in reproducibility and provenance capture over other methods. The execution overhead at runtime to extract the package before execution is typically negligible compared with the runtime of the model itself.

4.4. Usability

Activity Generator makes wrapping a supported model fast and easy for the non-technical user. It can present a range of input and output parameters to the user which can then be selectively exposed on the Activity and can also automatically handle the packing of the model application and its dependencies.

5. WRAPPING A REFERENCE TO AN APPLICATION IN ACTIVITY GENERATOR

Embedding an entire application and its data is not always practical due to the sheer size of the data involved. In these cases an alternative compromise arrangement must be found which attempts to balance size and execution speed against reproducibility and provenance. Activity Generator supports the wrapping of the large surface water modelling application Source. Source is a powerful desktop application which in addition

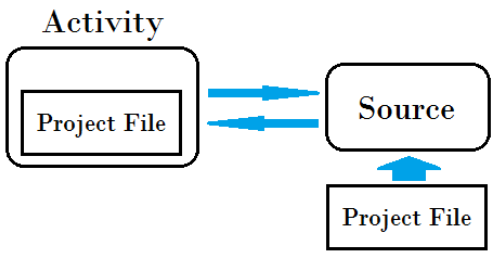

Figure 3. Source Activity and Source Application

to a rich graphical user interface also supports a service and command-line interface. At just over two hundred megabytes in size, Source is too large to embed in the Activity. An instance of a Source model is defined in a project file which contains all of the input and configuration data required to run the model. A range of input parameters can be configured within the project file such that they are exposed via the service interface for both setting and getting on a time-step by time-step basis during the temporal simulation. In order to wrap a model the project file is loaded into Activity Generator and the available inputs and outputs are determined. The user then selects which of the available inputs and outputs to make accessible on the Activity and the project file is embedded in the Activity assembly. Source can run locally or on a remote server and because of this the embedded project file is not used for execution, but a copy which is accessible to the local or remote Source application. The information required by a Source Activity to point to an instance of the Source application and to the duplicate project file is specified by additional input properties on the Source Activity.

5.1. Reproducibility

The ability of the Activity to reliably reproduce its results is limited by its reliance on two things; the availability of the correct version of the Source application and the existence of the duplicate project file. Both the availability of the application and the existence of the project file are beyond the control of the Activity and should either not be present and correct, the Activity cannot re-run. The Source application is evolving and complete backwards compatibility of results is not always guaranteed as new research improves internal models and issues get fixed. This means that the version of Source that is used is very important.

5.2. Provenance

The project file is embedded within the Activity assembly in order to provide the foundation for the Activities provenance. The model's default inputs and configuration are central to understanding what the Activity does. When combined with the input data applied to the Activity at the workflow level, the information captured on what the Activity does is very thorough. The current lack of version checks on both the Source application and the project file are the main factors which introduce uncertainty in the provenance.

5.3. Efficiency

The Source application itself is over two hundred megabytes in size, and its project files can range from one to several hundred megabytes. Most project files reside on the smaller side of this range and so are deemed appropriate to store in the Activity for provenance. The Source application must remain external as its presence would be an unacceptable burden on the size of the Trident database (the Trident database by default uses Microsoft SQL Server 2008 which is limited to four gigabytes, ten for 2008 R2). Some databases Execution speed is limited by the transfer rate of input data from the Activity to the Source application and the general overhead imposed by the service interface. This limitation can be mitigated by hosting the Source application locally or on a high-end server with a fast network connection between client and host.

A special advantage Source has over fully wrapped models is that if a Source model is to be run multiple times during a workflow, the Source server keeps the model loaded in memory after the first run, removing the need to load the project file into memory each time a run of the application is triggered, making subsequent runs much faster.

5.4. Usability

From the users' perspective the process of wrapping Source is similar to that of wrapping IQQM and Realm. The user is presented with a list of possible input and output parameters to expose and then selects the

desired parameters with Activity Generator automating the remainder of the wrapping process. From an underlying wrapping perspective Source varies in a small but significant way from the other models. Only certain types of inputs can be addressed via Source's service interface and as such any inputs required to be exposed by the Activity must be set to the correct type within the Source model prior to wrapping.

6. DISCUSSION AND CONCLUSIONS

There are many different techniques that can be used to expose new and existing pieces of functionality as Activities. By taking the best approach when creating these Activities based on the form of the functionality to be wrapped and on user requirements for the Activities, reproducibility, provenance and usability can be optimised. Creating these Activities and workflows has many advantages over traditional approaches like scripting. Although requiring more initial effort, encapsulating a program in an Activity and placing it within a workflow provides improvements to reproducibility, provenance and reuse. Many Activities are generic by design and can form part of an Activity suite which can be shared and used by many Trident users in their workflows. This makes it easier for others to reuse existing work with minimal effort because a well constructed Activity will have captured the functionality, addressed reproducibility and provenance, and hidden as much of the low-level complexity as possible from the user, all in a standardised form. Table 1 rates each wrapping technique on a scale of one to five stars (one being poor and five being excellent) against the four areas of reproducibility, provenance, efficiency and usability.

	Reproducibility	Provenance	Efficiency	Usability
Native C#	★★★★	★★★★	★★★★★	★
Assisted Code/Script	★★★	★★★★	★★★	★★★
Entire App	★★★★★	★★★★★	★★	★★★★
Referenced App	★★	★★	★★★	★★★★

Table 1. Ratings for each wrapping technique (1 to 5 stars)

ACKNOWLEDGMENTS

Activity Generator was originally developed as part of the Hydrologists Workbench project, an alliance between CSIRO's Water for a Healthy Country Flagship and the Bureau of Meteorology. Designed and programmed by Qifeng Bai, David Hehir and Shane Seaton. Paper reviewed by Dave Penton.

REFERENCES

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., & Mock, S. (2004) Kepler: an extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004*. Proceedings 16th International Conference on (pp. 423-424), IEEE.

Barga, R., Jackson, J., Araujo, N., Guo, D., Gautam, N., & Simmhan, Y. (2008) The trident scientific workflow workbench. In *eScience, 2008*. IEEE Fourth International Conference on (pp. 317-318), IEEE.

Cuddy, S., Fitch, P. (2010) Hydrologists Workbench: a Hydrological Domain Workflow Toolkit. Paper presented at the *International Congress on Environmental Modelling and Software*, Ottawa, Canada.

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*.

Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., … & Myers, J. (2007) Examining the challenges of scientific workflows. *Computer, 40*(12), 24-32.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., ... & Li, P. (2004) Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics, 20*(17), 3045-3054.