International Environmental Modelling and Software Society (iEMSs)
2010 International Congress on Environmental Modelling and Software
Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada
David A. Swayne, Wanhong Yang, A. A. Voinov, A. Rizzoli, T. Filatova (Eds.)
http://www.iemss.org/iemss2010/index.php?n=Main.Proceedings

On the appropriate granularity of activities in a scientific workflow applied to an optimization problem

Perraud, Jean-Michel[a], **Qifeng Bai**[a], **David Hehir**[a,b]
[(a)]CSIRO Land and Water, Canberra, Australia
[(b)]Australian National University, Canberra, Australia
email: jean-michel.perraud@csiro.au

Abstract: Scientific workflow management is an active area of research and development responding to an increase in the complexity of computational models, data analysis and data size. In the authors' experience, the distinction between scientific workflow software (SWS) and modelling frameworks or toolsets is at best not clear in the minds of users or developers, at least in the hydrology domain where the interest in concept of scientific workflow appears rather recent. It is understandably tempting for some users to assume that these new software tools aim to replace their existing modelling tools, with varying expectation depending on their prior satisfaction. While it is arguably clear that SWS can play a role in improving practices for high-level orchestration and traceability of scientific workflows, we explore in this paper the granularity at which activities can be usefully defined. We describe a case study, the calibration of a model, wrapping the components of a modelling framework (TIME) from the Trident and Kepler SWS. The problem is decomposed in several workflows comprising activities of differing granularities. We assess each approach against a set of criteria such as runtime performance and flexibility, discuss the feasibility and trade-off. The main findings are the design benefits stemming from having to clearly identify separate activities in the process, and that the difficulty of decomposing the optimization problem into finer-grained activities increases most markedly when needing iterative control flow capabilities.

Keywords: Scientific workflow software; granularity; optimization

1. INTRODUCTION

Modelling environmental systems in a holistic manner leads to the integration of usually disparate software systems and manual tasks. Unsurprisingly, the resulting pipelines of modelling tasks and analyses reach a size and level of complexity that can be daunting compared to that of smaller components of the modelling system, and needing enhanced capabilities for the capture of provenance and metadata. Recent examples of such projects in Australia are the Sustainable Yields projects (http://www.csiro.au/partnerships/SYP.html). These projects usually highlight the need for tools to better handle long-lived workflows. These workflows capture modelling processes whose execution can span anything from hours to months, possibly needing execution on different computing environments over their execution lifespan (Shukla and Schmidt [2006]).

In the authors' experience, scientists and modellers will tend to approach the task of handling long-lived workflow either by using ad-hoc batch files, shell scripts or other type of glue code as a means for tool integration and automation (McPhillips et al. [2009]), or by trying to somehow wrap heterogeneous software into their modelling software of predilection. Modelling software systems are primarily aimed at helping to represent a real-world system. This model-centric view of the modelling process may not be, in the authors' opinion, the most conducive to capturing information on data and model provenance as first class entities in the software system. If considered, this tends to be in the form of tools for the management of scenario and configuration information that are model-specific, and not generic enough to be applied to arbitrary workflows.

Conversely, for some scientists and modellers newly introduced to scientific workflow software (SWS), everything can end up looking like a workflow. This seems to be particularly the case if they perceive shortcomings in their main modelling tools. Arguably, many actions referred to as ”models” in the terminology of modelling software platforms are indeed pipelines of analysis potentially amenable to be usefully captured as a workflow. However, the value of representing fluxes and states of the physical entities modelled into a scientific workflow rather than more “traditional” modelling software is very debatable. The boundary between scientific workflow tool and modelling platform is somewhat ill-defined, quite possibly gradual and varying depending on the specific SWS used. There are consequently questions on which granularity is appropriate for activities in these SWS. In a modelling and analysis pipeline that needs orchestration at several levels of granularity ranging typically from high-level process batching down to the stepping through space and time of a simulation model, how appropriate are SWS to handle each of these levels of granularity?

1.1. Software

A sample of six fairly different workflow systems and their application to the scientific domain are reviewed in Curcin and Ghanem [2008]. For the present paper the authors consider one SWS listed in Curcin and Ghanem [2008], Kepler (https://kepler-project.org), and another SWS not listed in that publication, Trident (http://connect.microsoft.com/trident).

Kepler (Altintas et al. [2004]) is a Java tool based on the Ptolemy II system, which supports multiple models of computation based on the director/actor paradigm. This gives Kepler the potential to cover a large spectrum of granularities, ranging from the orchestration of high-level tasks to iterating over a temporal environmental simulation model. We consider in this paper the version 1.0 of Kepler. We may use the term ‘activity’ in this paper in lieu of the term ‘actor’ used in the Kepler terminology.

Trident (Microsoft [2009]) is a SWS tool released by Microsoft Research. It is based on Windows Workflow Foundation, currently using version 3 thereof (WF3), part of the .NET framework version 3.5. We consider in this paper version 1.0 of Trident.

We mainly use in this case study the SWS Trident and the TIME environmental modelling framework (Rahman et al. [2003]) to assess the practical feasibility of a calibration task as a workflow. TIME is a software development framework for creating, testing and delivering environmental simulation models. It includes support for the representation, management and visualisation of a variety of data types, as well as support for testing, integrating and calibrating simulation models.

2. CASE STUDY

We explore in this paper several approaches to perform the calibration of a simulation model, i.e. an optimization task. Some relevant contextual and more technically detailed information on this case study can be found in Perraud et al. [2009], while general considerations as to the use of SWS in the hydrology domain are in Guru et al. [2009]. The main activities of the conceptual workflow are depicted in the UML activity diagram in Figure 1. This modelling exercise is of particular interest to assess the granularity of a workflow as it has the following characteristics:

- The core of the system is a temporal simulation model, typically developed upon a modelling framework

- It is possible to decompose the overall workflow in three high-level steps, each in turn comprising several nested steps.
- The step of optimization is in essence a step "looping" over some sub-steps. Here the term 'loop' may actually cover serial or parallel evaluations, or a mix thereof.

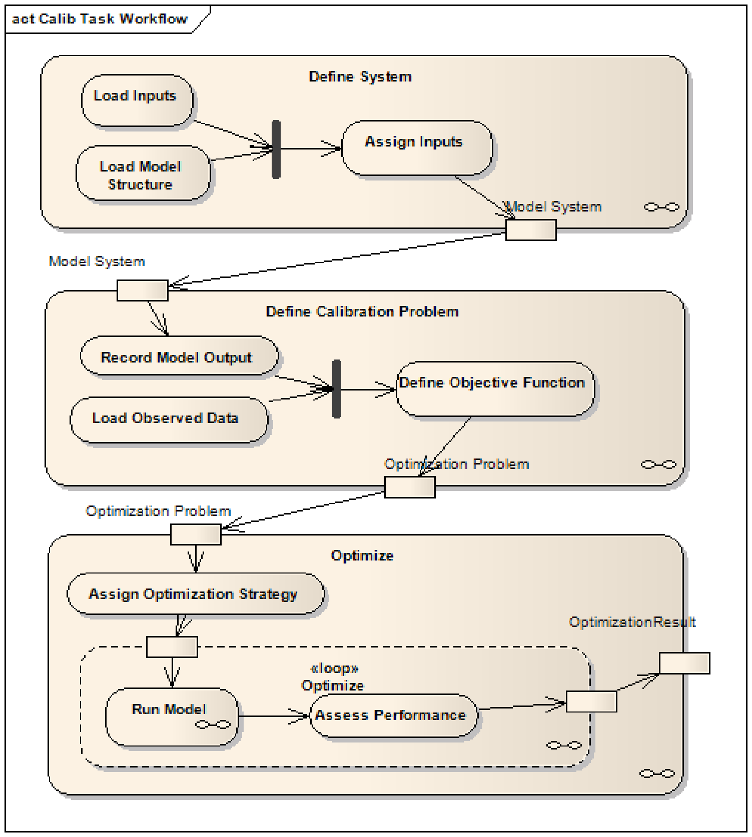

Figure 1. Calibration task conceptual workflow

There are several nested levels of granularities at which the sequencing of the steps in Figure 1 can be expressed usefully via a workflow. Starting from the high level granularity:

1. The entire task can be expressed as a single activity. While this may look at the first consideration as a "degenerate" case, this has value when used as part of an iteration over several differing models, for instance when having to calibrate several water catchments, or comparing the performance of alternate model structures on one or more catchments.
2. The pipeline of defining the model (the system), defining the optimization problem, and performing the optimization. The high level conceptual steps follow those in Talbi [2009], where more information can be found on optimization problems and related software frameworks.
3. Decomposing in the SWS the activity named 'Optimize' in Figure 1. There are a couple of possibilities at this level. The first is to offer in the workflow the possibility of composing a sequential pipeline of several optimization algorithms, as is quite common for instance to start by using a global search algorithm followed by a local search that uses more or less the gradient of the response curve. Pictorially in Figure 1 this corresponds to working on the step of "Optimization Strategy". The second possibility is to express the optimization algorithm itself in the SWS, and corresponds to the aggregate of the "Strategy" and the nested "Optimize" loop. This is arguably the most interesting part to work on as this puts to the test SWS in terms of expressiveness (can the algorithm(s) be expressed by the system), control flow (serial loops and/or opportunities for

parallelization e.g. for population based strategies) and data structure (can the inner information in the optimization process be handled in the SWS).

4. The finest granularity we may consider for explicit handling by the SWS is the "Run Model" activity in Figure 1. This takes over the time stepping, handling of input and output and in essence removes the need for a third party modelling framework.

3. CRITERIA

Each approach is primarily assessed against the criteria listed in Table 1. It should be noted that these criteria are chosen for the specific calibration case study, and are not a comprehensive list. McPhillips et al. [2009] for instance give a different list for general workflows.

Table 1 Main criteria considered for assessing the workflows

Criteria	Rationale
Practical feasibility	Are there technical or conceptual "show stoppers"? Note that an assessment on this criterion is somewhat dependant on the assessment of the others.
Cost of wrapping	The code required to include the library functionalities should be minimal
Versatility	The workflow can be configured such that most likely options in the case study are covered (for instance, applicable to several model systems)
Extensibility	The core workflow can be extended with other activities.
Clarity	The purpose and expected behaviour of the workflow should be conveyed by the visual appearance
Performance	The runtime is within a maximum of ~200% of the runtime obtained when using directly the tools and modelling components in TIME.

4. IMPLEMENTATIONS

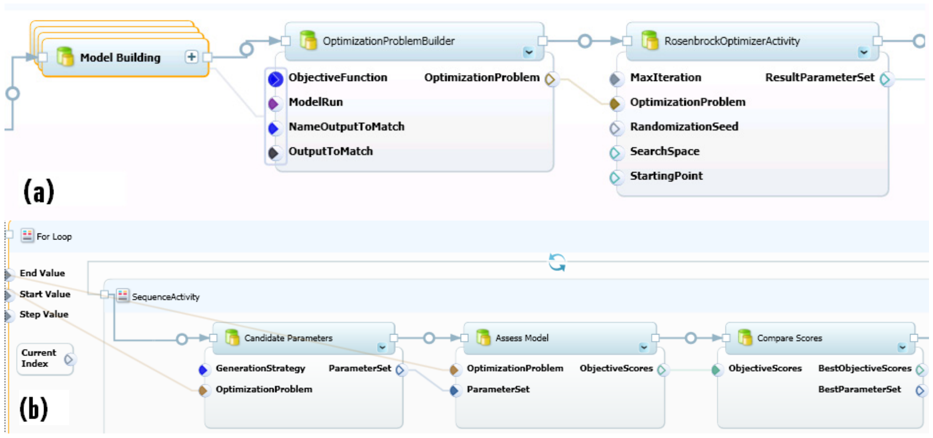

Figure 2. Implementations in Trident of two of the approaches, namely (a) for approach (2) and (b) for approach (3), in the second case only the simplest decomposition possible for the iterative loop

Figure 2 is a visual screen capture of two of the four solutions when implemented in Trident. The four approaches considered follow. Note that these are not necessarily mutually exclusive approaches. These approaches will be referred to in the rest of this paper as numbered thereafter:

1. Having a single activity for the whole calibration task
2. The high level calibration task is decomposed in three steps (model definition, optimization definition, and optimization)
3. The iterative process of the calibration algorithm is explicitly handled in the workflow
4. The time stepping of the model is handled in the workflow.

5. RESULTS

A summary of the qualitative assessment of each approach is listed in Table 2. Overall the approach (2) appears the best level of decomposition, allowing some degree of visual composition and extension for the user. Approach (3) is found to be feasible but difficult in both Kepler and Trident, more due to the respective fundamental design choices of their workflow engines than simply idiosyncrasies. It remains an appealing approach as it has the prospect to allow users to compose their pipeline of optimization with the greatest level of flexibility, something than in our personal experience many users not keen to program are asking for.

The remainder of this section discusses the salient points of the assessment of each approach that lead to the summary assessment in Table 2, and elaborates on some of the related technical characteristics of the SWS.

Table 2. Qualitative assessment of implementation approaches (+++ ideal, --- not acceptable, 0 neutral, ? not assessed)

	1 Single activity	**2 Model and optimization problem builders**	**3 Optimization algorithm in the SWS**	**4 Model iterations in the SWS**
Practical Feasibility	Yes	Yes	Difficult	No. (marginal in Kepler)
Cost of wrapping	+++	++	--	---
Versatility	0	+	+++	?
Extensibility	0	+	++	?
Clarity	-	++	+	?
Performance	+++	++	-	---

The implementation at the finer granularity (4) is assessed as not practically feasible, using either Trident or Kepler. Simple tests show that both platforms exhibit a prohibitive overhead in data exchange between activities. In Trident, the model of execution for activities implies that a new activity instance must be created at each time step. In effect each conceptual activity is stateless by default. It is feasible to maintain and restore the state of an activity (or part thereof) between iterative steps but the overhead is important. As the execution of the model is usually at least the significant majority of the runtime, the solution drastically fails on the performance criteria. This result is hardly a surprise given the performance optimization required even without the overhead of a SWS, as described in Perraud et al. [2009] which showed the significant performance tuning steps and architectural changes required for enabling multi-threading to speed up calibration.

The highest granularity solution (1) is feasible on both platforms with rather little wrapping 'glue' code, although a fair bit more for Kepler than for Trident, due to the difference of virtual machine with TIME, but this is a small concern for this paper. The information of the input and outputs of the activity, being high level, can be captured with simple types that both workflow software tools can handle 'natively'.

The approach (2) brings some interesting design questions and considerations to light when seeking implementation through both Kepler and Trident. This decomposition leads to the necessity of passing between workflow activities the definition of the model (the system being optimised) and of the optimization task itself (conceptually a superset of the former). The type system of Kepler would require translating this information to simpler types, passing it around as 'anonymous 'objects, or declaring additional specific types to Kepler. While all appear possible, the first approach is model specific, the second not ideal in terms of clarity, and the third is relatively costly in terms of code. Trident uses the standard .NET type system for the activity inputs/outputs and in this respect is much more accommodating to defining custom types to pass between activities.

The approach (3) is the first one that brings an explicit looping structure in the solution workflow (Figure 2-b). This approach definitely brings to light some behaviours, capabilities and limitations of both Kepler and Trident. Thus, this is the approach we discuss most in the present section.

In Kepler, the models of computation can be regarded as a framework for component-based design, where the framework defines the interaction mechanism between the components. The scheduling of execution is a global decision for a given workflow execution (Brooks et al. [2005]). The simplest and arguably most used director is the synchronous data flow director (SDF). Emulating a "for" loop in Kepler can also be achieved using the "PN" (process networks) director. It is of particular interest as this director is potentially useful for parallel and distributed processing. It is also appropriate for iterations where there are feedback loops between actors, which the SDF director could not handle. One problem we have identified is cases with nested loops using PN directors. When a step in the process fires a 'Stop' signal, it is not limited to the current PN domain (current iteration) and is propagated to outer loops, which means it will stop the outer iteration. While simple loops can be implemented with relative ease, the feasibility of nested control flows is uncertain at the time of writing, at least if approaching the task from a point of view of a traditional, imperative programming. Another aspect already noticed in approach (2) implemented in Kepler is the data system. Expressing the optimization process at a finer granularity requires further constructs (parameter sets, population of points, objective function calculators, etc.) to be handled. They need to be re-cast into Kepler's native data types, passed as anonymous objects, or added as first class data constructs. The undeniable benefits in terms of data polymorphism of the Kepler data types have a clear downside when wanting to handle domain-specific constructs.

If implemented in Trident, the simplest case for decomposing the optimization step (approach 3) can use a 'for' loop, as shown in Figure 2-b. The workflow composition and its runtime behaviour are somewhat intuitive, although a large amount of execution logic is still embedded within each activity executed. One particular characteristic of Windows Workflow Foundation version 3 (WF3) quickly becomes apparent when implementing the activity that compares objective scores between iterations. WF3 activities, or activity instances to put it more correctly, follow an activity automaton, typically the three states Initialized, Executing and Closed (Shukla and Schmidt [2006]). An instance cannot get back into an executing state when closed. At every step of the iteration, a new activity instance is created. Retaining the memory of the "best known parameter set so far" in the score comparer in Figure 2-b is not straightforward, as opposed to when iterating using in-memory stateful objects as is the paradigm with TIME components and most object based systems. The good side of this in WF3 is that the interactions between activities are primarily and natively dictated by the definition of bindings between the inputs and outputs of these activities, rather than relying on the state of activities from their last execution. This arguably fosters the clarity of what a workflow is doing, but this is a significant departure from traditional programming or scripting, aside perhaps from stateless transactions such as those found in Web-based systems.

6. DISCUSSION

One outcome of the exercise the authors found interesting is the incentives for certain design felt when using workflow software. Chief among these is the lead towards using something akin to the Strategy design pattern (Gamma et al. [2004]), and decoupling the steps in the workflow. The main constructs in the process are arguably the model definition, the optimization problem definition including the objective function, and the parameter sets trialed during the optimization. Given the execution model of the workflow software, even the simplest decomposition of the case study leads to using these constructs to transfer information between workflow activities. While the use of these constructs in an imperative, object oriented programming context is also beneficial, implementation leakage and implicit coupling between the different participating objects can occur more easily. One downside of the data handling and execution model in Trident is the temptation to use the Singleton pattern to maintain state in an iterative loop in a manner similar to imperative programming. This would have negative consequences for parallelizing the iteration of the optimization algorithm.

One enticing prospect of SWS for scientific workflows requiring iteration is the greater emphasis on declarative programming over imperative programming, the ‘what’ over the ‘how’. The approach aims to be more conducive to write scalable workflows, easier to parallelize (Chappell [2009]). It is worth noting that it comes at a time where there is also a renewed interest in functional programming, of a similar philosophy, with recent functional language additions both on the .NET and Java virtual machines. Declarative programming does require a change in mindset, perhaps more so for users versed in programming.

Facilities to parallelize tasks are highly desirable for many heuristic optimization problems, most of them being readily data-parallel with little exchange of messages between sub-tasks. For both products used for this paper, we find limits to the usability of such facilities. If and when underlying capabilities are present, they require a level of investigation that in effect limits the availability to users with a significant level of programming skills. Currently it appears more feasible to implement the parallelization of computing tasks within an activity rather than composing it at the workflow level. This is admittedly an assessment that may be biased by prior work described in Perraud et al [2009]. The design and implementation of simple, entry-level parallel workflow iterators for the simpler parallel problems is a task the authors are undertaking as a follow-up to this case study. We anticipate that the granularity of parallelization that can be expressed usefully through workflows is larger than that described in Perraud et al. [2009], which is located within the model. Rather, an evolutionary optimization algorithm would benefit from workflow iterators parallelizing the evaluation of model configurations i.e. different parameterizations.

7. CONCLUSION

Optimization tasks are common in many domains, stemming from the generic nature of most algorithms. The case study in this paper is chosen as it spans several granularities ranging from the temporal iteration of the model to the high level workflow defining the model, optimization and executing that optimization. We find that decomposing the optimization problem as a workflow improves the expressiveness when compared to what users have to do with scripting engines or programming directly to the application programming interface of optimization toolboxes and modelling frameworks. Four approaches distinguished primarily by the level of granularity of the decomposition are assessed. The finest level of granularity is not practically feasible due mostly to performance degradation. The intermediate approaches that decompose respectively the overall task and the optimization process itself have positive incentives. These stem from the workflow software, and the ability to design sensible constructs defining the temporal model and the optimization task to decouple steps in the workflow. We find that the nested control flows necessary to express the optimization algorithms in use in the hydrology domain are difficult to implement by graphical means in both SWS, due to fundamental characteristics of the software, albeit different ones in each case. While it has some clear positive aspects elsewhere, the lack of native maintenance of state values between iterations in Trident is the main logistical hurdle. In Kepler the main issue is the apparent complications of using the PN director to emulate iterative control flows, notably nested control flows. Notwithstanding these limitations both workflow software have the potential by design to support iterative control flow structures, notably to better support the parallelization of portions of workflows. We suggest that both platforms could benefit from better or at least more readily usable iterative control flow facilities to express, among other things, optimization algorithms as workflows.

ACKNOWLEDGMENTS

This work is part of Hydrologists' Workbench project in the water information research and development alliance between CSIRO's Water for a Healthy Country Flagship and the Bureau of Meteorology.

We thank Paul Box, Joel M. Rahman, and two anonymous reviewers for their valued contributions.

We gratefully acknowledge the help from the Trident team at Microsoft Research and from contributors to the Kepler users' mailing list.

REFERENCES

Altintas, I., C. Berkley, E. Jaeger, M. Jones, B. Ludäscher and S. Mock, Kepler: An Extensible System for Design and Execution of Scientific Workflows, 2004, Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04), 2004

Brooks, C., E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, H. Zheng (eds.), Heterogeneous Concurrent Modeling and Design in Java (Volume 3: Ptolemy II Domains), http://ptolemy.eecs.berkeley.edu/papers/05/ptIIdesign3-domains/ptIIdesign3-domains.pdf (Last accessed 2010-05-01), 2005.

Curcin V. and M. Ghanem, Scientific workflow systems – can one size fit all?, Proceedings of the 2008 IEEE, CIBEC'08, 2008

Gamma, E., R. Helm, R. Johnson, and J. Vlissides, Design Patterns: elements of reusable object oriented software, Addison Wesley, 1994.

Chappell, D., The Workflow Way: Understanding Windows Workflow Foundation, April 2009, http://msdn.microsoft.com/en-us/library/dd851337.aspx (Last accessed 2010-05-01), 2009

Guru, S. M, M. Kearney, P. Fitch, C. Peters. Challenges in using scientific workflow tools in the hydrology domain. In Anderssen, R.S., R.D. Braddock and L.T.H. Newham (eds) 18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation. Modelling and Simulation Society of Australia and New Zealand and International Association for Mathematics and Computers in Simulation, pp. 1059-1065. ISBN: 978-0-9758400-7-8. July 2009

Microsoft Corporation, Project Trident: An Introduction, Microsoft Project Trident: A Scientific Workflow Workbench Version 1.0a – https://connect.microsoft.com/trident (Last accessed 2010-05-01). July 9, 2009.

McPhillips, T., Bowers, S., Zinn, D., and Ludäscher, B., Scientific workflow design for mere mortals, Future Generation Computer Systems, Volume 25, Issue 5, p.541-551, 2009

Perraud J.-M., J. Vleeshouwer, M. Stenson, R.J. Bridgart, Multi-threading and performance tuning a hydrologic model: a case study. In Anderssen, R.S., R.D. Braddock and L.T.H. Newham (eds) 18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation. Modelling and Simulation Society of Australia and New Zealand and International Association for Mathematics and Computers in Simulation, pp. 1059-1065. ISBN: 978-0-9758400-7-8. http://www.mssanz.org.au/modsim09/C5/perraud.pdf, July 2009.

Rahman, J.M., S.P. Seaton, J-M. Perraud, H. Hotham, D.I. Verrelli and J.R. Coleman, It's TIME for a new environmental modelling framework, presented at MODSIM Congress, Townsville, Australia, July 14-17, 2003.

Shukla, D. and B. Schmidt , Essential Windows Workflow Foundation, Addison-Wesley Professional, ISBN-10: 0-321-39983-8, ISBN-13: 978-0-321-39983-0, pp. 480, 2006.

Talbi E.-G., Metaheuristics: from design to implementation, ISBN: 978-0-470-27858-1, Wiley, 624 pp., 2009.