Future Development Environments for Computational Scientists

Andreas Heil, Matthew J. Smith and Alexander Brändle
Computational Science Laboratory, Microsoft Research
7 JJ Thomson Avenue, Cambridge CB3 0FB, United Kingdom
aheil@acm.org, {mattsmi, alexbr}@microsoft.com

ABSTRACT
Computational Scientists are both creators and end-users of scientific models. Different aspects to their work target different audiences and generally require different development approaches. Here we report outcomes of an experimental collaboration between Software Engineers and Computational Scientists to create a new development environment to encompass diverse end user groups.

Categories and Subject Descriptors
D.1.7 [**Software**]: Programming Techniques – *Visual Programming.* D.3.2 [**Software**]: Language Classification – *Data-flow languages.* I.6.7 J.2 [**Computer Applications**]: Physical Sciences and Engineering – *Earth and atmospheric sciences.*

General Terms
Design, Experimentation, Human Factors, Languages.

Keywords
Computational Science, Scientific Workflow, Scientific Dataflow, Scientific Application Composition.

1. INTRODUCTION
Computing has dramatically changed the way scientific research is done and communicated. Models of systems and processes are routinely developed and simulated in all areas of science, and computational methods are typically utilised in data collection and analysis.

Scientists in the future will increasingly become literate in methods used by Software Engineers, enabling them to more effectively exploit the capabilities of new computational tools and resources. This paper describes lessons learned through collaboration between Software Engineers and Computational Scientists to jointly research new model development environments targeted at Computational Scientists, and the target audiences of Computational Scientists, as end users.

The research of Computational Scientists typically involves the development of new computational models and methods. Complex multi-component models, usually bringing together previously developed and new sub-components into a larger dynamical model, are now frequently developed and exposed as services. The communication of scientific models and their predictions is now a major area of science, with policy makers increasingly turning to scientists for predictions of complex dynamical systems. Scientific model development is increasingly done collaboratively, with different scientific teams working on sub-components of the larger model, becoming analogous to professional software development. It is therefore not surprising that the majority of scientists consider developing scientific software as important for their own research [1].

Despite the need for software to help conduct and communicate scientific research, Computational Scientists are generally "end-user programmers" [2] i.e. they are not typically involved in the professional software development for those purposes. While there is a great deal of variation in the level of programming expertise amongst Computational Scientists, they typically have entirely different goals to Software Engineers when creating and composing software (by "Software Engineers", we mean the group of professionals developing software as their primary task). Traditional software engineering approaches are therefore seldom applied when models and methods are developed by Computational Scientists, even though they could plausibly advance the state of the art. For example, Computational Scientists rarely employ the formal debugging methods so frequently used in traditional software engineering. A lack of such methods makes it extremely difficult to identify and distinguish between software bugs, model errors and approximation errors in large complex models, especially when the "correct" (i.e. bug free) predictions of a model may not even be known [1].

1.1 Composing Scientific Applications
Scientific applications, in contrast to the structure of traditional software applications, usually consist of various and changing connected components, often chained together in workflows to repeatedly perform particular processes (Figure 1). In the early stages of exploring a research question the main focus of a Computational Scientist is often on the development of a model, and in constructing the appropriate workflow, to generate new scientific insight. Computational efficiency is usually a much lower priority, even though processing the data or running a model can be time consuming and will potentially become more important in later stages of the process.

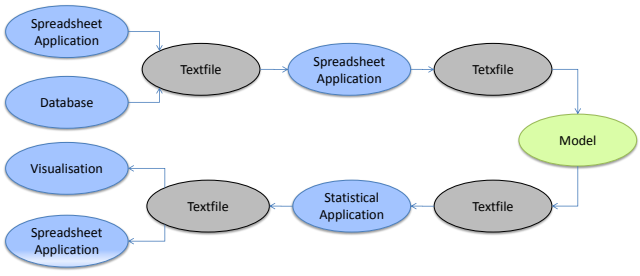

Figure 1. A simple scientific workflow for processing data

Computational Scientists often need to "chain together" different sub-models and analysis, sometimes doing so as part of multi-institutional collaborative projects and involving off-site datasets. This leads to systems that are far more complex than that depicted in Fig. 1. Significant model building programs therefore frequently include the involvement of Software Engineers who are skilled in developing complex multi-component applications. Such composition of scientific applications has been addressed recently in a wide range of research projects, however these are mainly driven by Software Engineers with the focus on the technical aspects of bridging well-established workflow technologies with scientific applications, and integrating heterogeneous systems and components. A simple way of composing these sub-models and analysis in forms of services could improve development process significantly.

1.2 Communicating Models and Data

Scientists have two broad audiences; other scientists wanting to investigate, adapt and use various aspects of the model, and non-scientists typically interested in the predictions of the model and, to a lesser extent, how the model works. Therefore, Computational Scientists are likely to want to represent their models and findings with different degrees of abstraction depending on their collaborators, audience and context (Figure 2). For example, scientists working collaboratively may discuss their model at the level of the actual code. Other collaborations, such as at an interdisciplinary level, may benefit from a higher level of abstraction. In contrast, communicating research methods and findings to policy makers may utilise a very abstract depiction of the model, with the emphasis placed on the model predictions rather than the methods.

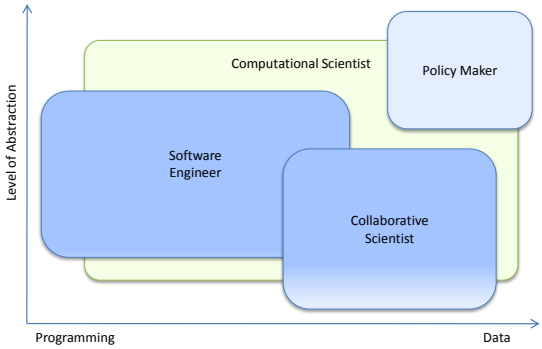

Figure 2. Development Intersections for Computational Scientists

2. EUD FOR COMPUTATIONAL SCIENCE

The Microsoft Computational Science Studio (MSCSS) provides a research prototype environment allowing different levels of abstraction in the development and composition of scientific applications. It is based on a dataflow paradigm, rather than a control flow paradigm, for composing applications. Spreadsheet applications like Microsoft Excel are well known examples adopting the dataflow paradigm. MSCSS includes (1) a shell scripting tool allowing the formal scripting of dataflows, (2) a visual designer and visual programming language for composing scientific applications/experiments using dataflows and (3) a graphical designer to create visual applications based on the underlying experiments. In Section 3 we will examine the lessons learned while building and testing these tools.

2.1 Dataflow Scripting

To combine model components in the form of services, we provided a dataflow shell (DFShell) that allows applications/experiments to be composed based on a scripting language. DFShell is a lightweight scripting environment for executing dataflows. The default behavior of these is to cause output variables to be automatically recalculated when the values of input variables change. To achieve this, DFShell allows the definition of (1) service endpoints including port descriptions, (2) data buses and (3) mappings between service port descriptions previously defined data busses.

2.2 Dataflow Editor

At a higher level of abstraction we enable scientists to identify and connect data sources, computational services, scripting components, user interfaces and other visualisation elements to computational models of varying complexity through the use of a Dataflow Editor (Figure 3).

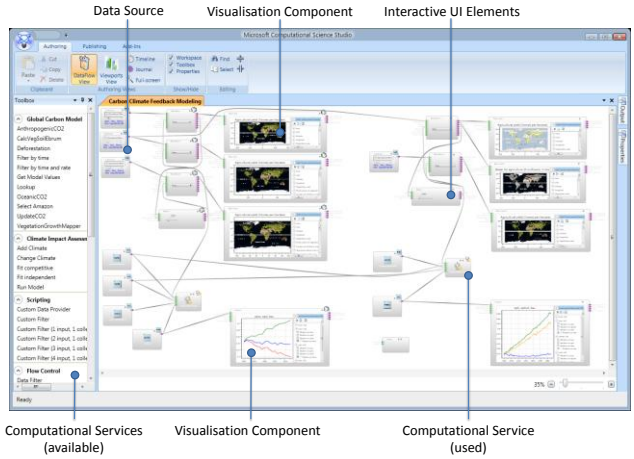

Figure 3. MSCSS Dataflow Editor

This environment allows the development of data driven applications, but additionally incorporates various additional components to control the dataflow (such as requiring the user to indicate when a particular data bus is to be switched on). We therefore also included common user interface elements such as buttons and sliders to interact with the services.

2.3 Visual UI Editor

The highest abstraction level is provided through a Visual User Interface (UI) Editor (Figure 4 which is based on Figure 3).

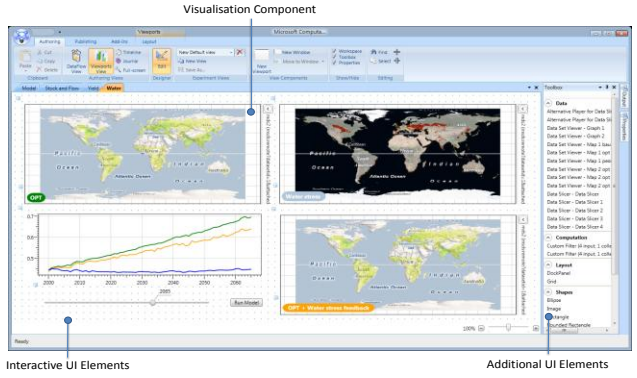

Figure 4. MSCSS User-Interface Editor

The Visual UI Editor allows a rich user interface to be constructed based on the "experiment" in the Dataflow Editor. Data visualisations and UI controls can be arranged to provide a user interface based on the experiment that meets the specific communication requirements. Multiple views can be created on the same experiment for different communication purposes. A key feature of this method is that the UI can be constructed while remaining dynamically connected to the model, such that changes to the model or data can lead to automatic updates in the UI.

3. LESSONS LEARNED

The Microsoft Computational Science Studio (MSCSS) was developed as a prototype through collaboration between Computational Scientists and Software Engineers. Including Computational Scientists in this process led to valuable insights into end-user requirements for future software environments.

3.1 Dataflow Scripting

Benefits: The dataflow scripting language makes it much easier for Computational Scientists to communicate their intentions to traditional Software Engineers. Software Engineers are normally familiar with the process of scripting data flows. Based on the script, they are able to perform systematic analyses of the dataflows without knowing the scientific basis for the model, and the specific details of the model implementation. It also makes it easier to debug applications originally developed by Computational Scientists.

Drawbacks: The Computational Scientist has to become familiar with the paradigm of building scientific applications out of dataflows. Although spreadsheet applications are widely used the construction of complex applications via dataflows is not currently widespread practice in Computational Science and the benefits of the approach to Computational Scientists have not been fully explored. In addition, the structure of scientific applications, especially in complex multicomponent models, are not always based on the automatic triggering of components in response to changes in input variables. Instead, Computational Scientists usually require a more diverse range of triggering events than the one implemented in the current version of the scripting language.

3.2 Dataflow Editor

Benefits: The visual appearance of the dataflow encourages users to explore the model and experiment with its structure, and makes it much easier to communicate the structure of the model to other users. The immediate visual feedback resulting from changes in the dataflow allows the structure and assumptions of the model to be more intuitively understood, even enabling a degree of debugging and model verification by a less experience computer user. Providing a common user interface also allows the scientist to easily add or change control dependencies. Moreover, in this case the Computational Scientist does not necessarily need to fully understand all of the input and output formats to compose functioning workflows. The ability to drop in data in various data formats (CSV, NetCDF) also allows easy exploration of the data.

Drawbacks: The visual language does, as all visual programming languages, have a tendency to visually overburden the user. While schematic representations are well received on a coarse grained level, they become confusing at a certain level of detail. Mechanisms are required that allow models to become more visually structured. Therefore it is currently not completely intuitive for Computational Scientists to develop new models within this environment. Currently, we provide a migration- and mitigation- strategy by explicitly allowing the user to mix visual programming elements with traditional computational elements (e.g. scripting or precompiled modules) but building new computational services currently still requires the support for a Software Engineer.

3.3 Visual UI Editor

Benefits: Computational Scientists can communicate models and their predictions using a clear UI allowing for a variety of static and dynamic components. Dynamic user interfaces can be constructed relatively easily and can be updated automatically when input data, model components or the model structure changes. This new way of packaging models together with user interfaces addresses an expanding future "market" for assisting collaboration and communication in scientific research.

Drawbacks: The UI editor is currently limited in the number of components available to communicate the data and model structure. The creation of a new visualisation component currently still requires the assistance of a Software Engineer.

3.4 Conclusions

There are both significant advantages and disadvantages to the approach taken in MSCSS in terms of providing a Development Environment for Computational Scientists. Further research is needed to identify shortcomings to this type of Development Environment. Reusing and adapting existing code and procedures is common software engineering but it remains to be seen whether Computational Scientists would equally significantly benefit from encoding their methods into dataflows. In contrast, the MSCSS already shows potential to benefit those Computational Scientists working with more complex models. We believe scientists working on complex models as part of multi-institutional efforts (such as climate change models) would benefit most from this technology in its present form. In such cases the overall model structure can be represented both graphically and textually, but can also be modified and run, allowing sub components to be created, modified and "plugged in" relatively easily. Our Visual UI editor potentially also removes a lot of the overhead in communicating model structure and predictions to diverse audiences. We believe that targeting Computational Scientists for End User Development is a likely to be a growth area, with the inevitable increase in the use and reliance on predictions of complex multi component models into the future.

4. ACKNOWLEDGMENTS

This work was made possible by Drew Purves, Martin Calsyn, Vassily Lyutsarev and Benjamin Schröter.

5. REFERENCES

[1] Hannay, J., Langtangen, H.P., MacLeod, C., Pfahl, D., Singer, J, and Wilson, G. 2009. How Do Scientists Develop and Use Scientific Software? Software Engineering for Computational Science and Engineering. DOI= http://dx.doi.org/10.1109/SECSE.2009.5069155.

[2] Myers, B.A., Burnett, M.M., Wiedenbeck S., Ko A.J., Rosson M.B. 2009. End User Software Engineering: CHI'2009 Special Interest Group Meeting. Boston, MA. USA.