National Research Council Canada
Institute for Information Technology

Conseil national de recherches Canada
Institut de technologie de l'information

NRC · CNRC

*Supporting the Everyday Work of Scientists: Automating Scientific Workflows **

Vigder, M., Vinson, N.G., Singer, J., Stewart, D., Mews, K.
August 2008

* published in IEEE Software. August 2008. NRC 50330.

Supporting the Everyday Work of Scientists: Automating Scientific Workflows

Mark Vigder, Norman G. Vinson, Janice Singer, Darlene Stewart, Keith Mews
National Research Council Canada

1. Keywords

J.2 Physical Sciences and Engineering < J Computer Applications, H.4.1.g Workflow management < H.4.1 Office Automation < H.4 Information Technology and Systems Applications < H Information Technology and Systems, H.5.2.q User-centered design < H.5.2 User Interfaces < H.5 Information Interfaces and Representation (HCI) < H Information Technology and Systems

2. Abstract

This paper describes an action research project that we undertook with National Research Council Canada (NRC) scientists. Based on discussions about their difficulties in using software to collect data and manage processes, we identified three requirements for increasing research productivity: ease of use for end-users; managing scientific workflows; and facilitating software interoperability. Based on these requirements, we developed a software framework, Sweet, to assist in the automation of scientific workflows.

Throughout the iterative development process, and through a series of structured interviews, we evaluated how the framework was used in practice, and identified increases in productivity and effectiveness and their causes. While the framework provides resources for writing application wrappers, it was easier to code the applications' functionality directly into the framework using OSS components. Ease of use for the end-user and flexible and fully parameterized workflow representations were key elements of the framework's success.

3. Introduction

Science and software have become integrally linked. Scientists use software for generating and analyzing data, sharing and modeling research results, managing data sets, and creating reports, among many other purposes. Unfortunately, the software available for supporting these tasks, while indispensable, also presents serious difficulties. This occurs for three primary reasons. First, much of the software is either written or customized by the scientists and technologists themselves to control specialized hardware or processes unique to their organization, or scientific domain. Difficulties arise because the scientists and technologists are not typically trained in software engineering methodologies, and therefore do not possess the necessary knowledge to manage, maintain and evolve the software and information artefacts associated with it. Second, even when a professional software support staff is involved, the number of customizations quickly becomes unmanageable. Finally, other difficulties arise from insufficient interoperability, with ad-hoc solutions often being inefficient.

To address these problems, we have undertaken an action research project in collaboration with scientists at the National Research Council (NRC) Canada. In an action research paradigm[1], the researchers work collaboratively with the problem owners to investigate the causes of the problem, and implement a solution. Thus, the work described in this paper focuses primarily on the Institute for Ocean Technology (IOT). The research project began when IOT scientists contacted us to help them improve their utilization and management of the software that allows them to study scale models of marine structures (ship hulls, offshore oil rigs, etc.) under various different conditions.

To ensure our framework would have broad application within IOT, we concentrated on commonly used, fairly well-defined sequences of data manipulation procedures. These sequences involved activities such as numeric transformations, format changes, analysis, and file management. We refer to these sequences as scientific workflows. Working with the scientists, we developed a scientific workflow management software system. We evaluated the effectiveness of our system in the course of iterative development and through semi-structured interviews. Respecting a fundamental tenet of action research, we insisted on using real-world problems in order to support external validity[2].

Following the IOT's lead, our research focused on improving workflow management related to the collection, analysis, and management of data produced by sensors and other instruments, and the subsequent creation of reports. To ensure generalization from IOT's ocean engineering research to other scientific fields, we are currently working with researchers from other disciplines to iteratively develop, extend, and apply our software framework. Our focus differentiates our work from much of the software engineering and computer science research in scientific software, which has tended to focus on high performance computing and modeling. Instead our goal is to improve workflow management related to the collection of data through to its reporting. As data collection and analysis forms the core activities performed in many, if not all, scientific domains, our work is relevant to most scientific fields.

The rest of this paper proceeds as follows. First, we briefly discuss related research. In Section 2, we describe scientific workflows, and relate this to the Sweet Framework we developed. Section 3 provides an overview of the Sweet Framework. Section 5 describes the results of our evaluation. We conclude with ideas for future work, and some lessons learned.

4. Related Work

Our work relates to two areas of active research: end-user software engineering and workflow management systems.

End-user software engineering refers to research dedicated to improving the capability of end-users who need to perform programming or software engineering tasks. For many, if not all, of these end-users, the creation and maintenance of software is a secondary activity performed only in service of their real work. This scenario applies to many fields including science[3]. However, there is little research specifically focused on scientists as end-user software engineers. An exception would be the work of Letondal[4]. She has performed a number of studies with bioinformaticians. Letondal finds that end-user scientists have difficulties not in the programming aspects of their software creation, but rather in the management of the software and artefacts that are created – our focus in this article. Other work looking at end-user software engineering and science tends to focus on the development of specialized modeling environments[5] or high performance computing[3], rather than scientific workflows.

One of the primary requirements we encountered in our research was the need for non-programmers to manipulate scientific workflows. Business workflow management systems emerged in the 1990's and are well accepted in the business community. Scientific workflows differ from business workflows in that rather than coordinating activities between individuals and systems, scientific workflows coordinate data processing activities.

A number of scientific workflow infrastructures have been developed[5]. These approaches have often focused on High Performance Computing (HPC), where the workflow controls the data processing over a distributed grid system. This is in contrast to our focus, which is to improve research efficiency by automating and managing the workflows involving data collection and manipulation. Although these workflows are not computationally complex, they are mundane tasks of science that are often time consuming.

5. Easy To Use, Customizable, Scientific Workflows

5.1 The IOT Case

The IT issues experienced by IOT scientists can be illustrated by a typical scenario. At IOT experiments are conducted on various watercraft and watercraft components, or models thereof. Researchers monitor and record the performance of the object under study under different conditions in various large water tanks. For instance, an experiment may focus on a ship's hull undergoing various manoeuvres. A model of the hull is instrumented and towed the length of a tow tank multiple times under varying conditions. The conditions in the tank are controlled by specialized hardware, which in turn is controlled by software that includes a variety of parameter settings. One instance of towing a model the length of the tank is called a *run*. Data from one or more previous runs is often analysed to select the conditions for the current run. This form of data analysis is referred to as *on-line analysis* in contrast to in-depth *off-line analysis* performed after the experimented has been completed. This run-analysis-run cycle continues until a sufficient set of runs have been completed.

The above scenario involves two end-user roles: the *tank operator*, who is responsible for controlling the tank facility, gathering the data, and monitoring the sensors; and the *scientist*, who is responsible for analyzing the data after each run to determine the parameters for the subsequent run.

Software engineering and information technology support (SE/IT support) constitutes a third role in the IOT labs. As with many science labs, the role of IT support includes developing software and customizing off-the shelf applications, often through scripting.

5.2 The Sweet Framework

Based on the general scenario described above, we identified a number of requirements for software support. We used these to develop a software framework, the SoftWare Environment for Experimental Technologies (Sweet). The Sweet framework provides basic services common across a spectrum of engineering and science disciplines, though the initial implementation was directed specifically at IOT's requirements. Below we describe our three primary requirements and how Sweet addresses them.

5.2.1 Ease-of-use.

The scientists and tank operators are the end-users responsible for gathering and analysing data. They often found it difficult to perform tasks that were conceptually simple because these tasks required a significant amount of software and computer knowledge that was outside the users' expertise. Our goals were to automate the tasks that could be fully automated and to simplify the tasks that required end-user input.

To do so, Sweet incorporates the implicit structure of IOT experiments. In Sweet, we defined a *runset* which is a grouping of individual runs that is presented to the end-user as a single object. All runs in a runset are based on a single *workflow template*, which is a parameterized workflow. This corresponds to the way in which many experiments are performed, with similar runs being executed in sequence with a few parameter changes. In Sweet, the end-user can add runs to the runset, execute batches of runs within a runset, change runset parameters affecting all runs of that runset or change each run's parameters independently of all other runs.

Additionally, Sweet allows the SE/IT support staff, working closely with the end-users, to create one workflow template to represent what was previously represented as multiple scripts with small variations. Parameterizations within the workflow templates represent the prior system's many script variants. End-users create an executable workflow from a workflow template by providing values for the parameters. With a smaller number of templates, the task of selecting the right template for an experiment becomes easier. Moreover, a GUI is automatically generated from the template to assist the user in selecting parameter values, thereby removing the requirement for end-users to program scripts.

5.2.2 Workflow Management

Scientific research often involves standardized activities that must be carried out in a specific order, such as the general process for water tank experiments detailed above. These activities are termed workflows, and in IOT's case they involve acquiring and analyzing data and generating reports. We observed that often steps that could be easily automated were performed manually in an error prone fashion. In other cases the workflows were automated via small programs (typically shell scripts). These small programs tended to be reused in an ad hoc manner, with multiple copies and versions existing within the organization and with insufficient version control or configuration management. Scientists and technicians would search the organization for the scripts that most closely matched their needs and then customize them for the specific task. These customizations were often performed by end-users with no software training and only a superficial understanding of the scripts.

This resulted in a number of problems. Without a proper repository and configuration management, scripts were hard to find. Script changes were not always recorded, so valuable information regarding data collection and processing was frequently lost. Depending on the complexity of the scripts, editing them could be an error prone, time-consuming task necessitating significant amounts of programming knowledge. While the scripting system provided a great deal of flexibility, it was difficult to use and manage.

Our approach to resolving these difficulties is to shift the need for programming skills from the end-users to other classes of users we refer to as template developers and tool developers (Figure 1). Using the Sweet framework, template developers work with the end users to map out the workflows and identify workflow variations that can be parameterized. Template developers require some scientific domain knowledge and some software engineering knowledge. They are typically members of the SE/IT support staff.

Rather than using a directed graph notation for representing workflows, we took the approach of representing them directly in an executable dynamic language, Python[6]. The motivation for this was to use a technique already familiar to the organization, i.e., shell scripting. Moving from a shell script to a dynamic language involved a technology change, but it did not involve a paradigm shift, and the organization felt comfortable changing the language used to represent their scripts. To this end, Sweet was intentionally designed to facilitate, not displace, existing processes and, on the end-user side, to start off small and lightweight and grow over time as user needs warranted.

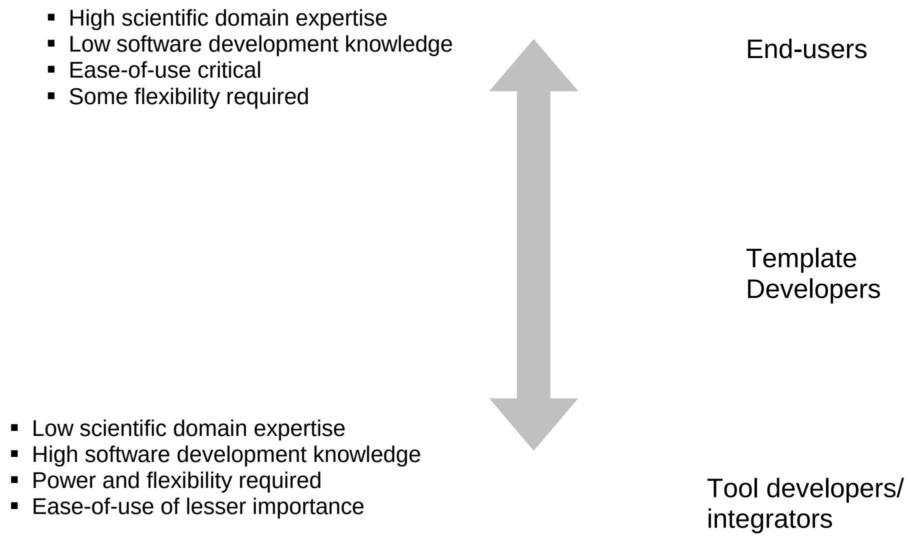

High scientific domain expertise
Low software development knowledge
Ease-of-use critical
Some flexibility required
End-users
Template Developers
Low scientific domain expertise
High software development knowledge
Power and flexibility required
Ease-of-use of lesser importance
Tool developers/ integrators

Figure 1. User roles within IOT

Using a well established dynamic language as a workflow representation had a number of additional benefits.

With limited manpower available to develop the framework, we did not have the resources to develop the workflow language, run-time environment, and development environment. Therefore, unlike most existing scientific workflow infrastructures, we used something off-the-shelf, i.e., Python. Python provided a number of support tools either directly or through the Open Source Software (OSS) community. Moreover, these tools were used extensively by the workflow template developers.

Since Python is a powerful full-featured language, once we had chosen to represent the workflows in Python, it became an obvious choice to implement the entire Sweet framework in Python as well. This provided us with a number of advantages. By developing our entire infrastructure in Python, there was a seamless integration between the workflows and the underlying implementation of the Sweet framework. Handles to data structures could be passed easily and object representations were the same in the workflows and the infrastructure. The dynamic nature of Python made it the ideal language for manipulating the templates, metadata and runsets, which are all represented in Python.

The main disadvantage of using a dynamic language for the workflow representation is that it lacks a formal syntax and semantics. This makes analysis of the workflow, for example to detect possible areas of concurrency, or deadlock analysis, impractical. As well, providing a graphical editor for building workflows, as is done with many of the graph-based approaches, is not feasible.

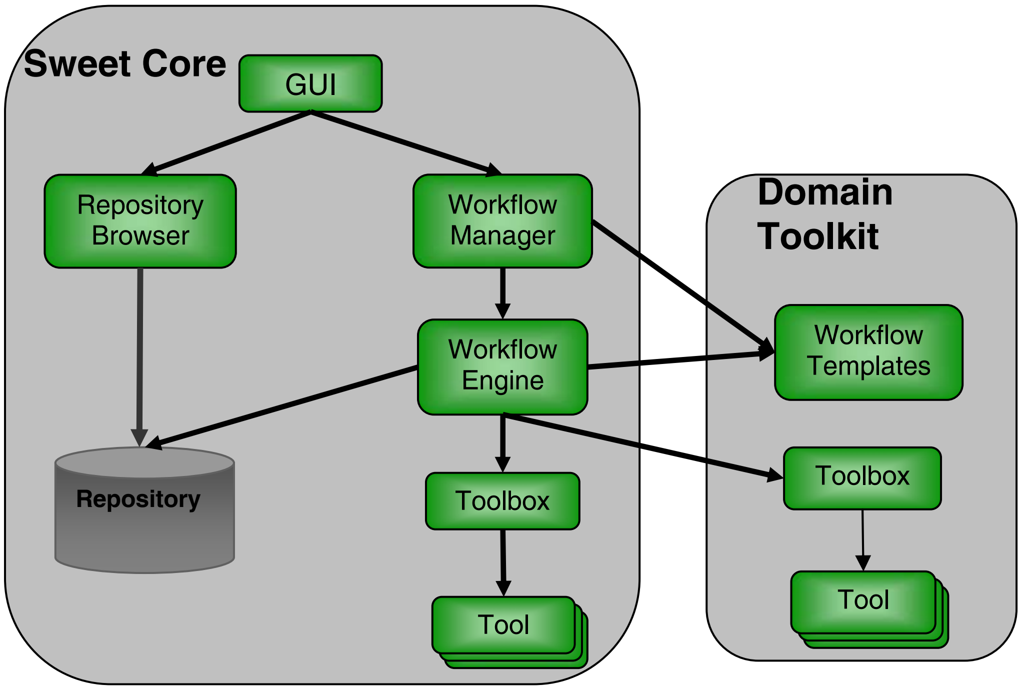


Figure 2. Architecture of the Sweet Framework.

Without an easy-to-use graphical editor, developing workflows requires some programming knowledge. However, since IOT has a relatively small and fixed number of workflows, and staff with the necessary skills are available for development, this is not a serious problem.

5.2.3 Software Integration

Software tools now manage the full spectrum of scientific and engineering activities, including data acquisition, data analysis and report generation. These tools include legacy software written in Fortran or C, commercial data analysis tools and highly specialized data acquisition software tools interfacing to proprietary hardware. The various software tools utilize different data formats, process models, exception handling mechanisms, and logging capabilities with no accepted standards for integration. Moving information between applications is one of the most labour intensive and time consuming steps in many science organizations, and requires scientists and technicians to perform data conversion and software integration functions – skills typically outside their area of expertise.

Our approach to dealing with software integration is to wrap applications with Python wrappers. A number of utilities facilitating the writing of wrappers are included within the Sweet framework, for example, scanning log files for error conditions and capturing the I/O from the wrapped application. The wrappers are collected into a ‘toolbox.’ Workflows dynamically link to the tools in the toolbox to invoke the necessary services. Wrappers were developed for a number of the Fortran programs still in use as well as for some commercial scientific applications.

6. System architecture

The high-level architecture of the Sweet system is illustrated in Figure 2. The basic framework consists of the *Sweet Core* used by all organizations regardless of scientific domain and the *Domain Toolkit* that allows for customization of the framework based on domain specific elements.

The core consists of the following elements:

Common Tools. A set of common tools applicable across a wide range of scientific disciplines. The tools are collected within a *Toolbox*.

Workflow Engine. Workflows, written in Python, can be executed directly. However a number of services are built on top of the basic interpreter, including parameterized workflows, metadata descriptions of the workflows and data recording.

Repository. The repository is a database storing information about the execution of workflows. It is queried through a *Repository Browser*.

Workflow Manager. The workflow manager provides a means for users to group related workflows.

Sweet GUI. The GUI provides the user interface for selecting, organizing, customizing and invoking workflows. Parts of the GUI are dynamically constructed from the workflow templates.

The Domain Toolkit consists of domain specific software tools, and the workflow templates for the organization. Currently we have a domain toolkit that is being used within IOT, and are developing one for NRC’s Institute for Aerospace Research (IAR).

The workflow templates of Sweet are parameterized representations of workflows. The template developer describes the template and its parameters using metadata within the template. Keeping the metadata within the template facilitates development and maintenance, as only file is needed. The metadata describes the template and various parameter characteristics including: the name and purpose of the parameter; its grouping which allows parameters to be grouped together; the type of the parameter defining the valid values that it can assume; and a default value.

Workflow templates are constructed using standard programming constructs. The overhead required by the workflow template developers to create a template is not significantly more than writing a script to do the same operation. For the Sweet framework, the developers’ overhead includes adding a declaration that the program is a template, adding metadata, and linking to the software toolbox.

An example of a simple template is shown in Figure 3. The template loads a data set, plots some data, and invokes a tool that allows a user to interactively select significant segments of the data. The string in triple quotes is part of the template meta-data. The `import` statements bring in the required parts of the environment. The toolbox being imported provides a mapping between services

```
"""Basic online analysis.
** Preprocessing
* project_title = Project Title
*    Leave empty to use project title from DAC file.
* included_channels = Included Channels
*    Enter a list of names of channels to be included in analysis.
* excluded_channels = Excluded Channels
*    Enter a list of names of channels to be excluded from analysis.
** Runset parameters:
* file_format(DAC File Format) = The format of the DAC input file.
* reanalysis_mode = Reanalysis Mode (True or False)
"""
from sweet.template import template
from iot.toolbox import Toolbox
@template(project_title='',
          included_channels=[],
          excluded_channels=[],
          custom_processor=(lambda x:x),
          file_format=FileFormat("VMS"),
          reanalysis_mode = False)
def basic_demo(self, data_file_name=""):
    """
    ** Run parameters:
    * data_file_name = DAC File Name
    """
    tb = Toolbox()
    dac_file = tb.dac_file(data_file_name)
    channels = dac_file.read()
    # Create an instance of a plotter and plot the data
    plotter = tb.plotter(report=report)
    ...
    plotter.plot(channels, pages)
    # Use the interactive selector tool
    selector = tb.segment_selector(
       interactive_mode=not self.reanalysis_mode,
       report=report, channels_to_display=...)
    segments = selector.select(channels)
```

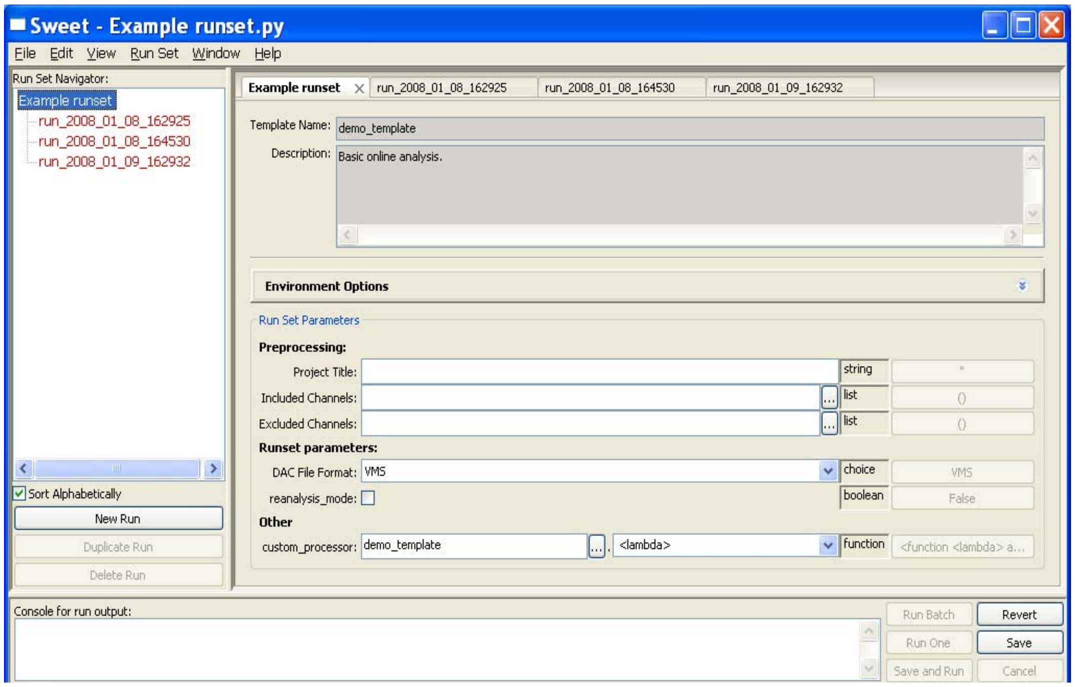
Figure 3. Example of a workflow template.

requested in the template and the underlying software program providing these services. The `@template` statement declares this Python script as a workflow template, giving the default values for any parameters. The body of the function creates a toolbox that is the link to the underlying software tools and then invokes tools as needed.

The Sweet framework reads the template and metadata and dynamically constructs a GUI interface that guides the user through the setting of the parameters. The GUI generated from the template of Figure 3 is shown in Figure 4. Shown is a runset with three corresponding runs.

7. Evaluating the Sweet Framework

The Sweet framework was evaluated through iterative development and through semi-structured interviews with end-users.



Template Name: demo_template

Description: Basic online analysis.

Preprocessing:

Project Title:		string	''
Included Channels:		list	()
Excluded Channels:		list	()

Runset parameters:

DAC File Format:	VMS	choice	VMS
reanalysis_mode:	☐	boolean	False

Other

custom_processor:	demo_template <lambda>	function	<function <lambda> a...

Figure 4. GUI generated from template.

7.1 *Iterative Development*

We followed a process of iterative development with IOT's SE/IT support staff. The end-users were not involved during the development as we did not have commitment or buy-in from them when the project was initiated. Nonetheless, SE/IT support staff worked closely with the scientists and tank operators during requirement gathering and analysis. Consequently, the development process was informed by the end-users' considerable knowledge of their processes and requirements.

Our initial collaboration focused on writing Python scripts to replace the scripts that were in use by IOT. As we saw what services were required, we began to develop the framework within which the scripts would execute and began turning the scripts into more complex workflows. This was done in collaboration with SE/IT support staff who provided detailed feedback on the framework as it evolved.

As the framework achieved a sufficient level of stability and functionality it was introduced to a small number of tank operators and scientists.

7.2 *Interviews*

We followed a qualitative approach in analyzing semi-structured interviews with representative end-users[7]. Five IOT Sweet users (IT/SE support, tank operators, and scientists) participated in our interviews.

Detailed notes were taken during each interview, and an audio recording served as a backup. Two authors used NVivo, a qualitative analysis application, (www.qsrinternational.com) to categorize the statements in the interview notes. The categories were expanded and merged as necessary. By the time the fifth interview was analyzed, no new categories emerged suggesting that we had saturated the analysis.

Before any interviews were conducted, the research protocol was vetted by the NRC research ethics board[8].

7.3 Results

One reason for Sweet's success was the combination of software expertise and scientific domain expertise among the SE/IT support staff. Although the external software developers were able to develop the framework, development of the workflows required someone with a much better knowledge of the end-users' processes. The SE/IT support staff allowed us to proceed along an iterative development path without intruding on the end-users until we felt well prepared.

While we were looking for information on ease-of-use in our interviews, we also found descriptions of the use of the Sweet software. We describe the uses of Sweet because we are finding that they are also represented in other labs.

7.3.1 Sweet Uses

The uses of Sweet fall into three general categories: data integrity checking, data analysis and collaboration. Tank operators used Sweet to ensure data integrity by verifying that there were no problems, such as faulty sensors, that invalidated the data. Scientists used Sweet for similar purposes, but they monitored data to ensure that the values remained in reasonable ranges. These activities are supported by Sweet's multi-channel displays, out of bounds data value alerts, and rapid loading of data. These functions allow users to monitor incoming data following each run.

Sweet provides data analysis capability by invoking services from external software tools. Currently, there is a limited amount of analysis being performed. This analysis consists of data normalization, some data transforms, and some reporting capabilities. Nonetheless, scientists find that even these basic capabilities are of use to them because by performing data analysis between runs they can optimize the conditions they select for the following run.

Sweet also supports collaboration by producing PDF files reporting runs and data channel graphs that can be shared easily with off-site collaborators.

Although the issues of data analysis, collaboration, and data integrity were uncovered by our interviews of Sweet users, we believe that they more generally describe some primary uses of scientific software by scientists.

7.3.2 Ease of Use

Perhaps the greatest impact of ease-of-use is productivity increase. Previous to Sweet, the scientists responsible for the experiment had to make themselves available when the script was being written because some hardware parameters had to be incorporated into the script while the experiment was set up. Sweet abstracted away the need to encode hardware parameters in the experiment set-up, relieving the experimenter from being present. Second, Sweet allows tank operators to perform data integrity checks following every trial, rather than after fewer than 10% of the trails. This reduces the number of trials that must be re-run due to data corruption caused by set up problems. Another productivity gain relates to the training of students. Before Sweet, students were unable to learn the scripting language in the short time frame they had available, thus forcing the scientists to set up the experiments. With Sweet, students were able to set up experiments with minimal training. Similarly, scientists now require much less help from SE/IT support in setting up experiments, allowing SE/IT support to perform other tasks.

Moreover, SE/IT support was able to develop just a few templates to replace an extremely large number of scripts, making it much easier to find, update, and maintain the relevant templates.

Prior to Sweet, the scientists had to send hard copy print outs of results to off-site colleagues. With Sweet, emailing PDFs has become the primary mode of sharing data.

7.3.3 Workflow Management

One observation we had was the large reduction in project specific workflows. Previous to using Sweet, each project had at least one project specific script, resulting in hundreds of variant scripts. With Sweet, these are all represented using a small number of workflow templates. Each project need only record the project specific parameter values. In addition to the project specific workflows, we expect the count of general purpose workflow templates to be less than forty. This justifies our approach to template development, which was to emphasize flexibility and power over ease-of-use, and to have template developers work closely with end-users in analyzing and parameterizing the templates.

The level of effort required to develop the workflow templates was appropriate for the knowledge and availability of the template developers. The templates were fairly small and quick to write (one to two pages of Python code), and the underlying tools were easily accessed through the dynamic binding of the toolbox. The template developers spent most of their effort writing common tools accessible by the different templates. Some of these were rewrites of the old legacy code, while others were general purpose graphing and reporting tools.

One feature of Sweet is that workflow template parameters can be custom functions written in Python. However, only one end-user took advantage of this capability.

On the other hand, the tools provided either directly or indirectly through Python were used extensively by the template developers. They used tools such as language editors, version control repositories, and issue trackers during template development even though they had not used them in the past. Thus Python not only relieved the framework development team from creating several workflow development utilities, it also provided the template developers with useful tools.

7.3.4 Software Integration

One of the initial objectives of Sweet was to facilitate interoperability between the different applications used by the end-users. Although this remains an objective, and has been achieved to some extent, the interoperability of the different applications has not been as extensive as we expected. Except for a few applications (some proprietary), the organization found it easier to rewrite services in Python rather than using the wrapped legacy applications we provided. However, as more complex analyses are integrated into the workflows, we expect that this situation will change and that wrappers around complex software applications will be required.

A similar phenomenon is occurring with another research institute with whom we are working, the Institute for Aerospace Research. They made extensive use of commercial software for their data analysis. However, as we start developing the workflows for them, it is becoming apparent that, for the majority of their workflows, they are using only a very small part of the capability of the commercial software and that this functionality can be easily provided by OSS software written in Python.

8. Future Work

With the successful deployment of the Sweet framework within one research institute, our future work will move in two directions. The first is to adapt the framework to other scientific domains and organizations. Our initial work with the Institute for Aerospace Research (IAR) is promising, as both IAR and IOT have similar processes and data analysis needs. One major difference, which creates a significant challenge, is that IAR does not have a strong SE/IT support team. The tool integration and development and workflow template development will have to be done by external developers who do not have a good knowledge of the processes of the end-users. We are not sure how this will impact the project.

The second major feature to be added to the framework is Information Management (IM). Scientific organizations accumulate large amounts of information. The information artefacts must be archived in a searchable fashion. As well, all information artefacts should include their complete provenance. The provenance of an artefact details exactly how and when the artefact was created. This is required not only to know the reliability and validity of the information, but to recreate the analyses that initially created the artefact. The framework currently maintains a database of all runs executed, and tracks all information items created. This feature has not been extensively evaluated however and requires a number of development iterations to become a robust service.

Having a system being used on a daily basis within a scientific organization provides us the ability to experiment with these issues.

9. Conclusions

Using an action research approach, working with NRC scientists, we developed the Sweet Framework to manage the mundane, everyday software tasks that scientists encounter in the course of their work. We found requirements in three primary areas: ease-of-use, workflow management, and software integration. Sweet addresses all three of these through the parameterization of workflow templates and the dynamic build of a GUI customized around those templates. Although the creation of the workflow templates does require programming knowledge, Sweet has nonetheless proven valuable. First, the IT/SE staff is free to concentrate on other tasks now that they do not have to customize the scripts for each instantiation of an experiment. Second, Sweet has allowed a wider variety of users to create, run, and monitor experiments. Third, Sweet allows increased monitoring, thus reducing the need to rerun experimental trials. Overall, our work with the end-users and IT/SE support at IOT was successful, and we are now working with other NRC labs to extend our user base.

10. References

1. R. L. Baskerville, "Investigating Information Systems With Action Research," *Communications of the Association for Information Systems, 2,* Article 19, 1999,
2. J. Segal, "The Nature of Evidence in Empirical Software Engineering," In proceedings of *STEP '03: The Eleventh Annual International Workshop on Software Technology and Engineering Practice,* IEEE Computer Society, 2003, pp. 40–47.
3. J. Carver, "Empirical Studies in End-User Software Engineering and Viewing Scientific Programmers as End-Users," In proceedings of *Dagstuhl Seminar on End-User Software Engineering,* Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI) Schloss Dagstuhl, 2007.
4. C. Letondal, "Participatory Programming: Developing programmable bioinformatics tools for end-users," In H. Lieberman, F. Paterno, & V. Wulf (Eds.), End-User Development, Springer, 2005, pp. 207-242.
5. A. Begel, "End-User Programming for Scientists: Modeling Complex Systems," In proceedings of *Dagstuhl Seminar on End-User Software Engineering,* Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI) Schloss Dagstuhl, 2007,
6. G. C. Fox, and D. Gannon (Eds.), "Special Issue: Workflow in Grid Systems," *Concurrency and Computation: Practice and Experience, 18,* 10, 2006,
7. P. F. Dubois (Ed.), "Special Issue on Python in Scientific Computing," *Computing in Science and Engineering, 9,* 3, 2007.
8. C. B. Seaman, "Qualitative Methods," in F. Shull, J. A. Singer, and D. Sjoberg (Eds.), *Guide to Advanced Empirical Software Engineering*, Springer, 2008, pp.35-62.
9. N. G. Vinson, and J. A. Singer, "A Practical Guide to Ethical Research Involving Humans," in F. Shull, J. A. Singer, and D. Sjoberg (Eds.), *Guide to Advanced Empirical Software Engineering*, Springer, 2008, pp.229-256.