Software Automation in Scientific Research Organizations

Mark Vigder, Darlene Stewart, Janice Singer
National Research Council Canada
{Mark.Vigder|Darlene.Stewart|Janice.Singer}@nrc-cnrc.gc.ca

1 Introduction

Science, software, and Information Technology (IT) are becoming integrally linked. Scientists use software for generating and analyzing data, embodying research results, managing their data sets, creating reports, and many other purposes. Unfortunately, the software engineering and information technology issues that scientists must deal with in order to effectively use their software tools are often an impediment to their effective work as researchers.

This paper describes a software development project that we have undertaken with scientific research institutes for the development of a software framework to facilitate the workflows of scientists and technicians. The objectives of the framework are to:

- Provide a standard means of integrating the heterogeneous software tools used by research organizations.
- Automate the workflows of the scientists and technicians.
- Provide Information Management (IM) services for managing the large number of data and information artefacts created by scientific research organizations.

As part of the work, it was necessary to analyze how scientific research organizations developed and used software as part of their research process. This analysis resulted in the identification of a set of distinct roles involved with the software, and the overall IT infrastructure. This paper describes the roles that we have identified within the research organization and how these roles can be supported with software tools. A framework providing some of these services has been built and is currently in use within a scientific research organization. The building of this framework, and its deployment, are discussed.

2 Background

The framework was built as part of an action research project, where we as software engineering researchers worked as part of the team with the problem owners, the scientific and engineering research organizations. Solving the problems involved the development of theories as to how the organizations worked, building tools based on these theories, and conducting evaluations of the tools and theories.

The organizations involved were the Institute for Ocean Technology (NRC-IOT) and the Institute for Aerospace Research (NRC-IAR). The two organizations have similar modes of operation: they instrument the item of interest (e.g., physical models of marine structures, jet engines), gather time sequence data during an experimental run, and process the data to produce results. The IT issues experienced by IOT scientists can be illustrated by a typical scenario. At NRC-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 tow (water) tanks. For instance, an experiment may focus on the strength of 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 experiment 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 *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.

Based in large part on our action research, we identified a number of problems faced by these organizations:

- Numerous data analysis tools were used that did not integrate easily. These included data acquisition tools, data analysis tools, and report generation tools. Many of these tools were legacy tools, often written in Fortran.
- Scientists and technicians were often editing software themselves. This involved, among other things, modifying shell scripts, editing data analysis tools, or customizing report generation software.
- Data was not being managed properly. Not all information items, such as derived data and reports, were properly archived with their complete provenance. Data that was archived was not indexed in a way that would allow easy recovery at a later date.
- Many tasks that were being done manually could be automated within software.
- Software variants were not controlled. Many software variants were created and needed to be stored with individual projects.

3 Roles and activities within a research Organization

A research organization begins by acquiring data for analysis. The data can be acquired from a data repository, or by experimentation to gather raw data. Once the data has been acquired, a number of data analyses are performed in order to transform the data into a form from which useful knowledge can be extracted. This knowledge must be represented in a way that is suitable for distribution, for example by generating appropriate charts, graphs and tables or by depositing it in a repository.

Although the data analysis and report generation used in each experiment are different, within an organization they tend to follow regular patterns. A simple process would involve cleansing the experimental data, normalizing data from different sensors, performing a set of transforms on the data, and putting the data in a chart or graph. For different data sets this might require different transforms or normalizations, but the structure of the analysis processes, and the kinds of report artefacts produced, are similar.

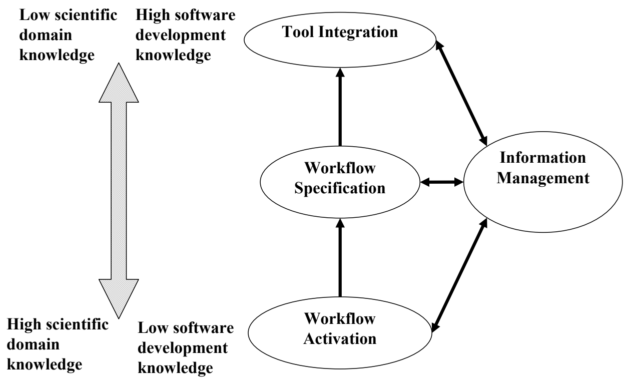

Low scientific domain knowledge
High software development knowledge
Tool Integration
Workflow Specification
Information Management
Workflow Activation
High scientific domain knowledge
Low software development knowledge

Figure 1. Activities in a scientific organization.

The processes that an organization uses to acquire, analyze, and disseminate information are defined as *workflows*. A workflow is a sequence of activities that are performed to generate a required output.

The activities underlying a workflow are supported by a set of software tools. These tools fall into a number of different categories. For example, some are computation bound software programs that perform the transforms on the raw or derived data; others are visualization tools that generate the graphs and charts for inclusion in the resulting reports.

Concurrently with all the workflows, a number of other activities must be supported by the organization in order to manage all the resources associated with the research. Records must be maintained of each analysis, all data must be archived, and configuration and change management of all IT resources must be performed. Putting in place these management activities allows all knowledge to be kept, mined, and reproduced at any point in the future.

Given the above discussion, a number of activities and their associated roles can be identified (**Error! Reference source not found.**). These are:

- *Tool integration* – integrate the heterogeneous off-the-shelf software tools used to support activities of a workflow. The associated role, the *tool integrator*, builds wrappers for the software tools to integrate them into the organizations IT infrastructure. The tool integrator needs strong software development skills, but is not necessarily an expert in the scientific domain.
- *Workflow specification* – formally specify the workflows used by an organization. The associated role, the *workflow specifier*, must understand the processes used by scientists to generate knowledge, the software tools available, and the means of specifying the scientific processes in an executable form. The workflow specifier needs a combined knowledge of software engineering and the scientific domain.

- *Workflow activation* – invokes the activities of a workflow in order to perform a data analysis. The associated role, the *end-user* (operator or scientist), invokes the workflow by first selecting the applicable workflow, tailoring it for the specific data analysis being performed, and initiating execution. This activity requires deep knowledge of the scientific process, but should not require software engineering knowledge.

Crosscutting all these activities and roles, is a fourth activity, *information management*. This involves the tracking, archiving, and managing all the information artefacts created. It is primarily an IT issue, and is similar regardless of the scientific domain. All roles within the organization have some responsibilities for information management.

4 The Sweet framework

In order to support the three identified roles, we developed a framework that can be applied across different scientific domains. The framework is called the **S**oft**W**are **E**nvironment for **E**xperimental **T**echnologies (Sweet) framework. The primary objective of the Sweet framework is to provide ease-of-use for the end-user role. However, to achieve this objective it was necessary to include support for all three roles. This section summarizes the features of the framework to support the different roles.

4.1.1 Tool Integrator

Software tools are any software code used within an organization. This includes stand-alone applications, services, and software libraries. Software tools can be custom written by the organization, or off-the-shelf software.

An external software tool is integrated into the Sweet framework by the tool integrator developing a wrapper for it. By developing a wrapper according to the standard of the Sweet framework, it will be available as a tool in the local toolbox and can be dynamically linked. Since the tools are dynamically loaded into the framework, different users can have different tools available at their workstation.

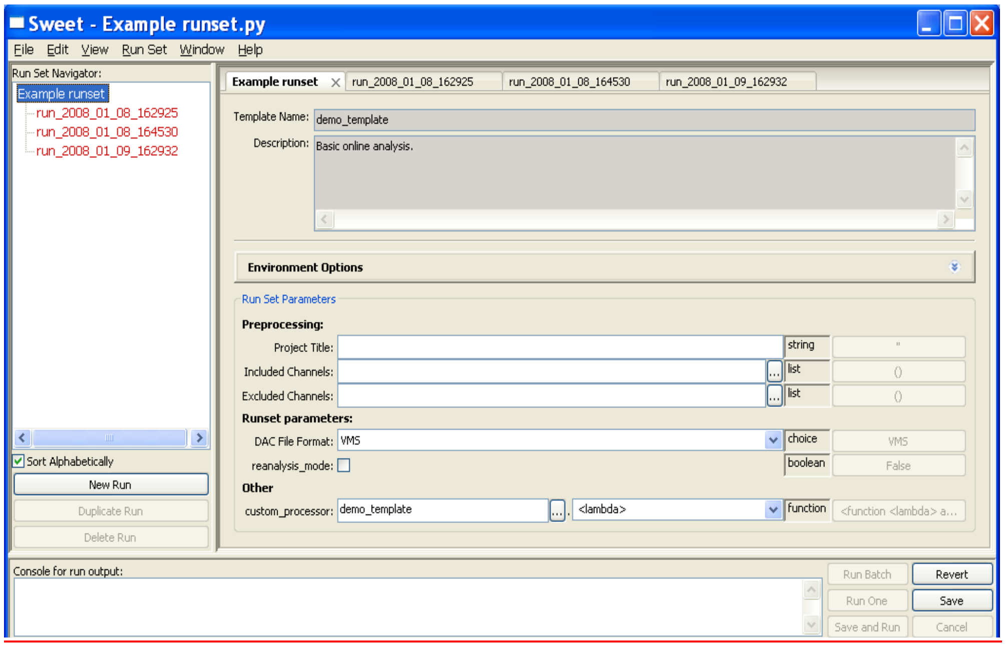
The Sweet Framework allows any type of tool to be integrated into the framework by developing a wrapper around it. Within the Sweet Framework, two types of services support the tool integrator in their role: general utilities for controlling the tool; and dynamic integration of the tool into the frameworks ‘toolbox’.

For developing wrappers for the software tools, a number of general purpose utilities exist within the Sweet framework, including output redirection; log file scanning and analysis; program control utilities; and exception handling. As well, other open-source tools exist for help with writing wrappers that interface to software libraries written in C or Fortran.

4.1.2 Workflow developer

The workflow developer is the role between the tool integrator and the end-user. In order to support this role, the services provided within the Sweet framework include:

Executable workflow representation. The Sweet framework uses a dynamic language, Python, for representing workflows. Using an existing language for workflows, rather than developing a new one, had numerous advantages, such as readily available off-the-



Example runset
run_2008_01_08_162925
run_2008_01_08_164530
run_2008_01_09_162932

Template Name: demo_template
Description: Basic online analysis.

Run Set Parameters

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 2. GUI generated from template.

shelf IDE’s, and a cross-platform interpreter. As well, using Python as both the workflow representation and the framework implementation greatly simplified the interface between the two, allowing for much easier integration of both control and data structures.

Parameterized workflows. Rather than hard coding all functionality into a workflow, workflow developers have wide latitude to parameterize the workflow. The parameterized workflow is called a *workflow template*. The parameterization requires the workflow developer to study the end-user processes to determine the commonality and variations between the processes. The commonality is hard coded into the template while the variations are represented by the parameters.

A number of standard types are included within the Sweet framework that are common to many disciplines. These include, for example, different numeric types, I/O types, lists, functions, etc. The workflow developer and the tool integrator can extend this list to their particular domain by adding the appropriate metadata descriptions and end-user functionality required.

Metadata workflow template descriptions. The workflow developer describes the workflow template and its parameterization using metadata facilities. These metadata descriptions are used by the framework to guide the end-user through the workflow configuration (see Section 5).

Dynamic binding to software tools. Using the dynamic binding features of Python, the tools available on a workstation are registered at run-time and placed in the toolbox. By importing the toolbox, the workflow developer has available any of the tools, including tools specialized to the particular scientific domain.

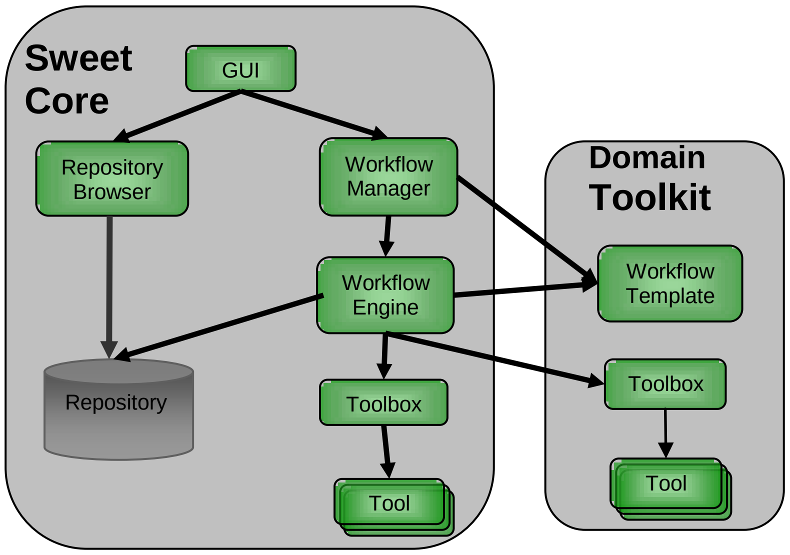


Figure 3. Architecture of the Sweet Framework.

4.1.3 End-User

The Sweet framework supports the end-user by helping them to configure the workflows and manage their workflow invocations. Using the metadata descriptions provided by the workflow developer, the framework builds a GUI for the end-user (Figure 2). Based on the parameter types users enter the appropriate data values. Various help and instructions can be included as part of the metadata.

End-users require a means of organizing the different workflows that they invoke. Within the Sweet framework this is achieved by naming each workflow execution (called a *run*) and organizing the runs into *runsets*.

Information management functionality is achieved through a set of services that track the data and information items as they are created, and store them into a database along with their metadata.

5 Implementation

The high-level architecture of the Sweet system is illustrated in Figure 3. 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 Sweet 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. 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 one 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 for the visual representation to the end-user; the type of the parameter defining the valid values that it can assume; and a default value. 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 4. 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 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 4 is shown in Figure 2. Shown is a runset with three corresponding runs.

6 Conclusions and observations

The Sweet framework has been implemented and is currently deployed within the NRC-IOT where it is being used primarily for on-line analysis. An evaluation of the effectiveness of this deployment for end-users has been done using structured interviews. Results of this study are described further in [3]. Plug-ins and workflows are currently being developed for use as a second research institute (Aerospace, Gas Turbine Laboratory).

```
"""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 4. Example of a workflow template.

Deployment at NRC-IOT has succeeded for a number of reasons.

- The use of dynamically generated GUI's, allowing the end-users to customize the tools as needed was very successful.
- A great deal of the success was due to the strong software engineering group within IOT. This group not only had expertise in software development, but also had deep knowledge about the scientific domain. Working with IAR is proving to be somewhat more challenging as they do not have the same level of software expertise internal to their organization.
- Many of the standard software engineering techniques (configuration management tools, automated testing, issue/bug tracking) were introduced into the organization and were quickly adopted.

Areas where the framework requires extensions, and on where we are actively researching include:

- Improved Information Management (IM) tools integrated into the Sweet core.

- Better support for exploratory scientific research where events and discoveries may change the pattern of work.
- Building plug-ins for different scientific domains to better understand the common services to be included in the core.
- Evaluations of the scientific organizations to determine the effectiveness of our current framework and to develop new theories and tools to support researchers.

7 References

1. G. C. Fox, and D. Gannon (Eds.), “Special Issue: Workflow in Grid Systems,” *Concurrency and Computation: Practice and Experience, 18,* 10, 2006.

2. P. F. Dubois (Ed.), “Special Issue on Python in Scientific Computing,” *Computing in Science and Engineering, 9,* 3, 2007.

3. M. Vigder, N. Vinson, J. Singer, D. Stewart and K. Mews, “Automating Scientific Workflows,” *IEEE Software,* Vol. 25, No. 4, July/August 2008.