ADVANCED WORKFLOW FOR EXPERIMENTAL CONTROL

D. Mannicke, N. Hauser, N. Xiong, ANSTO, Sydney, Australia

Abstract

GumTree is a java software product developed at ANSTO and used for experimental control as well as data visualization and treatment. In order to simplify the interaction with instruments and optimize the available time for scientists, a user friendly multi sample workflow has been developed for GumTree. Within this workflow users follow a step by step guide where they list available samples, setup instrument configurations and even specify sample environments. Users are then able to monitor the acquisition process in real-time and receive estimations about the completion time. In addition users can modify the previously entered information, even after the acquisition sequence has commenced. This paper will focus on how ANSTO integrated a multi sample workflow into GumTree [1], what approaches were taken to allow realistic time estimations, what programming patterns were used to separate the user interface from the execution of the acquisition, and the future opportunities the approach enables.

INTRODUCTION

Experiments have become more complex and involve different instrument configurations and sample environments. Our aim was to provide scientists with a user friendly application that would simplify the data acquisition process and the associated interactions with the instrument being used as well as help them to optimise the experimental time available.

For example with Quokka, a small angle neutron scattering beamline at ANSTO, it is very common to set up an experiment that requires measurements with 3 configurations, at least 7 samples and may also involve 5 different temperatures. This setup alone results in 105 single measurements which would be very time consuming to manually prepare.

In order to increase the acceptance of the provided solution by the scientific community, scientists and researchers were included in the development process. We asked about their expectations, ideas and experiences at other facilities. We received very different and sometimes contradicting responses. Finally we decided on a solution that delivered a compromise. The end result is suitable for the majority of experiments conducted on the beamlines.

Initially we assumed it would be sufficient to simply loop-over configurations, samples and temperatures in order to generate an acquisition sequence using existing scripting language commands. However, in practice many scientists desired more flexibility; they wanted to be able to adjust the order of the acquisition sequence including duplicate and remove measurements. The scientists also wanted the ability to make these changes after the acquisition sequence had commenced. The interactivity on a running command sequence could neither be implemented with a scripting language 'loop' construct nor with a simple 'command list' or 'command table', which are traditionally generated by unrolling nested loops.

A key requirement from the development perspective is that the system could be deployed and maintained on multiple beamlines with minimal effort. Therefore we had to develop a solution that keeps most of the implementation generic but provides sufficient flexibility to allow a customised user interface to be created for each beamline.

In addition, to improve the robustness it was decided to separate the system into a server and a rich client application. The server was developed to run in a headless mode on hardware with direct communication to the instrument control system. Furthermore, the server had to provide an interface that allows multiple clients to monitor and control the acquisition sequence.

IMPLEMENTATION

For the implementation of the user interface, the workflow was divided into four individual tabs where each tab represents a different aspect of the system (See Fig. 1). The first tab is dedicated to the sample stage; this includes specifying the properties of the used samples as well as the utilized sample holder. The second tab is concerned with setting up the desired beamline configurations which may include different sample to detector distances and aperture configurations. The next tab is concerned with sample environments; this is where a user can specify different temperature ramps or magnetic field strengths. On the final tab, the user is able to start and stop the acquisition as well as control and monitor the acquisition sequence.

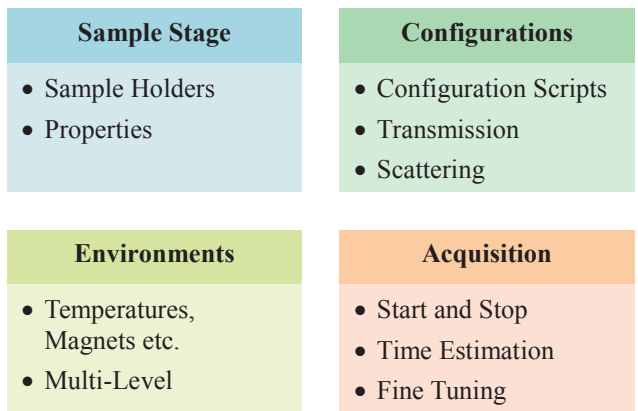

Figure 1: Workflow aspects

In order to keep the system easy to maintain for multiple beamlines we minimized the instrument specific components (See Fig. 2). For example on the server side, only an XML Schema Definition file, which is used as an

ISBN 978-3-95450-148-9

beamline definition file, is necessary to describe the elements of an acquisition sequence (e.g. Sample, Sample-List, Configuration etc.) including what properties it contains and their relationships. Furthermore, only one instrument specific python script is required that translates the instructions generated from the workflow into commands for the instrument control system. This product can be used on a variety of instrument control systems or hardware abstraction layers.

On the client side, the scientific user is presented with instrument relevant UI elements. To ensure that the client side development is kept to a minimum, a lightweight framework was developed to manage the client/server communication as well as the synchronisation of the client with the model.

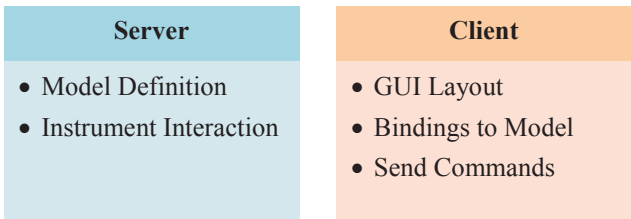

Figure 2: Instrument specific components

To minimize on-going maintenance efforts we ensured that most components are beamline independent (See Fig. 3); this allowed components to be shared across multiple beamlines while still providing the flexibility to cope with different instrument particulars. The beamline independent components on the server include the model database (which can be fully instantiated with the beamline definition file). Furthermore, all algorithms required for the time estimation as well as the communication protocols can be shared across all beamlines.

Figure 3: Beamline independent components

PROGRAMMING PATTERNS

Most influential programming design patterns were the *Model View ViewModel* (MVVM) in conjunction with the *Command* pattern. The MVVM pattern was developed by Microsoft to simplify event-driven programming of user interfaces. It consists of three components: *Model*, *ViewModel* and *View* as well as bindings and commands (See Table 1). The idea is to keep the *Model* and *ViewModel* independent from the code and technology used for the actual *View*. As a result most of the code relating to the *Model* and *ViewModel* can be shared across multiple beamlines and the beamline specific *View* can be easily maintained.

Table 1: Components of MVVM

Component	Description
Model	contains the bare information that describes the current state or content
ViewModel	provides bindable properties and commands and only the ViewModel has direct access to the Model
View	visualises the model via bindings to the properties and commands provided by the ViewModel

For further information, see [2], [3] and [4]; these resources provide great information and guided tutorials.

REALISTIC TIME ESTIMATION

Neutron scattering experiment sequences managed with this product can run for many hours, and are followed by sample changes that may take place at any time of day. Realistic time estimations were therefore a crucial feature for our scientists and thus we wanted to ensure that we are able to provide realistic values. We implemented a statistical approach that is generic enough to be used with different beamlines as well as able to capture the beamline specifics. Furthermore it was necessary to not just provide a single value for the estimation but also the uncertainty associated with it.

Since for us the beamline specific python script is the only code that directly interacts with the Instrument Control System we chose to add a custom profiler (via *sys.settrace*) into the python layer. This profiler is able to log all python calls, track the arguments and record the execution times of each function. Consequently we obtain a call graph with statistical metadata that also includes state transitions of the beamline configuration. This call graph then allows us to execute the code with different sets of input parameters without interacting with the beamline hardware. For a given experiment (which may contain multiple samples and configurations) an accumulated time estimation can be determined.

LESSONS LEARNED

The inclusion of the scientists in the overall design process resulted in positive feedback and ultimately increased their desire to see the project succeed. Involvement was facilitated by conducting review meetings. An agile approach to development was taken which relied heavily on the scientists being involved and providing feedback. This approach resulted in the success of the project and allowed us to understand the needs of the scientists as well as mange their expectations.

ISBN 978-3-95450-148-9

FUTURE

One of the future advancements that the implemented system enables is to use the collected statistics about the instrument response times to provide information about the history of the instrument performance regarding motor movements. This could further be enhanced to automatically notify instrument responsible if, for example, certain motor movements take significantly longer than predicted.

Another future development could involve providing an http web client that can be used to monitor the acquisition sequence via a web browser on a remote computer or smart phone. This would be particularly useful if an experiment takes multiple hours to complete.

CONCLUSION

The advanced workflow described in this paper provides features not implemented previously in beamline workflows. These features include reordering, duplicating and removing tasks from a running workflow and the statistical method used for time estimations.

REFERENCES

[1] T. Lam, N. Hauser, A. Gotz, P. Hathaway, F. Franceschini, H. Rayner, "GumTree, an integrated scientific experiment environment", Physica B 385-386, 1330-1332 (2006)
[2] http://www.codeproject.com/Articles/100175/Model-View-ViewModel-MVVM-Explained
[3] http://www.wpftutorial.net
[4] https://sourcemaking.com/design_patterns/command

ISBN 978-3-95450-148-9