Instrumentation in the Stockholm XRM lab

Mårten Selin
Department of Applied Physics, Royal Institute of Technology, Stockholm

Abstract

A software platform is presented that brings the instruments of the Stockholm X-ray microscope (XRM) together. The code is designed with flexibility and extensibility in mind. Its features are reported and promise to make the microscope operation more comfortable to the user.

Introduction

The Stockholm XRM’s performance steadily improves. The principles and successful operation has already been proved [1]. Now the focus is shifting towards achieving robust stable operation and demonstrating more biology applications in general, and tomography in particular. While this shift is certainly the goal and is a success in itself, it loads the operator with dual responsibilities. The operator has to be committed to the application and the microscope instruments.

While straightforward, it is presently a considerable amount of effort to have the system up and running, and after that it is tiresome to manually monitor, tweak and optimize all parameters of interest. The current system is a heterogeneous mix of instruments, some accessed through physical panels, some through the manufacturer’s own (or some general purpose) software and finally some through home-brewed scripts. It is needlessly to say hard to uphold a firm overview of the system’s status.

On many occasions, during investigation of a certain phenomenon or comparison with previous operation, it is important to have reliable logs of the microscope’s sensor data. It is therefore important to consistently measure and observe conditions in the setup. However, due to the overwhelming amount of information and unreasonable distribution among software and physical panels, it is not realistic (or humanitarian) to expect the operator to note down 5 - 10 measurements at 1 minute intervals while performing a 3-hour bio-imaging session.

All of these reasons emphasis the desire for more automatization and visual feedback. That is why members of the Soft X-ray group at BIOX investigated possibilities and eventually began programming an instrument software platform providing easy access through a graphical user interface (GUI), to an increasing set of instruments. Although at small scale at first, the potential is obvious. Properly presented, sensor data will provide the operator with better overview and insight of the system’s status. Automatic logging and comfortable control over instruments lessen the burden on the operator allowing him/her to focus on the more critical parts of the experiment. This promises to increase the productivity in the Stockholm XRM lab.
Design goals

The design goals of the software are listed below

1. Provide framework that allows easy and robust communication between user and microscope instruments.
2. Each GUI-tool and script should have easy access to whatever microscope instruments it requires.
3. Each instrument should be controllable from multiple GUI-tools and scripts.
4. As this is research, the experiment system is constantly changing. Therefore:
   a. The microscope instruments should be exchangeable where they provide the same basic functionality. E.g. upgrading the voltage reader should not have an impact on the software.
   b. It should be easy and straightforward to write drivers for new instruments so that the drivers conforms to the standards of the software and live happily in its ecosystem.
   c. It should be possible to simulate microscope instrument behavior, for testing and offline-development.
5. Logging. All relevant data should automatically be saved and a formatted report generated at the end of the experiment.

Implementation

LabVIEW

LabVIEW from National Instruments [2] is chosen as the platform for this programming-project. Some benefits are that it is an industry standard, contains a rich set of instrument drivers and tools for analyzing, makes building the GUI easy and finally that it encourages/enforces good multithreading practices. Disadvantages includes that it is proprietary, hides functionality in an invisible overhead and sometimes lacks desired flexibility.

Basic architecture

In order to meet the design goals, the architecture depicted in Fig. 1 is adopted. It compartmentalizes code according to different responsibilities in three layers.

![Diagram](image)

Figure 1 – The code is compartmentalized into three layers. Here a selection of the implemented functionality is shown. In the middle layer a set of instrument prototypes, one for each microscope instrument included, act as servers for the GUI-components and scripts of the top layer. The instrument prototypes load plugins for the actual hardware-interfacing code from the bottom layer.
**Top layer: GUI and scripts (clients)**

The code written in the top layer are basically the tools with which the operator interacts. It communicates with an abstract model (in the middle layer) of the instrument. This frees this part of the code from needing to know any particulars about the instrument, other than that it does its job.

Example: the cartographer-tool sends commands to the sample stage server, but it does not need to know specifically how to communicate with the Compustage. It need only know that supposedly the command makes the sample move as desired.

Design goal 1 is met when these GUI-tools are well coded and goal 2 is met by the architecture (Fig. 1).

**Middle layer: Instrument prototypes (servers)**

In the middle layer reside the instrument prototypes, implemented as servers. Each server is responsible for one of the microscope instruments. Upon initialization a server loads the appropriate plugin for its instrument. It manages communication, allocation of resources and other hardware-independent logistic tasks. Received data and status changes are broadcasted to all clients. All instrument-specific tasks are delegated to the loaded plugin.

Example: the X-ray CCD server allocates memory space to store images in and schedules the recurring readout of data, but leaves it to the plugin to perform the action of reading out data.

Since the server can interact with multiple clients simultaneously, design goal 3 is met.

**Bottom layer: Hardware drivers (plugins)**

Since the server handles all hardware-independent tasks, only the blanks associated with hardware-specific code needs to be filled in here. Thanks to this design, replacing one of the microscope instruments impose minimal coding effort and has no impact on other code. This design directly addresses design goal 4.a and 4.b. Design goal 4.c is met by writing “Dummy”-plugins, that simulate a microscope instrument behavior.

**Actor Framework**

Within LabVIEW it was decided that the Actor Framework [3] should be used. It provides means to run independent processes with a robust messaging structure. While the framework is extensive, its full capacity is not utilized. It would certainly be possible to develop a more lightweight framework for this task, and indeed the author briefly investigated this possibility, but it was concluded that AF provided the most robust and future-proof solution.

**Current implementation**

This section reports on the currently implemented microscope instruments and the GUI. As of now only a few of the potential instruments have been included in the software. However, the software as it is, is already routinely used during experiments in the lab. This is not because it is more advanced than previously used commercial software, but rather because it packages the desired functionality in a practical manner optimized for typical workflow during experiments. Furthermore, by combining the functionality of different microscope instruments new functionality is born not obtainable by any previous software. Because of this it is already an asset in the lab.

**Microscope instruments**

The currently implemented microscope instruments and their status are listed in Table 1.
Table 1 – Status of the currently implemented microscope instruments. The left column shows each instrument’s respective LabVIEW-symbol.

### Symbol

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="X-ray CCD" /></td>
<td>X-ray CCD</td>
<td>Implemented for the Princeton Instruments cameras that are compatible with the Scientific Instrument ToolKit (SITK) [4].</td>
</tr>
<tr>
<td><img src="image" alt="Sample stage" /></td>
<td>Sample stage</td>
<td>Currently a modified TEM-sample stage (Compustage) from FEI is used for sample stage. The plugin communicates over a local network with the TEM-computer.</td>
</tr>
<tr>
<td><img src="image" alt="Visual microscope camera" /></td>
<td>Visual microscope camera</td>
<td>This camera is located on top of the sample-chamber and looks down on the sample and zone plate through a visual microscope. Currently a plugin for all cameras that are compatible with NI IMAQdx [2] is implemented.</td>
</tr>
<tr>
<td><img src="image" alt="Voltage monitor" /></td>
<td>Voltage monitor</td>
<td>This component measures the analog voltage signals of the microscope’s sensors. Currently they are the chamber pressures and the cryostat pressure. Implemented for one plugin using NI DAQmx [2].</td>
</tr>
</tbody>
</table>

### Graphical user interface

Figures 2-4 show the most used graphical user interface elements.

Figure 2 – The launchpad is the main GUI-window, where the microscope instrument servers are managed (top left), log output is shown (middle), as well as relevant system data such as chamber pressure (top left).
Figure 3 – The imager tool controls the X-ray CCD. It starts single or continuous acquisitions of desired exposure time and binning. Some simple image analysis tools, useful during an experiment run, are implemented. The images are automatically saved on disc and in the logbook, with relevant data (e.g., time, acquisition parameters, system status) attached.

Figure 4 – Combining the images acquired by the X-ray CCD with the sample stage’s current position allows a map to be stitched together in the cartographer tool. Clicking on the map brings the sample stage to the corresponding position, so it is easy to navigate around on a sample grid and find interesting specimens to image.

Logging
All relevant sensor data and status messages are recorded by the logbook-tool (see Fig. 2). After an experiment finishes it compiles a report. Thumbnail images of all X-ray CCD acquisitions are attached. Thus the tool, with no effort from the operator, provides easily accessible documentation for later review of that day’s experiment. Thus design goal 5 is also met.
Upcoming implementations

New X-ray CCD-plugin

Recently a new X-ray CCD camera from Andor has been purchased, for which new hardware driver-code needs to written. However, since an SDK for LabVIEW is provided and due to the modularized architecture of the software, minimal effort is anticipated.

Source stage controller

Code for the Thermionics stepper motor controller (SMC), that drives the laser focus-positioning and liquid jet-positioning stages, has already been written but is not yet transferred to the platform of this report. Doing this would provide opportunities for more automatization of the plasma operation.

Zone plate stage controller

Also here some code is written for the current Physik Instrumente-stage that moves the zone plate, but not yet transferred to this report’s software platform. Implementation would allow, among other things, making automated focus series. Furthermore, the valuable zone plate can be protected against, e.g., mistyping the target position by defining soft limits.

X-ray photodiode

Continuous measurement and plotting of the pulse-peak value of the X-ray photodiode would give valuable insight of the plasma status. The signal could also be used as feedback for a plasma-source optimizing tool.

Summary

A new software platform including support for some of the most important microscope instruments have been implemented. Its modularized design makes it very flexible and able to adapt to changes in the microscope system. Already with just a few of the microscope instruments currently included, it is of great use and an asset during microscope operation.

Acknowledgements

The author gratefully acknowledges Emelie Fogelqvist and Mikael Kördel for valuable discussions and testing the code and Anders Holmberg for LabVIEW-support and discussions.

References