FP7 – Work package 10.2

Preliminary study on major design issues of the integrated beamline software platform

Table of Contents

Illustrations

1. Introduction

1.1 Status of software on beamlines at ESRF

Over the years the Bliss group at ESRF, acting both as a support team for instruments control and as a software development force, introduced standard tools and common practices on ESRF beamlines. That was a big challenge : each beamline has its own issues and way of doing experiments ; however, since the early days effort has been put in trying to find generic solutions to common beamline data acquisition and experiment control problematics.

This reasoning does not imply that predefined solutions are hopelessly tried to be applied to specific problems : it is quite the opposite. Specific problems on beamlines are opportunities to improve the generic software and solutions, whenever it is possible. The generic approach helps a lot on several aspects : all Bliss members share the same know-how, and it becomes easier to spread state-of-the-art techniques when people share the same background. Then, every beamline can easily benefit of some software development triggered by another one. And even more importantly, the overall quality of service is improved compared to a situation where everybody develops and maintains his own piece of software : bugs have a higher probability to occur and thus to be solved, a wider range of cases are taken into account, and software usually has already been tested. When better solutions are found to some problems the standard software is updated accordingly, enforcing a kind of “natural selection” within the software ecosystem.

To allow such an organization within the Bliss group, software tools need to fulfill some criteria. Flexibility comes in mind, but also stability, durability and efficiency. This last quality may be difficult to quantify. Of course, an efficient software can be the one that simply “does the job”. Often, tough, more subjective criteria apply like ease of use, ease of configuration, required time to do a task, user-friendliness, etc.

Software on beamlines can be classified in 4 groups :

- Drivers and other low-level software (e.g hardware test software)
- Device or instruments control
- Data acquisition& experiments control
- Graphical interfaces for experiments

An integrated beamline software platform covers the 3 last items from device or instruments control to graphical interfaces. There are also other aspects that need to be taken into account into an integrated beamline software platform :

- Configuration
- Software deployment

The Bliss group currently uses different technologies and different kind of software depending on each task group :

- MIDDLEWARE: Taco / Tango control systems are mainly used for device or instruments control
- SEQUENCER: CSS Spec is mainly used for executing data acquisition sequences
- GRAPHICAL INTERFACES: the Bliss Framework (based on Python / Trolltech Qt) is the main tool to build graphical interfaces on top of the two previous systems
- SOFTWARE DEPLOYMENT: it is achieved using a couple of in-house developed tools, Bliss Builder and Bliss Installer. The builder is creating RPM files, that can be deployed on beamlines via the installer.

1.2 Ways of improvement : the need for an integrated beamline software platform

The current software system has different problems :

- there is no centralized configuration
- there is an overlap between the different software components ; there is no overall architecture thought about since the beginning on how the different components should interact. Almost none of the tools are limited to a particular domain : it is possible to communicate with a device on a beamline using Spec directly, for example, instead of relying on a Tango server. It is also possible to write a data acquisition sequence in Python within the Bliss Framework. As there is no clear path, multiple solutions are easily found to a problem which can lead software odds and ends
- interoperability between the systems requires “glue code” to be maintained, since every system has its own preferred communication protocol
- the sequencer doesn't offer multitasking abilities
- the sequencer language is too limited
- the data format used to store acquired data doesn't fit our needs anymore
- standard data visualization tools (while data acquisition is going on) are missing features

2. Specifications for an integrated beamline software platform

2.1 Communicating with beamline instruments and devices

2.1.1 Heterogeneous hardware brought together

A beamline consists of dozens of heterogeneous devices (motors, encoders, pneumatic actuators, switches, cameras, etc.) and scientific instruments brought together. In the case of scientific devices sold by some company, there is no control on how the particular device will be connected to the beamline ; communication buses and protocols can vary from : serial line (RS232, RS432, etc.) with some ASCII-based or binary protocol, GPIB (IEEE 488) with some specific binary protocol, TCP/IP sockets again with some protocol, CameraLink, USB...

Sometimes it is possible to decide on a common way to address some particular devices, for example at ESRF the new ICEPAP motor controllers can connect to an Ethernet network and it is possible to "talk" with the controller using TCP/IP sockets. Then, all motors that can be plugged on an ICEPAP controllers can be controlled using this protocol, which is very convenient. But this is not the most usual case.

An integrated beamline software platform has to provide a solution for communicating with these different instruments and devices, in an uniform manner. Indeed, all the job of such a platform is to communicate with these components, to execute experiment control sequences, to acquire data and to display results.

2.1.2 Middleware

In order to provide such communication layer from the hardware to the integrated beamline software platform, middleware is to be used.
What is "middleware" ? Middleware is computer software that connects software components or applications. The software consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network.

2.1.3 Performance issues

Events
Big amount of data transfers

Reliability

2.2 Data acquisition & experiment control

Data acquisition and experiment control require operations to be executed one after the other : that's what we call sequences. Sequences are run by a sequencer. What is the role of a sequencer ? How sequences can be written ? User needs to have full control over what the sequencer is doing, and to be able to interact with it : how interaction with user can be done ? And what can be the architecture of a sequencer ?

2.2.1 Role of a sequencer

On a beamline, the sequencer is a key part : from the middleware (see previous section), it allows to get access to the hardware and helps implementing the logic behind any experiment. If motor movements are synchronized by hardware, or if data acquisition is made very close to the hardware for performance reasons, the role of the sequencer can be reduced. However, a sequencer is always needed in one form or another.

The sequencer executes sequences ; sequences are usually written in a computer programming language. As sequences need to be changed often, it is highly desirable to be able to write these sequences in an interpreted language.

The interpreted language for the sequencer should :

- be extensible (allows to interface existing libraries in C or C++)
- provide complex data types (tables, dictionaries, lists, sets...)
- have name spaces (avoid use of global space)
- offer debugging possibilities
- have the essential modern features of scripting languages nowadays : first-class functions, iterators, closures, OO capabilities, etc.

2.2.2 Writing sequences

In their day to day work, scientists use various computer tools, each one with their own logic and way of working, sometimes with their own programming language or macro facility. We can think of

gnuplot, Matlab, LabView, Igor... In the synchrotron world, we can add spec. Most of the time beamline scientists and users will only have a "working knowledge" of the basic tools they need to run their experiments. Some beamline scientists and users will even master real computer languages like Fortran or Python, but this clearly is not the common case. Not all scientists will write data acquisition and automation sequences, but it will happen quite often ; there are some really simple sequences, like repeating 10 times a scan with different parameters in a loop. The choice of the language for writing sequences is essential. What are the different approaches to the sequence writing problem ? Which language is best adapted for a sequencer ?

2.2.2.1 Graphical approach

The graphical approach consists of creating a sequence by assembling small code parts using a graphical tool. LabView is using this approach ; the Soleil synchrotron is also using the same approach with the Passerelle tool, on top of the Ptolemy project. The code is written by some software experts, then users only arrange new sequences around some predefined functional bits. The algorithm for the sequences is described in terms of I/O flow between the small components.

2.2.2.2 DSL approach

The domain specific language (DSL) approach is the one chosen in some specific tools like Spec. The application embeds a domain specific language ; it is designed to make common operations very easy to write since it includes whole concepts around basic control structures and syntax.

2.2.2.3 Computer scripting language approach

This approach consists of using a full-featured computer scripting language as the sequencer language. It opens a lot of opportunities and brings maximum flexibility, at the cost of a non-adapted syntax and some inherent concerns : since all scripting languages are designed to be very general since the ground up.

Depending on the targeted audience, the choice of one approach can appear to be better than another one. The graphical approach still looks like sci-fi : it is probably great for simple sequences, but it certainly becomes exponentially complicated for non-trivial sequences. However, it may be a good idea to experiment. The real scripting language approach is probably not a good choice, because it will force users to turn themselves into computer engineers and it will essentially become counter-efficient. Finally, the best approach could be to follow the DSL one (language + concepts), with an advanced underlying scripting language (not reinventing the wheel on this topic !).

2.2.3 Interacting with users

At ESRF, the sequencer is also the first access to beamline experiments offered to the user : spec,

through its CLI (Command Line Interface), allows commands and functions to be executed by taking input from the computer keyboard.

A more than 10 years experience has proved that this kind of direct-access interface, together with an interpreted language for sequences, is a really nice way to bring flexibility, to allow a fast and easy implementation and to offer a high-level workbench when testing new hardware on beamlines. Moreover, after a reasonably small learning period, any user can drive her own experiment by typing or executing herself dedicated commands and functions from the sequencer CLI. The command line interface has to be enough to drive any experiment.

Several kind of users are using a beamline and need to have access to the sequencer :

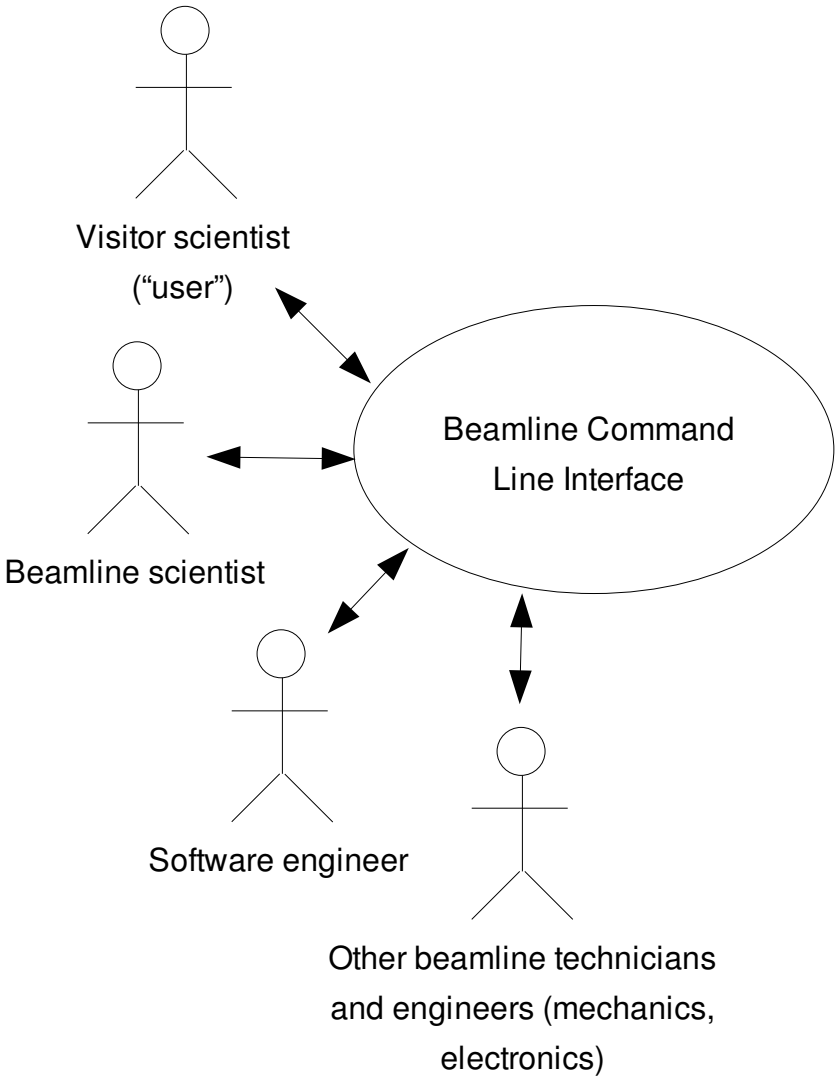

Fig. 1 : Different kind of users are interacting with the sequencer to control the beamline

Each kind of user brings its own culture ; the sequencer, through its command line interface, should allow each kind of user to be efficient with the tool very quickly. This can be achieved by coupling a simple graphical interface together with the command line prompt : nowadays, everyone operating on a beamline is used to computer graphical interfaces. Ergonomics should be studied carefully in order to produce a nice user-friendly tool, that would really help users of the sequencer. It has to be fast and to provide obvious access to all basic tasks, while allowing immediate handling for more experienced users.

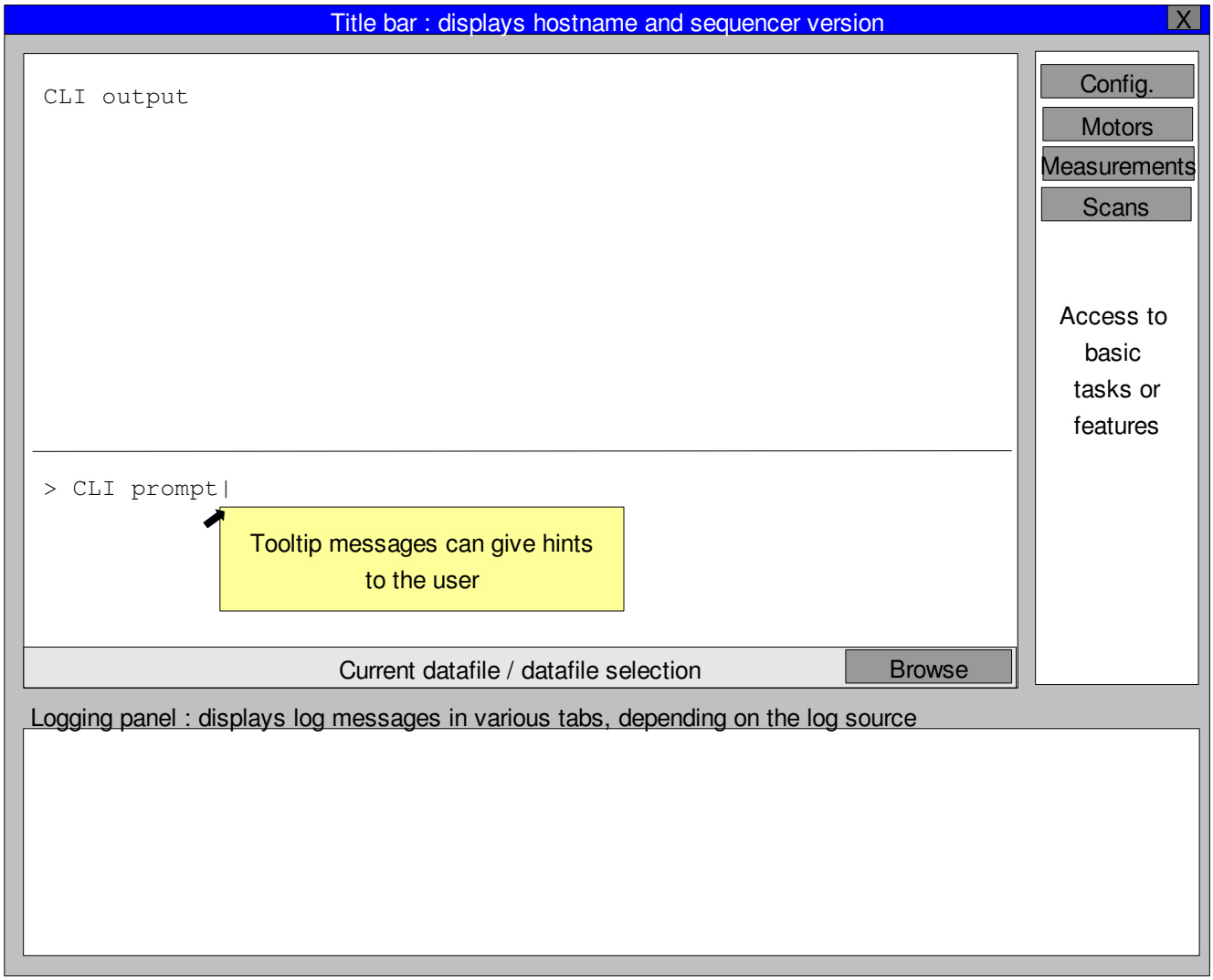

Fig. 2 : Example sequencer user interface

A full-featured integrated beamline software platform has to provide a sequencer, and through a CLI it should also play the role of the first interface for the beamline user (scientist, technician, software engineer...).

2.2.4 Architecture of a sequencer

KISS principle (Keep It Simple, Stupid)

Modularity

Client / Server ?

“Quality Insurance” : limit the amount of OS threads to the minimum (like spec...), use sandboxing to protect the sequencer core, deal with a limited amount of dependencies, etc.

2.3 Configuration

Any software system needs configuration. In the case of the integrated beamline software platform, configuration is essential – ideally, it would encapsulate all configuration needs for the different software components in a centralized way.

For the middleware, configuration indicates how to address hardware devices, i.e which kind of protocol, and where : serial line port number, IP address, GPIB address, internal PCI card address, etc. and contains all parameters that needs to be set on the hardware to make it work as expected.
For the sequencer, configuration represents the whole set of sequences and associated middleware devices ; one of the required feature is to be able to save “configuration snapshots” and to be able to retrieve them later, to allow users to save whole experiments setup with associated data acquisition objects.

Configuration has to be stored in a “database”. The “database” is not necessarily a SQL-compliant relational database like Oracle or MySQL. Any structured way of storing parameters is a potential candidate.

Some important criteria are :

- flexibility : the database should handle new kind of hardware easily, naturally ; copying configuration from one beamline to another must be straightforward
- reliability : the database should be protected from software crashes
- configuration parameters should be able to be changed very easily with a convenient editor
- browsing configuration should be easy to do
- searching capabilities

Experience has proved that a bunch of text files in a human-readable format is one of the best way to store configuration information. Classic Unix tools like “grep” are enough to be able to search through a whole set of files. Browsing is easy to do with “ls”. Several files are immune to software crashes (if

one file get corrupted, it does not corrupt all files). A simple text editor is enough to edit text files quickly and efficiently. Copying configuration from one beamline to another is very easy with standard tools as well – replacing text in several files is also easy to do to adapt from one beamline to another.

In the case of a binary format to store configuration information, some tools need to be shipped with the configuration database. It is very important to keep in mind that the tools have to be very powerful and ergonomic to compete with text files. One can reasonably think that a very good graphical editor for configuration will probably be better than a simple text editor, however this is clearly not always true in practice.

About the interaction between the configuration and the whole integrated beamline software platform, it should be foreseen to be able to disable a device, to re-enable it again etc. in a transparent manner, without losing any configuration information. Having a history is also a highly desirable feature : the ability to take configuration snapshots and to be able to restore them later is very convenient to have.

2.4 Graphical interfaces in an integrated beamline environment

The need for graphical interfaces can be characterized in 3 main domains on beamlines :

- to **monitor** beamline components, or physical quantities coming from sensors
- to **run experiments**
- to **analyze** data

The integrated beamline software platform should provide standard tools to build graphical interfaces and applications to fulfill these needs.

2.4.1 Monitoring

More than often, scientists want to have an overview of the state of beamline components at a glance. How much is the machine current ? Is it refilling ? Is the beamline safety shutter open ? Are motors moving ? Has the intensity read by a diode changed ? What is the temperature of my sample ? What is happening inside the experiment hutch ?

Complementary to that, they want to be able to access beamline components : to open shutters, to move motors, to change gains for a diode, to change cryo stream flux, etc.

A graphical interface is usually the more convenient way to provide both monitoring and basic controlling facilities to users. On MX beamlines at ESRF, two applications serve these purposes : the synopsis and the control panel. The synopsis shows position and state for all beamline motors, the synchrotron machine current, the state of shutters and two graphs with a 1 hour history, displaying counts for some diodes. The control panel presents more or less the same components, allowing users to do actions like moving a motor, opening/closing a shutter etc.

Whenever it is possible, there is a direct access between the graphical application and the middleware, to get access to the hardware. It removes layers, and gives the best performance. There is no need to pass by the sequencer : most of the time, there is no sequence to be executed in a monitoring application. As a corollary, graphical applications do not run sequences on their own, such tasks are delegated to the sequencer. Of course, there are exceptions ; that's why graphical interfaces of an integrated beamline software platform should be able to communicate transparently with one system or another, and stress should be put in clearly separating the “display” part from the “control” part.

2.4.2 Running experiments

A scientific experiment at a synchrotron beamline involves a lot of actuators, motors, sensors and detectors, to go from the sample(s) to data. Most of the time, experiments are changing from one run to another ; in rare situations, like on MX beamlines, the experiment is always the same.

A graphical interface to run experiments needs to be very flexible, as it will change as often as the experiment will change. The integrated beamline software platform should include a tool to be able to quickly generate graphical interfaces for experiments, and to easily modify existing ones.

Panels are needed for input parameters. Input parameters can be complex : for example, an user might want to click on a video display to specify coordinates for a scan.

2.4.3 Analyzing data

...

3. Conclusion