NOMAD – MORE THAN A SIMPLE SEQUENCER

P. Mutti, F. Cecillon, A. Elaazzouzi, Y. Le Goc, J. Locatelli, H. Ortiz, J. Ratel,
Institut Laue-Langevin, Grenoble, France

Abstract

NOMAD is the new instrument control software of the Institut Laue-Langevin (ILL). A highly sharable code among all the instruments' suite, a user oriented design for tailored functionality and the improvement of the instrument team's autonomy thanks to a uniform and ergonomic user interface are the essential elements guiding the software development. NOMAD implements a client/server approach. The server is the core business containing all the instrument methods and the hardware drivers, while the GUI provides all the necessary functionalities for the interaction between user and hardware. All instruments share the same executable while a set of XML configuration files adapts hardware needs and instrument methods to the specific experimental setup. Thanks to a complete graphical representation of experimental sequences, NOMAD provides an overview of past, present and future operations. Users have the freedom to build their own specific workflows using intuitive drag-and-drop technique. A complete drivers' database to connect and control all possible instrument components has been created, simplifying the inclusion of a new piece of equipment for an experiment. A web application makes available outside the ILL all the relevant information on the status of the experiment. A set of scientific methods facilitates the interaction between users and hardware giving access to instrument control and to complex operations within just one click on the interface.

INTRODUCTION

The Instrument Control Service (SCI) in close collaboration with the ILL's scientists has taken up the challenge to redefine the way experiments are performed. The initial part of the project has been dedicated to an exciting discussion with our scientists and users to define all the use-cases to be included in NOMAD. Such a discussion brought the attention on the following use-case:

UC1: *hardware installation*. Hardware added to the instrument needs to be incorporated into the software control. We have detailed how a new electronic device can be added dynamically to the control software.

UC2: *instrument component setup*. Once a new hardware is added it needs to be configured. This use-case takes into account the different roles that hardware components can have on the instrument control.

UC3: *instrument calibration*. After the hardware is correctly installed and configured, it needs to be calibrated to ensure a correct functioning of the instrument before performing an experiment. This may involve proper PID settings, detector calibration, etc…

UC4: *sample alignment*. In certain cases, a sample needs to be aligned. This is most commonly performed on diffraction-type instruments in which the diffraction pattern depends upon the orientation of the sample with respect to the neutron beam. In other types of instruments this alignment is sometimes known in advance but the instrument still needs to be setup correctly depending on the type of experiment being performed.

UC5: *performing the experiment*. This use-case describes the general steps involved in the execution of an experiment. It starts from the verification of the status of the instrument to continue with the execution of pre-defined list of commands. At the end of a well-defined sequence, data are stored and a detailed report of all the various steps undergone is produced.

All those different use-cases lead to the implementation presented in the current paper.

ARCHITECTURE

The Nomad application is a two-tier client server application [1]. Figure 1 shows a simplified component diagram of the application

Server

The Nomad Server application is the part of the program executing the main tasks required for the control of the instruments. The server written in C++ [2] has three main roles:

- Data provider: to provide the actual values of the variables of the system (e.g. the properties of the instrument).
- Plug-in container: to load, initialize and execute the business logic of the instrument. The state of the instrument is periodically updated by the reading of the connected devices. The business layer is organized into plug-ins separated into drivers (low level layer) and controllers (high-level layer).
- Command sequencer: to process the requests of the client. Those can be complex sequences of instructions onto controllers. A client user can launch batch processing by programming a combination of for loops with parallel execution of commands for the night.

The plug-in libraries are divided into controllers and drivers. These are dynamic libraries loaded by the server. They contain all the business code written for NOMAD including all the available drivers and controllers. Some controllers can be common to each instrument, but some controllers are instrument specific. We also have different levels of abstraction for controllers.

The controller and driver classes loaded by the plug-in libraries describe an object-oriented model of classes. Controller classes interact with controller or driver

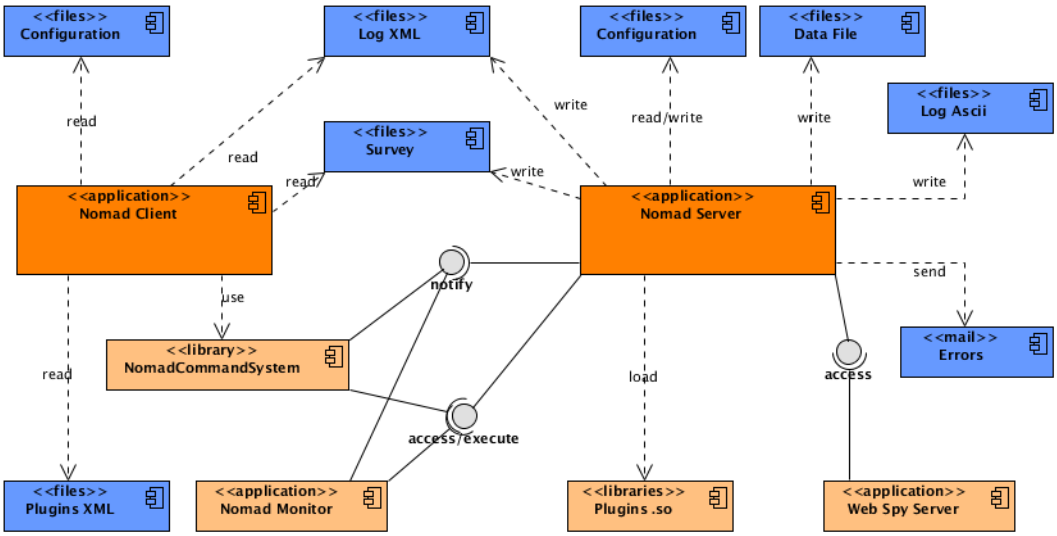

Figure 1: Simplified component diagram for the NOMAD application.

interfaces. We need to describe how we link the different controller and driver objects. These configurations are specific to each instrument. Two main configuration files, the *InstrumentConfig* and *HardwareConfig*, list the controllers and drivers required for a given instrument and how they are linked together. Additional instrument specific configuration files describes acquisition settings or scheduler rules indicating which controllers can execute tasks in parallel and which should be strictly sequential.

The results of the data acquisition are stored into several different file formats. ASCII data files contain all the relevant information organized in a text format following specific rules to be compatible with the existing ILL's data treatment programs. NeXus [3] files contain binary and compressed data. Nexus is a common data format share by neutron, x-ray and muon facilities. It is based on the Hierarchical Data Format (HDF) developed by U.S laboratories and extremely powerful for large data sets. LIST-MODE files are pure binary containing, beside a specific ILL header, the sequence of all detected events and the associate time-stamp to provide a maximum of flexibility to replay the experiment off-line.

NOMAD records all important instrument history information in daily log files. The ASCII version is directly accessible by the user via his favourite text editor. The server includes in this file some additional debugging information that can be helpful for the developers in case of unexpected behaviour. NOMAD is also saving a XML version of the log file. This can be accessed by the NOMAD client application and provides additional features to the user like filtering on specific actions and calendar navigation.

All instrument parameters are permanently monitored and stored into daily Survey XML files. Those data are displayed by the NOMAD client application in a dedicated plot window and the user is able to dynamically select and visualise a specific set of system's variable.

The server automatically notifies to the administrator or to the users all errors occurred during the execution.

Client

The Nomad Client application written in Java [4] SWT is designed to offer a practical Graphical User Interface (GUI) for the user to control the instrument. A dedicated effort has been made to provide an easy-to-use graphical programming of complex sequences of instructions. The NOMAD client application consists of three different graphical views specifically designed to satisfy different needs. The *Hardware* view is intended for the technical support and gives access to all hardware specific parameters like displacement limits, positioning speed, etc… The *Setting* view is mainly devoted to instrument responsible and allows configuring higher-level controllers. Finally the *Launch-Pad* view permits the creation and the execution of a specific workflow.

The *NomadCommandSystem* library is a fundamental component of the client architecture. It is designed as a bridge between the client and the server. It abstracts the way to request the server and it mainly uses Corba proxy objects to communicate with server. It implements some Corba IDL interfaces since it also acts as a server from the Corba point of view.

To minimise the coding work we have designed an XML file format to describe controllers and drivers' GUI without writing any Java SWT code. However, some specific GUI behaviours that cannot be obtained by XML description are directly coded in Java SWT. Some additional configurations files are required by the client application to start. They concern for example label text properties.

Nomad Monitor

We have designed a NOMAD Monitor application written in Java RCP to visualise the actual status of the NOMAD Server system. The application offers the list of the controllers and drivers showing all their property and

the associated values. Controller and driver objects are loaded by requesting the Nomad Server. Nomad Monitor enables debugging controller and driver execution and acts also a configuration manager giving the possibility to dynamically add a new controller or a new driver while the server is running.

Web Spy Server

To provide the user with a maximum of information to promptly react in case of a problem or simply to remotely verify the status of the measurement we have developed a web server that retrieves periodically from NOMAD Server all the actual values of properties. An instrument specific web page is then updated to provide all important information on the instrument (see Fig. 2).

Figure 2: Layout of the web spy for the instrument IN4.

INTER-APPLICATION COMMUNICATION

The CORBA [5] architecture has been chosen to make the connection between the C++ server and the Java client. The CORBA standard offers a flexible interoperability service through the definition of IDL interfaces and its implementations provide good performances in regard of other inter-application communication like, for instance, Web services. However we made the choice to minimize the dependency to CORBA by avoiding the number of CORBA objects for which life cycle is not so easy to manage in C++ and leave them only to the communication layer. For this reason, all our CORBA objects are wrapped into Adapter objects (client and server) that we call *Accessors*. Server and clients use a two-way communication for:

- access/execute: the clients request the server to get or set a value, to execute tasks, etc…
- provider: to provide the actual values of the variables of the system (e.g. the properties of the instrument)
- notify: the server notifies the client when a server variable has changed.

For the client to server requests (on a user request), the client contacts directly the server through the CORBA *Accessor* objects. The server to client request uses the *Observer* pattern to implement an event-handling system.

NOMAD server registers CORBA subscriber objects to dispatch events to them. The server and the clients both use a Producer/Consumer pattern for respectively send and process the event, resulting in an asynchronous event notification. Figure 3 shows a simplified example of server to client forward and backward communication. The Access Data model illustrate the different patterns, synchronous and asynchronous messages from the notification of the server of a value changed to the call of the client for getting the value.

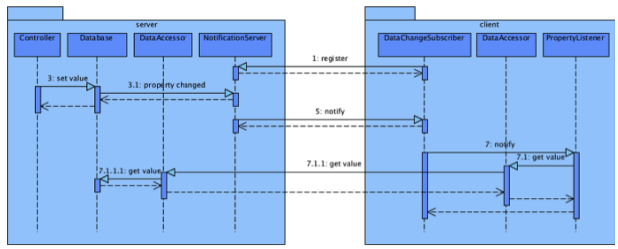

Figure 3: Data Access sequence.

The distributed Publisher/Subscriber objects when the client and the server communicate over a network offers great advantages to refresh the client only when needed and prevent from using client polling, but one drawback is that we can face client firewall port denials.

THE NOMAD SERVER LAYERS

The NOMAD server application is a multi-threaded application designed in three distinct layers.

- Data Provider is the part of the server that stores the data model, e.g. the hierarchy of controllers and drivers with their associated properties. We also store the list of commands accessible from each controller and driver and their current state. This is the data representation of the controllers and drivers running in NOMAD server for a specific instrument (loaded from the configuration files).
- Command can be considered as the business logic layer. It is responsible to execute the requested commands to the controllers and drivers. Each controller or driver accesses its property data from the Data Provider layer. We have designed a framework of classes for controller and drivers to facilitate the business code integration. We have chosen to implement an event-driven programming [6] model to react easily to the real instrument dynamic behaviour (listening to sensors). *AbstractController* is the abstract class to specialize. It contains access methods to associated properties that are bridges to the Data Provider data model. It implements the methods *execute*, *refreshSetValue*, *updateProperty*. The method *execute* contains the execution code of the different registered commands. The most important command is *start*. The *refreshSetValue* is the function implementing how the controller reacts when one of its properties change. This is the way to spread information from the high-level layer (controller) to the low-level layer (driver). The *updateProperty* is the function

describing how the controller reacts to the change of a property of one of its linked controllers or drivers. This is the way to spread information from the low-level layer to the high-level layer. NOMAD has at present about 250 controllers (from the most general to the most instrument specific) and 150 drivers available.

Error management controls the way NOMAD server reacts to the real behaviour of the instrument that can be seen as a complex system depending on many uncontrolled parameters. During the development phase we have rarely access to the real instrument therefore, all the tests performed in the lab cannot be exhaustive. To limit error's occurrence we run logical unitary and functional tests with a basic simulation of the system. A second series of tests aim the verification of the consistency of all instrument configurations. Unexpected behaviours can lead to exceptions, reported in the log files, or to crashes. The last are notified by mail to the developer's team together with the maximum of available information to allow further analysis of the problem. Errors connected to defective behaviour of the instrument generate a series of alarms or warnings that are notify by mail to the instrument responsible.

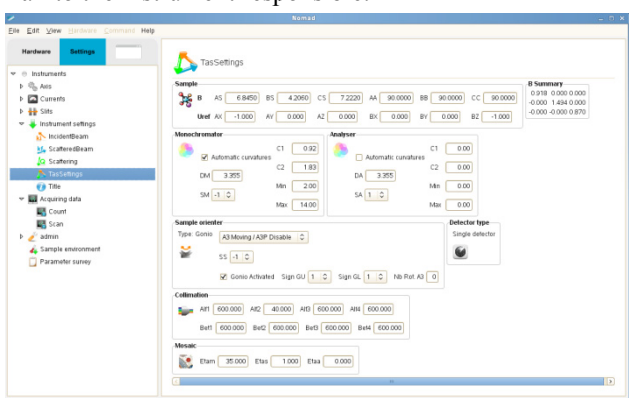

Figure 4: An example of abstract controller.

INSTRUMENT ABSTRACTION

Abstract controllers are used to hide from users the specific underlying hardware or control sequences. It provides an interface relevant to a specific function and then passes user requests to the more specific instances.

The main purpose of the instrument control software renewal is to free human resources and re-uses them to integrate prototyped versions of scientific methods into NOMAD. The main objective is to perform faster and more accurate measurement through intelligent scientific methods and thus increase the efficiency of instruments.

Handling instrument methods complexity, instruments control methods appears at first glance as complex and of large variety. The scientific analysis shows that instrument and scientific methods can be well organized into common elements with specific parameterization.

They are organized in term of primary and secondary spectrometer, sample orienteer, basic scattering and orientation strategies. Presently, the design of the scientific methods shows that only a few set of objects are actually needed. The present complexity and variety comes from the fact that mathematical models are merged into single routines, rigidifying the code. As well, science groups have different notations referring to the same scientific concepts. These will be handled by the GUI, letting the physics concepts behind untouched. Fig. 4 depicts and example of abstract controller that regroup in one single interface all the quantities needed to properly initialise a three-axis spectrometer and prepare it for the experiment. Complex instrument's setup but also alignment and acquisition sequences have been optimized and specifically coded in NOMAD to get the best out of the available beam-time and to facilitate users' tasks.

CONCLUSION

After an initial phase of heavy debugging, NOMAD is meanwhile installed on all ILL's instruments for motors and general hardware setup. So far, more than 20 instruments are fully controlled with it and more are added on a regular base. In addition, 10 installations have been performed to drive technical equipments within the Technical and Project Department. NOMAD includes meanwhile all the hardware devices present at the ILL and all the generic sample environment modules as well as all the instruments' specific ones. It provides the users the freedom to configure dynamically the equipment needed for a specific experiment. The full integration between sequencer, hardware devices and sample environment allows a much better event handling and full synchronization. As well, physical quantities, like energies or wavelength, are compounds of several hardware components. The control of the instrument directly through physical values, those who have a meaning for the user, implied a new way of managing interconnected hardware pieces. Giving control at a more physical than hardware level is in the spirit of opening our instruments suite to a broader audience.

REFERENCES

[1] G. Reese, "Database Programming with JDBC and Java" (2009); htth:// java.sun.com/developers/Books/jdbc.
[2] B. Stroustrup, "The C++ Programming Language", Addison-Wesley Professional (2000).
[3] http://www.nexusformat.org.
[4] K. Arnold, J. Gosling and D. Holmes, "The Java Programming Language", 3rd Edition Addison-Wesley Professional (2000).
[5] S. Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments", IEEE Communications Magazine, Vol. 14, No. 2 (1997).
[6] S. Ferg, "Event-Driven Programming: Introduction, Tutorial, History" (2006); http://Tutorial _EventDrivenProgramming.sourceforge.net