NEW APPROACH IN DEVELOPING OPEN XAL APPLICATIONS

C. Rosati[*] and E. Laface, European Spallation Source ERIC, Lund, Sweden

Abstract

Open XAL project is a pure-Java open source development environment used for creating accelerator physics applications, scripts and services.

Working with Open XAL requires developing a Java application with a prominent graphical user interface, allowing the final user to interact with the accelerator model, and to graphically view the results such interaction produced. Nevertheless the Open XAL support for specialized components[1] and for a document-view application framework[2], a lot of boilerplate code has still to be created, making the developer spending more time in UI than in accelerator physics code.

In this paper a new approach in developing Open XAL applications is explained. Here the developer is relieved of the UI-related common code by using software tools, allowing him to visually design the flow of data and events between the various elements of the applications (widgets and models), and automatically generate the application code, where code generation can be customized to use one of the available plugged languages (Java, Python, JS, . . .).

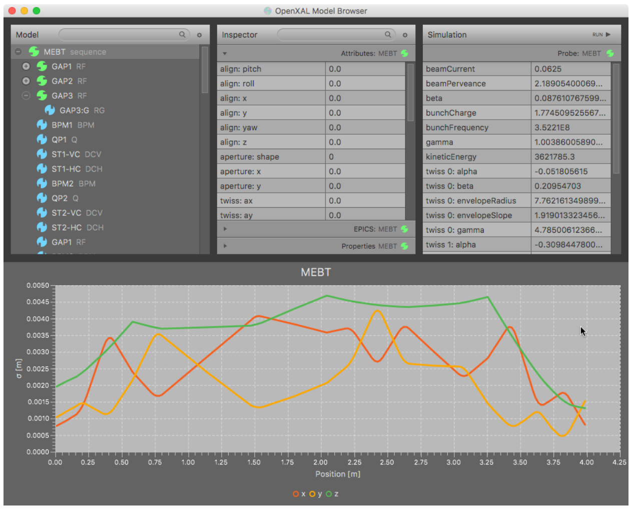
INTRODUCTION

Open XAL [1, 2], is an open source accelerator physics software platform written in Java and used for creating accelerator physics applications, scripts and services. The Open XAL project began in mid 2010 as a response to requests from the international accelerator physics community to adopt an open source accelerator physics platform based on XAL [3] from the Spallation Neutron Source (SNS) at Oak Ridge National Lab (ORNL) and establish a standard platform for accelerator physics software.

Writing a modern client application with a prominent graphical user interface (GUI, see for example Figure 1) requires writing a lot of code to exclusively handle the user interface. For example, the *Open XAL Model Browser* application is made of 950 lines of Java/JavaFX code (not counting blank and comment lines [4]). Of these, only 11 (**1% of the total code**) are "pure" Open XAL lines of code, used to run the ESS Linac Simulator (ELS) model [5].

It is our belief that a new tool for building Open XAL graphical client applications can be created, to automate the production of UI-related common code and to clearly define the boundaries between the GUI and the Open XAL development. Moreover, although this article makes specific reference to Java and JavaFX, we also believe possible to extend what herein described to other programming languages, e.g. Jython/Python, JS, C/C++, . . .

[*] Claudio.Rosati@esss.se
[1] Handling plotting, EPICS connection, etc.
[2] Relieving the developer of the burden related with this programming aspects.



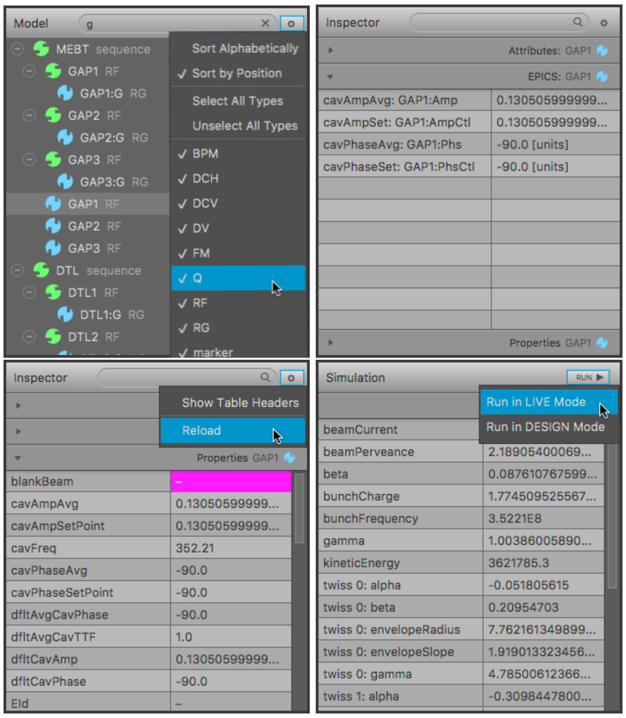
align: pitch	0.0
align: roll	0.0
align: x	0.0
align: y	0.0
align: yaw	0.0
align: z	0.0
aperture: shape	0
aperture: x	0.0
aperture: y	0.0
twiss: ax	0.0
twiss: ay	0.0

beamCurrent	0.0625
beamPerveance	2.18905400069...
beta	0.087610767599...
bunchCharge	1.774509525567...
bunchFrequency	3.5221E8
gamma	1.00386005890...
kineticEnergy	3621785.3
twiss 0: alpha	-0.051805615
twiss 0: beta	0.20954703
twiss 0: envelopeRadius	7.762161349899...
twiss 0: envelopeSlope	1.919013323456...
twiss 0: gamma	4.78500612366...
twiss 1: alpha	-0.3098447800...

Figure 1: Open XAL Model Browser displaying an ELS simulation.

OPEN XAL MODEL BROWSER

To prove our new approach we decided to develop a new client application (see Figure 1) using the traditional workflow, the Java programming language [6, 7], and the JavaFX GUI framework [7–10]. The application allows the user to browse the accelerator model, look at the selected node attributes, real-time EPICS values, and introspected prop-



cavAmpAvg: GAP1:Amp	0.130505999999...
cavAmpSet: GAP1:AmpCtl	0.130505999999...
cavPhaseAvg: GAP1:Phs	-90.0 [units]
cavPhaseSet: GAP1:PhsCtl	-90.0 [units]

blankBeam	–
cavAmpAvg	0.13050599999...
cavAmpSetPoint	0.13050599999...
cavFreq	352.21
cavPhaseAvg	-90.0
cavPhaseSetPoint	-90.0
dfltAvgCavPhase	-90.0
dfltAvgCavTTF	1.0
dfltCavAmp	0.13050599999...
dfltCavPhase	-90.0
Eld	–

beamCurrent	
beamPerveance	2.18905400069...
beta	0.087610767599...
bunchCharge	1.774509525567...
bunchFrequency	3.5221E8
gamma	1.00386005890...
kineticEnergy	3621785.3
twiss 0: alpha	-0.051805615
twiss 0: beta	0.20954703
twiss 0: envelopeRadius	7.762161349899...
twiss 0: envelopeSlope	1.919013323456...
twiss 0: gamma	4.78500612366...
twiss 1: alpha	-0.3098447800...

Figure 2: Top panels.

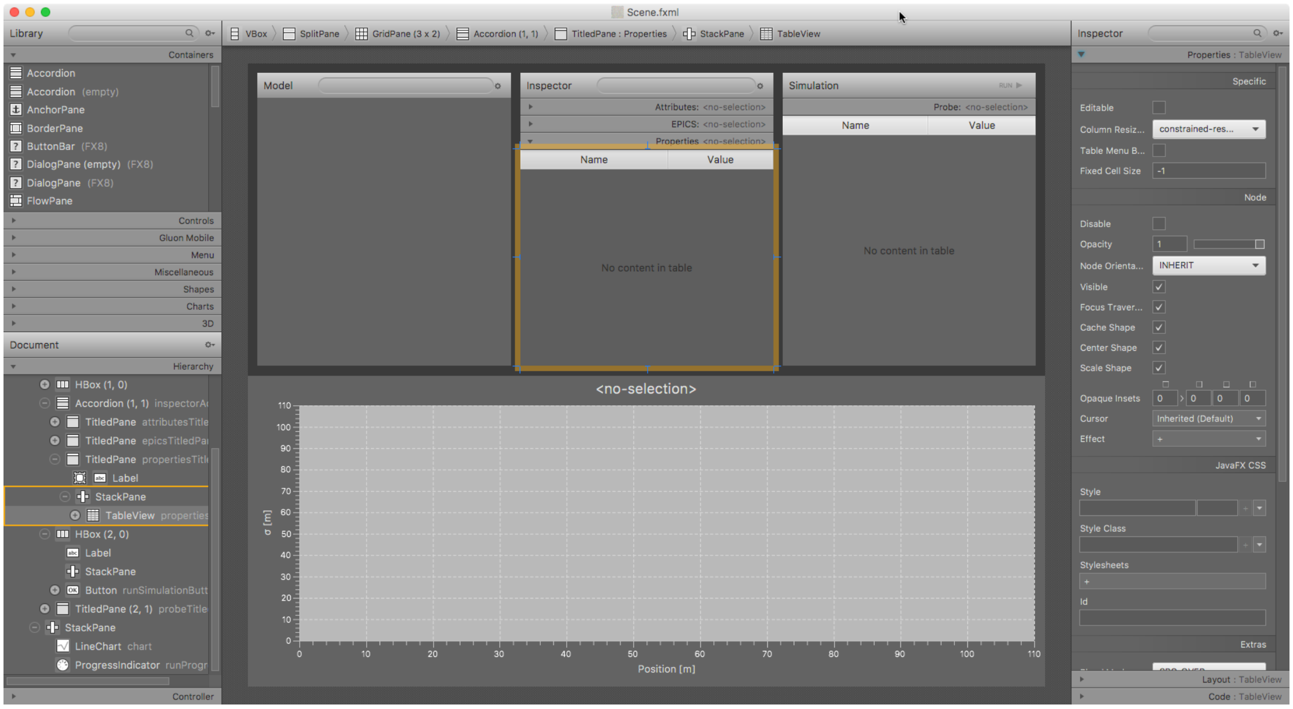


Figure 3: Scene Builder used to edit *Open XAL Model Browser* GUI.

erties. Moreover, if a top-level node is selected, the node probe can also be inspected and a simulation run (see Figure 2).

The model tree can be configured to show only certain nodes and/or nodes whose identifier matches a given search string. The inspector content can also be filtered in real-time, and table headings can be made visible to alter the columns size. As soon as the simulation run has completed, its results are displayed into the X-Y chart at the bottom.

The application GUI was built using *Scene Builder* [11] (see Figure 3), the standard open-source application provided by the JavaFX team[3]. The program code (10 classes, 950 lines of code, 258 lines of comments, 373 blank lines, [4]) was written in the Java programming language [6] using the NetBeans Integrated Development Environment (IDE, [14]).

THE NEW APPROACH

The new approach reckon on a graphical tool, *Open XAL Modeller*, that will relieve the Open XAL programmer from writing most (if not all) the UI code of a client application.

Designing the User Interface

The JavaFX team created a tool to facilitate building user interfaces based on the new Java UI Toolkit technology: Scene Builder [11]. In its "Getting Started" document [15] the following is written:

> JavaFX Scene Builder provides a visual layout environment that lets you quickly design user interfaces (UI) for JavaFX applications without needing to write any code. [...] JavaFX Scene Builder provides a simple yet intuitive interface that can help even non-programmers to quickly prototype interactive applications that connect GUI components to the application logic.

[3] Now hosted by Gluon [12, 13].

The approach followed by the JavaFX team defines clear boundaries between whom designs the user interface and who will code the application logic behind that UI, the latter being, in our vision, the Open XAL Modeller tool.

The Scene Builder application demonstrated to be very well made and easy to use. Moreover, the possibility of integrate it into a wider application [16] makes Scene Builder the perfect choice as the framework for building the graphical user interface of an Open XAL client applications.

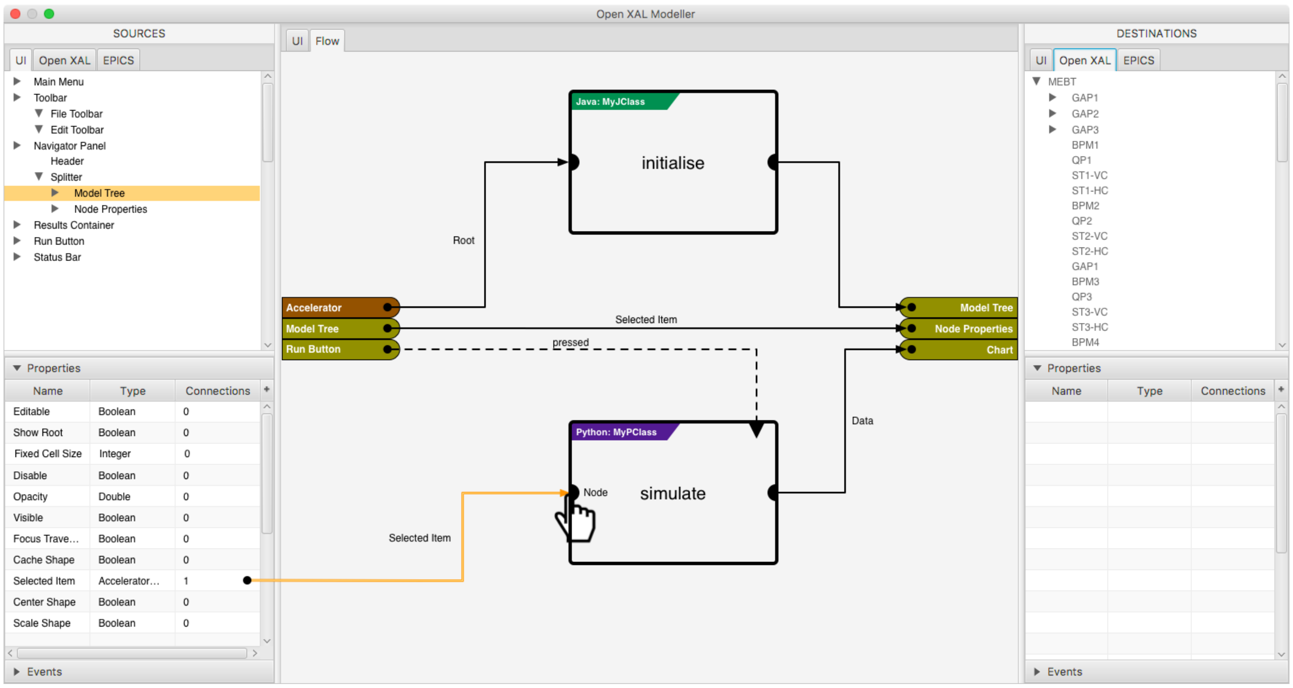
Thanks to the Scene Builder kit *Application Programming Interface* (API, [16]) every parts of the Scene Builder application (see Figure 3) can be embedded into another application (the Open XAL Modeller tool, in our case), making possible to edit the user interface and contemporarily accessing its components structure and properties: a perfect integration.

Designing the Application Logic

To connect the graphical user interface with the application logic, a visual programming tool will allow to define the bindings between JavaFX properties [17, 18], and to use events [19] to trigger execution of code (see Figure 4).

Inside this editor the Open XAL programmer will be able to:

- graphically bind JavaFX properties, dragging an item from the source *Properties* table into another item of the



Name	Type	Connections
Editable	Boolean	0
Show Root	Boolean	0
Fixed Cell Size	Integer	0
Disable	Boolean	0
Opacity	Double	0
Visible	Boolean	0
Focus Trave...	Boolean	0
Cache Shape	Boolean	0
Selected Item	Accelerator...	1
Center Shape	Boolean	0
Scale Shape	Boolean	0

Figure 4: Flow editor used to design the application logic.

destination one. The tool will automatically create the code necessary to convert data into a different data type if necessary (see *Selected Item* connection in Figure 4), or insert a *code node* where to manually implement the type mapping using the preferred programming language (Java, C/C++, Jython, Python, ...). Filter code nodes can also be added on request to narrow the bound data (see *initialise* node in Figure 4);

- add code nodes bound to JavaFX properties and events, exporting new properties whose values are computed when input data change and/or events are triggered.

Once the execution flow is designed, the tool will be able to create the full set of project files[4](Java, Python, C/C++ classes, `fxml` and other resource files).

Classes will be generated in pair: a read-only abstract superclass containing all of the boilerplate code and the abstract definition of methods corresponding to code nodes, and a subclass to be filled by the Open XAL programmer, containing the implementation of the abstract methods. Subclasses are automatically updated only to add new methods corresponding to new code nodes. Superclasses are, instead, automatically updated when the user interface or the execution flow changes.

When non-Java code nodes are used, the Java superclass will be created with a concrete definition of the method corresponding to the code node, where the statements to properly call the *foreign language* counterpart will be included.

A specific NetBeans IDE plugin [14, 23, 24] could be developed to facilitate jumping back and forward between the IDE[5]and Open XAL Modeller, improving even further the development experience.

CONCLUSIONS

Developing Open XAL client graphical applications requires coding the graphical user interface and the interaction with the application logic, resulting in a program where coding is mostly focused on the UI and the relative common programming statements, rather than on the accelerator physics application logic itself. For example, the size of the application logic of the program used as a reference for this paper is only the 1% of the total code lines.

We at the European Spallation Source believe it possible to develop a tool capable to automatically create most of the user interface boilerplate code from

- the `fxml` file(s) describing the graphical user interface(s) created through the JavaFX Scene Builder application [11–13, 15],
- the execution flow graph (see Figure 4) depicting the binding relationships between the application components.

The Open XAL Modeller tool we intend to build will be a module of a larger pure Java/JavaFX application based on *Drombler FX* [25], an open source modular application framework for JavaFX, and will be the first of a series of new tools [26] aimed to modernize and improve the stand-alone control system software used at European Spallation Source by the Integrated Control System group.

[4] The project itself being created through a specific Maven archetype [20], or a custom built ant task [21, 22].

[5] Used to compile, debug, test and deploy the client application.

REFERENCES

[1] OpenXAL Website. [Online]. Available: https://github.com/openxal/openxal

[2] A. Zhukov and C. K. Allen, "Open XAL Status Report 2017," in *This Conference*, no. THPVA093, 2017.

[3] J. Galambos *et al.*, "XAL application programming structure," in *Particle Accelerator Conference, 2005. PAC 2005. Proceedings of the*. IEEE, 2005, pp. 79–83.

[4] CLOC: Count Lines of Code Website. [Online]. Available: https://github.com/AlDanial/cloc

[5] E. Laface *et al.*, "The ESS Linac Simulator: a first benchmark with TraceWin," *Proceedings of IPAC*, 2013. [Online]. Available: http://accelconf.web.cern.ch/accelconf/ipac2013/papers/tupwo046.pdf

[6] J. Gosling *et al.* (2015, 2) The Java® Language Specification. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se8/html/index.html

[7] Java Platform, Standard Edition (Java SE) 8, Client Technologies Website. [Online]. Available: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm

[8] H. Schildt, *Introducing JavaFX^{TM} 8 Programming*, 1st ed. McGraw-Hill Education, 2015.

[9] J. Vos *et al.*, *Pro JavaFX 8: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients*, 1st ed. Apress Media, 2014.

[10] H. Ebbers, *Mastering JavaFX® 8 Controls*, 1st ed. McGraw-Hill Education, 2014.

[11] JavaFX Scene Builder: User Guide. [Online]. Available: http://docs.oracle.com/javase/8/scene-builder-2/user-guide/index.html

[12] Scene Builder Website. [Online]. Available: http://gluonhq.com/products/scene-builder/

[13] Scene Builder Documentation. [Online]. Available: http://docs.gluonhq.com/scenebuilder/

[14] NetBeans IDE Website. [Online]. Available: https://netbeans.org

[15] JavaFX Scene Builder: Getting Started with JavaFX Scene Builder. [Online]. Available: http://docs.oracle.com/javase/8/scene-builder-2/get-started-tutorial/index.html

[16] Scene Builder Kit and 64-bit Scene Builder Website. [Online]. Available: http://gluonhq.com/scene-builder-kit-and-64-bit-scene-builder/

[17] JavaFX: Properties and Binding Tutorial. [Online]. Available: https://docs.oracle.com/javase/8/javafx/properties-binding-tutorial/

[18] JavaFX Property Architecture. [Online]. Available: https://wiki.openjdk.java.net/display/OpenJFX/JavaFX+Property+Architecture

[19] JavaFX: Handling Events. [Online]. Available: https://docs.oracle.com/javase/8/javafx/events-tutorial/index.html

[20] Introduction to Archetypes. [Online]. Available: https://maven.apache.org/guides/introduction/introduction-to-archetypes.html

[21] Developing with Apache Ant. [Online]. Available: https://ant.apache.org/manual/develop.html

[22] Tutorial: Writing Tasks. [Online]. Available: https://ant.apache.org/manual/tutorial-writing-tasks.html

[23] H. Böck, *The Definitive Guide to NetBeans^{TM} Platform 7*, 1st ed. Apress Media, 2011.

[24] J. Wexbridge and W. Nyland, *NetBeans Platform for Beginners*, 1st ed. Leanpub, 2014.

[25] Drombler FX Website. [Online]. Available: http://www.drombler.org/drombler-fx/

[26] C. Rosati and K. Shroff, "JavaFX and CS-Studio: Benefits and Disadvantages in Developing the Next Generation of Control System Software," in *16th International Conference on Accelerator and Large Experimental Physics Control Systems, 2017. ICALEPCS 2017. Proceedings of the*, submitted.