ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA180

VISUALISATION OF REAL-TIME FRONT-END SOFTWARE ARCHITECTURE (FESA) DEVELOPMENTS AT CERN

A. Topaloudis, CERN, Geneva, Switzerland
C. Rachex, Polytech Grenoble, Saint Martin d'Hères, France

Abstract

The Front-End Software Architecture (FESA) framework is the basis for most real-time software development for accelerator control at CERN. FESA designs are defined in an XML document which is validated against a schema to enforce framework constraints, and are used to automatically generate C++ boilerplate code in which the developer can then implement specific code. Design files can rapidly grow in complexity making the overview of the resulting system almost impossible to understand. One way to overcome this is to benefit from a graph-based representation of the design, with XML fragments summarized into logical blocks and association between the blocks depicted by arrows. As the intricacy of the graph is analogous to a potential complex design, it is also essential to provide an interactive Graphical User Interface (GUI) for parameterising and editing the graph generation in order to fine-tune a simpler and cleaner illustration of a FESA design. This paper describes such a GUI (FESA Graph Editor) and outlines how it benefits the design and documentation process of the FESA-design-document.

INTRODUCTION

The control system of the accelerator complex at CERN can be divided in three physical layers and can therefore be described as having a 3-tier architecture. The top tier consists of dedicated computers for running operational and expert high-level client applications, while the middle tier consists of servers that implement business and supervision logic. The lower tier is composed of embedded front-end computers (FECs) running real-time software, to control and monitor the accelerator equipment [1].

Software in the lower tier is developed using the FESA framework in order to standardise, speed-up and simplify the software development process [2]. FESA is a complete environment where developers model their software according to the framework's standards, which results in generated C++ boilerplate code. This generated code includes the necessary real-time scheduling classes and sophisticated mechanisms to ensure data consistency in multi-threaded environments [3]. Thus, not only do they benefit from ready-made solutions accelerating the development time, but also from a common structure that facilitates its long-term support and maintenance.

Figure 1 shows how structures can be divided into three major segments that developers need to define. The *Server* part comprises the software's Application Programmable Interface (API), organized in so called *Properties* accessible to the control system. Each *Property* can contain a group of readable and/or writable value-items. The *Real Time* part is organized in C++ classes called *Actions* and is where the low-level access to the hardware typically takes place. The configuration which triggers such *Actions* (i.e. events the software must react on) also belongs to this part. Finally, the *Data Store* is a set of fields and custom structures creating the internal data model, which is shared by the aforementioned parts.

The definition of all these parts is stored in an XML document called a FESA-design, which is validated against a schema to impose the framework's constraints [2]. While the XML format is convenient for its validation and code generation, it becomes cumbersome to edit or visualise, especially as the complexity of the software grows.

The FESA framework currently complements the documentation of a FESA-design with a graph generated by a python script with the help of the graphviz library [4]. Although this proof-of-concept is very inspiring, the structure and the static nature of the graph often makes the result unusable.

This paper describes a new graphical representation of the FESA design as an alternative to the XML text. In addition, it describes a GUI that facilitates editing to make the resulting graph cleaner and more user-friendly, addressing the issues with the existing graphical visualisation.

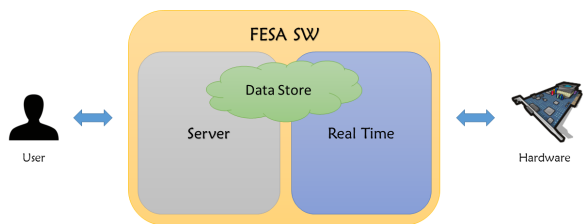

Figure 1: FESA software structure.

PROBLEMS

The current version of the framework (FESA 3) is integrated in the development environment as an Eclipse [5] plug-in. It provides two ways of viewing and editing a design document: a FESA design view, which is a prettified XML editor, and the eclipse's text editor to access the source document directly [1].

In a FESA-design, there exist numerous elements which can refer to other elements often located far from each other in the document. Consequently, both ways of viewing in Eclipse lack the ability to give a good overview of the software described, making its maintenance troublesome. This

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA180

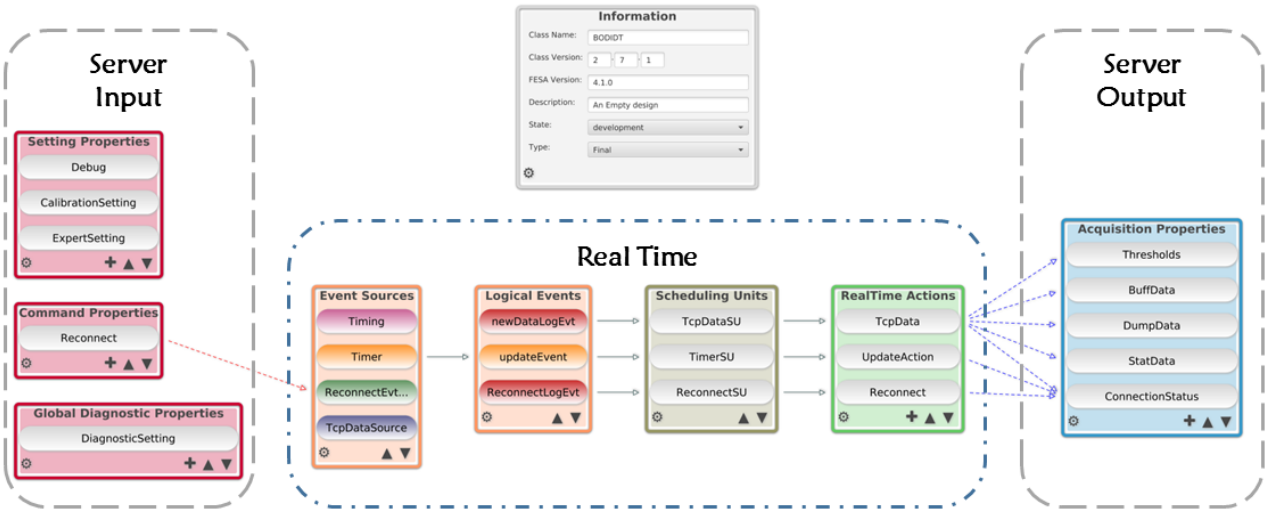

Figure 2: Structured graphical fragmentation of a FESA design.

is even more evident in composite software resulting from complex and lengthy designs since the number of such elements and their inter-references grows.

GRAPHICAL REPRESENTATION OF A SOFTWARE DESIGN

One way to obtain the overview of the software design, is to abstract it in a graph summarizing XML fragments into logical blocks and depicting the association between them by arrows as seen in Fig. 2.

Design Fragmentation

Visualising a FESA-design in a graph, benefits from its structured fragmentation. Figure 2 illustrates such an organization, grouping the elements from the different parts. Elements referring to the *Real-Time* part are placed in the centre, being at the core of the software. Those referring to the *Server* part are then split on either side of the graph.

By placing the *Properties* which contain writable value-items (*Input Server* part) on the left and those containing read-only items (*Output Server* part) on the right, gives a more natural way of reading the graph(left-to-right). Thus, studying the graph reveals the user inputs which are needed by the software to run the real-time part and shows what acquired data is given back to the user.

Overview and Documentation

As seen in Fig. 2, by fragmenting the design and visualising it in a graph, the internal details (i.e. *Data Store*) can easily be hidden, while simultaneously emphasizing the important aspects of the software's core organization (*Real-Time* part) and its public API (*Server* part). This, results in a clean, yet descriptive, overview. The documentation of the real-time software developed with FESA thus becomes richer and simpler, contributing to its long-term maintenance as well as encouraging collaboration among the different stakeholders.

Error Detection

In a graph-based representation of a FESA-design, developers can also benefit from quicker error detection. The association between the logical groups are visible in the form of arrows, facilitating the identification of missing or incorrect connections.

FESA GRAPH EDITOR

Even the structured fragmentation of a FESA-design in a graph, can eventually lead to unmanageable complexity. This yields the need to make the graph editable, to result in a cleaner illustration of the design while still benefitting from its graphical summary.

To overcome this problem, and improve user experience, a FESA Graph editor was developed to ease the transformation of a FESA-design document to its graphical representation.

Background

The FESA Graph Editor is a stand-alone application developed in Java to operate consistently across diverse platforms and to make direct use of existing Java libraries provided by the FESA team. These libraries form a model of a given FESA design document, which is encapsulated in the application's dynamic model called FModel. Therefore, FESA Graph Editor extends the model's capabilities to enable the creation and composition of the software's graphical abstraction.

For building the UI the JavaFX toolkit was chosen, as it currently provides the richest set of graphics and media packages for Java. Most of the UI is defined in an XML-based mark-up language called FXML, as it is suitable for easy development and maintenance of complex UIs [6]. Due to the lack of native support for interactive graphs, an open-source library was used. The *Graph Editor* was developed by Tesis DYNAware for creating and editing graph-like diagrams in JavaFX [7].

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA180

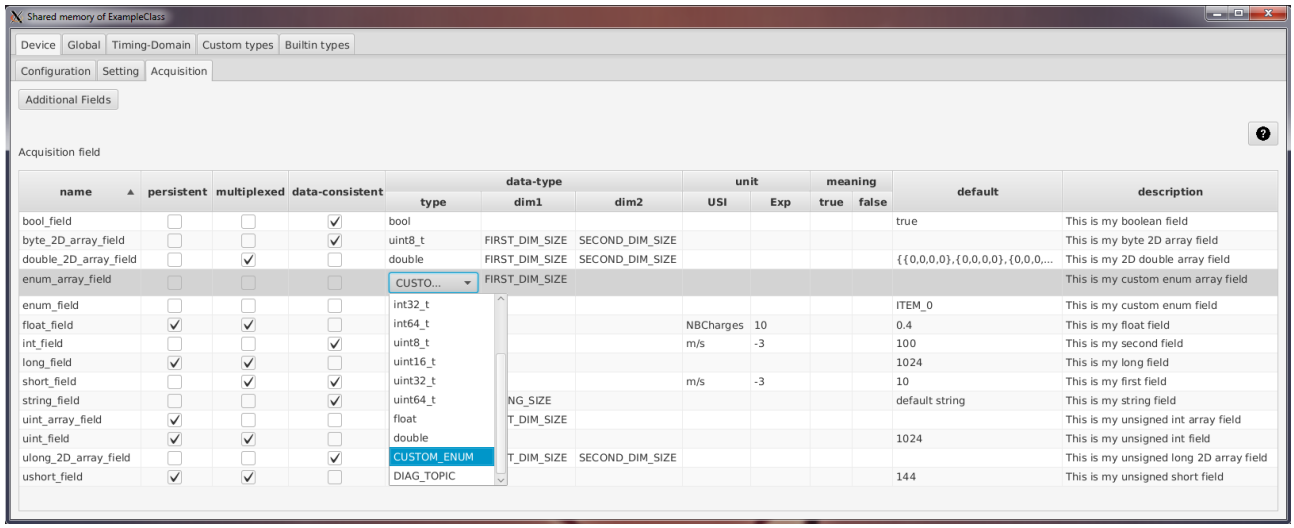

Figure 3: Data Store summary in classified tables.

In order to react to user input to the graph, the library implements the Model View Controller (MVC) pattern, making use of the *Eclipse Modeling Framework* (EMF) [8]. The Model of the graph is based on the aforementioned FModel and dictates the components of the FESA design that will be visible in the graph. The way these components are depicted is based on a set of Java classes called skins, that define the JavaFX nodes which represent them. These nodes are laid down in a container canvas that, together with the skins, compose the View. Finally, the Controller notifies the Model when the user changes the View (edits the graph) and vice versa (when changes in the Model need to be reflected in the View) for providing an effortless synchronization between the two.

User Interface

Exploiting the library's wide customization support, the FESA Graph Editor defines its own custom skins as an extension of the default set, to benefit from a tailored depiction of a FESA-design. To that end, FESA components of the same type (e.g. *Real-Time Actions*), are represented by boxes which may contain children (i.e. the actual defined real-time actions) in the form of inner nodes. A toolbar at the bottom of each box, allows customization of the box, in terms of order and colour of the inner nodes, as well other functions specific to the group. To avoid overpopulating the graph, the boxes become visible only if they contain at least one inner element. Since the boxes extend the library's default skins, they also support a drag-and-drop functionality enabling their repainting on the canvas to result in a cleaner graph.

The actual FESA components are represented by nodes that can be connected from either end to depict an association between them. Taking advantage of the library's selection API, the nodes can be re-arranged within their enclosing box, and shown with a menu listing the available functions they support. As a result, the information from a FESA-design can be abstracted without ever being lost, becoming visible only when the details are needed. As an example, in the overview of the graph the general API components are visible (e.g. the list of *Acquisition Properties* in the homonym box composing the **Server Output** part) but the details of a selected *Property* (i.e. the list of its value-fields) only becomes visible when requested in the form of an editable, pop-up table.

It is evident that the more FESA components belong to a group, the bigger the box will be. To support such large graphs resulting from complex designs, the area of the container canvas of the application is larger than what is visible. For navigating the whole area, the library offers convenient zooming and panning mechanisms. In addition, it offers an interactive, pop-up mini-map, which helps identify which area of the graph is currently visible and enables a direct change of the region of interest.

To give a complete editing user experience, the application implements a *Command Stack* to keep track of the changes performed on the graph. This permits cancelling or recovering an action, offering full undo / redo functionality. When the resulting graph satisfies the user requirements, it can be extracted in the following ways:

- saved as a PNG image.
- saved as a graph for reload to the application at a future date.
- copied to the clipboard for quicker sharing.

Although the *Real-Time* and *Server* parts of the FESA software are conveniently illustrated in a graphical format with boxes and arrows, *Data Store* is better summarized in tables. Since the list of fields that it contains can be large and are only needed internally in the software, its contribution to the design overview is limited. However, it remains of high importance when it comes to studying a FESA design in greater detail and thus, a user-friendly view can be made visible on demand.

Data Store's fields are grouped in tables along with their attributes for better visibility. Such tables are organized in

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA180

tabs according to the different categories of the fields for easier classification. Their cell type can differ (e.g. text-field, check-box, combo-box) according to the value type they represent. This is achieved by binding the tables with the FModel, offering a constant synchronization between the two.

An example of a *Data Store* illustration can be seen in Fig. 3 and is very similar to the table grouping details from a *Property* (i.e. the list of its value-items).

FESA Integration

A large effort was made to integrate as much of the FESA framework's functionalities as possible starting with its model. A FESA toolbar was therefore added to the application, allowing the user to launch three basic operations:

- *Validation* to validate the loaded FModel against the framework constrains.
- *Synchronization* to generate the C++ code based on the loaded FModel.
- *Upgrade* to upgrade the framework version in the loaded FModel.

Integration of the framework into the application is highly promising for producing a complete visual development environment to improve the user-experience when creating and editing the design of FESA software.

CONCLUSION & OUTLOOK

In order to get an overview of software developed with the FESA framework and ensure its coherent documentation and for subsequent long-term maintenance, a graphical abstraction of the design is essential. This paper proposes a structured, pictorial fragmentation of such a design, with logical blocks representing its high-level XML fragments and arrows illustrating their associations.

As the resulting graph may still be complex, the need for editing it, is crucial. Therefore, a graphical overview application with in-built editor was developed to allow the graph customisation, resulting in a cleaner illustration and thus a more contractive software overview.

The next step being envisaged is a complete visual layout tool, were users will be able to design and edit their software structure without needing direct access to an XML document.

ACKNOWLEDGEMENT

The team would like to thank Frederic Hoguin from the CERN FESA team for his considerable contribution to this work through a lot of useful advice, as well Bartosz Przemyslaw Bielawski from the CERN BE-RF group for inspiration.

REFERENCES

[1] S. Deghaye and E. Fortescue-Beck, "Introduction to the BE/CO control system", https://be-dep-co.web.cern.ch/sites/be-dep-co.web.cern.ch/files/site_documents/BECO%20Accelerator%20Control%20System%202016%20Part%201.pdf

[2] M. Arruat *et al.*, "Front-End Software Architecture", in *Proc. ICALEPCS'07*, Knoxville, TN, USA, Oct. 2007, pp. 310–312.

[3] S. Hoguin and S. Deghaye, "Solving the Synchronization Problem in Multi-Core Embedded Real-Time Systems", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 942–946, doi:10.18429/JACoW-ICALEPCS2015-WEPGF102

[4] Graphviz, http://www.graphviz.org/.

[5] Eclipse, https://eclipse.org/home/index.php

[6] JavaFX, http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html

[7] Graph Editor, https://github.com/tesis-dynaware/graph-editor

[8] Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/.