EQUIPMENT SOFTWARE MODELLING FOR ACCELERATOR CONTROLS

Michel Arruat, Stephen Jackson, Jean-Luc Nougaret, Maciej Peryt,
Accelerators and Beams Department,
CERN, CH1211 Geneva 23

Equipment software controls hardware actuators and sensors interacting with the particle beam. Traditionally, this has been achieved with the use of procedural languages and low-level techniques. Two years ago, the structure and behaviour of such software was formalised, and an object-oriented programming framework based on the resulting formal model was developed. This enables equipment-specialists to capture their design in a standard form before implementing the specifics of their system. This paper describes how we elaborated this model to incorporate the high-level language in which the equipment software model would be realised, as well as the entity-relationship schemas needed to represent instances of such equipment in a relational database. After describing the increasing levels of abstractions of equipment software, we will follow the development process by succinctly describing the successive stages of code-generation, implementation and deployment whereby abstract models are ultimately converted into concrete processes running on a real-time operating system.

INTRODUCTION

In 2003, CERN launched the front-end software architecture project, known as FESA [1], in order to homogenise the real-time software that provides access to the accelerators' physical pieces of equipments. In this section, we recall the purposes of equipment software and of the FESA framework within which it is developed.

Equipment software

Particle accelerators are fitted with terminal devices that can be sensors, actuators or a combination of both. From a remote control room, operators access these devices across the control system infrastructure which consists of layers of hardware, software and communication protocols. A crucial part of the control infrastructure, equipment software is located at the junction of two worlds: on one hand, it communicates with the control-room's computers and handles operator requests (property interface). On the other hand, it must deal directly with hardware. Equipment software provides a stable and homogeneous functional abstraction on top of accelerator equipment (sensors, actuators…) whose hardware implementation is heterogeneous and evolves over time. Figure 1 depicts the complementary components of an equipment-software.

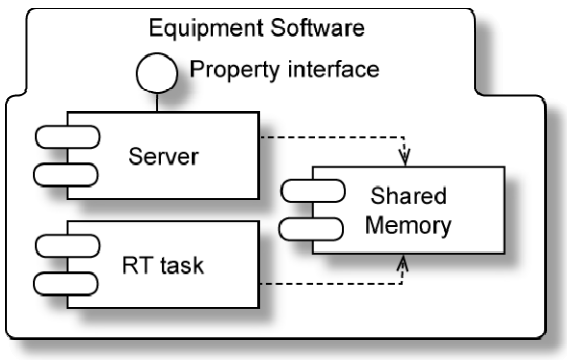

Figure 1: equipment software's binary components

Equipment software is incarnated by three binary components: a server process implements the services that respond to client requests, a real-time task performs repetitive access to the hardware according to time-constraints, and the two are decoupled and exchange data through a shared-memory segment.

Development framework

Equipment specialists are the people responsible of writing embedded real-time programs that control physical pieces of equipment. The FESA object-orientated framework encapsulates recurrent aspects of equipment-software development as a reusable software package that can be tailored – or customized – on a case-by-case basis. The main purpose of the framework is to define the overall structure of equipment software. As with most frameworks, the way classes fit together is more important than the functionality of any one class. In addition, it simplifies the task of writing an equipment class by capturing structural elements common to any real-time equipment software. Figure 2 illustrates how one creates custom equipment-software by applying and refining the core framework.

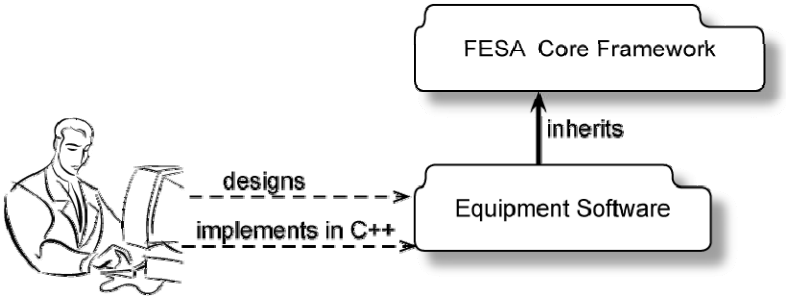

Figure 2: equipment software development with the FESA framework

Concretely an equipment specialist defines a set of concrete C++ classes that derive the framework's base layer and implements them.

Equipment software modelling

FESA has had a significant impact on the level of abstraction at which equipment specialists develop equipment-software. Initially, they were programming procedures in C. They now put an emphasis on object-orientated modelling of real-time components. Modelling of an equipment-software consists in recasting the problem at hand into a standard form. To this end, the equipment-specialist must think of an equipment-design as a set of conceptual objects:

- A public interface of the equipment class, as a set of operational properties.
- A device model, holding data that form the hardware-device's software-counterpart.
- A set of server-actions, implementing the property services.
- A set of real-time actions, accessing the hardware layer.
- A set of logical events, acting as the equipment-software's pacemakers.
- A set of triggering-rules, binding logical events and real-time actions.

FROM MODELLING TO META-MODELLING

Initially, we formally captured the FESA core framework's architecture as a set of UML diagrams. Over the last two years, we extended the formal model in order to also encompass the respective degrees-of-freedom and constraints for modelling custom equipment-software. This caused a shift in abstraction: from modelling the framework itself, we moved onto modelling the various ways in which the core framework can be refined by equipment-specialists; i.e. from capturing the generic aspects of equipment-software as a standard model, we moved towards capturing its specific aspects as a model of all possible custom models.

Hierarchy of abstraction levels

The equipment-configuration stage which was present in the early version of the FESA framework iteratively evolved into some sort of business-domain language for specifying equipment-software models in terms of structure and behaviour. Hence, the above-mentioned two-level view of the framework is actually better depicted by the diagram of Figure 3:

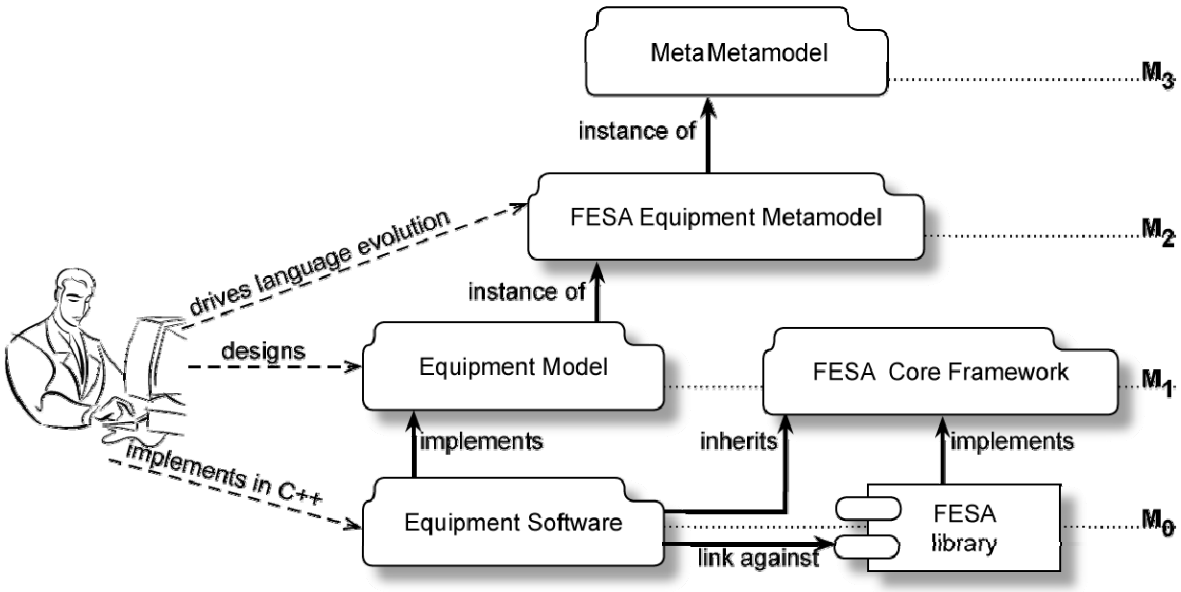

Figure 3: equipment software development with the extended FESA framework

FESA modelling spans across successive levels of abstraction. Tagging these levels according to the meta-modelling terminology defined by the OMG meta-object facility (MOF) [2], we notice that FESA modelling involves abstraction layers from M0 to M3.

- M0: the object of interest, in our case the C++ equipment software, which links against the FESA framework's library.
- M1: the model of the object of interest. This model is two-folds. The FESA C++ framework model captures the generic aspects whereas the custom part of the model, encoded as an XML design document, specifies the equipment's structure and behaviour.
- M2: the FESA meta-model, i.e. the model of all possible M1-level models. It is represented as a W3C Schema which encodes the equipment specification language as an XSD document.
- M3: the meta-meta-model, i.e. the language for specifying the FESA meta-model. In our case, we deliberately restrict ourselves to the W3C Schema specification. By comparison, the apex of the OMG pyramid of abstraction levels formally culminates with the MOF meta-modelling language.

Prior to introducing the FESA framework, equipment-software development was mostly restricted to level M0, with no formal design stage. Now, equipment specialists routinely work at levels M1, M0 for respectively designing and coding equipment-software. They also contribute up to level M2 when they submit new requirements. Requirements indeed often translate directly into meta-model extensions. To cover new requirements, the meta-model evolved incrementally over more than one hundred iterations in less than two years. The FESA meta-model has progressively converged into a shared high-level language designed by and for the equipment-specialists to address their specific needs.

Meta-model of equipment software

The FESA meta-model is encoded as an XML Schema. Elements represent the complementary pieces of information that may be part of an equipment-software's design model. Such elements form whole-part hierarchies according to their position in the XML tree. Some elements may be subject to additional constraints that binds them together. The diagram of Figure 4 illustrates the entity-relationship graph that captures the grammar of a language for specifying the model of an equipment-software. For the sake of simplicity, we only show some key elements and relationships. The detailed meta-model is about two orders of magnitude bigger in terms of number of nodes and constraints. The tree hierarchy accommodates increasing levels of details from coarse-grain to low-grain as one proceeds from root to leaf.

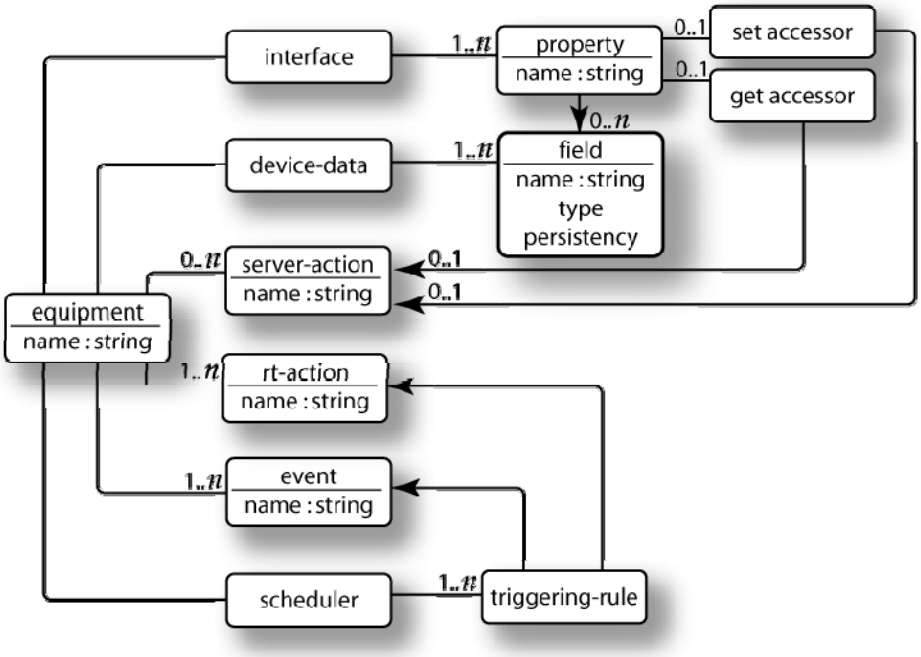

Figure 4: simplified graphical representation of the FESA meta-model

In the above diagram, solid lines represent whole-part relationships with indication on the cardinality of the part nodes. Arrows represent referencing from one element to another. For instance, the bottom part of the graphical diagram reads as follows: "equipment always features a scheduler. The scheduler consists of a set of one or several triggering-rules, where each triggering-rule binds an event with the real-time action it triggers." Equipment models are XML documents which are themselves instances of the meta-model [3]. Equipment specialists create such models with a dedicated design-tool and assemble them element-by-element. The design tool enforces the schema constraints in order to ensure that all created models are well-formed.

A digression about the meta-meta-model

As opposed to the M0, M1, and M2 levels which grow in-house and which are bound to the business domain of accelerator controls, we rely on the standard XML W3C Schema specification as the language for encoding the FESA meta-model. This choice is somehow arbitrary and driven by practical considerations. XML brings obvious advantages: as a standard, it brings in a collection of XML technologies for model-parsing and code-generation. But there is a downside: the choice of W3C XML Schemas artificially restricts the FESA "grammar" to map on a tree of XML elements. Moreover, Schemas currently only support simple element uniqueness and referencing constraints, which poses some severe limitations on the expressiveness of Schema-based meta-models. Note that this choice also departs from the OMG view, which relies on the reflexive MOF meta-modelling language at the apex of the abstraction hierarchy.

Database schema derived from the meta-model

The FESA meta-model of an equipment-software includes a mini-language for specifying the so-called data-model of the equipment. This data-model lists a set of fields which accurately provides a snapshot of the underlying hardware device's state and parameters. There is one data set per instance of any FESA equipment class. Hence the data-model can be viewed as the meta-data of all the instance-data of a given class. Configuration parameters are associated to the hardware configuration and are constants stored in the database. In FESA jargon, they are referred-to as "final" fields. Operational parameters represent the variable settings which are applied from the control room. In the FESA jargon, they are referred-to as "persistent" fields. Both the final and the persistent fields are

stored in an Oracle database [4] which stands at the heart of the FESA data-management system. The database's schema partially derives from the FESA meta-model, as shown in Figure 5.

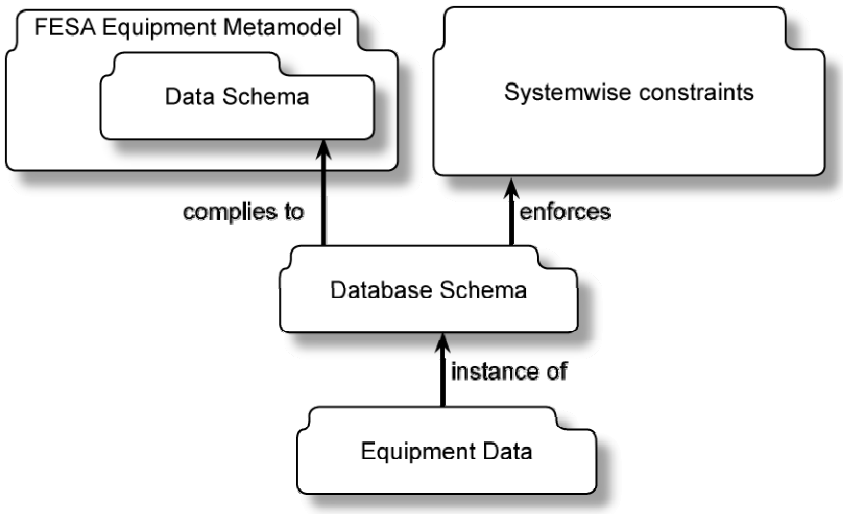

Figure 5: relationship between the meta-model and the database's meta-data

The data-model part of the FESA meta-model is embedded in the database's overall entity-relationship schema. The database schema encompasses additional constraints which are either system-wise, i.e. beyond scope of a single equipment-software, or which cannot be embedded in the meta-model. Indeed, XML Schemas fail to capture constraints beyond referencing an element from another. Therefore FESA meta-data rely on entity-relationship diagrams typical of relational databases.

CONCRETE BENEFITS OF RAISING ABSTRACTION

In about two years, FESA has successfully forced a switch from C to C++ as the primary language for writing front-end equipment software, and introduced the meta-model as a high-level specification language for modelling equipment-software. This improves knowledge sharing and productivity.

Sharing knowledge across accelerator control domains

In spite of the huge diversity of devices, FESA has successfully standardized the language for describing equipment software. Equipment software controlling devices as diverse as beam loss monitors, beam position pickups, bending magnet power supplies, RF cavities or beam kickers can now all be described in a common language. Although specific business domains of accelerator controls always require ad hoc customization and device-specific expertise, the meta-model already provides a common ground among equipment-specialists from different domains for sharing knowledge and transposing experience from one domain to another. Hence the main achievement of the abstract modelling approach is assessed by the range of business domains it covers.

Managing framework evolution

FESA must accommodate evolving user needs. Relying on a high-level specification of equipment-software in the form of XML design-documents makes it rather straightforward to track usage patterns of the framework. For instance, automated scripts routinely process the repository of equipment-design documents and extract information about the relative usage ratio of the various elements that form the FESA meta-model. Since such elements encode user features and translate as dependencies from the C++ core framework, such metrics are useful to weight the respective popularity of the framework's different features and to steer its evolution.

Generating executable code from models

The introduction of a high-level language for specifying an equipment-software's structure and behaviour means that the equivalent C++ constructs can be generated automatically rather then hand-

coded. Harnessing XML technology, code-generation recipes are encoded as XSLT style-sheets, which process XML-encoded equipment-models. Equipment-specialists rely on an automated code-generation phase for sub-classing their concrete classes from the framework's base classes, and then implement the server-actions and real-time actions by filling-in skeleton methods. This process is illustrated on Figure 6. Early metrics on about 50 equipment classes show that on average, 65 % of framework's customization code is now obtained through automatic generation. The remaining 35 % left to hand-coding is mostly devoted to custom treatments and hardware handling. After generating the binary components of an equipment class, the equipment-specialist may deploy them on some front-end computers. Then, they provide a software-device counterpart to each installed hardware-device, by repeatedly instantiating the equipment class. Instantiation consists in populating the database tables that are derived from the model of the class beforehand, as described in a previous section.

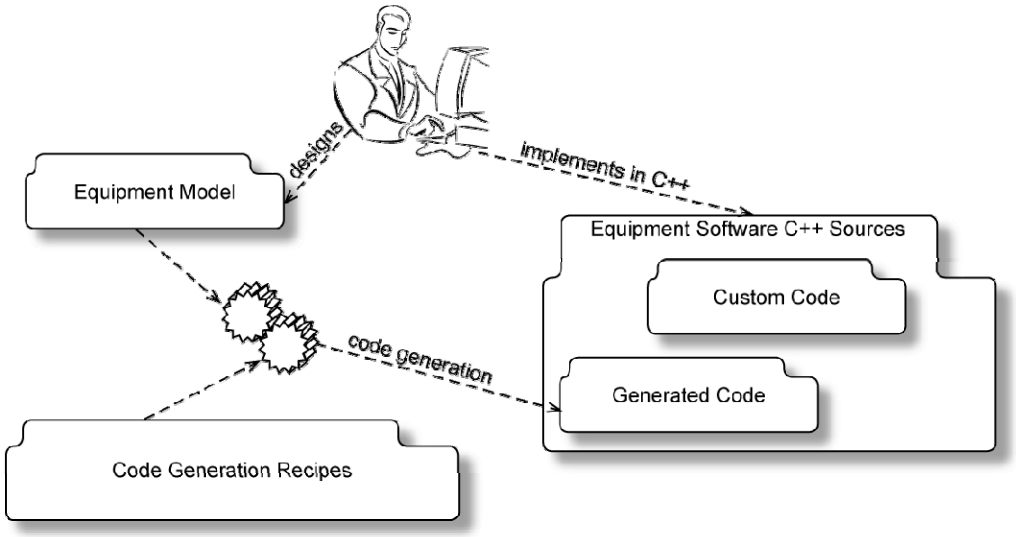

Figure 6: semi-automated conversion of design-models into executable code

How far shall we go into the direction of a defining a new language and of automated code-generation? The borderline between high-level modeling and hand-coding is largely a matter of taste and choice, and partly a matter of performance. Considering that FESA developers are programming experts, the current partitioning between high-level modeling and implementation languages is probably adequate. First, the framework base layer insulates equipment-specialists from dealing with system-level programming and intricate topics such as shared-memory access and multi-threading. Second, the meta-model allows them to tailor the framework in a pick-and-click stage. Last but not least, equipment-specialists largely rely on C/C++ hand-coding to implement and optimize action loops which carry out heavy processing and interact with hardware. Such a role-distribution amongst an object-orientated framework, a meta-modeling approach and manual C/C++ coding is conservative. In that respect, this paper falls short of proposing a revolutionary paradigm for equipment software development. Our contribution instead fits well into the technology track of this conference by describing how to harness technologies of meta-modeling and code-generation as a powerful complement to the prevalent approach of hand-crafted programming of embedded systems.

References

[1] CERN Front-End Software Architecture for Accelerator Controls. A. Guerrero, J.-J. Gras, J.-L. Nougaret, M. Ludwig, M. Arruat, S. Jackson. ICALEPCS, Gyeongju, Korea, October 2003.

[2] Meta Object Facility (MOF) Specification. Version 1.4. April 2002. Object Management Group.

[3] Use of XML Technologies for Data-driven Accelerator Controls. M. Arruat, S. Jackson, J-L. Nougaret, M. Peryt. ICALEPCS, Geneva, Switzerland, October 2005.

[4] The Directory Service for the CERN Accelerator Control Application Programs. J. Cuperus, M. Peryt, E. Roux, J. Schinzel. ICALEPCS, Geneva, Switzerland, October 2005.