A PROPOSAL FOR MODELING THE CONTROL SYSTEM FOR THE SPANISH LIGHT SOURCE IN UML

D. Beltran*, LLS, Barcelona, Spain
M. Gonzalez, CERN, Geneva, Switzerlan

Abstract

CELLS (Consorcio para la construcción, equipamiento y explotación del Laboratorio de Luz Sincrotrón), the first Spanish synchrotron is now in the latest stages of its design, previously to its construction. The control system for this 3rd generation light source is proposed to be done using software models, and object oriented techniques. In this paper the main ideas of this methodology are presented as well as the preliminary ideas of the development process for the Spanish synchrotron control system, with its analysis and design models. In this methodology the user requirements are captured with the use-case diagrams and specified by the activity diagrams. The design models are realized by the class diagrams for the static structure and the sequence diagrams and state machines for the dynamic structure. The first ideas for the architecture are also presented, as well as the software organization in packages. Also, a prototype for a detector data acquisition system using this methodology has been developed, and the experiences are also described. Finally the advantages of this methodology are discussed.

THE SPANISH LIGHT SOURCE

The first Spanish synchrotron radiation facility [1] has been approved, and it will be built in the area of Cerdanyola del Valles (Barcelona), Catalunya, Spain. This synchrotron is funded by the Spanish Ministry of Science and Technology and the Generalitat of Catalunya (Catalan Government) with a 50% contribution from each party. The construction has just started and, it is expected that light will be delivered to the beamline users in 2010.

The facility will have the classical layout, with a linear accelerator, a booster and a storage ring, connected by two transfer lines. The storage ring will be at least 2.5 GeV in energy, quite possibly 3 GeV, and will have at least 12 sections for insertion devices. Useful radiation for photon energies of up to 25 keV is in the design specifications. The possibility to have nominal energy injection and mini-beta sections for low gad ID's is now under serious consideration. The aim is to achieve emittances in the 5 nmrad region or better. Note that these are somewhat better specifications that those initially presented to the authorities [2] and which formed the basis of project approval. The capital project contemplates the building of five beam lines initially, even though the total possible volume is at least 36 beamlines. The process of defining the scientific and technical objectives for these first five beam lines has started through the usual procedure of wide consultation to the future user community and appropriate experts.

* dbeltran@esrf.fr

THE DEVELOPMENT PROCESS

For simple systems, it is perfectly feasible for a single person to sequentially define the whole problem, design, build the software, and then test the end product. However, in complex systems, as a light source is, a team of analysts, designers and programmers will be required to build the system, and their activities should be coordinated. In this case a software development process is needed in order to transform the user requirements into a software product. Among several processes (extreme programming [3] or the classical programming, like PSS05 [4]), we propose the usage of the unified process [5], which is a process based on a modelling language that is iterative, architecture centric and use-case driven. It is organized around four phases: inception, elaboration, construction and transition. It is further organized around five workflows:

- Requirements: describes what the system should do for its users and under what constraints.
- Analysis: analyses the requirements, described in the previous workflow, by refining and structuring them.
- Design: formulates models that focus on non functional requirements and the solution domain, and that prepares for the implementation and test of the system.
- Implementation: the essential purpose of this workflow is to implement the system in terms of components, that is, source code, scripts, binaries, executables, and the like.
- Test: verifies the result from implementation by testing each build, as well as final versions of the system to be released.

This model developed during the unified process provides a clear understanding of the requirements between the users and the developers, allows the selection of a suitable architecture and facilitate the management of the project. The model provides semantically rich representations of the software system under development, it will be used in the communication between the different groups involved in the project, and it will be used in the documentation. Furthermore, a good model of the control system will facilitate its maintenance and its future upgrades, as well as its testing and commissioning. Finally, the model will allow the

identification of the risks, and its mitigation before the point at which they come up in the development process.

The project will be organized in packages in the model what will ease the assignment of tasks within the team. In addition, it will allow to clearly specify the parts that could be subcontracted to external companies. Also, this organization will ease the software sharing and re-using.

In this paper we present a prototype developed with this methodology (unified process), following all the workflows described. The experiences developing this prototype are described.

For the Spanish facility control system the model itself is introduced with the use cases and actors. Afterwards the identified packages and analysis classes are reviewed.

The modelling language: UML

We have created the model in UML [6] (Unified Modeling Language) because it is an industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. This language appeared in 1996 by G. Booch, J. Rumbaugh and I. Jacobson, and it is a merge of several notations existing at that moment, most notable Booch, OOSE (Object-Oriented Software Engineering) and OMT (Object Modeling Technique). Today is of widespread use in software projects. This language was accepted by the OMG (Object Management Group) for its standardization and maintenance.

There are lot of CASE (Computer Aided Software Engineering)-tools which support UML. We have evaluated two of them: Umbrello v1.2 [7], an open source tool running under Linux and Rational Rose v6.5 [8], a commercial tool running under Windows. We have decided to develop the project using the second one, because it is closer to the UML standard (it contains all the diagrams and their artefacts) and it is more professional (better finished).

USER REQUIREMENTS CAPTURE

The purpose of this workflow is to drive the development toward the right system. This is achieved by describing the system requirements (i.e., the conditions or capabilities to which the system must conform) well enough so that an agreement can be reached between the users (machine physicists and operators) and the developers on what the system should and should not do.

The requirements are captured in UML by the use-case diagram, which describes what the system does for each type of user. In this diagram are two different artifacts: actors (users) and use-cases (service provided by the system).

Identifying the actors, we have identified the external environment of the system. They could be humans (operators, machine physicists and scientists) or hardware systems (vacuum devices, diagnostics, magnet power supplies, etc).

Any use case specifies a sequence of actions, including variants, that the accelerator control system performs and that yields an observable result of value to a particular actor.

The use cases capture the functional requirements of the facility. But the non functional requirements, such as performance, availability and security are also present in the diagram associated either to a particular use case (tagged value) or with any particular one (note).

Picture 1 presents the main use-case diagram, where the main uses cases in a synchrotron radiation facility are presented.

These use cases describe what the control system does but it does not specify how it does it. Then we have specified the behaviour of every use case by describing a flow of events in text: it includes how and when the use case starts and ends, the basic flow and alternative and exceptional flows of the behaviour.

The human actors interact with the system using the use cases, as it is shown in the figure 1. Also the use cases collaborate with the subsystems (diagnostics, magnet power supplies, RF, vacuum, timing, facility interlocks, beamline control, experimental DAQ), although the connections are not shown in the previous diagram. These subsystems will talk with the hardware actors.

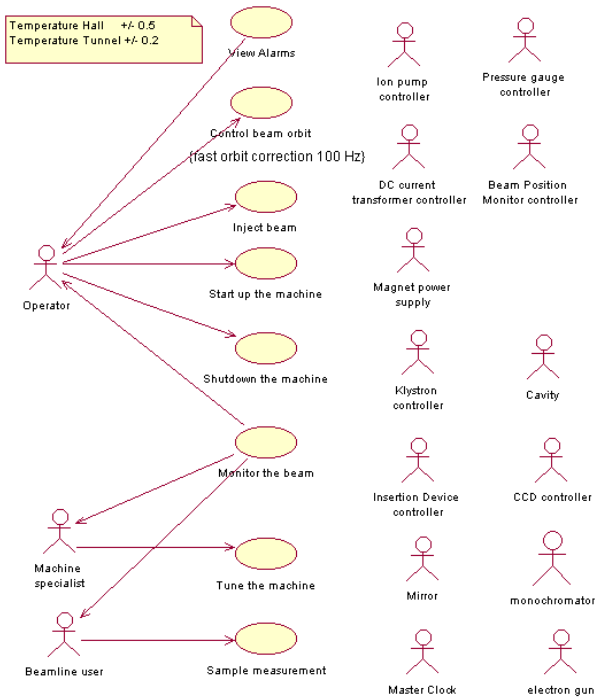

Figure 1: Main use case diagram.

In this paper we have picked up one of the most important use cases for a synchrotron (Inject Beam) and we have developed it with more detail. This is presented in figure number 2.

This use case incorporates the functionality of three other use cases (Control Linac, Control Booster and Control Storage Ring). This is modelled by the <<include>> relationship.

The injection could be done with top-up, which is a special injection mode. This exceptional behaviour is modelled by the <<extend>> relationship.

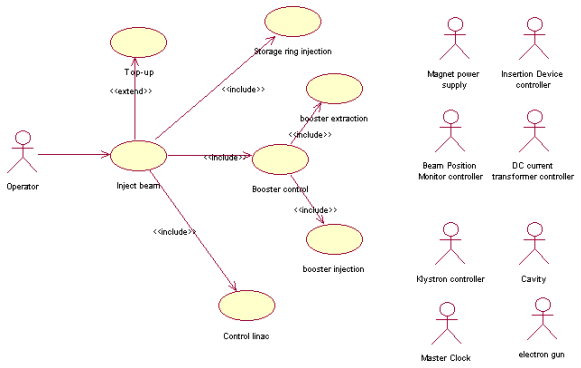

Figure 2: Inject Beam use case diagram.

ANALYSIS MODEL

The analysis workflow refines the use cases (use case realization) described in the previous diagram, in order to achieve a more precise understanding of the requirements. So the analysis model will be created, and it will grow incrementally as more and more use cases are analysed. This is a conceptual model, as it is an abstraction of the system and avoids implementation issues (it is applicable to several designs).

The next step is to refine the primary way in which the operator executes the Inject Beam use case. This is done in the activity diagram (see figure 3), which is essentially a flowchart, showing flow of control from activity to activity.

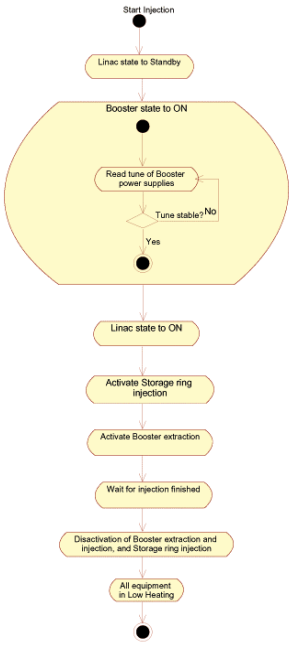

Figure 3: Activity diagram for the Inject Beam use case.

This diagram has different activities executed sequentially. The activity “Booster state to ON” has a nested diagram for modelling the waiting for the end of a certain action (stability of the booster magnet power supplies), which has a branch to specify an iteration.

This activity diagram presents the main sequence of actions, although exceptional flow of events (scenarios) has been specified.

DESIGN

The design model describes the physical realization of the use cases by focusing on functional and non-functional requirements, together with other constraints related to the implementation environment.

Architectural design

The purpose of the architectural design is to define a structure that will be preserved through the entire software life cycle. Due to the complexity of the project, we propose the usage of a layers pattern (see figure 4) that defines how to organize the design model in layers, meaning that components in one layer can reference components only in layers directly below. It reduces dependencies in that lower layers are not aware of any details or interfaces in the upper layer. Moreover, it helps us to identify what to reuse, and it provides a structure to help us make decisions about what to buy (and subcontract) or to build ourselves.

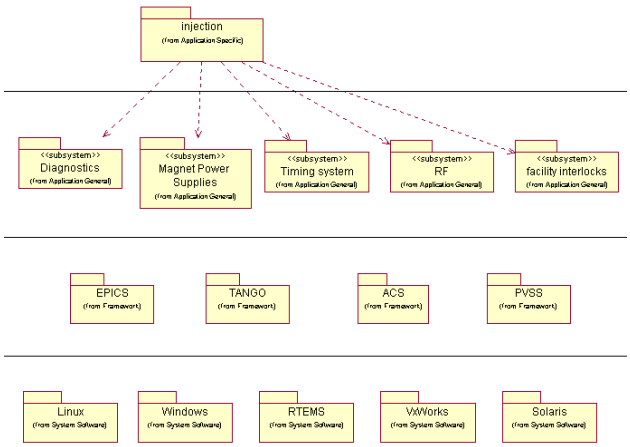

Figure 4: Layered architecture.

This system has individual applications at the top (application-specific layer). Below it, there is the application-general layer, which contains subsystems [9] that are not specific to a single application and can be reused for many different applications within the same domain. These design subsystems decompose the implementation work into more manageable pieces handled by different development teams, possible concurrently. The architecture of these two layers is created from the architecturally relevant use cases (as the Inject Beam is).

The architecture of the lower two layers (framework and system software) can be established without considering the details of the use cases. In this approach are presented the several candidates under study.

Design class

Once the design model has been decomposed in more manageable pieces (design subsystems), we have created and developed its components: the design classes. They will be characterized by a static view (class diagram) and a dynamic one (statechart diagram). In the following the magnet power supply class with its sub-classes are presented, which belong to the magnet power supply subsystem.

The class diagram (see figure 5) shows a set of classes, interfaces, collaborations and their relationship. It is a structural diagram to visualize, specify, construct and document the static aspects of a system.

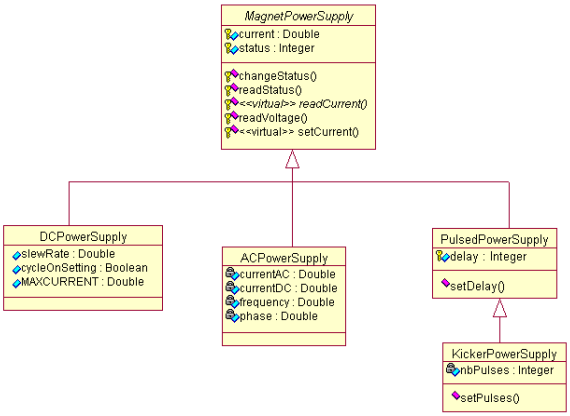

Figure 5: Magnet power supply class diagram.

This diagram contains an abstract class (MagnetPowerSupply) with the basic operations and attributes common to all magnet power supplies. Some of the operations are defined virtual (as ReadCurrent) so the implementation will be particular to any sub-class.

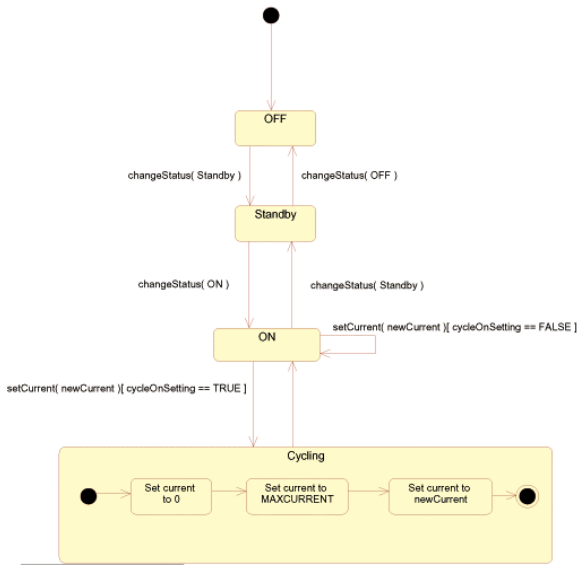

Figure 6: DC magnet power supply state machine.

There is a state machine associated with any class. A state machine is a behaviour that specifies the sequences of states an object goes though during its lifetime in response events (or commands), together with its responses to those events. The state machine is used to model the internal behaviour of an object, as it is show in figure 6 for a DC magnet power supply.

This state machine has a composite state (cycling), where is contained a nested state machine. Also, a guard condition has been added to indicate the transition that should be taken from the ON state when the command "setCurrent" is executed. It takes the value of the internal attribute "cycleOnSetting" to determine this condition.

We would like to note that this statechart specifies a machine that runs continuously; there is no final state.

Dynamic view

Now that we have an outline of the design classes needed to realize the use case, we will describe how their corresponding design objects interact. This is done using sequence diagrams containing the participating actors, design objects, and message transmissions between them.

The messages sent between objects are the public class methods defined in the in the class diagram: the tool detects which are the public methods of the class and they are the only offered. But an object could send a message to itself. In this case, also the private and protected methods could be sent.

The picture 7 presents a sequence diagram for the injection in the storage ring. The objects that participate in the interaction are at the top, across the X axis, placing the actor (Operator) that initiates the interaction on the left, and increasingly more subordinate objects to the right. The messages sent and received by objects and actors are placed along the Y axis, in order of increasing time from top to bottom. The focus of control is a tall, thin rectangle that shows the period of time which an object is performing an action. The top of the rectangle is aligned with the start of the action; the bottom is aligned with its completion.

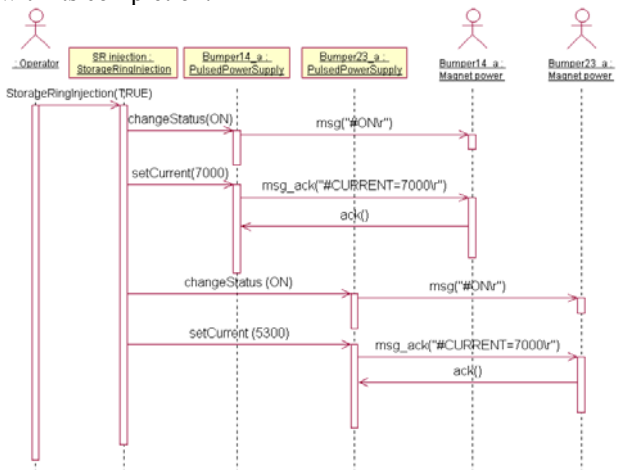

Figure 7: Sequence diagram for the injection in the storage ring.

It represents all the commands sent to the power supplies and in which order when the operator activates the injection in the storage ring. Note that for any message sent by the SRInjection object to the PulsedPowerSupply objects, these send a message to the

equipment through a fieldbus (msg_ack). When the hardware finishes the action, returns an acknowledgment to the object sending the message.

DAQ SYSTEM PROTOTYPE

We have developed a data acquisition system (DAQ) for a gas filled detector following the unified process introduced before. This project has been used as prototype to get real experience following the development workflows and using the UML diagrams.

In the requirements capture, three actors (Beamline User, Acquisition Card and Storage Area) and eight use cases (Start Acquisition, Collect One Image, Configure System, Calibrate System, Monitor Acquisition, View Image, Stop Acquisition and Save Images) were identified and described, with the help of the users, to the level required for the development.

Given the not so big size of the application, the analysis and the design phases were carried together. The class diagram contains two classes: a device server implemented in C++ and graphical interface in Python.

After the experience gained during the project work, we are convinced that the UML diagrams help us in finding a common understanding with the users of what the system should do. The use case artifact and the graphical representation were fundamental in the discussions with the beamline users about how they should interact with the acquisition system and with the hardware engineers about the cards. Achieving a good communication with them allowed us a clear understanding of the problem, which was the key for developing the right software system.

The class diagram became central technique for the object-oriented analysis. It also has a greatest range of modelling concepts. Together with the sequential diagram, it allows the refinement of the class, after a couple of iterations.

Finally, the UML diagrams were very useful for the generation of documentation, both about the architecture and its components (classes).

For the near future, we would like to investigate further other facilities provided by the tools. The automatic code generation from the class and component diagrams will facilitate the synchronization of the model and the code. We also consider very important the inclusion of a n Integrated Development Environment (IDE) to shorten the cycle of editing-compiling-testing the code. Finally, our configuration management tool will also have to be integrated, specially bearing in mind that the process has to scale up to the development for the full synchrotron.

CONCLUSIONS

The unified process, which is being proposed as for the software development of the control system for the Spanish light source does not impose any restriction in the software development process. The main difference with the traditional techniques is that the program is primarily manifested as a model. The UML model generated during the unified process facilitates the understanding between the users and the developers, choose a suitable architecture and facilitate the management of the project. Furthermore, a good model of the control system will facilitate its maintenance and its future upgrades, as well as its testing and commissioning. And finally, this model will be primary used for documenting the system.

We have developed a C++ and Python application following this model-driven development strategy. The development of this application has confirmed the advantages previously described of this methodology, and that confirm it as a good candidate for developing the control system of the Spanish synchrotron. At the moment, we are working at extending our experience to cope with all the tools that will be required during the development.

REFERENCES

[1] http://www.cells.es
[2] http://www.lls.ifae.es/
[3] http://www.xprogramming.com/xpmag/whatisxp.htm
[4] http://www.v-modell.de/director/products/je014.htm
[5] I. Jacobson, G. Booch, J. Rumbaugh, “The Unified Software Development Process”, Addison-Wesley, 1999.
[6] http://www.omg.org/uml/
[7] http://sourceforge.net/projects/uml/
[8] http://www.rational.com/products/rose/index.jsp
[9] A subsystem is a grouping of elements (classes, interfaces and other subsystems).