The use of ROOM in the design of data-acquisition software components

W. Carena, R. Divià, P. Vande Vyvre and A. Vascotto (for the ALICE collaboration)
CERN, CH 1211 Geneva 23, Switzerland

Abstract

The Event Builder and Distribution System (EBDS) is a component of the data-acquisition architecture of the ALICE experiment at CERN. The purpose of the EBDS is to dispatch the sub-events originated in the detector front-end electronics to the processors of the Event Filter Farm, where the full events are assembled.

For the design of the EBDS, we use the Real-time Object-Oriented Modeling method (ROOM), which was chosen because of its powerful modeling paradigm, well suited to this type of application. The use of ROOM is aided by the ObjecTime Developer tool set, which fully supports the method and covers all the aspects of the development cycle, from analysis to code generation. Fast prototyping and simulation bring a new perspective to the designer, who can advance by gradual refinements.

We describe how ROOM has been used to design a model of both the EBDS and its environment, and the results obtained from the simulation. We also review the experience acquired with ROOM and the ObjecTime tool, and indicate what benefits and obstacles we encountered.

I. Introduction

The ALICE experiment [1] will be performed at the LHC machine, currently under construction at CERN. The purpose of the experiment is the study of the quark–gluon plasma created in high-energy nuclear collisions. The ALICE detector is specialized in the detection of heavy-ion nucleus–nucleus interactions.

The ALICE data-acquisition (DAQ) architecture is based on parallel streams of data, as shown in Figure 1.

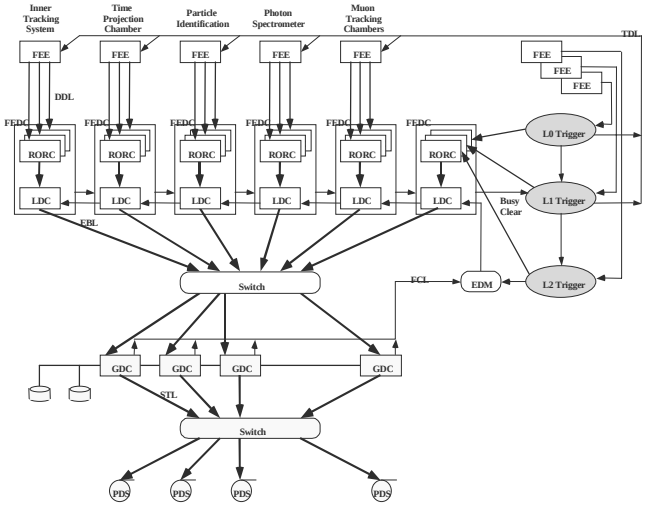

Figure 1: ALICE data-acquisition architecture

The data generated in the detector are transferred through about a hundred optical links. The data are received in the Front-End Digital Crates (FEDC) by the Read-Out Receiver Cards (RORC), which are buffer memories capable of storing several fragments of events. The Local Data Concentrators (LDC) in the FEDC are embedded processors that collect the event fragments and construct the sub-events, which are then sent over the network to one of the Global Data Collectors (GDC). The GDCs are processors as well: their role is to put together all the sub-events, build the full events and send them to the recording system. The GDC will also have analysis, filtering and storage functions.

The expected data rates are about 100 Mbyte/sec on each of the parallel streams, for an aggregate bandwidth of 2.5 Gbyte/sec.

The network linking LDCs and GDCs is based on widely used standards and off-the-shelf commercial components. We expect the performance of the Local Area Network technology available in 2005 (startup of the experiment) to be adequate to our requirements and affordable.

II. Event Building and Distribution System

The problem of building events is always present in data-acquisition systems with parallel streams of data. The sub-events coming from different machines must be transferred to a place where the full event can be assembled. In the ALICE data-acquisition architecture there is not a single object that can be identified as the event builder [2]. The responsibility for building the events is rather distributed among several processors and processes. The LDCs send the sub-events of a given event to the same GDC. The network links all the machines together, thus allowing all the GDCs to be accessed by all the LDCs (see Figure 2).

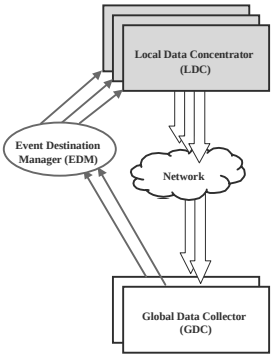

Figure 2: Interconnections between LDCs, GDCs and EDM

However, as far as the event building is concerned, the network has only a passive role, which consists of dispatching the sub-events to their destination. The decision on the destination is made elsewhere, in the way described below.

A process called Event Destination Manager (EDM) maintains a list of available GDCs and periodically sends it to the LDCs. The GDCs inform the EDM of their availability, according to their load. The LDCs use the list of available GDCs to establish the event destination. Each GDC list has a scope of validity referred to a range of event numbers. All the components of the EBDS process are synchronized through the event number.

The EBDS as a whole provides the following functions:

- Synchronization of the LDCs on the choice of the destination GDC.
- Back pressure from the GDCs to the LDCs in case of GDC congestion.

We are confronted here with a system that has the following features:

- There are real-time constraints, since it is necessary to keep up with the data flow.
- It involves inter-process communication and the establishment of transactions protocols.
- The algorithm used to determine the choice of the destination contains some heuristic elements.

All these features suggest that it is important to make a simulation of the system during the design phase. Furthermore, it would not be easy to adapt a hard-wired implementation to all the possible parameters and algorithms that we want to test. We therefore made, in 1997, a study of several major software design methods [3].

III. The Choice of ROOM as Design and Simulation Method

We eventually chose ROOM [4]. The aspects of ROOM that appeared to us particularly attractive are the following:

- The development process is based on the creation and the growth of models. The models capture our requirements and our view of the expected behavior. A fully-grown model eventually behaves as the target system, and it therefore becomes *the system*.
- The operational approach eliminates semantic discontinuities. There is a single set of modeling abstractions that are used throughout the development process, from analysis to design and implementation.
- A tool set exists (ObjecTime Developer [5]), which supports all the features of the method. In particular it provides:
  - A simulation run-time system that allows even incomplete models to be executed. This fosters an incremental and iterative development process and fast prototyping.
  - Code generation in C++.
  - A target run-time system where the developed application can run. The tool provides the visibility of the target process. Many platforms and operating systems are supported.
- There were indications of a convergence between ROOM and the Unified Modeling Language (UML) [6]. The UML was going to adopt ROOM to extend its formalism to real-time processes.

IV. The ROOM paradigms

ROOM is a method specific to the real-time domain, designed to handle the concepts of timeliness, dynamic structure of objects, reactiveness, concurrency and distribution.

The main paradigm used in the ROOM models is that of active objects, called *actors*, which communicate among them by exchanging messages (called *signals*).

The ROOM developer may use two distinct abstraction levels in his model, for which two different notations will be used:

- The schematic level, which uses the ROOM graphic language.
- The detailed level, in which a traditional programming language may be used, such as C++.

A ROOM model may be developed and made to grow along three independent dimensions:

- The structure, which represents the static topological aspects of the system. It describes the system components and their relationships in term of communication (ports and bindings) and containment (actor decomposition). The representation is made by graphs, as shown in Figure 3.

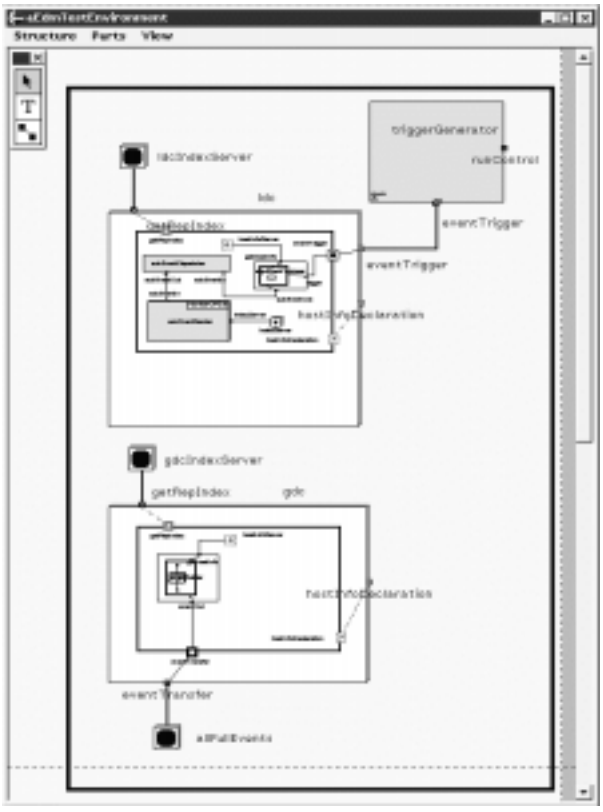

Figure 3: Structural view of a model

- The behavior, which represents the dynamic aspects of the system. It describes how the system changes over time. The representation is made by finite-state machines, as shown in Figure 4.

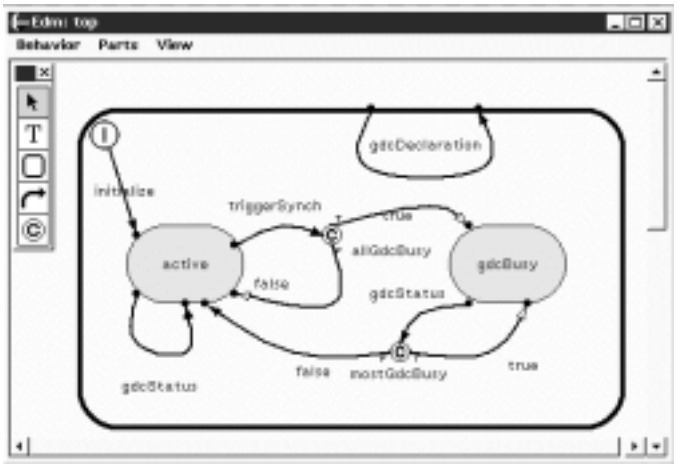

Figure 4: Behavioural view of a model

- The inheritance, which represents the class hierarchy. It concerns the classes rather than the instantiated objects. It allows abstraction and re-use of models.

V. The EBDS ROOM model

A ROOM model of the EBDS has been made using the ObjecTime tool set. It consists of two distinct parts.

The first part, called the EBDS environment, constitutes the simulation environment used to test the EDM. It includes a trigger generator, a number of LDCs and a number of GDCs. The trigger generator sends triggers to a number of LDC actors. The LDCs generate events and dispatch them to the GDCs. The LDCs contain three actors: the event generator, the event dispatcher and the event sender. The decision on where to send the event is taken by the dispatcher.

The second part is the EDM itself. It is made of an actor that keeps track of the status of the GDCs and periodically updates a status table. The update of the tables demands synchronization between the EDM and the event dispatchers in the LDCs.

The model has been built and then exercised in simulation mode. It has revealed a number of defects in the protocol, which have been rectified.

Currently, we have split the model, leaving the EDM separated from the EBDS environment. The environment continues to run in simulation mode, while the EDM runs in target mode on a remote machine, as a normal application. Therefore, we now have a powerful tool that allows us to test algorithms and system performance.

VI. Experience with ROOM and ObjecTime

We have used the ROOM method with the ObjecTime tool set for several months, over the last two years, to design and simulate an application of sizeable dimension.

The whole idea to work on a model from the very beginning, and to be able to execute the model - even if partially developed - is extremely attractive, especially in the field of high-energy physics data-acquisition systems. In a development environment, where the requirements are often fuzzy and change occasionally, the possibility to work on graphical models and to make prototypes at an early stage is very welcome.

A. *Strong points*

The ROOM paradigms confirmed that they are extremely powerful to make models in a well-defined domain of application, namely the real-time domain. We do not think that ROOM could be applied to other domains (e.g. physics-event reconstruction or analysis): this is not the ambition of the ROOM authors either.

The only slight inconsistency in the language, if any, is the fact that incoming and outgoing signals belong to different abstraction levels. While incoming messages are implicitly processed at the schematic level, when they trigger a state transition, outgoing signals must be explicitly generated in the code at the detailed level.

The ObjecTime toolkit is well made. It covers all the aspects of ROOM and gives access to all the features of the graphical language. The notation used matches exactly the one described in the manifesto book [4]. Due to the inherent complexity of ROOM, the notation is quite complicated and sometimes difficult to master.

The toolkit is very robust, even though it happened to crash or hang occasionally.

The toolkit provides a lot of facilities to manage the models in a collaborative team environment. Parallel concurrent versions (called *updates*) may be developed from a common baseline (called *context*). Eventually, various updates may be merged into a new context. Each user owns a *workspace* that preserves his own window configuration and any other settings he/she may have established.

The simulation environment is very powerful. It gives a lot of visual hints of the course of model execution. It allows setting of break-points and trace-points, and entering messages on a logbook. It is possible to inspect variables and almost all the aspects of the simulated model.

B. *Weak points in target mode*

The target mode gives the same sort of facilities as the simulation mode; unfortunately a model that runs happily in the simulation run-time system (SimulationRTS) is not guaranteed to run as well in the target run-time system (TargetRTS). Sometimes the tool crashes when using the Target Observability capability if the model contains errors and the application loses contact with the tool.

In order to run on distributed systems, models must be split by hand. Separated models must be built for each target processor.

The bindings between actors running on different targets must be converted into a special kind of communication called service access/provision point (SAP/SPP). This is normally

dedicated to interconnecting different logical layers of the model. In other words, the communication between different target processors is confined to the client/server paradigm.

The implementation of SAP/SPP over TCP/IP sockets is currently limited to the exchange of textual messages (ASCII-encoded), which may be highly inefficient for bulk data.

Moreover, there is a number of unfortunate inconsistencies between the SimulationRTS and the TargetRTS that must be taken into account, such as the indexing of multiple instantiated actors, differences in the routines for handling simulated time and real time, and different time formats. Some visual hints change when you run in TargetRTS, such as the messages appearing in the inject port's Trace window.

C. Other weak points

There are in ObjecTime a number of minor inconveniences, which do not seem to affect the overall good impression of the tool set; still, they make life a little more complicated for the designer. Here are a few of them.

The original human interface is neither instinctive nor user friendly. The use of buttons and menus in the window decoration is most unnatural. Things are a bit better since they have introduced the Microsoft Windows look and feel in version 5.2 of the tool set.

The tool set provides its own window management within its main window. Windows are very widely used for any entity you require to see; therefore, while you work, more and more windows are stacked on top of one another. Unfortunately, the concepts of inheritance and containment are not propagated to the windows; therefore, the user must manually close windows one by one in order to tidy up a cluttered screen.

There is no graphical indication of the status of SAPs and SPPs. Since SAPs and SPPs are necessary in a distributed environment, substantial features of the model do not appear in the graph, making it much less self-explanatory.

The remnants of an early proprietary language (RPL) appear here and there. You are not expected to learn it, but you cannot ignore it altogether, since some native classes are defined using the RPL and this will affect the derived class. When inspecting the variables at run time, the RPL will come to the surface again.

The run-time error messages are often obscure, in both working modes.

The built-in statistic tools available are rather poor.

VII. CONCLUSIONS

We have used ROOM and the ObjecTime toolkit for the development of the Event Builder and Distribution System of the ALICE experiment. We have found ROOM to be a useful development method for projects having a strong real-time connotation. The paradigms adopted by ROOM are very powerful to succinctly capture the features of the models. ObjecTime is a complete and robust development tool, notwithstanding some annoying awkwardness.

VIII. REFERENCES

[1] ALICE collaboration, Technical Proposal, CERN/LHCC 95-71, 1995.

[2] H. Beker, W. Carena, R. Divià, P. Vande Vyvre, A. Vascotto, ALICE Event Building and Distribution System - User Requirements Document, ALICE internal note INT-96-10, 1996.

[3] M. Macowicz, ALICE Data Acquisition System Control: Assessment of methods and tools for the development, ALICE internal note INT-98-02, 1998.

[4] B. Selic, G. Gullekson, P. T. Ward, Real-Time Object-Oriented Modeling, John Wiley & Sons, 1994.

[5] ObjecTime User Guide, ObjectTime Limited, 340 March Road, Kanata, Ontario, Canada K2K 2E4.

[6] J. Rambaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, 1999.