ALICE Data Acquisition System Control:
Assessment of methods and tools for the development

Maciej Macowicz
CERN EP/AID

Abstract

The purpose of the note is to assess the methods and tools to be used in the development of the ALICE DAQ System Control (ALICE-DSC) and to present the current status of the ALICE-DSC prototype. We first introduce the ALICE-DSC, then we define the criteria for assessing methods and tools to use during the ALICE-DSC development. In the following part of the note we overview reactive programming languages and CASE tools with respect to these criteria, then we present the design of the ALICE-DSC using ObjecTime: the CASE tool we chosen. In conclusion we outline the strengths and weaknesses of ObjecTime and we present the future workplan.

Contents

1 Introduction 2
2 Structure of the ALICE-DSC 2
3 Assessment of methods and tools 3
   3.1 Criteria for evaluation 3
   3.2 Programming languages evaluated 5
   3.3 CASE tools evaluated 6
4 ROOM and ObjecTime 8
5 ALICE-DSC using ObjecTime 10
6 Conclusion 12
1 Introduction

The ALICE experiment [1] at LHC (CERN) is foreseen for 2005. The ALICE Data Acquisition (ALD) group is working on the design of the Data Acquisition System (DAQ), presented on figure 1. The hardware used in the ALICE DAQ system consists of a number of processors (currently workstations and VME boards running flavours of Unix) connected via a network (currently Fast Ethernet). The ALICE DAQ system will perform a number of activities, controlled by the ALICE DAQ System Control (ALICE-DSC).

In this note we present the assessment of methods and tools to be used in the development of the ALICE-DSC. We start by introducing the structure of the ALICE-DSC and by defining the criteria for the assessment. Then we present the programming languages and the CASE tools we assessed with respect to these criteria. The assessment shows that the most suitable method for the development of ALICE-DSC is ROOM (Real-Time Object-Oriented Modelling) [19]; therefore, in the next part we overview the method and introduce the associated CASE tool ObjecTime.

The following part of the note targets the design of the ALICE-DSC prototype and ObjecTime experience we acquired. Finally in conclusion we judge the suitability of ObjecTime and we present the workplan for the future.

2 Structure of the ALICE-DSC

The purpose of the ALICE DAQ System Control is to control the activities of the ALICE DAQ system. The basic set of user requirements related to ALICE-DSC has already been identified [14].

The ALICE-DSC is being designed as a distributed object-based system (figure 2). The controllable elements in the ALICE DAQ consist of external objects (software processes) running on the DAQ hardware and acting directly on this hardware. External objects behave as stateless servers: upon the receipt of a command, they perform an action
Figure 2: Structure of the ALICE-DSC

and then return the related status information. External objects are also able to generate asynchronous status information when unsolicited events occur in the environment.

The DAQ activity is defined by the sequence of commands sent to the external objects. In order to define such activity, the ALICE-DSC contains several logical objects which behave in a state-oriented manner and send sequences of commands to related processes. Logical objects can be distributed through processors in the ALICE-DAQ and communicate via messages.

3 Assessment of methods and tools

3.1 Criteria for evaluation

The structure of the ALICE-DSC requires a development method and related tool(s). We identified three groups of criteria for evaluation of the suitability of methods and tools for the development.

Logical objects

One should be able to decompose the system under development in small independent parts. Such parts, logical objects should be characterised by following properties:

- **hierarchical organisation**: objects can be either organised in a flat manner or nested at any level of hierarchy. For the ALICE-DSC development we need the nested organisation.

- **communication paradigm**: objects should be able to communicate according to paradigms such as point-to-point and broadcast. The semantics of the communication can be synchronous or
asynchronous, although for practical reasons we prefer the asynchronous communication in the ALICE-DSC\(^1\).

- **state-oriented reactive behaviour**: the description of the behaviour is given using formalisms such as statecharts, synchronous automata or imperative programming languages. For the needs of ALICE-DSC we require a variant of finite state machines with nested states.

### Behaviour modelling process

The most important facet of the ALICE-DSC is its behaviour. We consider that the modelling of the behaviour is an important activity during the development process, so there is a number of criteria for evaluating tools with respect to this facet:

- **coverage of analysis, design and implementation**: since we consider the modelling of the behaviour as the most extensive activity in the ALICE-DSC development, we need the tool covering the broadest part of the modelling.

- **manipulation of the behavioural description**: depends on the formalism chosen to describe the behaviour. Since the most common formalism is based on state machines, we need a tool with the appropriate graphical or textual facilities.

- **verification of the behavioural description**: the possibilities of the verification depend on the formalism used to describe the behaviour. In most cases one can expect weak verification, allowing to detect unreachable states, unbound transitions and other static errors. However, the strong verification allowing to detect causality errors such as deadlocks is possible only if the formalism has solid theoretical foundations (precise semantics).

- **simulation facilities**: the tools used in ALICE-DSC development should provide the programmer with the simulation of the behaviour according to a given description.

- **support for debugging**: the tools used in ALICE-DSC development should provide the programmer with the observation of the behaviour of target. Such observation can be considered as high-level behaviour-oriented debugging and should be as non-intrusive as possible.

### Passage to the real world

The tools used in ALICE-DSC development should provide the programmer with the following facilities related to the deployment of the system:

- **external objects** are specified by their interface: the tool should allow the definition of such interfaces and should provide the facilities for the communication between logical and external objects.

- **control through Application Programmer Interface (API)**: the tool should provide the interface for submitting commands from external applications.

- **programming language for the behaviour**: it should be possible to enrich the behavioural description by programming in a standard language (e.g. C or C++).

\(^1\)The term synchronous is related to the propagation of events: we consider a system is synchronous if all the objects receive messages at the same time (no communication time should be taken into account). The system is asynchronous if the communication time is observable.
- **hardware platforms for development and deployment**: The ALICE-DSC prototype is being developed on Unix workstations (currently Solaris/AIX) and is deployed on Unix workstations and VME boards (currently MVME-2600 under AOS).

- **support for distributed programming**: The programmer should be provided with the tools taking into account the distributed nature of the ALICE-DSC. The support for distribution should be based on TCP/IP sockets. The distribution should be done in a semi-automatic manner, according to the configuration, giving the physical location of logical objects. The tool should help the programmer in setting up the topology of connections and should load the respective logical objects onto given locations.

At the beginning of the evaluation we were convinced that a reactive programming language would be sufficient for the behaviour modelling process, but during the evaluation we realized that the behavioural aspects of the ALICE-DSC require more methodological approach, involving the application of systematic development method and supporting CASE tool.

### 3.2 Programming languages evaluated

**CHSM** (Concurrent Hierarchical State Machine) is the extension of C++ allowing the description of reactive behaviour by CHSM-charts, i.e. state machines with nested states [13, 17]. The semantics of CHSM-charts is close to Harel's statecharts [10].

CHSM is a preprocessor producing target C++ code from textual representation of CHSM-charts augmented with C++ statements. CHSM is implemented in C++, lex and yacc. Basic debugging facilities such as displaying of states and transitions are also available. We prototyped a remote observation tool for CHSM-charts.

CHSM does not include the verification of the behaviour, except syntax errors. It is to note that exhaustive verification of statecharts can be done only when additional constraints on the communication (synchrony hypothesis) can be assumed [2].

CHSM allows us to express only the behavioural aspects of the system under development, so its application is limited to describing one logical object; the structure of the system under development should be given by other tools and formalisms. For example, in the ATLAS collaboration, CHSM is used for completing OMT model of Run Control [16]. The experience of ATLAS shows that the behaviour modelling process requires simulation and observation facilities [6]. Finally, no distribution aspects can be taken into account in the behaviour.

**Argos** [15] is a variant of statecharts (namely Argos-charts) based on perfect synchronous communication hypothesis [9]. This hypothesis allows us to perform the exhaustive verification of the behavioural description and to detect major causality errors (such as e.g. deadlocks) in advance.

We prototyped the ALICE-DSC behavioural part using the Argos development environment implemented at GMD (German National Research Center for Information Technology). This environment contains the graphical editor of Argos-charts, compiler, verifier and simulator. It is written in Tcl/Tk (editor and simulator) and CAML-Light (compiler and verifier). Argos compiler produces the skeleton of target code in C, the programmer should complete this code by specific routines for interactions with the environment under control. Finally, structural aspects can be expressed using nested logical processes.

Argos appeared as a convenient programming environment for developing automata (behavioural aspects) but it does not include directly the facilities related to distribution. It is a tool suited to hardware/software co-design (very simple controllers) rather than complex, distributed control systems.
Finally, it is worth noting that Argos can be compiled into OC target code, which can be distributed automatically according to SPMD paradigm \cite{8}; such distribution is based on physical location of stimuli.

**Esterel** \cite{4} is a reactive programming language based also on perfect synchronous communication hypothesis. The complete development environment (compiler, verifiers, simulator) is available on Unix platforms from INRIA (French National Institute for Research in Computer Science and Control). Esterel allows us to decompose the system under development in terms of flat logical processes. The process behaves in a manner described through special reactive statements. The Esterel compiler produces the skeleton of C code which should be completed with stimuli acquisition and emission routines. External objects can be defined in Esterel in terms of asynchronous external tasks (implemented as Unix processes).

During the prototyping of the ALICE-DSC using Esterel we also tried the automatic distribution of code. Nevertheless, the main drawback of Esterel is the complexity of reactive statements used to describe the behaviour.

### Conclusion on the use of languages

We decided to use none of the languages described above. At modelling level, the problems of behavioural description should be coupled with the structure and none of the languages allow to do this. The use of reactive programming language to describe the behaviour and other means to describe the structure is error-prone, since there is no global vision (including structure, distribution and behaviour) of the system under development. The overview programming language facilities is presented in the table 1.

### 3.3 CASE tools evaluated

We selected the CASE tools suitable to the real time systems between those evaluated at CRIM \cite{7}.

**OMT-based CASE tools**

The OMT (Object Modelling Technique) method was introduced by Rumbaugh \cite{18} as a general-purpose, object-oriented software development method. When trying to apply OMT to model ALICE-DSC, we encountered a number of problems related to the structure of logical objects. In fact, OMT is based on flat model of objects, so the hierarchy of objects can only be expressed by associations and aggregations, which is not suitable to our needs. The behaviour of objects is described by statecharts with transitions bound to method’s calls.

We first tried to prototype the ALICE-DSC using the CASE tool **Rational Rose** release 3 (very unstable in the Unix environment). The coverage of the development process by Rose is limited to the analysis and design stages, since the code generated by this CASE tool consists of skeletons of classes and should be completed manually. Rose does not provide the generation of code for the dynamic part of the model (statecharts), nor the customisation of the code generator.

The second OMT-based CASE tool we tried was **Software through Pictures (StP)** from Aonix. StP appeared as a tool much more stable and flexible than Rose: it allows the programmer to customise the generation of code. StP/OMT is being used by the ATLAS DAQ group at CERN, and produces the CHSM code for the dynamic part of the model. However, StP does not provide complete support for the deployment and while it remains useful in analysis and design stages, its utility is very limited in the implementation stage.
In conclusion, concerning the use of OMT method in the ALICE-DSC prototype development:

1. The method is not completely suitable to the needs we identified (no hierarchy of logical objects).

2. CASE tools available for OMT method introduce big discontinuity in the development process.

**Statemate and Rhapsody**

Statemate [11] is a CASE tool supporting specific method which targets real-time systems. Interesting from the practical point of view, (no discontinuities in the development process), this method provides no support for distributed programming. In fact, Statemate is much more adapted to hardware/software co-design inside small controllers than to the development of complex distributed control systems.

Rhapsody is the successor of Statemate which integrates the UML framework. The field of application of Rhapsody remains the same as it is for Statemate.

**ObjecTime**

Finally, ObjecTime is a CASE tool based on the ROOM (Real-Time Object-Oriented Modelling) method and targets distributed, real-time systems. Since the ObjecTime toolkit ensures the complete coverage of the ALICE-DSC development process, we decided to evaluate it in depth.
### Table 2: Main CASE tools evaluated

<table>
<thead>
<tr>
<th>Logical Objects</th>
<th>ObjectTime</th>
<th>Statemate</th>
<th>Rose+StP/OMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchy</td>
<td>nested actors</td>
<td>nested objects</td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>one-to-one and broadcast</td>
<td>synchronous broadcast</td>
<td></td>
</tr>
<tr>
<td>Behavioural description</td>
<td>ROOMcharts</td>
<td>statecharts</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Behaviour modelling process</th>
<th>ObjectTime</th>
<th>Statemate</th>
<th>Rose+StP/OMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Editing</td>
<td>graphical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verification</td>
<td>weak</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulation</td>
<td>complete simulator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Debugging</td>
<td>observation of target</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Passage to the Real World</th>
<th>ObjectTime</th>
<th>Statemate</th>
<th>Rose+StP/OMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>External objects</td>
<td>using the API library provided with the tool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control through API</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming language supported</td>
<td>C++</td>
<td>C, Ada, VHDL</td>
<td>C++, Java, Ada</td>
</tr>
<tr>
<td>Hardware platforms</td>
<td>Unix+WinNT</td>
<td>Unix+Windows</td>
<td>Unix+WinNT</td>
</tr>
<tr>
<td>Distributed programming</td>
<td>manual</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4 ROOM and ObjecTime

#### Elements of the ROOM method

The ROOM method [19] targets the development of distributed real-time systems. It is based on two principles:

- The system developer should be able to progressively construct and test complex elements, using simpler elements (executable models can be used to represent such simpler elements).

- The system developer should be able to represent the system with the same set of concepts through all phases of the development (there is no semantic gap between development phases).

The means provided by the ROOM method during the development are the following:

- A graphical modelling language, which allows to describe the system under development in terms of active objects (actors).

- A compiler, which translates the system description expressed in the modelling language.

- A virtual machine, which is able to execute the compiled system description.

The system under development is considered at three abstraction levels:

- at the system level, the system under development is split in layers, described in terms of services it provide. Layers are organised in hierarchical manner: they provide the services to upper layer
through Service Provision Points (SPP) and access the services offered by lower layer through Service Access Points (SAP). The structure of system level is analogous to the OSI/ISO network model.

- at the **concurrency level**, the contents of a layer is specified in terms of actors. An actor possesses a number of **ports** for communicating with other actors via **messages**. The communication is **typed** in the sense that each actor obeys to a specific **protocol**.

The design at concurrency level is done following three modelling dimensions:

- the structure of the actor, i.e. its ports and its internal composition (the actors can be nested).
- the behaviour of the actor, expressed in terms of finite-state machines in which the transitions are bound to messages appearing on the actor’s ports.
- the inheritance relations between actors and protocols.

At concurrency level the **visual modelling language** (VDL) is used.

- at the **detail level**, the system description should be completed with specific code written in the target programming language (C++ or C).

**Development cycle with ROOM**

The authors of the ROOM method propose a software development process based on three phases:

- during the **requirements phase** one has to specify the external behaviour of the system. The ROOM requirements model is built and used for interaction with customers and between team members. The requirements model is intended to answer the question “**What the system should do?**”. It is worth noting that the establishment of the requirements can be done using Use-Case Maps [5], a variant of Use-Cases [12].

- during the **architecture/design phase** the ROOM requirements model is transformed into ROOM design model, focused on the question “**how do the system in order to meet the requirements ?**”. At the design phase the environment under control is “**stubbed out**” in the design model and the model is typically executed using the simulator.

- during the **implementation phase** the design model is progressively enriched with more and more information about the environment under control, until it becomes the deployed system. This involves:

  - replacement of each stub by real elements interacting with the environment under control.
  - progressive port of the ROOM design model form the simulator to the target platform.

**ObjecTime toolkit facilities**

ObjecTime is the only CASE tool supporting the ROOM method; it consists of:

- **Editors**, to represent graphically the structure and the behaviour of the actors. The behaviour editor allows the programmer to define conditions and actions for the state transitions in the behaviour.
• **Browsers**, to browse lists of actors, protocol classes or data classes. Version management is also available at model level in browsers.

• **Monitors**, to watch the model during its execution.

• **Compiler**, translating the model into target language (C++ or C) source file. The compiler accepts the graphical description of structure and behaviour, completed with target language code for actions and conditions.

• **Virtual machine implementation**, to execute the ROOM design model on development platform (in *simulation mode*) and on deployment platform (in *target mode*). The virtual machine code should be linked with code produced by the compiler.

5 **ALICE-DSC using ObjecTime**

**General architecture**

As we mentioned above, we assume the controllable elements in the ALICE DAQ to be software processes, which behave as stateless servers. We put these processes in a special, external objects layer (figure 4). Other layers we defined in the ALICE-DSC design model are the following:

- the **HCI layer** contains the human-computer interface (application *external* to ObjecTime).

- the **Run layer** contains the actor *Run* coordinating the behaviour of DAQ subsystems such as Front-End Processors or Workstations. It is worth noting that the notion of DAQ subsystem is flexible enough to cover DAQ hardware subsystems (e.g. DAQ hosts) as well as DAQ logical subsystems (e.g. event builder which runs on several DAQ hosts).
- the **DAQ subsystems layer** contains the actors controlling the behaviour of the related DAQ subsystems. Such actors are listening to commands from the *Run* object and send the commands to the appropriate external objects.

**Interface to the external world**

The ObjecTime model is connected to the external world (i.e. to external applications) via the layering mechanisms. From a programmer’s point of view, the connection is implemented by Service Access Points (SAP) and Service Provision Points (SPP).

ObjecTime Run-Time systems (i.e. implementations of ROOM virtual machines) provide the implementation of SAP/SPP using TCP/IP sockets, available to external applications through predefined C++ classes. We developed external objects and Human-Computer Interface using C++ and Tcl/Tk toolkit.

When the communicating entities are running on the same host, the communication via SAP/SPP can also be bound to the “Target IPC” mechanisms such as shared memory or message queues.

**Structure of the Front-End Processor**

There are five actors and three external objects running on Front-End Processor (figure 5):

- the *Daemon* is responsible for the correct initialisation of the *Controller*, and for the communication between the upper-level layer and the controller.

- the *Controller* controls the overall activities of the Front-End Processor and contains the logical objects *Monitor*, *Readout* and *Recorder*, controlling the respective external objects.

- **External objects** are stateless “servers” (Unix processes), executing **requests** submitted by respective logical objects and reporting unexpected events occurred in the environment under control. External objects communicate with respective logical objects via SAP/SPP bound to TCP/IP sockets.
6 Conclusion

ROOM and ObjecTime strengths

We found the ROOM method and the ObjecTime tool well suited for the ALICE-DSC development. The ROOM method captures the hierarchical and reactive nature of the ALICE-DSC and provides an object-oriented modelling facilities at a very high abstraction level (actors and protocols). The ObjecTime toolkit completely supports the method.

ROOM also allows us to take into account the requirements related to the system under development. ObjecTime provides the traceability of the requirements throughout the development phases.

Another strong point of ObjecTime is the early executability of design models, completed with a set of simulators and observers, providing a truly incremental development cycle.

Finally, ObjecTime is a complete development environment including generation of executables, support for deployment, version management and generation of design documents.

Weaknesses: ROOM method

Currently the ROOM method targets only distributed, real-time systems. This means that a ROOM model can only cover the “control” aspects of the system under development. Other aspects of the system (such as e.g. data handling) should be designed using other methods and associated CASE tools. The interoperability between CASE tools seems then to be very important in perspective of systematic development.
The ROOM method should evolve towards the UML-RT method; interoperability between Objec-Time and Rational Rose is foreseen for the mid 1998.

Weaknesses: ObjecTime toolkit

The main problem we encountered during the evaluation was the poorness of the documentation. While the manipulation of the toolkit is intuitive and the learning period relatively short, the integration of external applications can be problematic because of the poor documentation of the run-time systems, both on paper and on-line versions.

Future

We developed the complete prototype of the ALICE-DSC for the Front-End Processor only. The next step of the project will consist in the integration of ObjecTime models into the existing Run Control System of ALICE-DAQ prototype [3]. This will require the following steps:

1. the precise description of currently used external objects, such as e.g. readout and recorder.

2. the design and development of the ALICE-DSC loader, to load the ObjecTime model on distributed DAQ hardware. This step will also include the evaluation of the scalability of deployed ObjecTime models.

3. the completion of the ObjecTime model to control the ALICE-DAQ activities.

We have been using ObjecTime for evaluation period of six weeks, granted by the manufacturer; now we are waiting for the test licence, expected for January 1998. We plan to complete the prototyping phase of the ALICE-DSC in order to obtain a product similar to the currently used DATE run control system.

References


