OASIS: A NEW SYSTEM TO ACQUIRE AND DISPLAY THE ANALOG SIGNALS FOR LHC

S. Deghaye, D. Jacquet, I. Kozsar, J. Serrano, CERN, Geneva, Switzerland

Abstract

To cope with the user requirements [1] for the Large Hadron Collider (LHC) era, a new system has to be developed for the acquisition and display of analog signals in the accelerator domain. OASIS, the Open Analog Signals Information System, extends the capabilities of the existing 'new Analogue observation system' (nAos) in many fronts: new acquisition hardware is supported to provide higher analog bandwidth to the users, a new distributed triggering scheme has been designed for greater flexibility and the software has been completely redesigned with openness in mind to allow easy upgrading and maintainability. This paper describes the architecture chosen to satisfy the new user requirements. The solution involves three tiers and makes use of the J2EE platform to communicate between the GUI and the middle tier. The front-end tier runs on Linux PCs in CompactPCI format, to support fast digitizer modules. A first prototype system, used for the extraction tests from the Super Proton Synchrotron (SPS) to the LHC, is also presented.

MOTIVATION

In addition to the user requirements brought by the LHC, there are also strong hardware and software arguments to move from nAos to OASIS.

Manufacturers are no longer building high performance digitizers in the VXI format. Nowadays >1GS/s modules are in CompactPCI format (CPCI).

Concerning software, the maintenance of the current system starts to become difficult with a Graphical User Interface (GUI) written in C using X/Motif, a domain where experts are becoming a rare resource. Moreover the application running in the front-end computers is not based on our standard framework [2] neither for the equipment access server nor for the real-time tasks.

Finally OASIS will give us the opportunity to exploit in a more efficient manner the digitizer modules, which are the most expensive parts of the system, and to distribute more coherently the responsibilities among the different layers, which will bring openness to the system.

The remaining of this paper will focus on the architecture of the system. Section 2 gives a view of the general architecture. Then, sections 3, 4 and 5 give more details on the front-end layer, the middle tier layer and the application layer respectively. The 6th and last section will give test results and what is foreseen for the following developments.

GENERAL ARCHITECTURE

We chose for the OASIS system a three-tier architecture as depicted in figure 1.

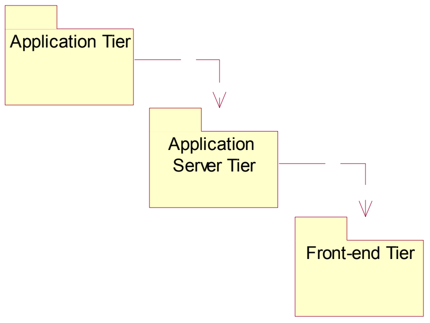

Figure 1: General architecture

The front-end tier has the responsibility to handle the hardware, the digitizer and the multiplexer modules. It provides hardware independent interfaces to the upper tiers.

The application server tier has the responsibility to manage the resources, which are the acquisition channels; it routes the analogue signals to the best oscilloscope input and ensures the settings coherence among different modules. It is the mandatory passage for all the applications wanting to access the hardware. It brings into the system a level of indirection between the acquisition data and the hardware that carries out this acquisition allowing us to re-assign the digitiser hardware at run time.

The application tier is itself separated in two layers. The lower layer, a client package, is a model of the virtual oscilloscope system. It gives access to all the classical oscilloscope features plus OASIS specific functionality. The other layer is the GUI, based on a VirtualScope component and uses the services offered by the client package. The VirtualScope component is intended to be reused in every application that has to work with analogue signal acquisitions. It is foreseen to have several applications with different features depending on their domain of use. The first application that has been developed is a viewer for the SPS extraction tests that were made during September 2003.

The communication between the front-end tier and the application server tier uses the Control Middleware [3] (CMW), a CORBA based communication protocol that allows the application server tier to get, set and subscribe to the properties exposed by the equipment access server.

For the dialog between the two upper tiers, we use either RMI/IIOP for synchronous commands or JMS for the asynchronous ones.

FRONT-END TIER

The front-end computer is based on a CompactPCI crate with an x86 CPU board running GNU/Linux. We use Acqiris DC270 digitizers [4] and Pickering matrices [5] for the trigger signal routing. The modules are accessed through a kernel mode device driver either written by us or provided by the manufacturer.

It is foreseen to install these crates in technical buildings as close to the analogue signal sources as possible. Since those places are not very accessible, we needed to improve the reliability of the crate and that is why we bought diskless CPU boards and we made a diskless network bootable image (NBI) of the GNU/Linux operating system using the mknbi-linux program [6].

The higher level software is based on our standard framework [2] composed of an equipment access server using the CMW and a real-time task communicating with the server through a shared memory segment.

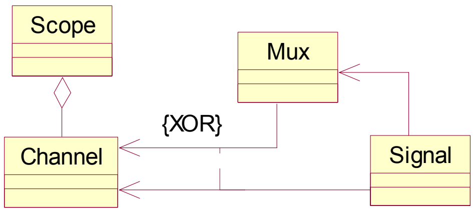

Figure 2: Signal interfaces exposed by the front-end tier.

The front-end computers provide the application server tier with several interfaces corresponding to the different services implemented by the hardware. Obviously, those interfaces need to be independent of the hardware that performs the function.

Figure 2 shows the analogue signal part of the system. We have channels, which are contained in scopes, and signals, which represent the analogue signals to be acquired. We can connect these signals to the channel input either directly or using the multiplexer. This last option will increase the number of signals that can be connected to a given crate. Of course this solution has a cost: the availability of each signal is decreased.

Figure 3 depicts the external trigger sub-system. Each oscilloscope module has an external trigger input. We can connect the trigger signal to this input either directly or through a matrix, an n-to-1 matrix. This solution is the most used one since it allows the user to change the trigger remotely.

The last feature is the possibility to add a counter between the matrix output and the external trigger input. This counter can count machine related clocks such as the revolution frequency.

Each interface presented in figures 2 and 3 has a set of properties the application server can work with. For example, the interface 'Channel' has a property AQN that contains the last acquired data.

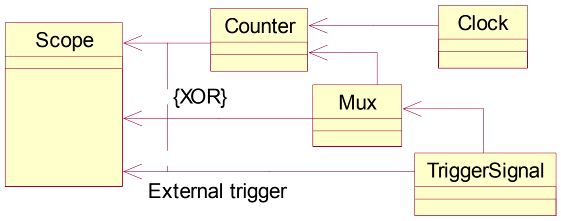

Figure 3: Trigger interfaces exposed by the front-end tier.

APPLICATION SERVER TIER

We use, for the middle tier computer, a standard Pentium IV machine clocked at 2.4 GHz with 1 GB of memory. This machine runs RedHat Linux Enterprise [7] and the Oracle J2EE container OC4J [8]. The latter is the container where we deploy our Enterprise Java Beans (EBJ) [9] that implement the application server logic.

For the OASIS system, we use three different types of EJB: container managed persistence (CMP) entity beans for the persistence, session beans and message driven beans (MDB) for synchronous and asynchronous use cases respectively.

With the CMP entity beans, we just need to develop an abstract class with the getter/setter methods for persistent data and to describe in an XML file how these data map to the Oracle database. Then the container is able to generate all the JDBC calls to implement the entity beans. Furthermore, we give the container the exclusive write access to the relational tables to improve access time by enabling the cache mechanism. For administrative purposes, we develop web pages that display the information stored in the database such as the current connections, the applications that use these connections, etc.

The session beans are used when the work carried out by the application server produces a result that is needed by the application to continue, for example, when the application asks for the trigger signals available to acquire a given signal. The heart of the application server is the 'ConnectionManager' session bean. This bean has the responsibility to connect the analog signal to the most appropriate hardware module taking into account the priority of the client asking for the connection. It has also the responsibility to publish the acquired data on the correct JMS topic after having resolved the indirection between the digitizing device, unknown by the application layer, and the logical connection, known by the application layer. This last part is the most 'real-time' part of the system since we aim at a delay between the end of the acquisition and the display in the application of about a hundred milliseconds.

Finally, we have a MDB to handle the settings changes made by the user. This bean has the responsibility to keep coherent the settings of the different hardware modules belonging to a virtual scope.

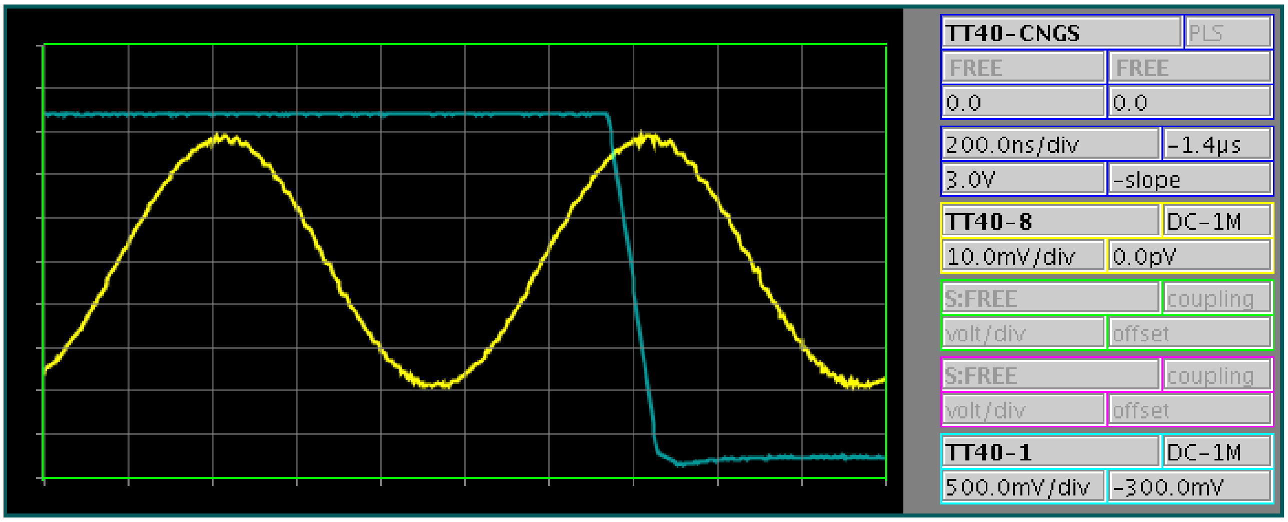

Figure 4: The virtual oscilloscope component.

As an example consider a virtual oscilloscope (VScope) with two signals connected but these signals are in fact digitalized by two different modules. When the user wants to change the trigger delay, the MDB has to find out which modules are used for this VScope and to change settings on both modules.

APPLICATION TIER

This last tier is the user interaction part. This is a standalone Java application launched using JavaWebStart. The application is composed of two parts.

The first part is the client package; it relies directly on the application server, communicating with it through either RMI/IIOP or JMS. It exposes to the upper layer a model of the virtual oscilloscope system with concepts like VScope and VTrace. It hides the use of J2EE from the GUI which allows us to change the application server implementation if needed.

The second part is the graphical user interface which is the non-generic part of the system, presenting specific features that depend on the application domain. The GUI uses services provided by the client package like signal menu, signals set definition and connection/disconnection demands. The central part of any application involved in analogue signals observation is the VirtualScope, of which a snapshot can be seen in figure 4. Its development is made around the EdPlot bean which is a plot bean developed at CERN. The VirtualScope component behaves like a four-trace oscilloscope and is the view/controller counterpart of the client package. Moreover, the GUIs that host the VirtualScope, one or several instances of the VirtualScope, can use complex features provided by it like cursor measure, envelop drawing and surveillance and reference display.

TEST RESULTS AND FUTURE

In September 2003, there were two days of SPS extraction tests. These were the first milestone for the OASIS system since we had to provide a way to monitor the kicker pulses used to extract the beam and the circulating beam. This very first prototype of the OASIS system only provided basic functionality. The prototype worked well even if some tuning is still to be done on the transport delay between the front-end computer and the application.

In the future, we plan to extend the system to the VXI modules used in the previous nAos system. For that, we will need to develop front-end software that controls the VXI modules and expose the same interfaces as the ones depicted in figures 2 and 3.

On the application server side, we plan to replace the application server machine by a HP ProLiant DL380 G3 [10] with two Pentium Xeon at 2.8 GHz and 2 GB of RAM. Furthermore, we will continue adding functionality such as advanced routing algorithms in order to improve the signal availability.

We will also continue the development of the VirtualScope component.

REFERENCES

[1] S. Deghaye, “OASIS: Requirements specification”, CERN LHC-CP Signals Working Group (SiWG) Internal Note 2003.
[2] A. Guerrero et al., “CERN Front-end Software Architecture for accelerator controls”, ICALEPCS’03, Korea, October 2003.
[3] K. Kostro et al., “Controls Middleware (CMW): Status and use”, ICALEPCS’03, Korea, October 2003.
[4] http://www.acqiris.com/Products/Digitizers/
[5] http://www.pickeringswitch.com
[6] http://www.etherboot.org
[7] http://www.redhat.com/software/rhel/
[8] http://otn.oracle.com/tech/java/oc4j/
[9] E. Roman et al., “Mastering Enterprise JavaBeans”, second edition, Wiley Computer Publishing, 2002.
[10]http://h18004.www1.hp.com/products/servers/proliantdl380/