OASIS: STATUS REPORT

S. Deghaye, L. Bojtar, Y. Georgievskiy, J. Serrano.

CERN, Geneva, Switzerland.

ABSTRACT

OASIS, the Open Analogue Signal Information System, is the LHC era system for the acquisition and display of analogue signals in the accelerator domain. OASIS is based on a three-tier architecture. The upper layer is an application written in Java. It is responsible for waveform display and user interaction. The application server, in the middle tier, uses the OC4J J2EE container and is based on the Enterprise Java Beans. This part takes care, among other things, of the signal connection and the management of the connection settings. The front-end tier is implemented within the FESA framework and controls oscilloscopes and analogue matrices in CompactPCI and VXI format.

Today, most of the performance critical features such as the mountain range mode or the scrolling mode have been implemented. However, the problem of remote trigger has still to be addressed. The system was successfully put in production for the TT40/TI8 tests with a small configuration. The next, and bigger, installation is the LEIR machine where OASIS has to be available twenty four hours a day for a longer period and has to handle more than 200 signals.

INTRODUCTION

OASIS is a system for the acquisition and display of analogue signals in the accelerator domain. The signals, distributed all around the accelerators, are digitalised by oscilloscopes sitting in front-end computers (FEC). The acquired data is sent through the Ethernet network and displayed on a workstation running a dedicated application. When the bandwidth requirement allows it, the analogue signals are multiplexed by analogue matrices which are connected to the oscilloscope channels. This scheme takes into account the fact that not all signals available are observed at the same time and allows us to save some digitisers, the most expensive devices in the system. The FECs are installed next to the signal sources in order to preserve the signal integrity as much as possible.

OASIS provides the Virtual oscilloscope abstraction (Vscope). A Vscope is a software oscilloscope that takes its data from different hardware oscilloscopes and displays it as if it came from the same module. Thanks to this scheme, we are able to observe several signals as if they were next to each other while they are actually distant from hundreds of meters. Of course, for this to work, we need to have the same trigger pulse and OASIS must keep in synchronisation the acquisition settings used by the different connections belonging to the same Vscope. OASIS also proposes other convenient functionalities such as predefined set of signals, mountain range display or peak detection mode.

GENERAL ARCHITECTURE

The architecture chosen to implement the OASIS system is three-tier. Figure 1 shows a view of the system along with the main technologies used to implement it.

The front-end tier controls the hardware modules (oscilloscopes, multiplexers, and counters) and provides to the upper layers a hardware independent interface using FESA [1]. Should you be specially interested by this interface, [2] explains it in minute details.

The application server manages the resources provided by the front-end interface and assigns them to the connections requested by the clients. The connection algorithm always tries to maximise the number of concurrent acquisitions. It is based on a priority mechanism whereby existing connections can be deleted if resource shortage requires it. It has also a master/slave mode thanks to which connections held by a client, the master, can be shared in read-only mode with others, the slaves. In addition, the application server implements the Vscopes abstraction and controls the associated acquisition settings to keep them coherent and give the virtual oscilloscope image. The middle tier implementation is based on the J2EE Enterprise JavaBeans standard [3] and runs in the Oracle Container for J2EE (OC4J) [4].

The application tier, written in Java, provides the users with the graphical user interface (GUI); one can see the Vscope component in the topmost part of Figure 1. In addition, this tier exposes a Java client interface. This interface is the only client entry point to the OASIS system. Anyone who would like to profit from the OASIS services has to interface at the application level whether or not he wishes to reuse the graphical components available.

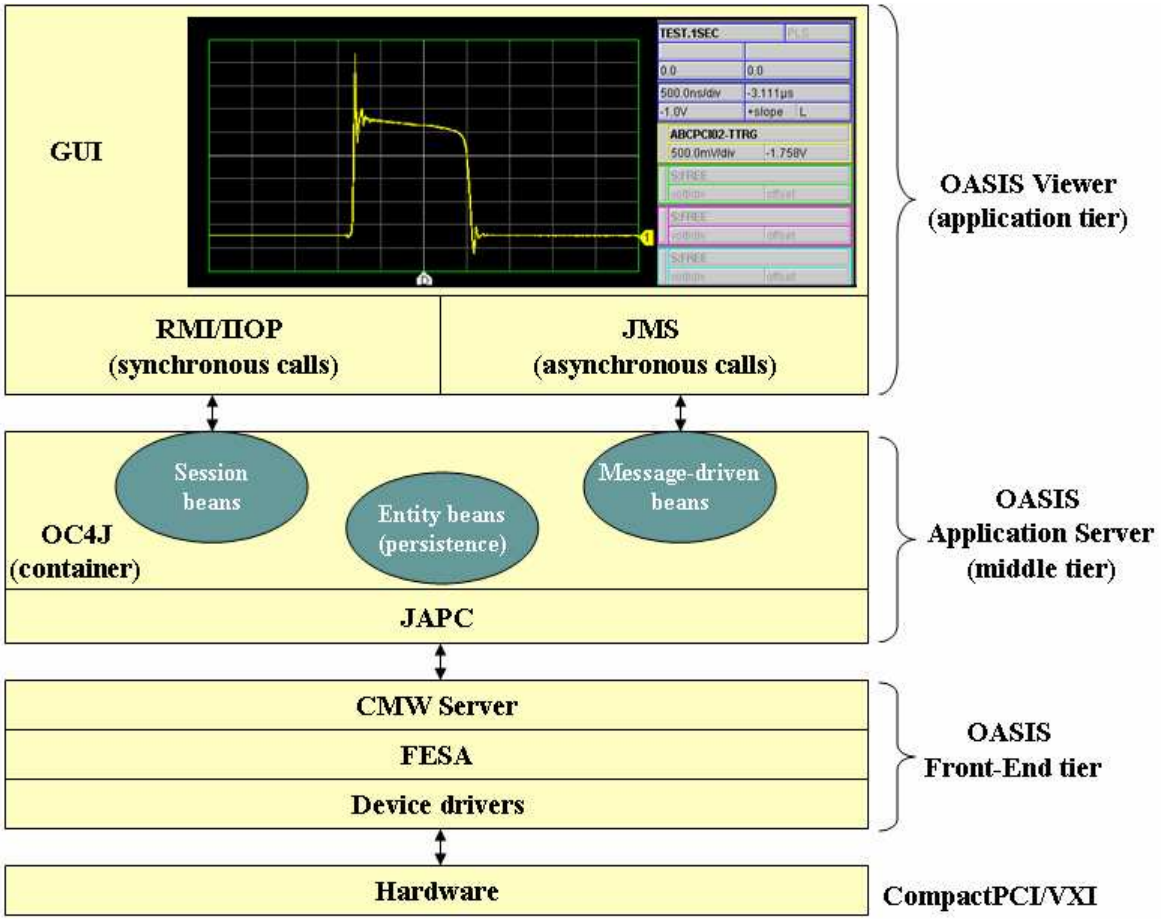

Figure 1: The OASIS architecture

This architecture suits very well the OASIS needs since important processing can be centralised in a single place, the OASIS application server. Since the server is the only component to have a global view of the current system state, the connection requests and the acquisition setting management are performed there. Furthermore, the Vscopes are created and managed on the server as well, which eases their sharing (master/slave mode) and the implementation of functions where the hardware and the Vscope have different settings, such as in the scrolling mode for example.

Nonetheless, this architecture has a major drawback, it introduces an overhead in the data processing and this can be fatal to some performance critical features. Before explaining how we have worked around this limitation, next section gives the general scheme for the inter-tier communication.

INTER-TIER COMMUNICATION

The communication between the middle tier and the front-end tier is based on the Control Middleware (CMW) [5] and uses the standard AB/CO JAPC [6] services to access it.

For the communication between the EJBs living on the application server and the application tier, the J2EE proposes two solutions. We can use either RMI/IIOP for a synchronous communication between the Java application and a Session bean or JMS for an asynchronous communication with a Message Driven Bean (MDB). We decided to use both to have some flexibility in the implementation of the different use cases.

Synchronous communication

This kind of communication is used in two situations, when the application cannot go further without the result, for example the menu browsing methods, or when the request leads to an application state change, such as the connection request method. In this latter case, keeping the GUI free without waiting for the requested action to complete would lead to unstable situations. Indeed, the state of the application being under modification by the server, not all the actions previously available are still allowed. For example, when we connect a signal on a channel, this one becomes busy and cannot be selected for another connection request.

On the server side, the requests are received by Session beans. Depending on the method called, the container running the beans may start a transaction that will guarantee the coherence of the system state in case of error. For example, the connect method is executed in such a context and, therefore, should a request fail for an unforeseen reason, the container will automatically roll back every modification made since the transaction start-up. In OASIS, the Session bean does not implement the behaviour, at maximum it checks the parameters, but forwards it to a POJO that performs the requested action. It is therefore ensured that a change in the technology or in the communication type is easier.

This communication mode has several advantages. It is easy to return a value or an exception. If needed, we can return any object that supports serialisation and the client will receive it immediately. The same applies to the exceptions. The synchronous communication is also easier to put in place. It is more or less just a method call and the result is strongly typed so we benefit from the compile time type checking. Of course, the induced strong coupling has also a bunch of drawbacks. If the server is slow or the processing rather long then the application is slow as well. The communication is always initiated by the application, never by the server. This can be a problem when an action undertaken by a client has an effect on another one such as when a Vscope is shared by two applications (master/slave mode), for example. Finally, if a request results in more changes than expected, we cannot return them since the method has a strongly typed return value.

Asynchronous communication

The JMS communication is used in the two cases where we meet the limitations of the synchronous mode explained above. The first and more important one is the virtual oscilloscope settings. The virtual oscilloscope is implemented in the application server by two classes, the Vscope and the Vchannel, backed up by entity beans. These two classes are made of properties, most of them being the copy of the settings defined in the Front-end interface [2]. The property values are initialised by the server upon connection and can be changed by the user through the OASIS Viewer. The asynchronous communication is also used for the administrative channel that every application has and which is used, among other things, to kill a connection or check periodically that the applications are still alive.

Why do we need to use this communication mode in these two situations? JMS was the only solution for the administrative channel since all the messages going through it are initiated by the server. For the Vscope settings, there are several reasons. First, the processing of a setting change request can be slow. Indeed, a Vscope can be composed of up to four different oscilloscopes and some setting changes imply operating electromechanical devices which are known to be slow. With JMS, the client is not blocked and can continue while the actions on the hardware are being made. Second, the settings in an oscilloscope are often coupled between each other. For example, the maximum pre-trigger delay time depends on the time span. This means it is very likely that we have several changes for one change request. With JMS, the updates are just published to the client and the number of changes to send does not pose any problems. Finally, when we have a Vscope with slave clients, we also have to send the changes to several destinations; a completely transparent operation with JMS.

The JMS communication has other advantages as well. Since the client is not blocked during a setting change, (s)he can send several of them in parallel. They will be treated one after the other on the server and should two of them collide, the last one will be rejected and no update will be published for it. Moreover, the setting changes being the most common request received by the server, we are free to make some load balancing if the growth of the system requires it.

Unfortunately, this scheme has a major drawback for OASIS. The acquired waveform is a property just like the others which means that an update arrives first on the server in a CMW message and then

the server publishes it to the client(s). While it is very flexible since it allows the server to re-assign a connection to other hardware modules without notifying the clients, the delay induced by this 3-tier scheme is too long, at least for some acquisition modes such as the scrolling mode or the mountain range mode.

COMMUNICATION FOR PERFORMANCE CRITICAL FEATURES

In OASIS, we have several use cases where the time is critical. Since all the accelerators are time multiplexed with a slot period of 1.2 second, the basic period, the time taken by an acquired waveform to go from the oscilloscope to the user screen has to be smaller than this period. Furthermore, the user has often in his view range other fast systems that are also synchronous with the machines, such as the video observation system for example. Therefore, in order to avoid confusion on the user side, we have to minimise the transfer time as much as possible. The given soft real-time constraint is a few hundred milliseconds. The scrolling and mountain range acquisition mode also require a very small and quite constant transfer time since they both send small pieces of data at a higher pace. Typically the scrolling mode sends a 10-point packet every 50 ms and, in LEIR, the mountain range mode sends a whole 500-point waveform every 200 ms up to 10 times per basic period. As we can expect the 3-tier communication scheme (CMW + JMS) is a bit too slow for these modes with around 500 ms per transfer. For this reason, we designed another communication mode reserved for the acquired waveforms and that keeps as much as possible the flexibility offered by the CMW + JMS based communication. It is presented in figure 2 along with the 3-tier asynchronous scheme.

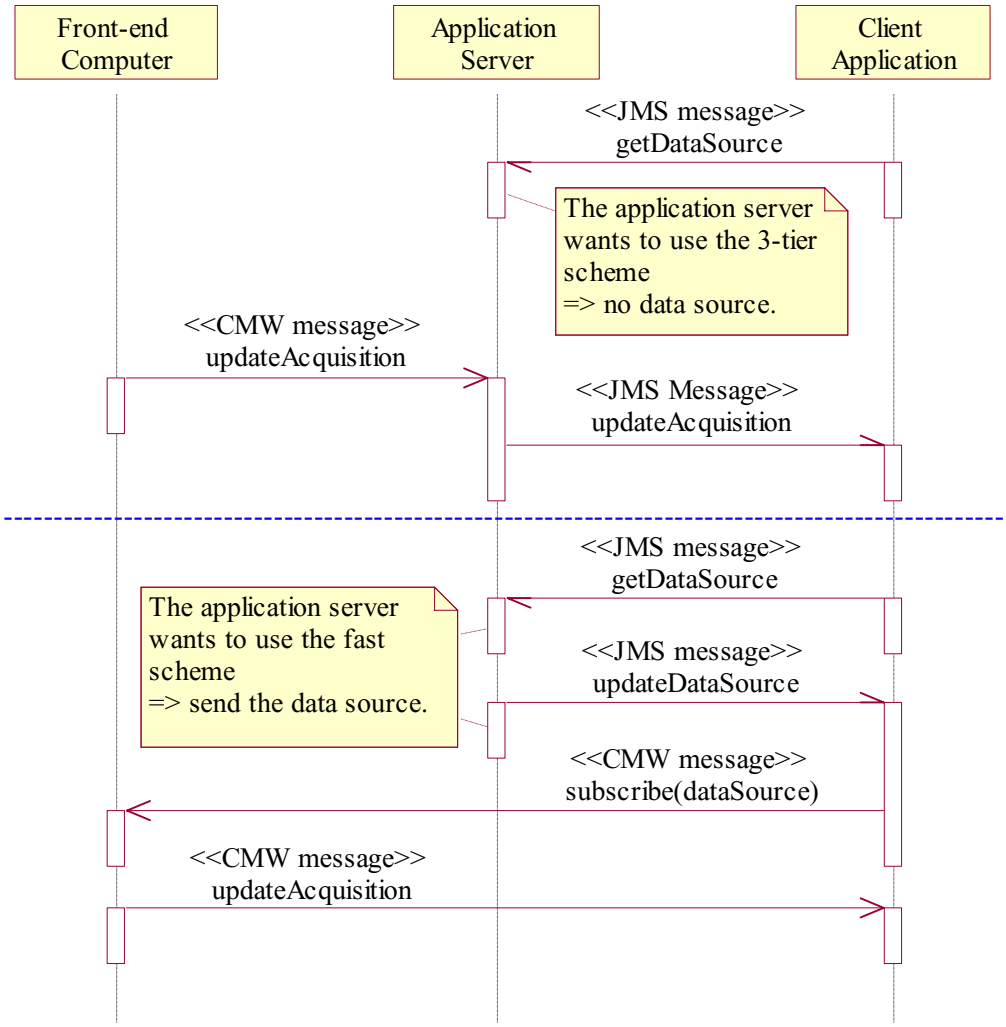

Figure 2: Waveform transfer. On the top, the 3-tier scheme where the data passes by the middle tier. On the bottom, the fast scheme where the data goes directly to the clients.

In the fast mode, we move the responsibility of subscribing to the acquisition property from the application server to the client application. However, since it is the server's task to decide which hardware to use, in FESA words which device to use, we added to the Vchannel class a property called *dataSource*. The new property gives the device name and the property name in a format that is directly usable with the JAPC API so the application does not have to interpret it. On a data source update, the client is requested to unsubscribe to the previous source, if any, and subscribe to the new one. If there is no data source published, that means the acquired waveform will come through the normal setting channel using JMS. The latter mode is still used in some cases such as when a common processing has to be done by the server, for example.

Of course, when using the fast mode, we loose all the advantages brought by the 3-tier architecture and the front-end computer has to send the data several times if there are one or several clients in slave mode. Furthermore, we might have coupling effects between the FESA server and the Java application when something goes wrong such as when an application or a front-end server dies.

On the other hand, this scheme allows us to achieve the maximum performance since the only AB/CO standard protocol for the front-end computer is the CMW and we use only one CMW connection. The advantages we loose are only lost for the acquisition property. Finally, this implementation also lets the server decide which mode to use for a connection. The application server can even change it during the connection lifetime; it simply has to publish the new data source value. Hence, we keep the flexibility needed for on-the-fly hardware re-assignment.

OPERATION

The OASIS system has already been put three times in operation. The very first time was for the TT40 test (September 2003) where a prototype with basic functionalities was deployed [7]. Six signals were available. This test allowed us to validate the principles used although not much stress was put on the system. Then, in September 2004, we deployed the next version for the TI8 tests. We still had only 6 signals but these were acquired twice, once for on-line observation with the OASIS Viewer and once for the logging. Furthermore, thanks to the port of the front-end software to FESA and to the implementation of the peak detection mode, it was possible to observe the whole kicker pulse and the single circulating bunch at the same time which was impossible with the previous version since the signals have very different time scales (more than three orders of magnitude).

In April this year, we have put in production OASIS for LEIR. This time, the configuration is more demanding. We have two oscilloscope crates handling more than 200 signals and a trigger crate centralising the main LEIR machine events. The system has to be available 24/7 until the end of the year. The number of clients is also more important since we have now almost 2 or 3 clients all the time observing between 1 and 12 signals.

The first problem we had with this operational installation was related to the synchronisation between the waveforms coming from different crates. Indeed, up to then, we had just one crate for the waveforms and so only a loose synchronisation was enough to produce an “all arrived at the same time” effect. Now with two data sources, we can have very short delays, fractions of a second, and it is already enough to confuse the user. To solve that, we developed a trigger logic on the application side that compares the time tags set by the front-end computers and notifies the graphical part only when all the waveforms are available. The main drawback of this solution is that it obliged us to install timing receivers in all crates in order to ensure that the tag stamping is precise enough. We have also had problems with functionalities that need to retrieve a lot of data from the database such as the predefined signal set feature. Although the problem is still under investigation, it seems it is related with the entity bean persistence layer. Indeed, we observed that there are a lot of round trips between the application server and the database although all the caches and timeouts are set to their maximum values. This problem can be very annoying since a simple read of a 12 signal configuration can last 15 seconds. If tuning the database connections does not solve the problem, we will have to think about a technology change abandoning the entity beans for another persistence layer such as Hibernate. Apart from these problems, the availability of the system is very good. The application server runs for weeks without any problem.

EVOLUTIONS

Future installations

In the coming weeks, we need to replace the current nAos software in the Linac 3 by the OASIS software in order to have a complete integration of the Linac 3 and LEIR signals. The replacement of nAos means installation of the new front-end software but also export of the existing data from the nAos database to the OASIS database including format conversion.

For the 2006 accelerator start-up and the commissioning of the new CERN Control Centre (CCC), we have to provide support for the Machine Development signals that were up to now acquired with desktop oscilloscopes directly installed in the control rooms. This includes support for 12-bit digitisers and an export function towards Lab View.

Finally, we are planning the replacement of the nAos system in the other accelerators. It will be a step by step phase out starting with the lowest energy machines and going to the higher ones.

Future developments

Concerning the Lab View export function, the first stage is to be able to send the acquisition settings and the acquired data from a running OASIS Viewer application to Lab View. In a next stage, we will investigate the possibility to have Lab View as a complete OASIS client – it should interface at the client package level in the application layer.

A requirement that has not been addressed yet, but will be soon, is the cabling delay compensation function. In a few words, the cabling of the analogue signals and of the trigger pulses induces some propagation delays. Since these are fixed, OASIS should be able to compensate them so that the users observe the signals without any distortion due to their location.

Integration of new hardware is also foreseen. We will integrate modules with higher resolution than the current 8-bit fast digitisers but also hardware with lower capacities such as ADCs modules.

On the trigger scheme side, we have to test our remote trigger scheme. With it, the real trigger is not distributed to all the oscilloscopes. Instead, a pre-trigger used to start the acquisitions is produced and time tagged precisely by a timing receiver [8]. The time position of the pre-trigger is computed from an estimation of the real trigger time position. Since all the pulses are time tagged with nanosecond precision, OASIS should be able to shift the waveforms by the correct amount of samples to correct the time error between the real trigger and the pre-trigger.

REFERENCES

[1] A. Guerrero *et al.*, “CERN Front-End Software Architecture for accelerator controls”, ICALEPCS’03, Gyeongiu, Korea, October 2003.

[2] S. Deghaye *et al.*, “Hardware Abstraction Layer in OASIS”, these proceedings.

[3] E. Roman *et al.*, “Mastering Enterprise JavaBeans”, second edition, Wiley Computer Publishing, 2002.

[4] http://otn.oracle.com/tech/java/oc4j/

[5] K. Kostro *et al.*, “Controls Middleware (CMW): Status and use”, ICALEPCS’03, Gyeongiu, Korea, October 2003.

[6] V. Baggiolini *et al.*, “JAPC - the Java API for Parameter Control”, these proceedings.

[7] S. Deghaye *et al.*, “OASIS: a new system to acquire and display the analog signals for LHC”, ICALEPCS’03, Gyeongiu, Korea, October 2003.

[8] J. Serrano *et al.*, “Nanosecond Level UTC Timing Generation and Stamping in CERN’S LHC”, ICALEPCS’03, Gyeongiu, Korea, October 2003.