OASIS EVOLUTION

S. Deghaye, L. Bojtar, C. Charrondiere, Y. Georgievskiy, F. Peters, CERN, Geneva, Switzerland.
I. Zharinov, JINR, Dubna, Russia.

Abstract

OASIS, the Open Analogue Signal Information System, was fully deployed in 2006 and now allows observation of more than 1900 analogue signals in the CERN accelerator complex. Our first operational experience in 2005 indicated that, for performance reasons, a change in the technology used to access the database was needed. Further experience throughout 2006 showed that an even bigger move was required in order to keep the system easy to maintain and improve. Initially based on the J2EE Enterprise Java Beans (EJB) and Java Messaging Service (JMS), the OASIS server was tightly coupled to OC4J, the Oracle's EJB container, and SonicMQ, a JMS broker. The upgrade to the latest version of these products being unnecessary complex and the architectural constrains being major drawbacks of the EJBs, it was decided to move completely away from those. The paper presents the new server architecture based on open-source products – Spring, ActiveMQ & Hibernate. It also presents the improvements done to the user request processing in order to reduce drastically the response time. Finally, the concept of Virtual Signal is introduced along with the new scalability constrain it brings into the system.

SYSTEM OVERVIEW

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 specific application, the OASIS viewer. 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 the available signals 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 has two main tasks. It has to manage the resources, namely the oscilloscopes, and to provide the Virtual oscilloscope abstraction (Vscope). Here, resource management means to affect oscilloscopes to connections in a way that maximises the number of concurrent acquisitions. A Vscope is a software oscilloscope that takes its data from different hardware modules 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 is based on a three-tier architecture. The front-end tier, the lowest one, has the responsibility to handle the hardware, the digitizer and the multiplexer modules. It provides a hardware independent interface [1] to the upper tiers. The application server (middle tier) manages the resources provided by the front-end interface and assigns them to the connections requested by the clients. In addition, it implements the Vscopes abstraction and controls the associated acquisition settings to keep them coherent and give the virtual oscilloscope image. The application tier is the tip of the iceberg and provides to the users a light-weight Graphical User Interface (GUI) and Java API to interact with the system.

ARCHITECTURE '06

Picture 1 shows the two upper tiers of the system along with the main technologies used at that time.

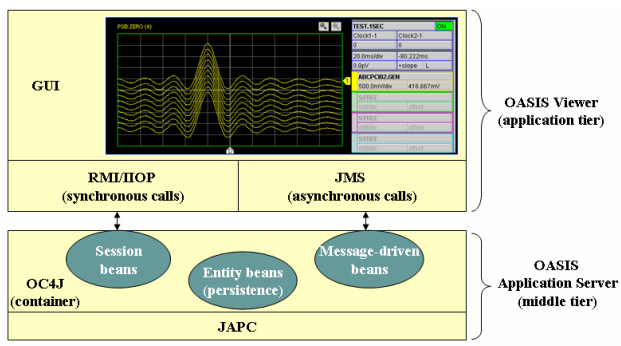

Figure 1 OASIS upper tiers - 2006 version.

The server part ran in an Oracle J2EE container (OC4J). The communication with the application tier was based on RMI and JMS depending on whether the call needed to be blocking or not. For the RMI communication, we had session beans while message driven beans (MDB) were used to handle JMS messages sent through the JMS broker, SonicMQ. Persistence was hidden behind entity beans and the transactions to access the database were demarcated declaratively using the J2EE descriptor files. More details on that version of the system can be found in [2] and [3].

Performance problems

Quickly after the first release and 24/7 operation of the system, we found out some actions had very poor performance. For example, the connection of a predefined 12 signal configuration took between 40 and 50 seconds. After investigation, it turned out just reading the

configuration from the database was extremely slow (typically 15 seconds [2]).

As a first set of actions, we tried to change the database connection settings, the timeouts and the isolation level trying to avoid as many database round-trip as possible. The results were still below user expectation. The second step was more radical since we decided to change the persistence layer and replace the entity beans by Hibernate, an Object/Relational Mapping (ORM) service [4]. The performance improved greatly with a reduction by factor 40 of the time needed to retrieve a stored configuration. From the user point of view, that change reduced the connection time by almost a factor 4 - from 50 seconds to 15 seconds for 12 signals. In addition, Hibernate proved to be much simpler than the entity beans. Indeed, only one Java class and one mapping description file are needed and the mapping can be generated from Javadoc comments. The possibility to use persistent objects directly on the client side was also seen as an advantage since less cumbersome Data Transfer Objects (DTO) were required.

Stability & maintenance problems

Unfortunately, we discovered during the 2006 run that the introduction of Hibernate had also destabilised the whole system and we experienced several server crashes per day. The main reasons were concurrency problems and, to a lesser extend, version problems between the third party components we were using. The crashes would not have been too much of a problem, since that resulted in a service unavailability for 30 seconds once or twice a day, if it was not for the half committed transactions, corrupting completely the database, they were leaving from time to time.

While trying to move all the components to their latest version, we hit several difficulties. Going from the antic version of OC4J to the latest version required from us a big investment to understand all the configuration files and various possibilities. Furthermore, the embedded services in the container were incompatible with CERN made components such as Control MiddleWare (CMW), a CORBA based middleware. SonicMQ also turned out to be unnecessarily complex to install and configure and the features we really needed were not justifying the licence cost. All those components were not well adapted to our needs. Therefore, we decided to change the infrastructure and to move to lightweight and possibly free solutions.

ARCHITECTURE '07

The selection of the new components was strongly influenced by the industry trends of that time. EJB was more and more disregarded as a valid solution mainly due to the undue complexity it brought and so-called lightweight containers were taking more and more importance. We decided to replace the OC4J container by one of the leading solutions, Spring [5]. The session & message driven beans were replaced by standard Java classes plus the Spring remoting facility. Hibernate has been kept for the persistence. SonicMQ was also replaced by ActiveMQ [6] which has the big advantages to be free, easy to install (unzip & run) and easy to configure. We also took the opportunity given by this refactoring to move to Java 6.

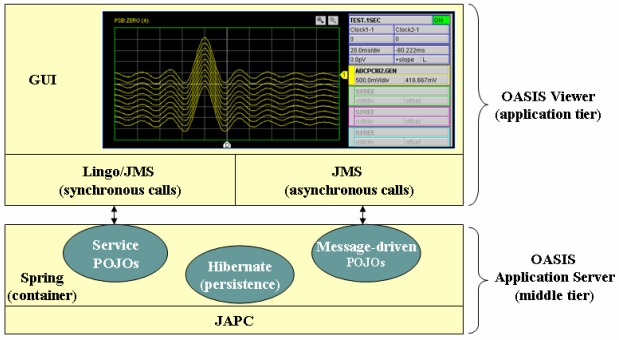

Figure 2 OASIS upper tiers - 2007 version

Persistence layer

The port of the entity beans to Hibernate was already done but we had now the possibility to use the Hibernate Spring integration. This, combined with the possibilities brought by Java 6 and the annotations, simplified greatly the code. Annotations are used to describe the mapping between the Java classes and the database tables (no XML file anymore). The transaction demarcations, also based on annotations, can affect any method of any Spring managed bean giving us a better granularity.

Server deployment

The server is now deployed as a stand alone Java Virtual Machine. There is no need any more to package the classes in special files (EAR) along with deployment descriptors. ActiveMQ being written in Java, it offers to possibility to be embedded in the server process. It is also a simplification since it is one less process to start and to look after.

Performance improvements

Several Java technologies were used to solve performance problems.

RMI was replaced by Lingo, a specialised JMS-based remote method invocation framework, which allows asynchronous calls. Use of the latter feature resulted in decreasing unnecessary wait time in case of non-returning methods. For example, disconnecting 12 signals is now immediate from the user point of view.

Given that OASIS is a multi-client software, several hints were also implemented to parallelise execution of non-interfering operations. A task thread pools per client is created and, thanks to Lingo, remote method invocations are treated as tasks. The result is a reduced wait time for consecutive calls from multiple clients and long operations made of several independent parts. For example, connecting a configuration made of three independent 4-trace Vscopes take only 4-5 seconds compared to the 40-50 seconds of the first version.

Finally, the JMS topic structure has been rewritten which significantly reduced memory consumption and number of threads per client, especially in cases of clients with many connections.

OPERATION

The new version of the system with all the modifications was released early March for the 2007 machine start-up. Since then, everything has been very stable except for some small FEC communication problems that oblige us to restart the application server every month. In August 06, we introduced a connection statistics feature and, as one can see on figure 3, with roughly 40'000 connection request per year, the system is well used. 1.2 % of the connection requests resulted in a situation unsolvable by the server. The 2839 connection requests leading to a 'no free channel' result could be reduced by adding more digitisers but as it is described in the next section, we wait for cheaper hardware for that.

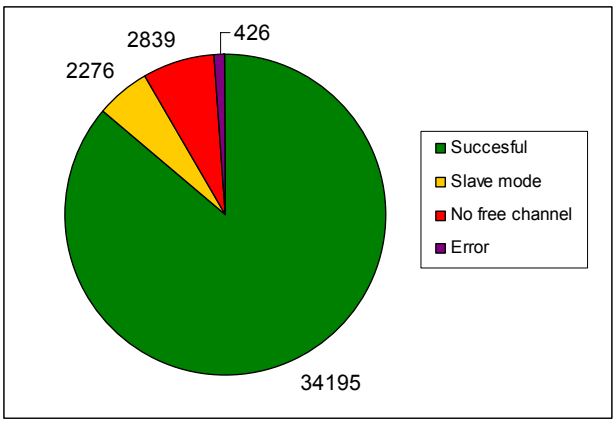

Figure 3 Connection requests from 09-2006 to 09-2007

ONGOING DEVELOPMENTS

ADC integration

The first phase of the project was to provide an open system for analogue signal observation with digitisers and there was already around one hundreds 1GSa/s digitisers installed. In order to reduce the cost per signal, the integration of modules with fewer oscilloscope-like features (no sensibility, no offset…) has started. As a test bed, we chose the SIS3320 [9] which is a 200 MSa/s 8-channel VME ADC. We are also studying the introduction of low sampling rate PCI digitisers - from 200 kSa/s to 1 MSa/s. In parallel, the integration of dedicated acquisition systems is also going on with for example the LEIR low-level RF digital system and the BPM acquisition system built for the CLIC Test Facility 3 (CTF3) by a LAPP/CERN collaboration [8].

Virtual signals

With the current version of the system, anyone on the CERN technical Ethernet network can look at any of the 2000 available signals if he was next to the source with a standard oscilloscope. While that has proved extremely useful, it would be interesting to, first, observe the signals in their real units e.g. Amperes, millimetre… and, second, observe signals that are actually the result of a computation on several real analogue signals – what we call a Virtual Signal. Support for signal unit, scaling factor and offset is done and should released soon.

Full virtual signal support is under development and requires modifications at all levels. Since the computation on the waveforms is CPU intensive, a scalable schema is required in order to support tens or even hundreds of virtual signals. The front-end tier being the less loaded part of the system and also the easiest to extend with additional CPUs, we are developing a virtual signal front-end component to perform the computation. This component is a class developed with the CERN Front-End Software Architecture (FESA) [10]. On the server side, several routing algorithm changes are needed. Indeed, on reception of a virtual signal connection request, the server has to decompose the signal into concrete signals (and this can be recursive) and to connect the concrete signals to available channels. Then, the server has to inform the virtual signal FESA class of the data sources to be used to retrieve the acquired waveforms and perform the computation.

CONCLUSIONS

After a first year of operation, we learnt a lot on the system we did the necessary modifications to have the required system availability. The revised architecture has been running for a year without major problems and with about 40000 connection requests per year, the system is heavily used. A second phase of the project has started with the aim of reducing signal cost and providing the possibilities to the operation to observe high level machine signals.

REFERENCES

[1] S. Deghaye *et al.*, "Hardware Abstraction Layer in OASIS", Geneva, Switzerland, October 2005.
[2] S. Deghaye *et al.*, "OASIS: a new system to acquire and display the analog signals for LHC", ICALEPCS'03, Gyeongiu, Korea, October 2003.
[3] S. Deghaye *et al.*, "OASIS: Status report", Geneva, Switzerland, October 2005.
[4] http://www.hibernate.org/
[5] http://www.springframework.org/
[6] http://activemq.apache.org/
[7] http://lingo.codehaus.org/
[8] http://www.struck.de/sis3320.htm
[9] L. Bellier *et al.*, "CTF3 BPM Acquisition System", these proceedings.
[10] A. Guerrero *et al.*, "CERN Front-end Software Architecture for accelerator controls", ICALEPCS'03, Korea, October 2003.