HARDWARE ABSTRACTION LAYER IN OASIS

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

CERN, Geneva, Switzerland.

ABSTRACT

OASIS, the Open Analogue Signal Information System, is based on a three-tier architecture. The front-end tier, the lowest part, controls the hardware and provides a uniform interface to the application server in the middle tier. However, we do not want to restrict the functionalities provided to the user to the level of the hardware with the fewest capabilities. To achieve these contradictory goals, the front-end tier model is made of several classes with relationships between them. Each class represents an element of the model e.g. an oscilloscope or an oscilloscope channel. The relationships allow us to navigate in the model e.g. to go from a channel to its enclosing oscilloscope. The model lets most of these elements be optional to account for the heterogeneity of the different installations. The class interfaces are made of properties where a property can be seen as a parameter of the controlled hardware. Each property has a standard structure that gives not only the current value of the parameter but also how it can vary. This way, the system is able to support high-end hardware with a lot of possibilities but also cheaper or older hardware with fewer capabilities.

Today, there are three implementations of the front-end model under development. The first two are oscilloscopes and analogue matrices in CompactPCI and VXI formats. The last one is the diagnostics part of the LEIR RF beam control - a completely digital component - showing that the model is flexible enough to support very different hardware.

OASIS IN BRIEF

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 to the upper tiers. The two upper tiers are outside the scope of this paper and are described in [1] and [2] where there is also more information on the architecture.

REQUIREMENTS

As stated above, the front-end tier has to provide the upper tiers with a hardware independent interface. It means that from the OASIS application server the latest Acqiris 4 GSa/s oscilloscopes [3] must look the same as the now 12 year old HP oscilloscopes [4] that were used previously. The 30

MHz CERN made multiplexer with its incomplete matrix must behave in the same manner as the 250 MHz Pickering module which has a complete combinatorial logic.

On the other hand, we do not want to restrict the functionalities provided to the user to the level of the hardware with the fewest capabilities. For example, the minimum time span for the HP module is 500 ns while we can go down to 25 ns, without any processing, with the Acqiris DC211 module. That means we need an interface that is able to give the hardware capabilities when we need them but hide them completely when we do not. The interface also needs to support missing functionalities such as the lack of signal coupling control on very fast digitisers, for example.

The support for heterogeneous installations is yet another requirement for the front-end interface. Indeed, while the digitizers are at the heart of every installation, the other components are more or less optional. To take two extreme cases, we can have a very simple installation with one oscilloscope module digitising one analogue signal with a fixed trigger pulse but we have also the Proton Synchrotron (PS) installation where the trigger pulses are multiplexed centrally and almost all digitisers (about 96 channels) have a signal multiplexer. The front-end interface should not hide these differences but must allow them and ease the work of the upper layers.

FRONT-END INTERFACE STRUCTURE

This section and the following give details on the solution designed to cover the requirements explained above. This section deals with the high level structure of the interface while the next one focuses on the property level structure.

The front-end interface is composed of several classes. Each class can be related to the others in some predefined ways using relationships. The interface can be split in two parts, the signal part which is concerned with the analogue signals and their acquisitions and the trigger part which focuses on the oscilloscope triggering.

Signal part

Figure 1 depicts the signal part of the front-end interface. At the centre of the model, there are the Scope and Channel classes. The channels always belong to a scope and a scope has at least one channel. The Signal class represents the analogue signals we want to observe. It is related to the Channel class either directly or through a Mux class instance.

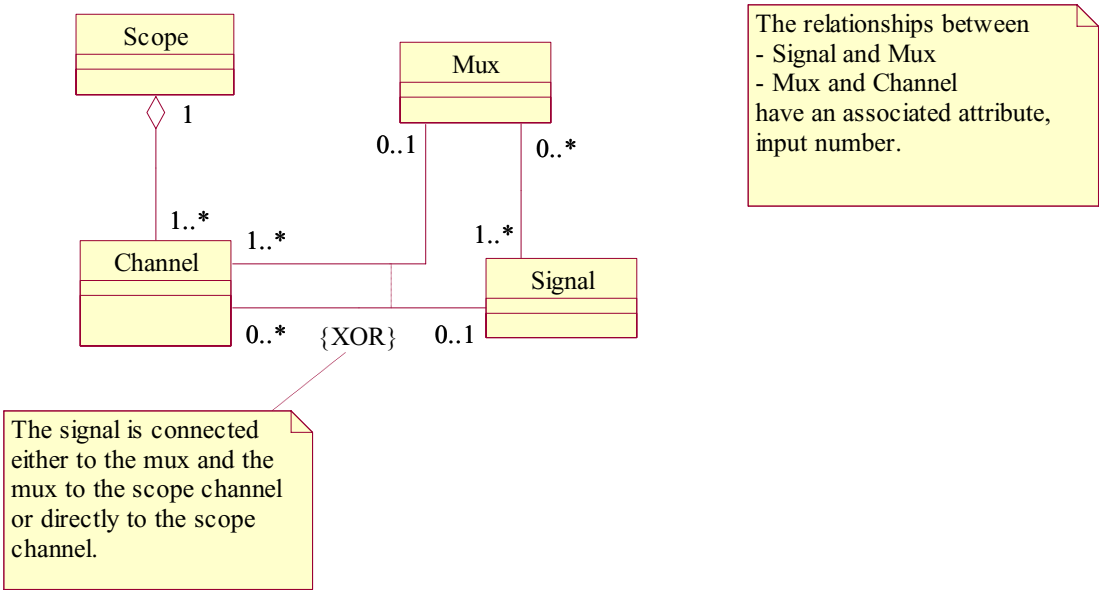

Figure 1: Analogue signal part of the front-end interface

The Mux class controls the multiplexing matrices that can be installed in front of the oscilloscope channels. The Mux class is optional. Of course, we expect a matrix to have more than one input and

one output and that is why we have an associated value representing the input or output number with every Signal-Mux and Mux-Channel link. In the Channel class, we find all the settings that are channel dependent such as the sensibility or the DC offset while the Scope class has the global settings - those which affect all the channels in a scope - such as time span or trigger delay time. The Scope and Channel classes are tightly related and a change in a scope setting has often an impact on the current value of channel settings. The Signal class is one of the simplest although it may evolve with time. Today, it only keeps the signal name. It is foreseen to add later information such as a unit with its scaling factor and the desired input impedance. The Mux class has the typical open/close methods and in order to deal with incomplete matrices a test method that tells whether an input/output combination is possible taking into account the current connections.

Trigger part

Figure 2 shows the trigger part of the front-end interface. The TriggerSignal class represents the trigger pulses we want to acquire the analogue signals with. The Mux class has the same role as the one used in the signal part except that it multiplexes trigger pulses instead of analogue signals. The Counter class controls the counters that may be installed at the multiplexer output. These counters are used to delay the trigger pulses using an external clock. The interface supposes that several clocks are available for counting. In the trigger part, only the TriggerSignal and the Scope Class are mandatory.

As the Signal class, the TriggerSignal class is only a data holder and, today, the only data available is the trigger name. Other information may be added later. For the Mux class, there are some properties that are useful only when the multiplexer is in this position (the trigger mux position). These are the PPM (pulse-to-pulse modulation) properties that allow us to choose the machine conditions under which a trigger is allowed to go through the matrix. This, of course, is only available for solid state matrices since such a thing on an electromechanical multiplexer would result in excessive wear out of the relays. The Counter class has properties that allow us to choose the clock to be used and the number of ticks to count before the trigger can go out.

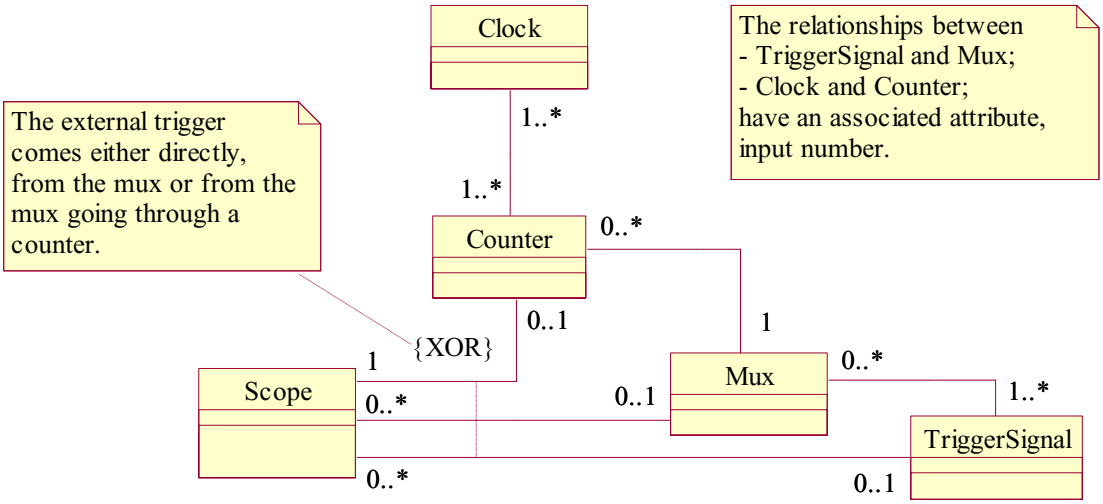

Figure 2: Trigger part of the front-end interface

One should note that the trigger part is connected to the scope and not to the channel. The interface assumes that the hardware we will have to control has one trigger logic per module (per oscilloscope). This can be seen as a limitation but most of the commercial oscilloscopes and even simple ADCs have this architecture.

PROPERTY STRUCTURE

In the high level structure described in the previous section, the requirements on hardware independence and on the support for installation heterogeneity are fulfilled. The possibility to enquiry the hardware capabilities is addressed at the property level.

We have two situations to support. First, two different modules may have different ranges of variation for the same property. Second, a property can be not applicable to a given module type.

The first case is solved by the introduction of the Setting type. When a property represents a setting, such as time span or trigger coupling for example, it has to give the current value and what are the valid values. Of course, depending on the hardware, the setting can vary in two different ways, either continuously or discretely. If the setting varies continuously, it should have a minimum and a maximum value. If it varies discretely, it should have a set of allowed values. This gives us the Setting type hierarchy, as depicted on figure 3. All the setting properties in the front-end interface use this hierarchy, except properties of type Boolean where the range in obvious.

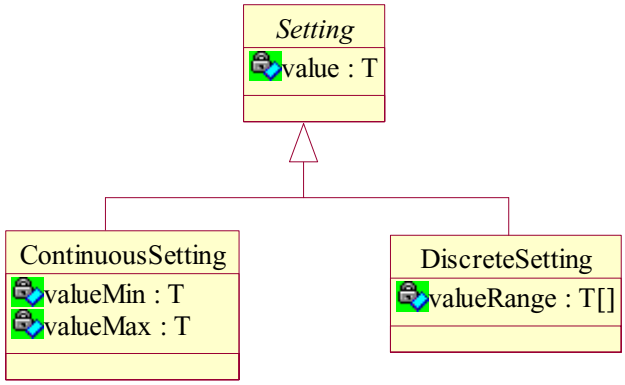

Figure 3: Setting type hierarchy

For the second situation, when the hardware does not support a setting, the property should be set to a default value and have a range with this single value. This way, the get returns something coherent; the current value is in the range and the range has only this value, the property is fixed. The set method can be just ignored. For example, the oscilloscope Acqiris DC211 has a fixed trigger coupling (DC, 50 Ohms). The get call on the trigger coupling property returns value = DC-50, range = [DC-50].

IMPLEMENTATION

The implementation of the front-end interface is based on FESA [5] for the classes and on the device relationship database and the directory service [6] for the links between instances.

To implement a class described in the previous sections in a FESA class is quite straightforward. The class has just to implement all the properties defined by respecting the property name and the name and type of the data items. However, the ContinuousSetting and DiscreteSetting type used in the front-end interface definition cannot be mapped directly to a FESA type. Indeed, FESA supports only primitive types (integer, float…) presented as scalar, array, or 2D array. Furthermore, the CMW [7] which is used for the network communication is data oriented, in other words, we cannot send an object but only a data structure. To solve this problem, we define a mapping between the ContinuousSetting and DiscreteSetting classes and two data structures that we use in FESA class designs each time we need to implement a property returning the Setting type. These structures, depicted in figure 4, have the data of the mapped class including the super class's data plus a type identifier. In this very simple case, a boolean is enough to distinguish between the two classes – an enumeration should be used if more types are needed. For example, the DiscreteProperty_DataType, which maps the DiscreteSetting class, has an item "value" coming from the Setting class, an item "range" coming from the DiscreteSetting class and an item "discrete" informing the client side that it represents a discrete property.

Today, we have at least one implementation for all the classes. Implementing the Scope and Channel classes, we have FESA classes for the control of Acqiris CompactPCI oscilloscopes (the DC series) and HP VXI oscilloscopes. The Mux class is implemented by the class controlling Pickering CompactPCI multiplexers [8] and also implemented by classes controlling VXI matrices (HP and CERN made). Finally, a class implements the counter interface to control the PS complex trigger counters. These classes have been tested successfully during the TI8 tests in late 2004 and during LEIR hardware commissioning with different Java and C++ clients.

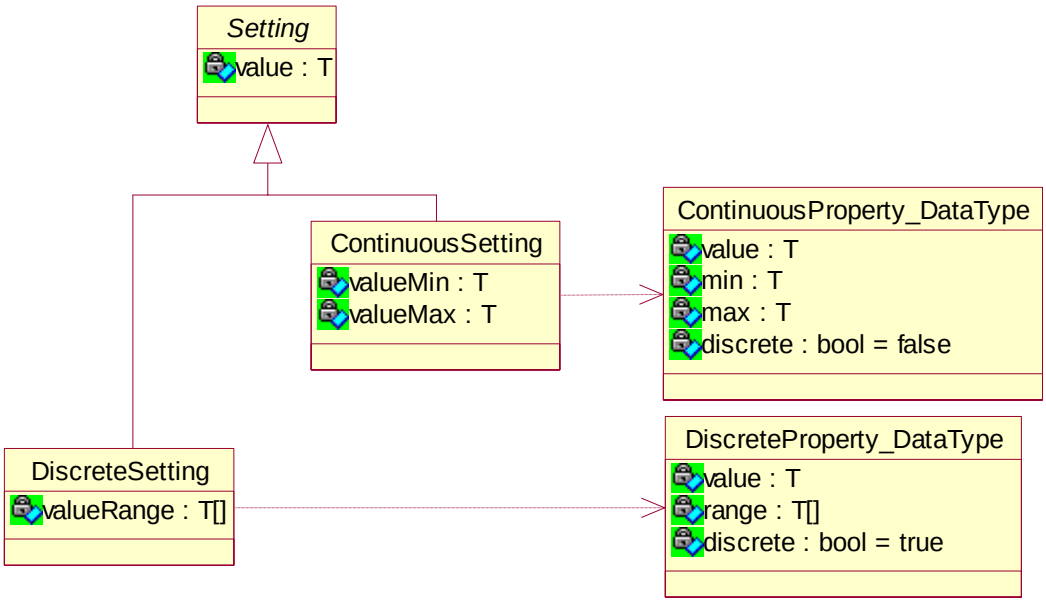

Figure 4: Mapping of the Setting hierarchy to FESA property data structure

FUTURE DEVELOPMENTS

In the coming months, we will increase the number of hardware types supported. For example, the implementation of the front-end interface to control the diagnostics part of the LEIR RF beam control - a completely digital component – has started. With this system, we have the opportunity to work with hardware very different from the one we had in mind when developing the interface. We also plan to integrate a newly developed digital multiplexer-counter VME module which should replace the old VXI hardware. In the future, we should also try to implement the Scope/Channel class interfaces above a module with reduced functionalities such as a scanning ADC.

We will also extend the front-end interface. A correct integration of the units and scaling factor is desirable for the Signal class. For the TriggerSignal and the Signal class, we are thinking about integration of cabling information, more precisely, propagation delays due to the cabling. With this information we would be able to automatically shift the acquired waveforms to compensate the delay induced by the cabling of the analogue signals and the trigger pulses. Work has also to be done on the support of modules with a higher resolution than 8 bits.

REFERENCE

[1] S. Deghaye *et al.*, “OASIS: a new system to acquire and display the analog signals for LHC”, ICALEPCS’03, Gyeongiu, Korea, October 2003.
[2] S. Deghaye *et al.*, “OASIS: Status report”, these proceedings.
[3] http://www.acqiris.com/products/digitizers/8-bit-cpci-6u-digitizers/dc211.html
[4] http://www.agilent.com
[5] A. Guerrero *et al.*, “CERN Front-End Software Architecture for accelerator controls”, ICALEPCS’03, Gyeongiu, Korea, October 2003.
[6] J. Cuperus *et al.*, “The Directory Service for the CERN accelerator control application programs”, these proceedings.
[7] K. Kostro *et al.*, “Controls Middleware (CMW): Status and use”, ICALEPCS’03, Gyeongiu, Korea, October 2003.
[8] http://www.pickeringswitch.com/dynamic/main.cgi?myaction=showprod;ref=2361