A software architecture for data acquisition, process and storage

June 13, 2016

Contents

Chapter 1

Introduction

Data acquisition is a key aspect of every experiment, since without it nothing can be learned. During the history of science, instrumentation and knowledge formed a feedback loop, where better instruments have provided humans with better understanding of natural fenomena which allows the develpment of more advanced instruments. This relation is cleary visible on the development of electronics during the past decades. Long running physics experiments are naturally sensitive to these developments and the design of their data acquisition systems must take this into account. As these systems become ever more complex, the concept of CODAC emerges. CODAC stands for Control, Data Access and Communication and can be considered an extension of the CODAS (control and data acquisition system) as it treats all systems that are part of an experiment (namely control of the experiment, data collection, data access and plant security) as a whole. Although te specific needs of an experiment must be taken into account when designing a CODAC, there are several key features that should be taken into acount. This thesis uses the implementation of a CODAC for a specific experiment as the starting point for a study on how to design a CODAC for modern, large physicals experiments, taking into account not just their specific needs but also a boarder view.

1.1 Fusion Research and Engineering

Fusion is a nuclear reaction where light elements are merged (fused) into a another, heavier, element. Although any element can be fused this way, it typicaly involves hydrogen and its isotopes, deuterium and tritium, being this the main reaction that powers our Sun. Since this reaction produces a great amount of energy, the main goal for fusion research is to apply it to produce electricity, as an efficient and clean alternative to existing power sources, namely fossil fules. Of all the fusion reactions available the deuterium-tritium technology is the most promissing: not only is the most energetic, it is also the one that occurs at faster rate. Besides, deuterium is abundant on ocean water, from where it can be easily extracted and tritium, although not occuring naturaly on Earth, can be obtained from Lithium which is a very common metal. However this reaction also generates fast neutrons that activate the structures of a fusion reactor, this being its biggest environmental issue. These same neutrons, on the other hand, can be used to breed tritium from lithium, meaning that there should be no need to handle tritium (which is radioactive itself) outside of the fusion chamber.

In order for fusion reactions to occur certain densities and temperatures are required and for these to be fullfilled the elements must be on a confined plasma state. There are three ways to confine a plasma: gravitational, magnetic and inertial. Gravitational confinement is the one responsible for energy production in stars, so it is not viable on Earth. Inertial confinement uses powerful lasers to compress fusion fuel pellets during short periods of time. Magnetic confinement uses the fact that ions and electrons follow magnetic lines to keep the plasma inside a partial-vacuum chamber. The most common magnetic configurations are the Tokamak and the Stellarator. Both have toroidal shapes, their main difference being how a poloidal magnetic field, necessary to stabilize the plasma, is generated: in a tokamak a plasma current is induced by varying the magnetic flux on the central core (like in a transformer); in a stellarator by specially shaped external coils. Until now, the tokamak is the most successful magnetic geometry and is widely considered the best candidate for a commercial fusion power plant.

1.2 COMPASS

Tokamak COMPASS (COMPact ASSembly) is installed in the Institute of Plasma Physics (IPP) of the Academy of Sciences of the Czech Republic since 2006. It was designed in the 1980s in the British Culham Science Centre to study plasma physics in circular and D shaped plasmas, with the purpose to prepare for ITER scenarios. In fact, the relations between its dimensions are very similar to those of ITER. At the turn of the 21st century, the Culham Science Centre started working with a new spherical tokamak, the Mega Ampere Spherical Tokamak (MAST), and so operation of COMPASS was discontinued. However, due to its relevance for the ITER project, it was offered to IPP - Prague which had experience with small tokamaks.

As before, the main objectives of COMPASS are related to ITER, specially:

- study of H-mode physics (there are only two other operational tokamaks in Europe capable of this regime);
- MHD equilibrium and instabilities;
- Plasma-wall interaction;
- Physics of runaways and disruption;
- Developments of advance diagnostic methods;

1.3 ISTTOK

Tokamak ISTTOK (Instituto Superior Técnico TOKamak) is installed in Lisbon, in the institute that gives it the name and is explored by IPFN. It is a small tokamak with a circular cross-section, a poloidal graphite limiter and an iron core transformer, that was built from the former TORTUR tokamak, which was de-commissioned by the Association EURATOM/FOM in 1988. The ISTTOK construction started, officially, on January 1st, 1990, date on which the contract of association EURATOM/IST enter into force. The operation of this experiment in an inductive regime started on February 1991. Its main objectives are:

- Teach and train personnel (mainly university students) in physics, engineering and technologies associated with nuclear fusion;
- Development of new diagnostic techniques and to develop and test digital instrumentation dedicated to control and data acquisition.

Since 2012, thanks to fast electronics developed at IPFN and real-time control, ISTTOK has been operating on AC discharges, where the direction of the plasma current changes quickly enough to allow the pulse to be continued, keeping a somewhat constant plasma density without saturating the iron core. This has allowed discharge duration to go from 25-40 ms up to more than 0.5 s (although the typical discharge is around 200 ms).

1.4 Motivation

With the installation of the COMPASS tokamak in Prague, it was necessary to install a CODAC system for the experiment. Over the years, new diagnostics and actuators were expected be installed and thus required to be integrated on the CODAC. The operation would be based on the novel ATCA control and acquisition boards developed on IPFN (Instituto de Plasmas e Fusão Nuclear). These possess advanced technical features, namely state-of-art FPGA for signal processing, multi-Gigabit/s point-to point serial links for super-fast data sharing across all the input channels and a sub nano-second synchronous timing/event distribution network. This system was similar to the one used on the ISTTOK tokamak in Lisbon. The knowledge obtained during the design, installation and maintenace can be extrapolated to other experiments, future and current, thus becoming the basis for the work presented on this thesis.

Chapter 2

Control, data acquisition and retrieval in international experiments

2.1 JET

The Joint European Torus (JET) is currently the largest tokamak in operation in the world. It was designed in the mid 1970s and its operation began in 1983. It is installed near Culham, Oxfordshire in the UK and is managed through the European Fusion Development Agreement (EFDA). Not only was here that the first controlled release of fusion energy happened, but it also detains the world record in fusion power (16 MW). In 2006 operation was started with an ITER-like magnetic configuration and in 2011 an ITER-like wall was installed, thus becoming the key on the preparation for ITER operation. For the scope of this thesis, JET represents a good example of a large experiment that has been running for several decades, during a time of significant technological advancements.

At the begining, its CODAS was designed around a modular, tree-like structure. It consisted of three levels:

- Level 1: Jet-wide supervisory control
- Level 2: Subsystem supervisory control
- Level 3: Local unit or component

Level 2 and most of level 3 software ran on level 2 computers, with some real-time control on microprocessors in level 3. This structure was also used to identify data in the database. The technology used was cutting edge for the time, with NORD computers and CAMAC front-end electronics. Most software was written in Fortran, with time-critical software written in Assembly and NPL (NORD Programming Language). Around 1990, this technology was showing its age: more powerfull hardware and modern software were available, but they were not supported on the NORD architecture. It was thus decided to switch to a UNIX based system (Solaris OS), to use ethernet networking and to replace the CAMAC system with VME and PC front-ends. This was no easy task: there were many CAMAC systems being used and they could not be replaced overnight, so they had to be interfaced with the new computers; tens of thousands of lines of code written in Assembly and NPL had to be ported to C; around half a million lines of Fortran code also had to be translated because of NORD extensions that were not supported by UNIX; some NORD features also had to be temporarly emulated; and all of this while the machine had to remain operational. Only in 1994 and after a major shutdown phase, was the migration to the new CODAS system considered complete.

This, as expected, would not end here and by 2008 the system had received another overhaul, being by then mostly based on PCI eXtension for Instrumentation (PXI) and LabVIEW, with computers now using GNU/Linux operating systems, specifically Linux RTAI for real-time applications. More modern hardware architectures, like Advanced Telecommunications Computer Architecture (ATCA) here started to being used for new subsystems. In 2010, a real-time control framework, called MARTe (Multi-threaded Application Real-Time executor), that can take advantage of multicore processors, was developed at JET and now is being used for some of its real-time control, including some critical subsystems like the vertical stabilisation control.

It is also interesting to look at how the system grew during its opertional history. In the early design it was estimated that the CODAS would require about 15 computers, together with a large mainframe for data storage and analysis. It was also expected that this would be enough for the entire life of JET. But as soon as JET became operational this number had already grown to 34 NORD minicomputers (20 online), in 1999 there were 122 SPARC computers being used (92 online) and the number continued to grow to the point were it is not easy to make an estimation anymore. As for the data generated, it was first expected that each shot would produce almost 1 MB of data, value that was surpassed as soon as JET started regular operation. Over the years the amout of data collected has been steadily increasing, roughly doubling every two years, reaching 10 GB by 2008. In order to cope with this growth the first measure taken (besides improvments made on the network) was to classify data has either 'urgent' or 'non-urgent', with the first being collected in the first few minutes after a pulse and the other later in the day, during quiet periods. More solutions (now with ITER also in mind) like data compression and selective acquisition rates are also being studied and may be implemented in a near future.

2.2 Wendelstein 7-X

The Wendelstein 7-X (W7-X) reactor is an experimental stellarator built in Greifswald, Germany, by the Max Planck Institute of Plasma Physics (IPP), and completed in October 2015. It is currently the largest stellarator in the world and it was designed to operate with up to 30 minutes of continuous plasma. Its first plasma was created in December 2015 and regular operation has yet to start.

The CODAC of W7-X is based around a modular system called Control and Data Acquisition Station (CoDaStation). It was designed with the specifics o W7-X in mind and prototypes were tested on other installations (like WEGA stellerator) before deployment. The core of the CoDaStation software is a generic framework that solely provides the necessary infrastructure to acquire and process signals. In order to achieve high modularity, there are many components on the system, each with well defined functions nad interations. On the data acquisition side there is:

- Buffers: signal packages containing data with the same *time group* (similar time-stamp)
- Signal providers and consumers: generate a signal and recieve one or mores signals, respectively
- Routers: transfer data from *signal providers* and *signal consumers*
- Routing jobs: define if routers act synchronously or asynchronously, according to urgency or special requirements

On the control side there is:

- Properties: the state of a CoDaStation component
- Controllables: retrieve *Properties* from the CoDaStation and translate them into component-specific actions
- Station states: define the overall state of the CoDaStation
- Controller: interface for external control systems to issue control commands

These come together on packages called AddOns, which are used to integrate subsystems thus providing CoDaStation with new functionalities. These can have *controllables*, *signal providers* and *signal consumers*, interacting with the main system through a fixed software interface. Thanks to this, new features can be easily tested outside W7-X and also CoDaStation components can be replaced by mockups. There are also *Supervisors*: software that operators use to monitor *Station* and *AddOn* states and act if necessary. This can also be done remotely, thakns to JXI based supervisors.

2.3 NIF

The National Ignition Facility (NIF) is a large laser-based inertial confinement fusion research device, located at the Lawrence Livermore National Laboratory in Livermore, California, in the United States of America. Construction began in 1997, being ready for operation in 2009 and by 2010 the National Ignition Campaign started with the goal to produce more fusion energy than deposited on the target. However the campaign ended in 2012 without reaching its main objective and currently NIF is mostly devoted to materials studies.

At the heart of the control and data acquisition system of NIF is the NIF Data Repository (NDR). The NDR integrates many autonomous database systems thus being called a federated database. It allows the collection of all information relevant to the experiment, specifically campaign plans, machine configuration, calibration data, raw experimental results and processed results. There are three main events that trigger data collection and automated data analysis: laser alignment, shot time capture, and post shot optics damage inspection. Recent data is stored on-line, least recently used data is migrated to near-line storage and older data storage is subjected to different policies depending on the type of data. In 2009 NIF was generating approximately 66 TB of data per year, most of them 2-D images.

NDR uses industrial and scientific standards for its operation, such as:

- Web Services Business Process Execution Language (WS-BPEL) for preparing shots and organizing data collection, transfer and analysis;
- HDF5 files for storing calibration, background and diagnostic data;
- Uniform Resource Names (URN) to identify data and accessing it through Web services.

One of the goals of NDR is foster collaboration between universities and research institutes. To achieve this, all data and other content can be shared on the Web, with researchers being able to decide what they want to share with everyone, a restrited group of colleagues or to keep private.

Chapter 3

CODAC Tools on ISTTOK and Compass

The CODAC in ISTTOK and Compass share several common tools and are thus somewhat similar. These tools were designed to be flexible, modular, independent from operating system and computer architecture. This is achieved by using non-proprietary open-source software, industry standards and cross-platform programming languages. These systems are open-source and their code is stored in a subversion repository.

3.1 FireSignal

FireSignal is the part of the CODAC that allows the operator to configure and monitor the experiments, also retrieving data from diagnostics and storing it in the database.

As a fully modular system FireSignal avoids dependencies of particular technologies. It is composed of five different modules, which provide all the functionality to the system:

- Central Server—works as bridge between all the other components. It is responsible for managing all the connections, commands and data broadcast.
- Database connector—controls the installed database, storing and fetching data when ordered by the Central Server, providing an abstract layer, independent of the database solution.
- Security Manager—the Central Server queries this component to authenticate users and nodes and to authorize all the operations. The security schema used (LDAP, PAM, etc.) is abstracted by the security manager.
- Hardware Clients—these are responsible for driving the devices and for data readout. Template nodes, with the communication to the Central

Server already implemented, are provided in different languages and can be extended to integrate any hardware device.

- User Clients—these are the graphical user-interfaces, which allow the interaction with the system. It is the only optional component as the system can still operate without direct human intervention, e.g. when FireSignal works as a system which receives orders from other data acquisition system.

All the components are connected through CORBA allowing them to run in different operating systems and to be written in different computer languages. Currently FireSignal is using Java and C++ as main languages.

3.1.1 Database connector

All data arrives tagged with two absolute time-stamps: acquisition start time, end time. These can later be used to generate time arrays and correlate data. The data also comes tagged with an event. A large number of what happens during the discharge can be considered an event. Examples of events include the discharge start trigger, external triggers and disruptions. Events are defined by a unique name and number and by an absolute time-stamp. In the case of ISTTOK the event shot is defined by the string '0x0000', where the unique number is the shot number and the time-stamp the time in which there is the formation of plasma. The accuracy of these times stamps isn't guaranted by FireSignal so an external timing system is necessary.

The installer for FireSignal originaly came with three database connector implementations: MySQL, PostgreSQL and PostgreSQL with external files for the binary data. A fourth, supporting Compass Database (CDB), was developed later. This one was also used as a base for a version using HDF5 (as opposed to plain binary files) files to store data, which isn't currently being used.

3.1.2 Hardware Clients

Hardware clients, also named nodes, provide the interface between the installed devices and the Central Server. Their main responsibilities are the configuration of hardware, tagging data and events with the correct time-stamps and sending them to the server. Nodes are organized as follows:

- Nodes contain Hardware devices.
- Hardware devices contain parameters.
- Parameters contain configuration fields.

This organization is not very different from a typical acquisition system, with crates (Nodes) containing boards (Hardware) with several channels (parameters). By default, each parameter in the same hardware has a different value for the same configuration fields but it's possible to have them share the same value.

Nodes are described in two types of XML files, one for the Node iself and another for each Hardware. The Node file is very simple, only containing the name and description for the node itself. The Hardware file contains the name and description for the hardware itself and also the list of parameters and their fields. Parameters and fields can be defined to be editable or not, it is possible to force fields be not be empty, have their value shared across the same Hardware and it also defines their type. Parameters have a mime-type that is used to determine how to store and display data (there is a plugin system that allows user defined mime-types). Likewise, there are several standard defined field types and it is possible to have user-defined types, thanks to the plugin system.

3.1.3 CORBA

As with all the FireSignal components, the communication between the Nodes and the Central Server is perform through CORBA. Each Node implements the FireSignal server–node communication standard. The number of functions is quite significant and ranges from hardware configuration details to event broadcasting and data acquisition. FireSignal is bundled with server–node communication standard implementations in two different languages: Java and Python (C++ is now being developed). This allows developers to focus only in the interface between Nodes and hardware without having to learn or concern about the communication protocol with the Central Server.

3.1.4 User Client

The user client is a standalone application, developed in Java and deployed using Java Web Start (JWS) technology, serving as a front-end of the FireSignal system. Distributing through JWS downloads all necessary libraries and if any of these are updated it will also update them automatically. The parameter configuration interface is automatically generated, using the XML Hardware configuration file and the default configuration options can extended by loading plugins. Besides configuration, it also allows the supervision of the states of each Node and Hardware, it allows remote participation, it includes a data viewer and it also has the possibility of performing data analysis using an external tool, but this feature is not currently implemented. The data viewer only supports one dimensional data, but it can be extended with plugins to show more complex data, accordind to the mime-type.

Users can connect to the system anywere in the world, given that they are registered. There is a basic chat room (powered by Jabber) and users are also able to send data references that they fin relevant to the discussion to each other. FireSignal recognizes three roles:

- Regular user: can see configurations, and data. Can use the chat and share data;

- Administrator: manages users;
- Operator: can configure the experiment and operate the machine.

It also supports groups with special permissions, for the case were a group of researchers is responsible for the configuration of a susbsystem but are not authorized to operate the machine.

3.2 SDAS

The Shared Data Access System (SDAS) is the part of the CODAC of ISTTOK that is reponsible to find and retrieve data from the database. SDAS is based on Remote Procedure Calls (RPC) specifically XML-RPC, where the calls are made by HTTP using the XML format. SDAS was designed to be used by many different associations, with the objective of users only using a single set of libraries to access any database of the adopting associations. To achieve this it is required for each association to develop a software component which would translate the SDAS calls into something meaningful to their data storage system. There are connectors already available for PostgreSQL and MySQL relational databases, with the option of storing data as a binary blob inside the database itself or as a binary file (raw or HDF5).

SDAS design shares some similarities with FireSignal. Data is also indexed by time (start time and end time) and events but a signal unique identifier doesn't necessarily need to follow the node-hardware-structure used by FireSignal. Data is also treated by default as a one-dimensional array of values but it is possible to extend SDAS to support more complex data, like images for example.In the case of varying acquisition rates, it is possible to have the data in slices, each with its own start and end times, and the system will recognize them as being part of the same data set. Data can have linear calibration coeficients applied to them, but on the other hand it is not possible to retrieve the raw data as it is stored: even without coeficients it always returns in double-precision floating-point format.

On the user side, there are libraries in Java, Python and C++ that can be used for programing or be integrated in data analysis software like Matlab, IDL and Octave. There is a simple data viewer in Java that has basic functionalities implemented, allowing users to search for signals and events and view data in either a plot or a table. There is also a Web tool that allows users to search on any web browser for all the data generated by a specific event and to download signals as binary files. SDAS as a basic authentication protocol that allows authorized users to save processed data on the main database. This feature is being used at ISTTOK to store the automated calculations of plasma density and plasma current after each shot.

3.3 MARTe

Developed at JET, MARTe is a modular framework designed for real-time projects, specially real-time control. It is built on a C++ real-time library named BaseLib2, that allows the same code to run in different operating systems. The main features of BaseLib2 are:

- configuration file, used to automatically create and configure objects;
- garbage collector for memory managment;
- HTTP interface vailable for all objects;
- built-in mathematical and algebraic tools;
- configurable logger, including a tool in Java to receive and filter logger messages.

Being a modular design, MARTe itself is made of different components. The basic ones are:

- Generic Application Module (GAM): the main element of MARTe and building blocks. They have three interfaces: one for configuration, one for input and one for output. GAMs that interface with hardware are called IOGAMs. It is also possible to have GAMs that simulate inputs for other GAMs, for debug purposes.
- Dynamic Data Buffer (DDB): optimixed memory bus that GAMs use to transfer data betwen each other. It also ensures the coherance of signals betwen GAMs.
- Real-Time Thread: container of GAMs and responsible for their sequential execution. MARTe must contain at least one real-time thread, but it can handle any number of them, running either concurrently or in parallel.

MARTe is currently being used at JET for the vertical stability control, at ISTTOK to control the plasma position and the primary current (for plasma current inversion for AC discharges), and in COMPASS to control the plasma position and shaping.

3.4 CDB

The Compass DataBase (CDB) is the part of the CODAC of Compass that is reponsible to find and retrieve data from the main database. Originaly Compass used an implementation of SDAS but do the perceived shortcomings a new solution was designed. The main features of CDB are:

- Nothing stored can be overwritten; instead, revisions are possible as corrective actions;

- Shots are separate from other events;
- A relational database is used to store metadata of the numerical data and also physical quantities names, units and information about the axis;
- Time can be stored as an array or just the start and end of the acquisition;
- Data files (HDF5) can be written directly on the main data storage without having to pass through a central server.

The core application is implemented in Python (pyCDB), with Cython being used to wrap the Python code in a C API. Matlab, IDL etc. clients can then be built using the C API. There is also a Java binding using Jython.

3.4.1 Integrating CDB and FireSignal

For CDB and FireSignal integration, two things were required:

1. Design CDB in such a way that FireSignal identifiers for channels and events would still be recognized and usable;
2. Develop a new FireSignal database controller module, for legacy Nodes.

The second point would still cause the issue of bottlenecking that CDB was trying to solve, but it was considered necessary not only because it would allow Compass to operate normally until all nodes had been converted to the new data storage method but it would also allow Nodes developed at IPFN to be tested on Compass before any major changes were applied. The first point was achieved by having functions on the API that convert between FireSignal IDs and CDB IDs and vice-versa. The biggest challenge regarding the second point was the fact that the CDB tools that existed to write the meta-data on the database were only available in Python, so a bridge between Java and Python had to be used. At first Java Embedded Python (JEP) was used as it was more straightforward than Jython, however Jython was easier to install and thus it was used for the Java API bridge. Since it didn't make sense to use two different tools to perform basicaly the same task, eventually Jython replaced JEP. The database controller acts like a regular FireSignal one but with some extra steps:

1. Because the two relational databases are being used in parallel, the controller stores meta-data on the FireSignal Database like usual;
2. Following CDB procedure, a request is made for a new signal ID;
3. The data and the FireSignal XML shot configuration file are stored in a single HDF5 file;
4. Meta-data is stored on CDB relational database.

Chapter 4

Subsystem Integration

4.1 MARTe - FireSignal Bridge

When MARTe was introduced for real-time control in ISTTOK and COMPASS tokamaks, it was necessary to integrate it with FireSignal. This integration meant to allow MARTe to receive triggers, send data to the main database and be configurable on the user interface of FireSignal. It was decided that the node would use the standard HTTP server provided with MARTe, since this would allow the node to fulfill the objectives without having to interfere with MARTe core. It was decided to program in Java in order to take advantage of its integrated HTTP and HTML tools.

4.1.1 State Machine

MARTe has its own internal state machine. The states can be cycled manually by the operator either on the console interface or the HTML interface on a web browser. This node completely automates this procedure: once the main event arrives, it will cycle automatically through the states, following closely equivalent states in FireSignal.

Some FireSignal states are mapped to MARTe commands. Although these are not really system states, they tell the operator that the node has sent the related command to MARTe, but the current state hasn't changed yet. There is a thread that constately monitors the MARTe state page and changes the Node

MARTe	FireSignal
idle	stopped
waiting_for_pre	configured
pulsing	running
post_pulse	running

Table 4.1: MARTe states and matching FireSignal states

MARTe	FireSignal
pulse_setup_completed	configure
pre	run
ejp	N/A
collection_completed	stopping

Table 4.2: MARTe commands and matching FireSignal intermediate states

status if needed. If necessary, the states can be changed by the operator on the MARTe interface and the Node will acknowledge the change. The system, however, needs to know when to issue the *ejp* command to finish the acquisition. For this, an extra HTML page is required with the information if there is plasma or not.

4.1.2 Configuration File Parser

A FireSignal Node requires some information on XML files before being built. Because not all the necessary information is available on the HTTP server at start, it was necessary to get this from the configuration file. BaseLib includes a C++ parser, but since it was decided to program the node in Java, making a native parser from scratch was considered a better option than using JNI.

The parser is event-based sequencial access, meaning that while it parses the file it will call methods (events) from a handler object when it finds keywords. Parsing events are sent in the order of the information on the document itself. Besides comments, it defines the following elements of a MARTe configuration file:

- single attributes: elements in the form *att = value*. They are defined with a single value, that can be a string or a numeric value.
- array attributes: elements in the form *att = {value1 value2 ...}*. Single values enclosed between *{...}*.
- blocks: elements in the form *block = { att = ... }*, with attributes and other blocks inside.

4.1.3 HTTP Communication

For the communication between MARTe and FireSignal two solutions were available: either developing a GAM implementing the functionalities of a FireSignal Node or developing a Node with HTTP communications implemented. Since the HTTP interface allows access to all configurations and data, the only advantage to have a GAM would be to have it all integrated on MARTe. The HTTP approach offers several other advantages: i) it is possible to develop the MARTe code without FireSignal, ii) the Node is not required to be running on the same machine as the real-time code, iii) any existing MARTe configuration file can be used.

4.2 Fast Acquisition Node

This C++ node was developed for PCIe (need reference) acquisition boards. These boards have 32 18-bit ADC channels that can acquire up to 2 Mega samples per second. The node accesses directly the board through driver commands.

With the node interface it is possible to configure:

- Clock source: choice between shared (common to all boards on the same crate) or local
- Trigger source: choice between shared or local
- Trigger type: choice between software (acquisition begins immediately) or hardware (waits for external trigger)
- Acquisition time:
- Trigger delay: configurable delay, in ms, from the trigger arrival and the acquisition start
- Delay time: delay to be added to the data timestamps, to compensate for possible trigger differences with other boards

Besides these fields, there are three other that are not configurable, being used for information and reference:

- Frequency: the acquisition rate is fixed at 2 Msamples/s.
- On board memory size
- DMA byte size

This node defines a main event that usually refers to the plasma start. When a new event arrives the configurations are sent to the boards, which then wait for trigger. In case software trigger is selected, a call is made to the driver, starting the acquisition.

Acquired data is tranfered to the node as a signed 32 bit integer array, with the extra bits padded with 0.

4.2.1 16-bit Node

From the data transmission and storage point of view, using 32 bits to store 18 bit values results in 43.75% of space being wasted. To improve this situation, two solutions were considered: compress the data or only store the 16 most significant bits. Some tests were performed to evaluate these solutions.

Using some acquisitions on ISTTOK when the shot failed, we analysed the noise level coming from the boards. Assuming it was gaussian noise, we took note of the standard deviation for the entire signal. Then we calculated the average for the available channels and took note of the maximum and minimum values, calculating the base 2 logarithm in order to obtain an aproximation on

Shot	Average Standard Deviation (log 2)	Min Standard Deviation (log 2)	Max Standard Deviation (log 2)	Number of channels
31652	19.63	18.99	20.19	21
31668	19.63	19.04	20.18	20
31916	19.83	19.14	20.21	27
31942	20.12	19.23	21.82	15
31972	20.05	19.49	21.48	25
32060	19.68	18.98	20.16	20
32116	19.86	19.30	20.33	24
32118	19.85	19.34	20.36	24
32120	19.88	19.44	20.33	24

Table 4.3: Some measurements done to estimate the base noise level at ISTTOK. Not all channels were available for every shot.

the number of equivalent bits (on a 32 bit integer). We can see some examples on the next table:

The varying number of channels come from the fact that, while some channels were malfunctioning, others were picking up residual signals that were not relevant to our study. We can conclude that around 66% of the baseline noise is contained by 20 bits and is thus safe to remove the 16 less significant bits without any noticible loss in signal.

The conversion is done in a simple and straightforward way: a 16 bit pointer is used to run over the data coming from the boards; the less significant part is ignored,while the other is stored in a 16 bit integer buffer that is sent to the central server of FireSignal. This way,without changing the firmware or the drivers, one can choose if he wishes to have the full 32 bits or 16 is enough.

Chapter 5

Future CODAC Design

As we have seen throughout this thesis, CODACs for large, long duration experiments face similar challenges. Many of these were experienced during the work at ISTTOK and Compass tokamaks. This chapter discusses how these challenges should be tackled on future designs.

5.1 Guiding Principles

During the work leading to this thesis it became clear that good CODAC design for a large experiment must follow three principles that are somewhat related:

- Scalability
- Adaptability
- Autonomy

5.1.1 Scalability

As we have seen from the case studies and from experience with ISTTOK and Compass, long duration experiences have a tendency to grow. New or improved diagnostics are a necessity to improve the scientific output and keep the laboratory relevant. This usually implies more acquisition channels and faster rates. Advancements on electronics will also push the desire for faster rates and more bits per sample. New techniques (like AC discharges on ISTTOK) can also increase the amount of data generated for each session. Because its desirable that core elements of a CODAC don't change radicaly during the experiment lifetime it needs to be ready for this growth. Radical changes can costly both in time and money, not only to upgrade the system but also to retrain the staff.

5.1.2 Adaptability

As new technology becomes available it tends to be less and less compatible with old technology. As such a CODAC must not be too dependent on a given technology, but it should be adaptable to changes that are certain to come, even if they are not obvious. On the other hand, it should be prepared to have different technologies working side by side. Using fusion resarch as an example, there is a tendency to use cutting-edge technology, as fusion energy itself is cutting-edge, however stable and robust technology is also desirable to protect the investment and the machine itself. That's why space programs still use last century technology, as they can't afford any level of failure.

5.1.3 Autonomy

If a CODAC should not be too dependent on a single technology, it should also not be dependent of a single software or hardware company. Turnkey solutions are apealing, specially when they offer a complete environment, but can be costly on the long run, as these solution may not offer the flexibility required by a large experiment. Being able to modify the system without legal restrictions is the easiest way to ensure scalability and adaptability. Of course this is not always possible, but a certain degree of autonomy must be ensured.

5.2 Proposals

5.2.1 General Design

One of the key features that a CODAC should have is undoubtably modularity. This concept is not new, as it was already present on the early designs of JET CODAS, but it is indeed essential if one wants to achieve scalability and adaptability. Idealy, a CODAC should be composed of several different modules, each with its own well-defined function and without overlaps. The connection between modules should be done in a generic and transparent way; each module should not be concerned about the inner workings of other modules. This will also allow modules to be tested outside the main system and be integrated without major changes.

Typically a hierarchical tree-like topology is used, where each component only communicates with the components directly above and below on the hierarchy. Such structure is also found in object oriented programing as it is an intuitive way to implement a transparent module-based structure. However this structue isn't flexible enough and can lead to bottlenecks and excessive complexity. Take for example what happened at COMPASS, where data transfer to the database was hitting a bottleneck at the Central Server of FireSignal, so a bypass between Nodes and Database Manager was implemented. Another example: there has also been some internal discussion on the ISTTOK team regarding a new trigger distribuition hardware and how it should handle events, which has lead to considering a less centralised event distribution. In conclusion, the structure the structure should allow flexibility, with each module being capable of direct communication with more than those directly above and below, in what is called a partially connected mesh topology.

5.2.2 Data collection and access

As it was mentioned, experimental data volume increase is a given on these types of experiments so it is important for a CODAC to be prepared for it. A solution similar to the one adopted at COMPASS, where data is written directly in a database cluster, is one of the best approaches, but it could be made more transparent.

5.2.3 User tools and interfaces

Chapter 6

Conclusions and Future Work