Evolution of the VLT instrument control system toward industry standards

Mario J. Kiekebusch [1a], Gianluca Chiozzi [a], Jens Knudustrup [a], Dan Popovic [a], Gerard Zins[b]
[a]European Southern Observatory, Karl-Schwarzschild-Strasse 2, D-85748 Garching bei München;
[b] Laboratoire d'Astrophysique de l'Observatoire de Grenoble, 414, Rue de la Piscine, 38041 Grenoble Cédex 9 France.

ABSTRACT

The VLT control system is a large distributed system consisting of Linux Workstations providing the high level coordination and interfaces to the users, and VME-based Local Control Units (LCU's) running the VxWorks real-time operating system with commercial and proprietary boards acting as the interface to the instrument functions. After more than 10 years of VLT operations, some of the applied technologies used by the astronomical instruments are being discontinued making it difficult to find adequate hardware for future projects. In order to deal with this obsolescence, the VLT Instrumentation Framework is being extended to adopt well established *Commercial Off The Shelf* (COTS) components connected through industry standard fieldbuses. This ensures a flexible state of the art hardware configuration for the next generation VLT instruments allowing the access to instrument devices via more compact and simpler control units like PC-based Programmable Logical Controllers (PLC's). It also makes it possible to control devices directly from the Instrument Workstation through a normal Ethernet connection. This paper outlines the requirements that motivated this work, as well as the architecture and the design of the framework extension. In addition, it describes the preliminary results on a use case which is a VLTI visitor instrument used as a pilot project to validate the concepts and the suitability of some COTS products like a PC-based PLCs, EtherCAT[8] and OPC UA[6] as solutions for instrument control.

Keywords: VLT, Instrument, Control, Software, Evolution, Fieldbus

1. INTRODUCTION

What is today the VLT Instrumentation Software (INS) is the result of the generalization and evolution of the control software developed for the first VLT instruments. The software artifacts created for the first generation instruments such as ISAAC, UVES and NACO together with the experience gained during the first years of operations laid the groundwork to develop incrementally an application framework which has become the basis for the instrument software of many VLT/VLTI instruments.

This application software framework provides the following:

- A standard and modular structure for VLT instrument software that defines the building blocks of the instrumentation software.

- Common software providing a set of libraries, utilities, rules, definitions and guidelines to support the development of VLT instrumentation software[1].

- Template Instrument to ease the implementation at the early phases of the development process by means of predefined and customizable instrumentation software.

The instrumentation software is subdivided into the following standard INS software packages[2]:

- **Instrument Control Software** (ICS): It controls and monitors instrument devices.

---

[1] mkiekebu@eso.org; phone +49 89 32006-0; fax: +49 89 320 2362.

- **Detector Control Software** (DCS): It performs all the tasks to control technical and scientific detector subsystems. It takes exposures for different read-out modes, it handles the image transfer and display, and store images as FITS files.

- **Real-Time Computer** (RTC): It carries out the data acquisition, reconstruction and control in order to compensate for the atmospheric aberrations.

- **Observation Software** (OS): It coordinates the execution of an exposure for a given observation mode and is the main interface to the instrument control software.

- **Maintenance Software** (MS): It is used for instrument configuration, checking and troubleshooting.

- **Observer Support Software** (OSS): It consists of tools to support the observer in the preparation of an observing block.

In the scope of this paper we will focus only on the extension of the ICS package, therefore the other components of the instrumentation software will not be described further.

1.1 Instrument Control Software

ICS is the software system dealing with the control of all devices belonging to an instrument such as filter wheels, calibration lamps, temperature and pressure sensors, shutters, etc., but excluding detectors and real-time computers which are handled by other software packages according to the organization of the VLT instrumentation software. The hardware architecture of the ICS follows the typical architecture of the VLT control system[3]. It includes an instrument WS running Scientific Linux and one or more LCUs acting as the interface with the hardware devices as it is depicted in Figure -2. An LCU (Figure -1) is an autonomous computing unit composed of a set of custom and commercial boards (CPU, digital and analog I/O, serial and motion controllers, etc.) connected through the VME bus[4]. Each LCU has several layers of software most of them developed in house providing the basic services to all VLT applications. This common software includes the message system, local database, time services, motor library and drivers [4].
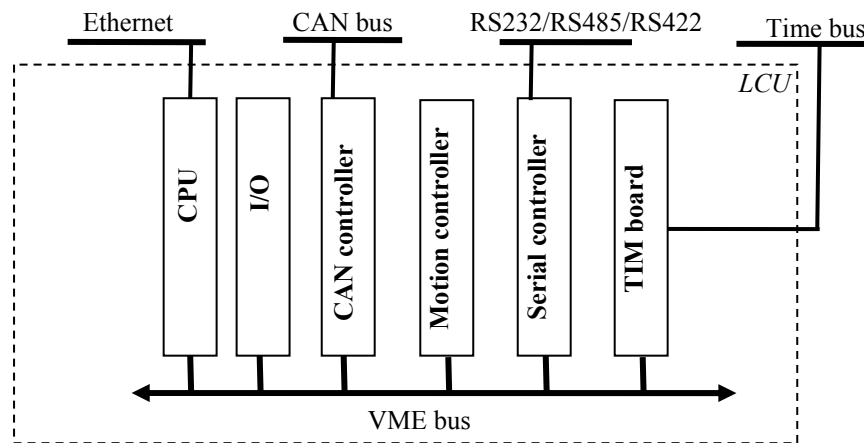

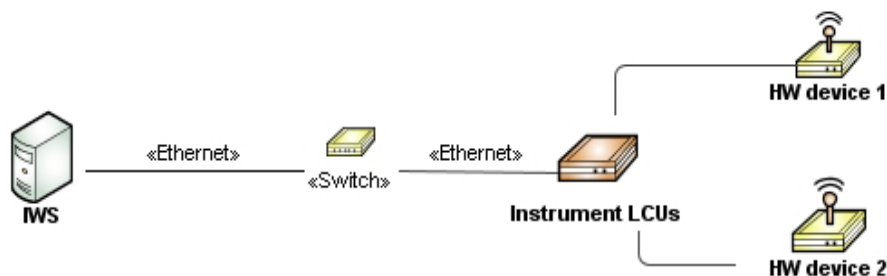Figure -1: Typical LCU architecture.

Figure -2: Example of actual instrument hardware architecture.

The ICS is composed of two parts, one running on the instrument WS and the other one running on the LCU.

The ICS WS part includes the following components:

- **ICS WS front-end Process** doing the following tasks:

  o Controlling the global status of ICS.

  o Forwarding setup commands to the LCU front-end processes.

  o Handling and coordinating replies from the LCUs front-end processes.

  o Retrieving status of all devices.

- **One additional process when** which simulates the LCU behaviour.

- **Graphical Interfaces,** automatically generated and customizable for instrument specifics needs.
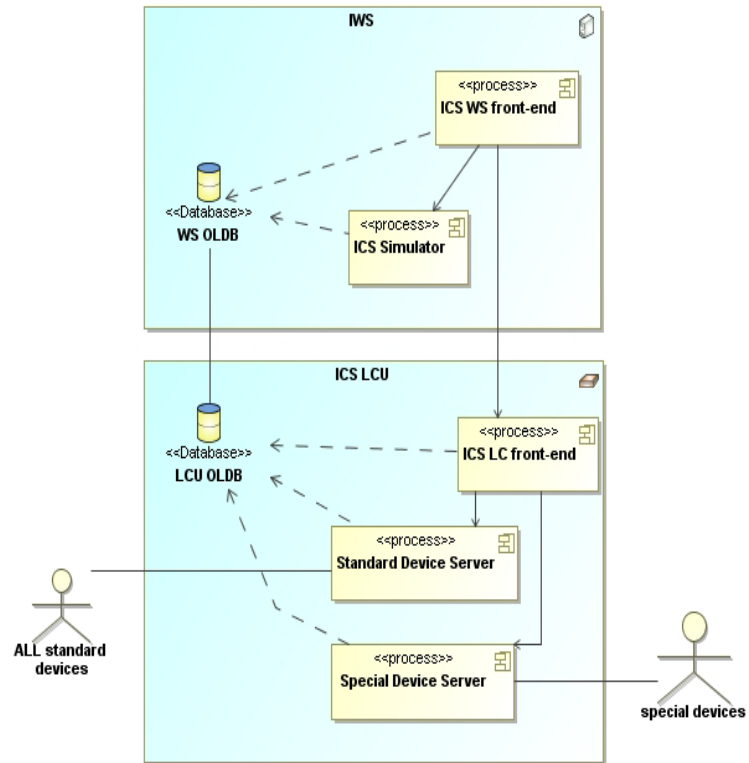
The ICS LCU part consists of :

- **ICS LCU front-end server** which acts as the front-end for all commands coming from the WS.

Figure -3: ICS Hardware architecture without the fieldbus extension.

- **Standard Device Manager,** a single server dealing with all requests for all standard devices.

- **Special Device Manager** (if required), a single server per device or group of devices of the same type implementing the control of instrument specific software requirements.

1.2 Concept of Standard and Special Devices

In the ICS nomenclature, standard devices are generic and configurable pieces of software controlling VLT standard hardware. The ICS standard devices are part of the instrumentation framework such that instrument developers just need to customize them by means of a set of configuration parameters. The following table shows the list of devices currently available as part of the ICS LCU software. The so-called ICS special software devices are all devices which are not part of the framework and are implemented to accommodate to specific instrument requirements. This generally occurs when the VLT standard hardware needs special treatment or when instruments use hardware that is not supported.

Table 1: Currently available ICS software devices.

Quantity	Device Type	Interface	ICS Devices
2	Digital Control	Digital I/O Board	Lamps and Shutters
12	Motor	Motion Control	ADC, Derotator, Filter Wheel, Slit Wheel, Optical function with continuous position, etc.
9	Sensor	Serial Port	ESO cabinet cooling controller, OMEGA temperature

				controller, Humidity sensor, Ester DC24 temperature controller, Yokogawa DAQ, Lakeshore temperature controller, etc.
1	Sensor	Digital Board	I/O	Digital sensors
1	Other	TIM board		Time synchronization device

2. JUSTIFICATION

2.1 Hardware Obsolescence

During the conceptual phase of the VLT control system in the early nineties, VME technology was the clear choice for implementing hardware control for telescopes and instruments. This technology had many benefits in performance, reliability and support for long life cycle developments. ESO has developed many specialized hardware and software solutions for this architecture in order to satisfy various requirements coming from the VLT projects.

In the last ten years, the industrial market, in particular the non real-time applications have been migrating toward less expensive and simpler solutions based on PLCs and fieldbuses. As the consequence, some of our hardware providers are starting to shift their product lines to follow the market trends making it more difficult to ensure the availability and support of our VME based technologies for future instruments and to replace failing parts. A concrete example of this situation happened some years ago when our main vendor for motion control boards announced the discontinuation of the support for the VME interface forcing ESO to embark on the development of a custom-made motion controller with all the associated costs this has implied.

Because ESO is still in the process of developing second generation VLT/VLTI instruments and because we might have even a third generation of instruments coming up, there is a clear need to take measures to alleviate this rising issue by searching for alternatives with lower long term sustainability risks.

2.2 PLC support

There has been a revolution in the area of industrial automation with the definition of international standards and with the development of Ethernet based fieldbuses, some of them with extensions for real-time communication. PLCs are becoming more powerful and supporting not only digital and analog I/O but also motion control, distributed control systems and process and safety control.

VLT instruments like VISIR and PRIMA are already employing PLCs in their respective control systems, being pioneers in this area, but without a proper infrastructure allowing a smooth integration with the VLT instrumentation software. Future VLT projects are also planning to incorporate PLCs especially for the control of safety related tasks like the interlock panel of the 4LGS Facility or in the ongoing development of a cryogenic controller for infrared detectors. In summary, we could say that PLCs are becoming a new standard for VLT instruments and therefore it is required to provide an interface for them within the ICS.

2.3 Complexity

Many of the tasks to control and monitor instrument devices do not require real-time capabilities or high computational power. The LCU is a general purpose real-time system that is very flexible and versatile but often too complex and costly for simple control requirements. The ICS framework provides templates and examples that facilitate the task of developing control software for the LCU but this is still not a straight forward process[5]. Some of the factors increasing this complexity are:

- VLT control software is a custom software framework and the learning curve is steep, especially for the ICS LCU code.

- Only ANSI C is supported as the programming language for the ICS LCU software. This means that we cannot make use of real object oriented capabilities that would simplify the coding of the devices.

- A real-time operating system is inherently more complex that normal UNIX operating systems.

- The large number of applications deployed and still being developed make the introduction of significant changes more difficult, as well as ensuring backward compatibility.

2.4 E-ELT Pathfinder

ESO has embarked on the E-ELT project and in this context we are looking for the most suitable instrument control technologies that could be used by the E-ELT instruments. We are very much interested in prototyping and doing field tests on real astronomical instruments to evaluate not only functional capabilities but also some performance issues associates to several months of operations under realistic environmental conditions. We strongly believe that, at this moment, any evolution of the VLT control system should be tried in the direction of the E-ELT technologies.

3. VLT ICS FIELDBUS EXTENSION

Based on the motivations explained previously, we have decided to explore the possibility of an extension of the ICS toward industry standards and fieldbuses. This means a flexible architecture that will allow deploying state of the art technology connected to the IWS through a simple network interface and without the need of custom and complex software. The goal is to be able to connect the hardware devices directly to the IWS via Ethernet when devices support it or connect them through a fieldbus commanded by a PLC for more elaborated applications. The above is illustrated in Figure -4.
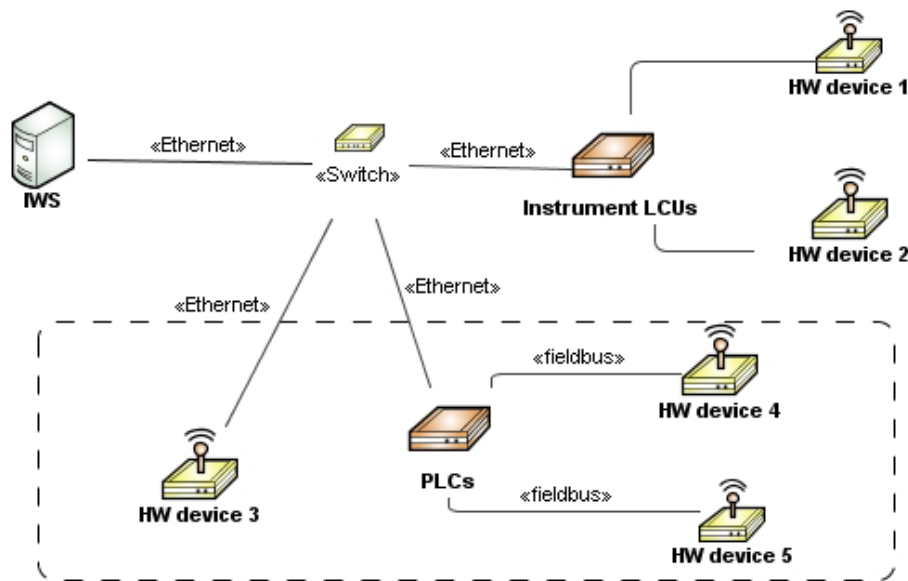

Figure -4: Example of an instrument control system architecture with the fieldbus extension.

The ICS fieldbus extension (ICSFB) is still under development and it is planned to be officially released with the VLT software version 2011. Nevertheless, we have already a beta version which has been tested in our lab and which comes with device drivers needed for our first users. The details of ICSFB are presented in the following sections.

3.1 Project Constraints

In order to facilitate a seamless integration of the ICSFB into the existing ICS architecture and avoid backward incompatibilities that could affect the operations of actual instruments, we have defined the following constraints:

- The ICS extension must be fully compatible with the command interfaces of the ICS LCU.

- The technologies to be used should not create a strong impact on the Paranal maintenance.

- The ICSFB should be compatible with the present device configuration of the ICS framework.

3.2 Design Principles

Based on the constraint of ICSFB and other requirements, we have defined the following design principles:

- The ICS extension is fieldbus independent. It is possible to support different PLC vendors and fieldbuses as long as a proper communication interface exists. Any hardware to be employed shall be previously defined as a VLT standard; exceptions from this principle must be duly justified.
- The ICS fieldbus extension runs on a private CCS environment but on the same IWS. This means that although the extension is running on the same machine, it is completed decoupled from the ICS WS software.
- The simulation capabilities of the software devices are decoupled from its implementation.
- The software architectures for standard and special devices are identical.
- The PLC shall be used only for the interaction with the hardware minimizing the programming to what is strictly necessary. As much business logic as possible shall be implemented in the ICSFB device driver.
- Each software device uses a communication interface dealing with all requests toward the physical device or the controller (PLC). This interface is fully independent from the software device implementation. The same device server might be deployed with different communication interfaces depending on the protocol and technologies to be used.

3.3 Architecture

There is one front-end control process (*ICSFB Control*) acting as the interface between device servers and the ICS WS main process in the same way as in the ICS LCU software. Unlike the ICS LCU part, the ICSFB uses two servers to control each device: Device Server and Device Simulator (Figure -5). This model tries partially to overcome the lack of multithreading support on the VLTSW and promote parallelism between devices.
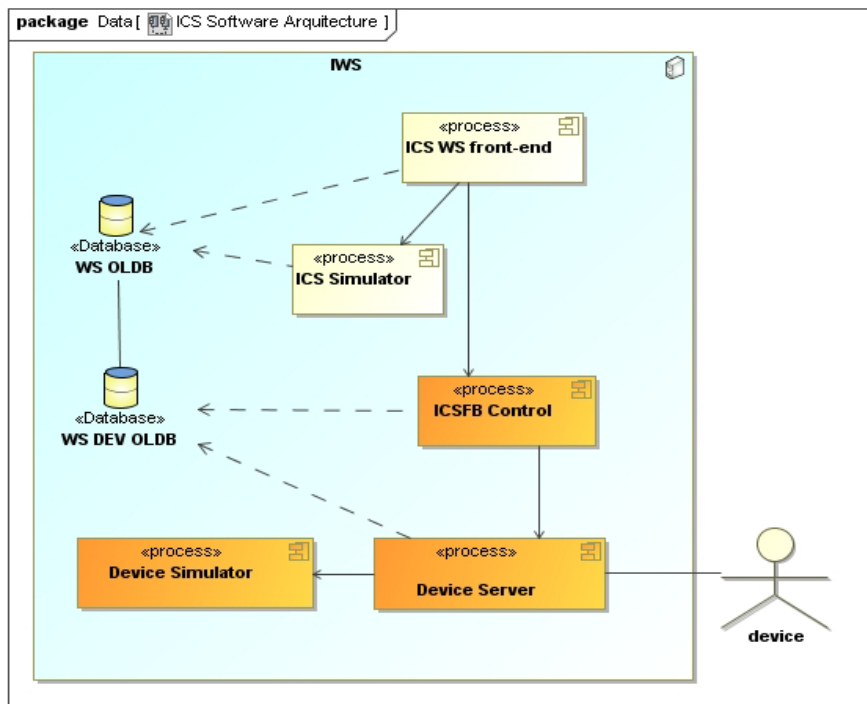

Figure -5: ICS Software architecture with the fieldbus extension

The communication between software processes is done through the VLT message system and via the online database (OLDB). The ICSFB CCS environment has its own database and the synchronization with the ICS WS part is achieved by means of the CCS scanning system. The ICS WS database structure for fieldbus devices has been simplified, but keeping compatibility with LCU devices so that the same graphical user interfaces could be reused. From the user point of view, there is no difference between devices controlled from traditional LCUs or devices handled by a PLC.

The overall architecture is very flexible and scalable allowing the control of complex VLT/VLTI instruments. One instrument could have multiple ICSs, each one having one or more ICSFB instances. Each ICSFB instance could handle several devices without presenting performance degradation[2]. It might be also possible to run ICSFB instances in a separate IWS when performance becomes an issue but this is not encouraged and should be properly justified. The amount of devices controlled by one PLC depends on the characteristics of devices and/or on the capabilities of the PLC. One PLC can control several devices making use of multitasking capabilities of modern PLCs.
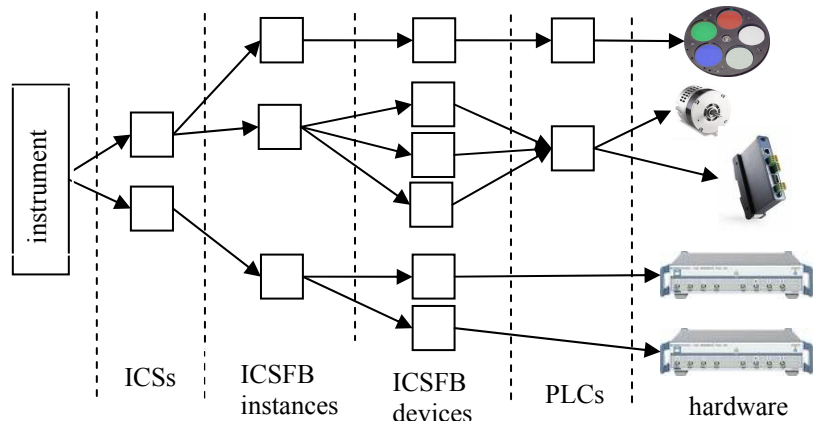

Figure -6: Overall ICS architecture with ICSFB.

3.4 Main Components

The ICSFB code consists of set of C++ classes that provides the basis for the framework and the foundation for implementing software devices, simulators and communication interfaces. Some of the core classes for the development of devices drivers and communication interfaces and the relationship between them are depicted in Figure -7. This diagram shows, as an example, a device driver for a lamp device that has one communication interface. At the instantiation of the device driver, two communication interfaces are assigned: one for normal operations and one for the simulation mode.
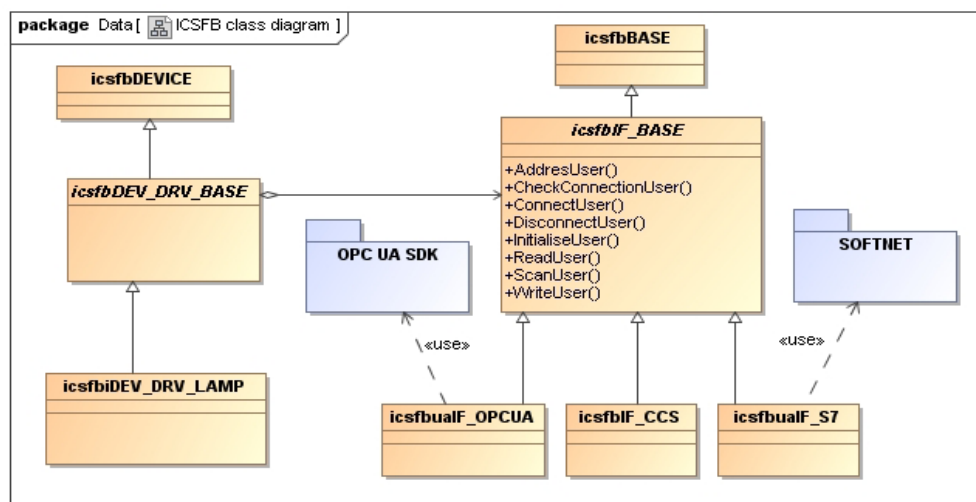

Figure -7: Some core classes of the ICSFB.

---

[2] According to the tests carried out on a standard IWS running 100 devices in one CCS environment (around 200 processes).

3.5 Communication Interface

ICSFB provides an abstract base class, "icsfbIF_BASE" (see Figure -7), that gives the basis for implementing the communication protocol between the device object and the controller. This class has a set of methods (see Table 2) that encapsulates the actions needed by any communication interface like for instance *connect*, *disconnect*, *read* and *write*. For each type of communication protocol supported, a class must be implemented, derived from this class.

Table 2. Main methods of base interface class (icsfbIF_BASE).

Methods	Description
AddressUser()	Register the address used to connect to the device. The method must parse and check the validity of the address, before registering it in the object.
CheckConnectionUser()	Check that the connection is in good condition.
ConnectUser()	Connect to the service provided for the device.
DisconnectUser()	Disconnect from the service provided for the device.
InitialiseUser()	May be implemented to initialize the basic housekeeping of the object. Note, no access to the HW should be implemented within this method.
ReadUser()	Read the value of one or several parameters.
ScanUser()	Scan the name space of the device and generate a vector with these names and the pertinent information about each parameter defined in the name space.
WriteUser()	Write the values of a given set of parameters to the device.

3.6 Device Driver

The base class allowing the creation of device drivers is "icsfbDEV_DRV_BASE". This class provides a set of abstract methods to be overriden by child classes adding the user implementations of each device. Following this scheme, all standard ICS commands like *setup, status, online, standby, off* are mapped into the methods of this class and derived classes should provide the specific implementation. In order not to block the functioning of each device driver while performing an action, there is another set of methods providing information about the status of the actions being executed.
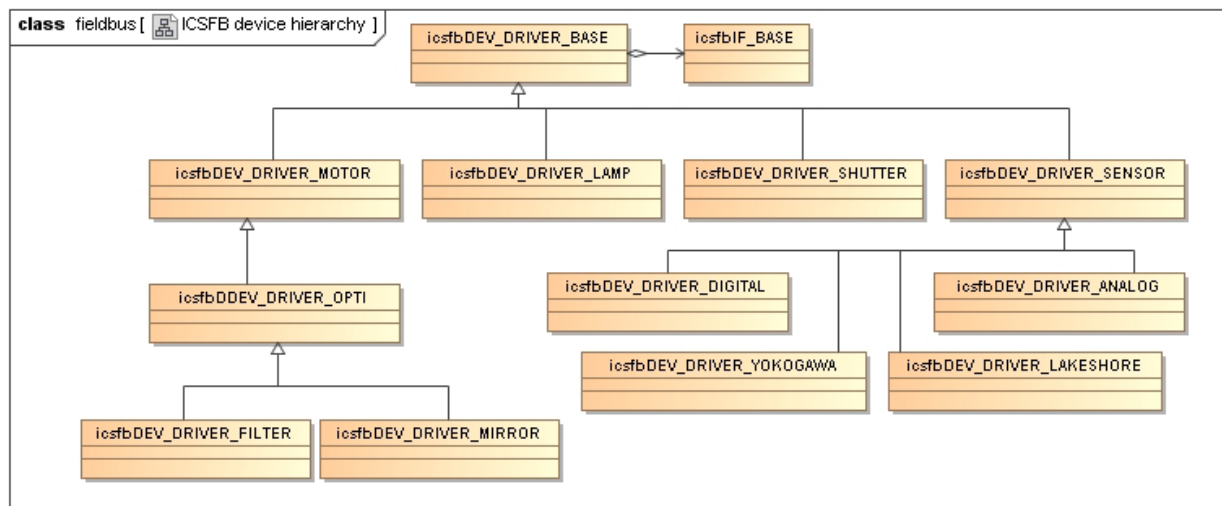

Figure -8: Example of class hierarchy for ICSFB device driver classes.

3.7 Industry Standards

The international standard selected to interface the VLT messaging system and the PLCs is OPC[6]. OPC is a well established and largely used communication interface in the area of industrial automation that has been historically supported only for Windows platforms but recently extended to a new and cross-platform specification named OPC Unified Architecture (UA). The usage of this standard will allow us to be independent of the vendor specific protocols and fieldbuses.

Although OPC UA is encouraged, we cannot use it at the VLT with traditional PLCs without the availability of an OPC UA Server for the Linux platform. At this moment, OPC UA Servers for Linux are not yet available on the market[10]. For this reason we have also implemented a communication interface for Siemens proprietary PLC protocol.

Table 3: Current ICSFB communication interfaces.

Interface	Description
OPC UA	Implements the OPC UA communication interface based on the Unified Automation OPC UA SDK[9].
S7	Implements the communication interface to Siemens PLCs via the access to Softnet Library. Softnet is a commercial library from Siemens supported under Linux that implements the communication with PLCs using the proprietary S7 protocol.
CCS	Implements the pure CCS interface for simulation purposes.

3.8 Complexity of ICSFB devices

Although it is probably too early to make evaluations and comparisons between the ICS LCU code and the ICSFB, it is interesting to see some SLOC metrics for both platforms. The following table shows the lines of code of a set of devices computed by *sloccount*[3]

Table 4: Lines of code of software devices in ICS LCU and ICSFB.

| | Traditional LCU | Field Bus Extension | |
	ANSI C	C++	PLC ST (structured text)
Lamp Device	4262	370	80
Shutter Device	3054	332	70
Digital Sensor	2059	233	50

The above shows a significant difference between the amount of code for the LCU ICS and the ICSFB that could be interpreted as a gain in simplicity. However, these are just preliminary results based on a beta version of the ICSFB devices so they are not conclusive.

The low number of lines of code in the PLC is explained by the simplicity of accessing the hardware within the PLC programming language and by the ICSFB principle of moving away the complexity from the low level part reducing it only to the mapping with the hardware and the definition of interface variables for the devices listed above.

---

[3] generated using David A. Wheeler's 'SLOCCount'.

```
(* Interface variables *)
bCtrlSwitch:        BOOL;                (*~ (OPC : 1 : comment) *)
bStatOpen:          BOOL;                (*~ (OPC : 1 : comment) *)
bStatClosed:        BOOL;                (*~ (OPC : 1 : comment) *)
bStatLocal:         BOOL;                (*~ (OPC : 1 : comment) *)
bStatFault:         BOOL;                (*~ (OPC : 1 : comment) *)
```

```
(* Physical signals *)
q_bCtrlSwitch       AT %Q*:    BOOL; (* Dig OUT for shutter control signal*)
i_bStatOpen         AT %I*:    BOOL; (* Dig IN for OPEN status signal*)
i_bStatClosed       AT %I*:    BOOL; (* Dig IN for CLOSED status signal*)
i_bStatLocal        AT %I*:    BOOL; (* Dig IN for LOCAL status signal*)
i_bStatFault        AT %I*:    BOOL; (* Dig IN for FAULT status signal*)
```

Figure -9: Extract of the ICSFB PLC code for a Shutter Device.

4. PIONIER INSTRUMENT – PILOT PROJECT

PIONIER is a VLTI visitor instrument being developed by *Laboratoire d'Astrophysique de Grenoble* (LAOG) and planned to be commissioned in Paranal during end of 2010. Based on the simple control requirements and due to the various constraints on the project like budget, power dissipation and flexibility, LAOG have decided to use COTS components instead of the traditional LCUs. PIONIER functions will be controlled by an Embedded PC (CX1030) and several EtherCAT modules like remote IO and stepper motion controllers from the company Beckhoff[7]. The CX1030 is an industrial PC, as well as a PC-based PLC suitable for medium performance control tasks. It hosts a real-time kernel controlling all the distributed I/O components through the EtherCAT fieldbus[8] and a Windows CE operating system for the non-realtime tasks including the industrial standard OPC UA server.

ESO, through the SDD/CIS department, has established a collaboration agreement with LAOG to develop the ICS extension so that it could be ready on time to be used as the control software for PIONIER. For its part, LAOG has made commitment to contribute in testing and commissioning this software.
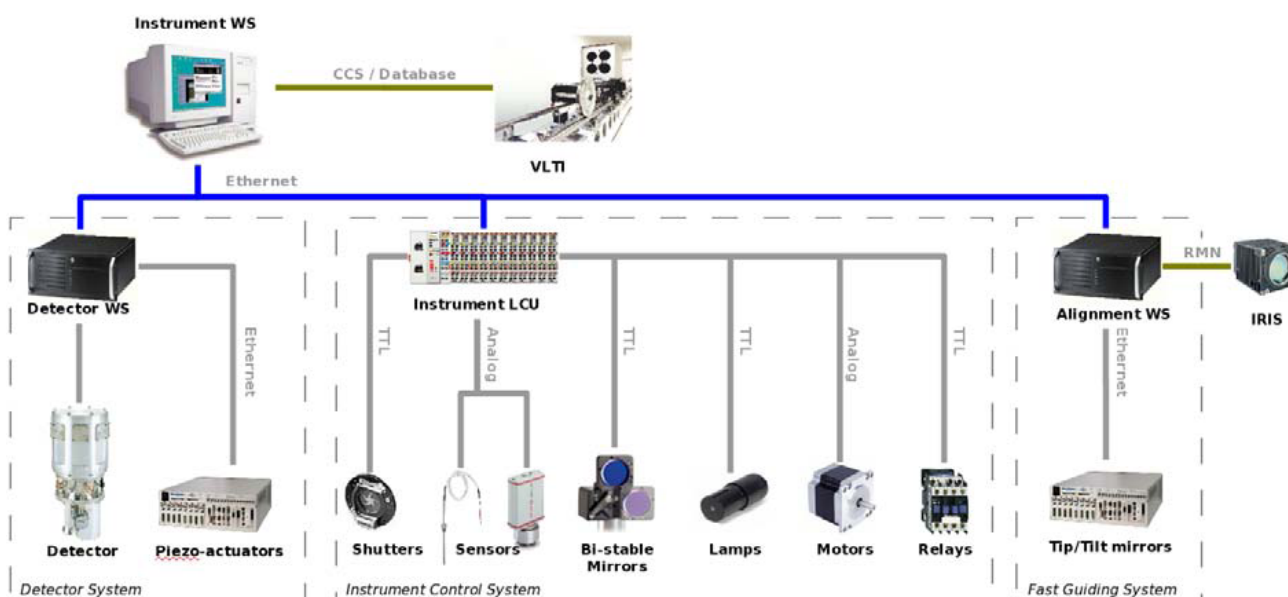

Figure -10: PIONIER Control System

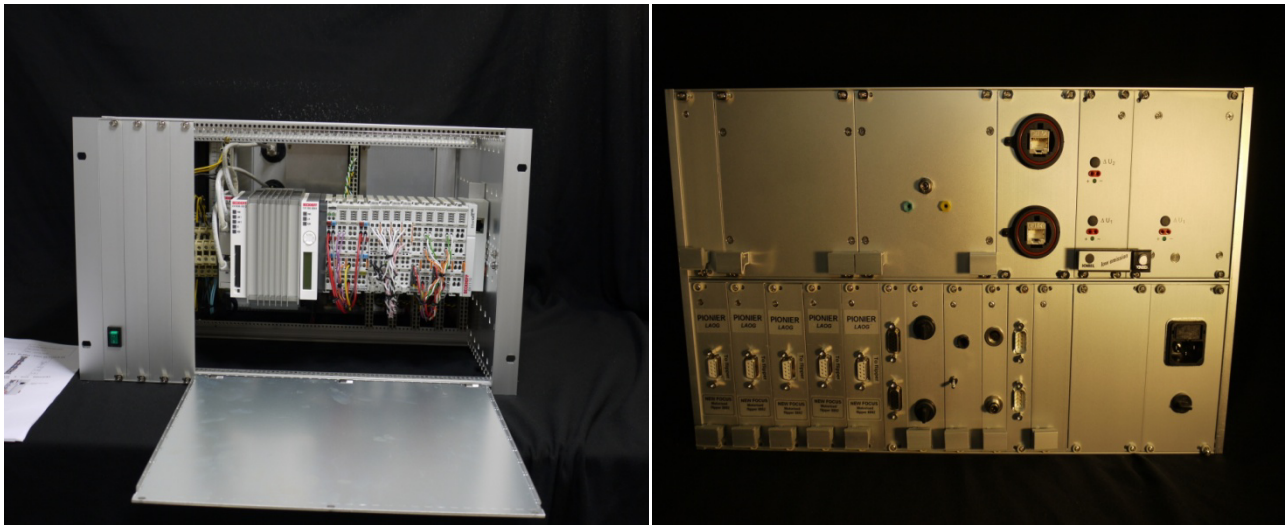Figure -11: PIONIER optical table.

Figure -12: PIONIER crate housing the Beckhoff Embedded PC and EtherCAT modules.

The Beckhoff system equipped with an embedded PC and EtherCAT fieldbus is used to control PIONIER motors, lamps, shutters and sensors. It is now being integrated with the instrument and it has been already tested with shutters and motors using the ICSFB software. According to the words of the PIONIER project manager "Some developments and tests have still to be done, but we have validated most of the critical issues:

- PLC programs - embedded programs on Beckhoff system
- OPC UA protocol - protocol used for communication between WS and Beckhoff system
- ICS extension - framework to support new fieldbus device drivers"

5. CONCLUSIONS

Throughout this paper we have presented the ICS fieldbus project which extends the current architecture of the ICS framework toward industry standards aiming to tackle the rising obsolesce problem of the VME-based LCUs. This new software not only tries to simplify the coding of instrument control software but it also attempts to introduce technologies envisaged for E-ELT instruments in order to use the VLT as a field test. We will have to wait until after the deployment, commissioning and operations of PIONIER to conclude about the success or failure of these solutions but the preliminary results obtained are already validating the efforts spent on this project.

The usage of PLCs and international standards simplify the low level control of devices. However it requires some time to learn how to use efficiently the tools to program and configure the PLCs and fieldbuses. OPC UA has a great potential to be used as the solution to interconnect PLCs coming from different vendors but it is not yet a mature specification. There is currently only one commercial SDK for Linux and no availability of OPC Servers from main PLC providers for this platform so a windows machine is required where to run the OPC UA Server. Embedded solutions like the *Beckhoff Embedded PC* overcome this problem by providing an integrated controller with Windows CE and the PLC real-time environment. Using communication interfaces for proprietary solutions is another way to solve this problem as it has been done in ICSFB for Siemens PLCs.

ACKNOWLEDGEMENTS

REFERENCES

[1] Allaert, E., INS Common Software Specification, ESO internal document, VLT-SPE-ESO-17240-0385
[2] Longinotti, A., VLT Instrumentation Software Specification, VLT-SPE-ESO-17212-0001 publicly available at ftp://ftp.eso.org/pub/vlt/vlt/pub/releases/VLT2009/vol-5a/VLT-SPE-ESO-17212-0001.pdf
[3] G Raffi et al., "The VLT control software in its final test phase", *Proc. of SPIE, Vol. 2871*, pp 996-1004, 1996.
[4] Gustafsson, B., "VLT local control unit real-time environment" Proc. SPIE, Vol. 21991014 (1994).
[5] Kaufer, A., "VLT(I) instrument operations and maintenance at the Paranal Observatory" Proc. SPIE, Vol. 6270
[6] http://www.opcfoundation.org
[7] http://www.beckhoff.de/
[8] http://www.ethercat.org/
[9] http://www.unified-automation.com/
[10] Kiekebusch, M., E-ELT Technology Demonstrator Report: OPC Unified Architecture Evaluation, ESO internal document, E-TRE-ESO-449-0529 Issue 1, (2009).