EVALUATION OF SOFTWARE AND ELECTRONICS TECHNOLOGIES FOR THE CONTROL OF THE E-ELT INSTRUMENTS: A CASE STUDY

P. Di Marcantonio#, R. Cirami, I. Coretti, INAF-OATs, via G.B. Tiepolo, 11, I-34143, Trieste, Italy
G. Chiozzi, M. Kiekebusch, ESO, Garching bei Munchen, 85748, Germany

Abstract

In the scope of the evaluation of architecture and technologies for the control system of the E-ELT (European-Extremely Large Telescope) instruments, a collaboration has been set up between the Instrumentation and Control Group of the INAF-OATs and the ESO Directorate of Engineering. The first result of this collaboration is the design and implementation of a prototype of a small but representative control system for an E-ELT instrument that has been setup at the INAF-OATs premises. The electronics has been based on PLCs (Programmable Logical Controller) and Ethernet based fieldbuses from different vendors but using international standards like the IEC 61131-3 and PLCopen Motion Control. The baseline design for the control software follows the architecture of the VLT (Very Large Telescope) Instrumentation application framework but it has been implemented using the ACS (ALMA Common Software), an open source software framework developed for the ALMA project and based on CORBA middleware. The communication among the software components is based on two models: CORBA calls for command/reply using the client/server paradigm and CORBA notification channel for distributing the devices status using the publisher/subscriber paradigm. The communication with the PLCs is based on OPC UA, an international standard for the communication with industrial controllers. The results of this work will contribute to the definition of the architecture of the control system that will be provided to all consortia responsible for the actual implementation of the E-ELT instruments. This paper presents the prototype motivation, its architecture, design and implementation.

INTRODUCTION

The mock-up instrument, to be controlled, is a kind of multi-object (optical) spectrograph composed by the following main subsystems and components:

- One spectrographic arm with an ADC (Atmospheric Dispersion Corrector) and a dedicated CCD;
- One imaging arm with a filter wheel and a dedicated CCD;
- One calibration system equipped with a simple on/off lamp (e.g. Thorium-Argon lamp) and one characterized by a warm-up time required to reach the necessary stability (e.g. Deuterium lamp).

The skeleton of this prototype is based on a real astronomical spectrograph that is being built at INAF-OATs premises and is representative of the components that we expect to be part of a real E-ELT instrument.

#dimarcan@oats.inaf.it

FUNCTIONAL REQUIREMENTS

A detailed set of requirements is a starting point to produce both adequate software and electronic architecture design. For our specific prototype we reused the work done both by ESO and INAF-OATs as part of the studies for the E-ELT Phase A instruments. Dedicated use cases were developed covering aspects like system/instrument start-up, simulation levels, handling of logs and errors, and, typical of an astronomical instrument, handling of the acquisition, observation and calibration phases.

We have reused, moreover, some successful paradigms employed in the VLT software instrumentation framework ([1] and reference therein) applying the experience gained during the software development for the VLT instruments.

Command/ eply

Sub-systems and, in general, devices of the system respond to commands (e.g. *SETUP*) which execute specific actions. Each command is composed by a string made up of one or more keywords (in FITS-like format) and parameters chained together (e.g. *INS.FILT1.NAME U*). Each keyword specifies the subsystem and device the message is addressed to.

State Machine

Devices and components needed to operate the prototype follow the typical VLT state machine. The states implemented are: OFF (not running), LOADED (started-up), STANDBY (software initialized), ONLINE (hardware initialized) [1].

Parallelism

Operations in our prototype are executed in parallel. At the device level this means that all devices are capable of moving in parallel to spare time and minimize initialization procedures. At the higher level (i.e. interaction with the user) it means that the corresponding sub-system does not block upon receiving a request.

SOFTWARE ARCHITECTURE

The baseline design for the prototype Control Software architecture follows a VLT-like approach [1]. Several blocks/packages constitute the overall architecture and could be summarized as follows:

- Low-level control package (identified as **ICS** – the *Instrument Control Software*), responsible for handling the vital part of the prototype (motorized devices and lamps in our case);

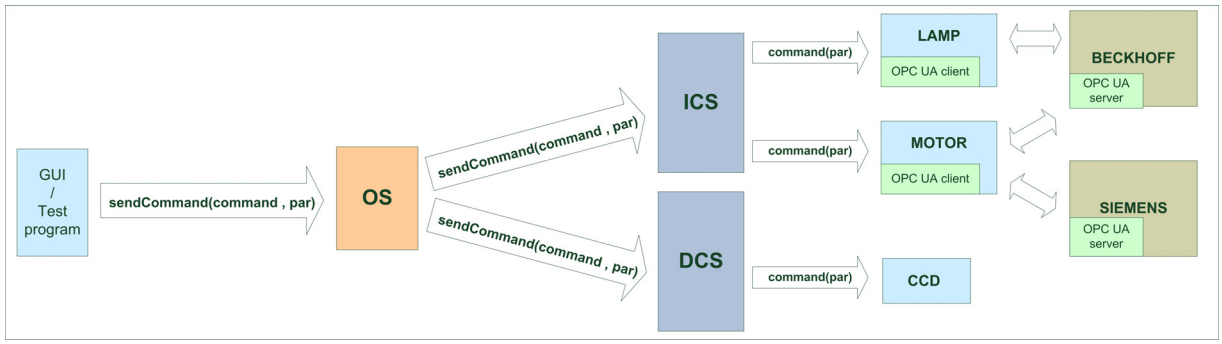

Figure 1: Mock-up prototype software architecture.

- High-level control package (identified as **OS** – the *Observation Software*), responsible for the coordination of activities of the detectors, low-level part and the telescope;
- Detector Control Software (**DCS**), responsible for controlling/handling all detectors.

We decided to use the ALMA ACS framework [2] as the infrastructure software to build our mock-up prototype instrument control software. The reason of this choice is essentially twofold:

- to use a modern, fully fledged, application framework to speed-up the prototype implementation
- to evaluate ALMA ACS complexity (learning curve), completeness, performance and ease-of-integration with other technologies in a view of its possible usage inside the E-ELT project.

The ACS philosophy is to model the system as a set of collaborating components located inside one or more containers contrary to the VLT case where the various software packages are implemented as a set of standalone processes.

Figure 1 shows the final prototype architecture. It is just a schematic picture and does not show all the involved entities (e.g. call-backs, threads). A GUI/test program sends a command to OS, which forwards it to the involved subsystems. Then the subsystem (at present ICS or DCS) sends the command to the appropriate devices. In the case of ICS, the connection between the ACS Components (LAMP or MOTOR) and the hardware (Beckhoff or Siemens PLCs) is realized through the OPC UA technology [3].

In the current version of the prototype the following C++ (ACS) Containers have been configured:

- *OSContainer*
- *ICSContainer*
- *DeviceContainer*, which hosts the LAMP1, LAMP2, FILT1 and ADC1 ACS Characteristic components;
- *DCSContainer*, which hosts the CCDIMAG and CCDSPECTRO ACS Characteristic components.

The motivation of having (so many) different containers even though the system is quite simple is to model it as a highly-distributed system exploiting the great flexibility of the ACS Component/Container model. This will facilitate future modifications. For example, OS and ICS, first implemented in C++, have been in parallel implemented in Java and deployed on a dedicated Java Container, in order to assess the feasibility of using this language for high level coordination tasks.

OS and ICS General Characteristics

In this prototype, the bootstrap of the system uses ACS activation on demand: each component is activated automatically only when requested by another component or by a client. In this way, if there is a need to interact with a specific device it can be just connected to a GUI and the corresponding component will be started up. On the other hand, if there is a request to activate the top-level OS, it will hierarchically activate ICS which in turn will be responsible to activate the needed devices. The ACS configuration database contains the needed deployment information.

The system is designed and implemented to allow executing operations fully in parallel. At the sub-system level (ICS, OS, DCS), command/parameters pairs are received (using a standard SETUP-command syntax implemented via CORBA method invocation) and the corresponding operations executed. Operations are non-blocking i.e. after checking the correctness of the received parameters the involved Component is free to accept new requests. At the OS level this is achieved via call-back mechanism. Before dispatching the incoming command to ICS or DCS, a call-back is instantiated and passed, together with the command, to the appropriate subsystem. In this way, it is responsibility of the involved subsystem to notify when the action has completed allowing OS to handle new requests. Once notified, OS propagates the answer back to the client (see Figure 2).

At the ICS level, a multi-threading paradigm is used instead. Parameters are parsed and sent to the appropriate device using a dedicated thread. Parallel actions are handled by separate threads and a thread join is used as a rendezvous point when all actions are completed (see Figure 2). This allows ICS to receive new commands if required (e.g. an emergency STOP or a SETUP on different devices).

The reason to use the call-back mechanism only at the OS level is due to the fact that OS (in principle) manages a limited number of sub-systems. Call-backs inside ACS are basically CORBA objects. They are therefore somehow "heavy" to activate and this could, at length, lead to some performance penalties.

Adopting these two different paradigms allowed us to compare the two options and to evaluate the easiness of their usage. The final impression is very positive: at the development level both paradigms are fully supported by ACS and also from performance point of view they behave as expected (see final section).

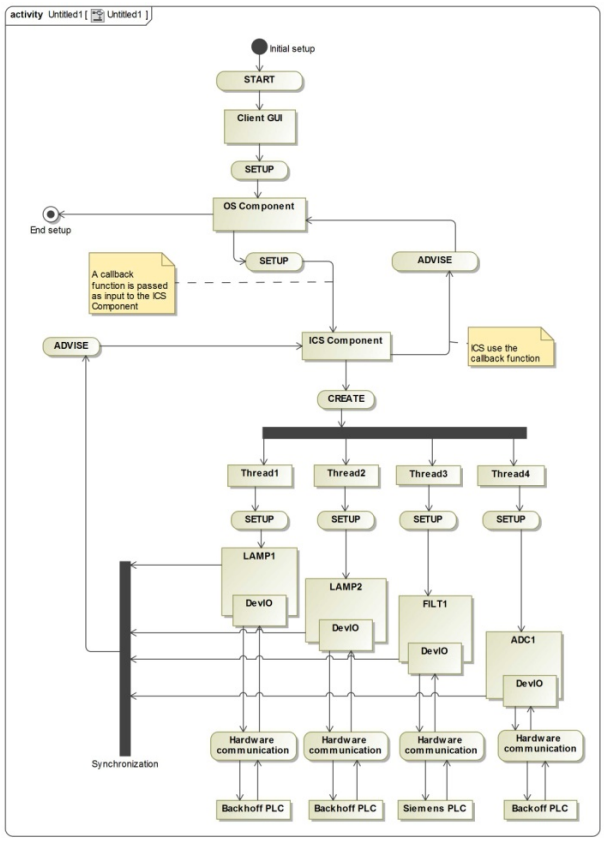

Figure 2: Activity diagram (only ICS part is shown).

The Java implementation for both OS and ICS components has been developed following for both cases the ICS multi-threaded architecture described above. This implementation has demonstrated that Java development is very convenient for these applications and has allowed us to demonstrate also the re-use of other system elements developed for ALMA (like the standard ALMA Master Component state machine or ALMA operator GUIs).

Devices and OPC UA tandard

Both ICS and DCS control hardware devices. Connection to the ICS PLC hardware is handled via OPC UA [3]. The OPC UA is a platform-independent standard through which various kinds of systems and devices can communicate by sending messages between clients and servers over TCP networks. In an OPC UA application we can usually find the following components:

- a "server", usually a thread on the process controller which exposes process data and methods that might run on another node and communicate with the hardware with proprietary protocols. It is supplied (typically) by the hardware manufacturer;
- a "client" that, by means of APIs, implements the OPC UA communication stack and allows the user application to access the data and methods exposed on the "server" side.

In the current prototype version the devices controlled by ICS are:

- LAMP1 – a simple on/off device (mimic a true ThAr lamp) controlled physically by a Beckhoff PLC;
- LAMP2 - a simple on/off device (mimic a true Deuterium lamp), with functionalities similar to Lamp1, but with a warm-up time i.e. the time that the lamp needs to reach full operative mode. Also this lamp is physically controlled by a Beckhoff PLC;
- FILT1 – a filter wheel with certain number of discrete positions controlled physically by a Siemens PLC;
- ADC1 – rotational device physically controlled by a Beckhoff PLC.

DCS controls the following devices:

- CCDImag – implements the interface to a Finger Lakes Instrumentation (FLI) CCD USB Camera.
- CCDSpectro – implemented with dummy responses.

We implemented our own OPC UA client inside the ACS by means of DevIO abstraction layer [2] using the OPC UA client provided by Unified Automation. The OPC UA SDK library is well wrapped inside just two classes: the first one implements all the steps that are necessary to correctly access (i.e. read, write, subscribe) a specific node through a DevIO interface; the second one handles all the connections/disconnections with the PLCs.

Another project based on ACS [4] is now taking this implementation as a starting point for writing a similar implementation on Java. Once this will be done, we will retrofit this prototype with a Java implementation of Device, to verify if also this lower level control part can be implemented conveniently in Java.

ELECTRONIC ARCHITECTURE

One of the aims of the prototype has been to evaluate possible hardware technologies that in terms of performance and cost could be employed in the actual implementation of the E-ELT instruments. Following the description given in the preceding sections (usage of PLCs and communication through OPC UA standard) two complete hardware infrastructures have been set-up to this purpose at our laboratories: one based on Beckhoff TwinCAT SoftPLC platform and the other on Siemens S7-300 PLC platform. Specifically the Beckhoff system is a CX9000 family embedded system equipped with I/O modules (EL1002, EL1014, EL2004) and the AX5201 servo drive able to control the AM3022 brushless synchronous servomotor. The Siemens system is a Compact CPU (CPU 314C-2 DP) equipped with the FM354 Servo Function module able to control a custom made DC drive used in ESO/VLT instruments.

In the Beckhoff case, the PLC software has been derived from the code developed for the ESO/VLT PIONIER instrument and uses the Motion Control (MC)

library conforming to PLCopen IEC 61134-3 as interface to the motors. In the Siemens case, due to the lack of PLCopen compliant libraries (cost and compatibility with our current hardware layout is under evaluation), dedicated code has been developed based on the Siemens FM354_354 block library. Note that the same subset of process variables as the one used in the Beckhoff implementation were exposed making the two platforms fully and transparently interchangeable. This is for sure one of the major achievements of this project: the prototype has shown that thanks to the ACS DevIO abstraction layer and a clever usage of the OPC UA standard, the overall system is unaware of the employed underlying technology/vendor products. By only changing few (ASCII) configuration files, but without having to recompile high-level ACS Components code or changing GUIs, it has been possible to control motors/lamps using platforms of the two different vendors with the possibility to interchange them in a completely seamless way. For the E-ELT instruments this has a quite big impact; it means that it could be possible to change the underlying low-level technology (e.g. due to obsolescence) without affecting the high-level framework or vice-versa in fully transparent way.

Performance

Performance measurement has been done with a mock-up for an ADC connected to the the Beckhoff platform. An ADC is a device able to compensate (counter-balance) the dispersion produced by the terrestrial atmosphere on the incoming starlight, composed by prisms that must be free to rotate. Their position depends on the celestial position of the object to be observed and must be therefore continuously updated.

The performance test that has been carried on is relatively simple: within a thread (spawned by the ACS Component Device associated to the ADC), the new position is calculated (simulating a real astronomical object) and then two calls are made trough OPC UA. The first call updates the target position; the second triggers the actual movement of the motor. To try to reach the real limit of the system, no check is performed on the positioning by the high-level software. In the case the motor is still positioning while a new "move" request is triggered, the target position gets simply updated and the motor continues its movement.

Measurements were made by time stamping, within the ACS code, when a thread is entered, before the first OPC UA call and at the thread exit.

At the PLC level, a physical output was mapped to the "move" flag and then sampled with an oscilloscope. No other tools where accessing the PLC by network when the test was made.

Analysis of these measurements shows that, as expected, most of the time is spent in the OPC UA communication (astronomical computation and thread life cycle management by ACS is negligible compared to the OPC UA calls). With our slow CX9010 system (consider that this is the cheapest Beckhoff CPU family able to run an OPC UA server) and a 10 ms PLC cycle the average time between two consecutive thread call is of the order of 80 ms as confirmed also in Figure 3 by an oscilloscope screen capture.

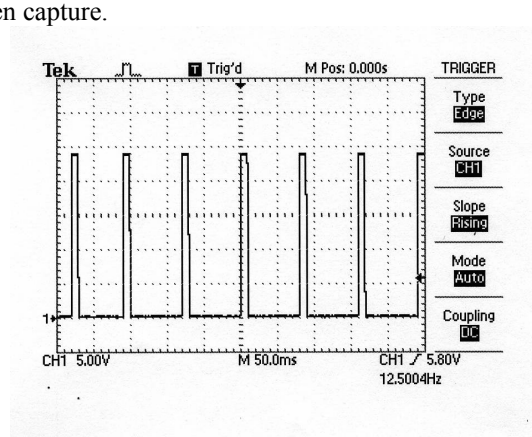

Figure 3: oscilloscope screen capture. Average time between two successive "move" commands is of the order of 80 ms.

Results can be certainly improved using a more performing CPU family (e.g. CX1030), but the actual measurements confirm already now that our (relatively) cheap hardware system connected to the ACS framework through OPC UA is able to sustain cycles of the order of 10 Hz which is well within the requirements asked for our applications.

CONCLUSIONS

The main outcome of our work is that the selected technologies fulfil the imposed requirements and are feasible alternatives to implement instrument control software for the E-ELT. ACS strong points are for sure the Component/Container paradigm, the transparent managing of the Component lifecycles, the DevIO interfaces and the various services offered by the framework which ease implementation (e.g. threads).

The major strength in using OPC UA is the transparent hardware management. Measurements show that performance achieved at the communication level are within specification for most astronomical requirements and therefore OPC UA, in perspective, could be further evaluated as an appealing technology to be employed for the future E-ELT instruments.

REFERENCES

[1] M.J.Kiekebusch et al., "Evolution of the VLT instrument control system toward industry standards", Proc. SPIE 7740, 77400T (2010)

[2] G. Chiozzi et al., "ALMA Common Software (ACS), status and development", ICALEPCS, TUP101, (2009)

[3] Mahnke et al., "OPC Unified Architecture", ISBN 978-3-540-68898-3, Springer (2009)

[4] I. Oya et al., "A Readout and Control System for CTA Prototype Telescope", ICALEPCS, MOPMU026, (2011)