An object-oriented event-driven architecture for the VLT Telescope Control Software

G.Chiozzi
European Southern Observatory, Karl-Schwarshild-Str. 2
D-85748 Garching bei Muenchen, Germany
Email: gchiozzi@eso.org

Abstract

The control software for the Very Large Telescope follows the "Standard Architecture" and is distributed over several workstations, that provide high-level and coordination services, and VME based systems, for real-time control purposes.

The adoption of object-oriented design techniques and the support of a C++ application framework for the implementation of Event-Driven systems reduces considerably the complexity of the applications. The framework provides a general application skeleton and services to automatically receive and analyse events. These are then dispatched to specialized objects, designed to handle specific events.

Since all the run-time and configuration data of the whole VLT is stored in a distributed real-time database, there is a strict coupling between the structure of the database and the applications. As a consequence, the real-time database, although based on the hierarchical model, has been structured to provide support for object-oriented design and implementation.

This paper describes the architecture of the Telescope Control Software (TCS) for the VLT and the object-oriented infrastructure on which it is based.

1 INTRODUCTION

The control software for the Very Large Telescope (described in more details elsewhere in these proceedings[5]) follows the "Standard Architecture"[7] and is distributed over several workstations, that provide high-level and coordination services, and VME based systems, for real-time control purposes. The communication between all the processes running on these machines is based on a message system and a distributed hierarchical database.

This architecture implies that all the applications, but in particular the coordination processes running at workstation level, must be ready at any moment to accept and handle a lot of different kinds of messages, such as new commands, alarms, and notifications from the controlled sub-components.

As a consequence, the design and the implementation of the coordination applications has an high degree of complexity, with an obvious impact on development and debugging time.

The adoption of object-oriented design techniques and the support of an application framework for the implementation of Event-Driven systems reduces considerably this complexity. The framework, based over a set of C++ classes, provides a general application skeleton and services to automatically receive and analyse events. These are then dispatched to specialized objects, designed to handle a specific set of events without having to take into account other parallel but independent conditions.

Given these services, the design and implementation of an application consists of the design of a set of smaller and independent objects specialized in the handling of specific events, such as a command, a change in some database values, the tic of a periodic timer, etc.

Since all the run-time and configuration data of the whole VLT is stored in a distributed real-time database, there is a strict coupling between the structure of the database and the applications. As a consequence, the real-time database, although based on the hierarchical model, has been structured to provide support for object-oriented design and implementation.

This architectural approach has a number of advantages, both in the design/development and maintenance phases; in particular it enforces the respect of standards and provides a mean of sharing and reusing code in the VLT software team.

2 VLT CONTROL SOFTWARE

The control software for the VLT is responsible for the control of the 4 main telescopes and of the auxiliary ones, the interferometer and the instruments attached to the light beams. The main telescopes can be operated individually or in combination; on an individual telescope multiple instruments can be used simultaneously (one for observation, the others for calibration, preparation or maintenance). This correspond to a distributed environment of 120-150 coordinating workstations and microprocessor based VME systems (Local Control Units, LCUs)[6].

The basic software for the VLT control system, called the Central Control Software (CCS)[3][4], is a large set of modules designed to provide services common to many applications. Most of these common services are available both on workstation and LCU platforms and provide on both the same application programming interface.

The main group of functions provided in the CCS are:

- message handling
- on-line real-time database
- error and alarm handling
- logging
- process I/O
- event monitoring

The real-time database definition language and a framework of C++ classes for the development of event driven applications provide support for object oriented programming. These are described in more detail in the following paragraphs.

3 TELESCOPE CONTROL SOFTWARE

The Telescope Control Software (TCS)[8] controls the main telescopes and the attached equipment that is common to instruments. In particular, for every telescope it controls the main axes (altitude and azimuth), the mirrors, three instrument adapters, the CCDs for auto guiding and active optics, the enclosure and other auxiliary equipment.

The complete VLT will have four equivalent copies of TCS, one per telescope.

3.1 Software architecture

The TCS software can be grouped into four categories:

- user interface
- coordinating software
- subsystem application software
- special interface software

The subsystem application software implements all the functions that can be performed locally in an LCU, without any knowledge of other components of the system. It controls basic actions and motions of subsystem devices such as the main structure of the telescope, mirror supports, adapters and the enclosure.

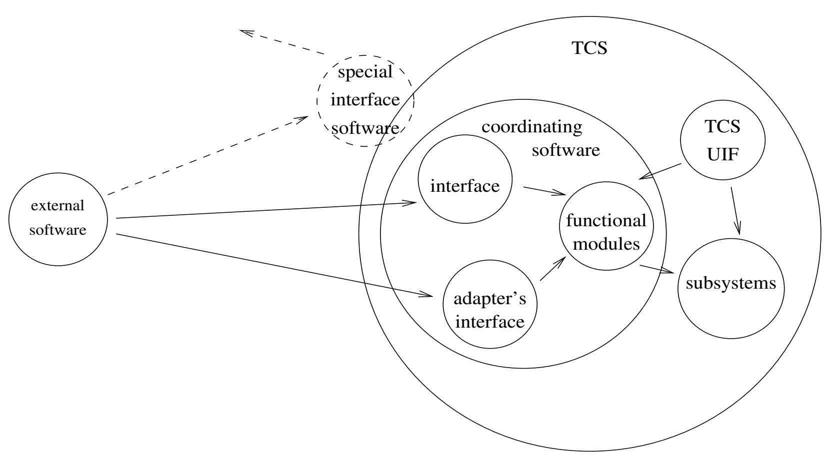

Fig. 1- categories of TCS software

The coordinating software is hierarchically above the subsystem software. It performs actions of combined, coordinating nature, and it often uses one or more subsystems to execute its actions. It runs mainly on the telescope control workstation, although there are major parts running in LCUs.

The special interface software consist of libraries to access, via TCS, external systems such as star catalogues and the Astronomical Site Monitor subsystem.

While the software running on the LCUs has been written in C using traditional techniques, the workstation software is designed using an object oriented methodology and coded in C++.

3.2 Coordinating software

The coordination software provides a public interface for external applications requiring services to TCS and a set of "functional" modules. These are the core modules of TCS software and perform all the coordination work, interacting with each other and with the subsystems.

Some of these modules are for:

- Presetting (setting the telescope to a new target)
- Tracking (maintaining object position following its movement in the sky, without guide star feedback)
- Autoguiding (corrections to telescope position based on guide star feedback)
- Enclosure control (dome rotation, doors, windscreen, louvers)
- Active optics (main mirror axial support and mirror two-position corrections)

All inter-module communication and all communication with subsystems, make use of the CCS message system. Each module has a command interface which is the one that is normally used for all inter-module communication and which is also used by the TCS user interface panels.

3.3 Coordination module's architecture

All coordination modules have the same architecture and their implementation is founded on an object oriented framework based on C++ classes and real-time database classes. Thanks to this approach, maintenance of already existing modules and development of new modules is much easier and, at the same time, VLT conventions and rules are automatically enforced because they are implemented in the base classes used in the implementation.

All the modules are implemented through one or more independent processes.

If a module is made up of more than one process, only one of them is responsible for providing the public interface to the external world, i.e. to receive commands from external modules and to send back the corresponding replies.

The other processes are just slave processes used to perform specific sub-tasks and to improve parallelism.

All coordination modules receive commands from external applications, analyse and elaborate the incoming data and initiate all the necessary actions sending requests to other coordinating modules and to subsystems. They must always be ready to process a new command within a typical "command response time" of 100 milliseconds and many actions must be initiated and handled in parallel as far as possible.

4 THE EVENT HANDLER

In order to meet this requirement, every process is designed in an event-driven way and the implementation is based around the event-handler provided by the evhHANDLER class, which is a core component of the C++ library provided by CCS[1].

Every process contains an instance of this class.

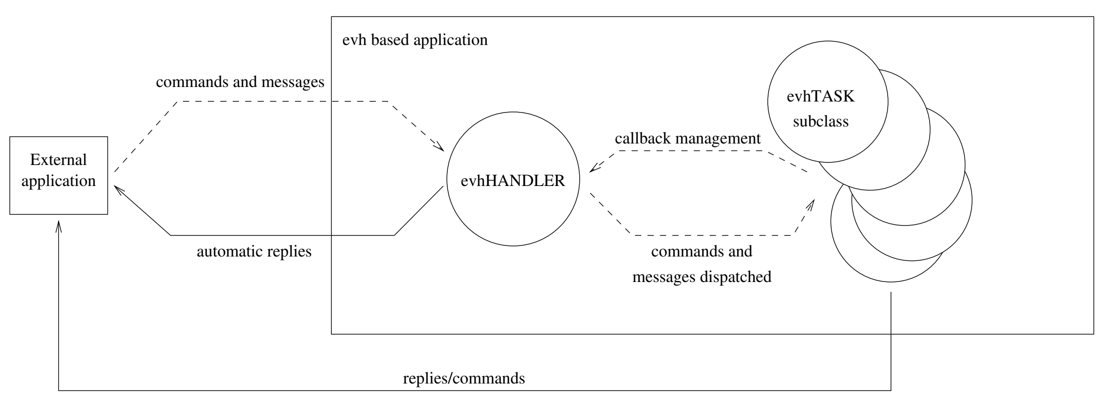

Fig.2 - Event Handler Architecture

This object receives and parses all the incoming events, such as:

- CCS commands
- CCS replies and error replies to sent commands
- CCS time out notifications

- CCS periodic timer notifications
- CCS Data Base Event messages
- UNIX signals
- UNIX file inputs

It then searches an internal database for a list of functions or object methods to be called in order to handle that specific event. Several callbacks, in principle unlimited, can be registered for a single event. For synchronization purposes, it is also possible to define callbacks to be invoked after a combination of events. The callback list is dynamic and the evhHANDLER class provides methods to add and delete elements from the list.

Every list of callbacks must return to the event handler within the “command response time” of 100 milliseconds and the event handler itself takes care of checking this condition, issuing error messages if it is not met (coordinating modules have only soft and not hard real-time requirements).

The design of every process consists mainly in the design of the independent objects providing the methods to be attached as callbacks to the event-handler.

Usually there is one object for every task (a TASK object) that can be executed in parallel inside a process and all the objects have their own independent “life”. Whenever an object is waiting for commands or for messages and other events from subsystems, the event-handler is ready to receive events and dispatch them to other objects.

5 THE TASK CLASSES

In order to help to implement the TASK objects and enforce at the same time that the VLT standard rules and protocols are followed, a wide set of base classes are provided.

For example, it is required that every VLT module must be able to handle a certain number of “standard commands”. As a consequence an evhSTD_COMMANDS class exist that implements all the standard commands in consistent and reasonable way. A fully compliant VLT application can be built with a few lines of C++ code just allocating an instance of evhHANDLER and one of evhSTD_COMMANDS classes. If a command must be implemented in a different way, the related callbacks can be overloaded or a new object can be allocated for its handling.

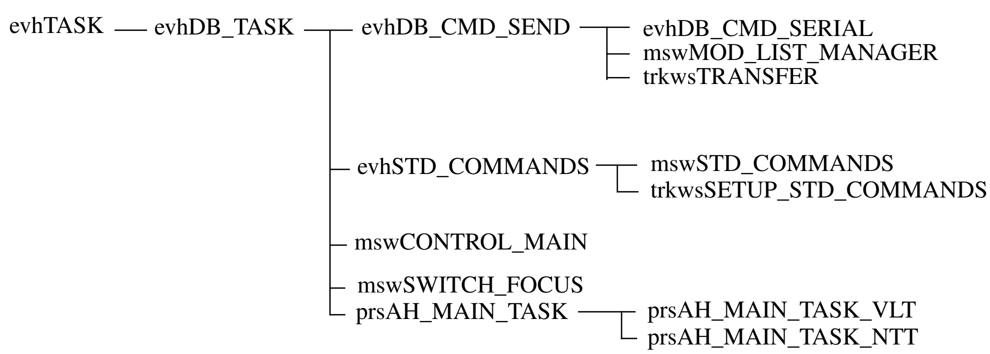

Fig.3 - Some of the classes in the hierarchy of evhTASK

Other classes provide a standard architecture to send commands to one or more external modules and collect the replies. The developer must just derive a sub-class where one or more callbacks are overloaded to specify the object’s behaviour when successful replies, error conditions or time-out events are received. This scheme can also handle complex synchronization and queuing protocols, saving a lot of development time and, most importantly, warranting that the same concept is implemented consistently and in the same way everywhere. This is of great benefit for maintenance and debugging.

6 THE REAL-TIME DATABASE

All data that can be of interest for external modules, to get a picture of the status of the system, or for system tuning and configuration, are stored in the real-time database.

This is a hierarchical database distributed on the different workstations and LCUs, mapping the physical and logical objects that constitute the VLT control system and describing how they are logically contained, one inside the other (how one component is “part-of” another one).

Each local database has a partial image only of the units of the system that the particular workstation or LCU has to use. The whole database is the merge of all the local sections.

There will be a lot of different places in the database describing objects with the same or a very similar internal structure and behaviour. For example the VLT system will have a lot of motors, encoders or moving axes that have the same general characteristics.

This description of the VLT real-time database fits very well in an object-oriented model, but the implementation of the database itself is based on a commercial hierarchical database (RTAP, from Hewlett-Packard) that does not support object oriented concepts.

For this reason we have developed a preprocessor and a class browser that extend the semantic and syntax of the RTAP database definition language, introducing the concept of class, with inheritance and overloading[2].

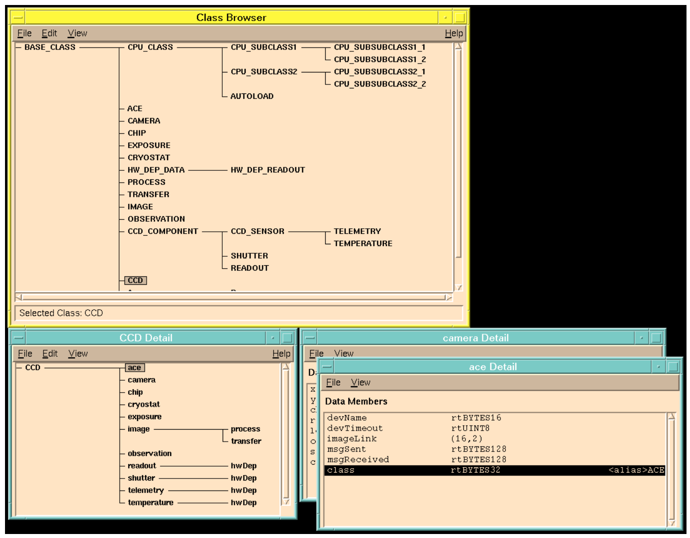

Fig.4 - User Interface for the DBL class Browser

RTAP syntax allows only to define "points" as data structures containing a set of "attributes" of predefined basic data types. The hierarchical structure is built defining a new point as a sub-point of an already existing one. If the same data structure (for example describing a motor) is used in more places, the whole definition of the database branch must be copied.

The extended syntax introduces the concept of "class" as a definition of a new structured data type that can be used as any other available data type. This means that a class instance can be used just in the same way as a native type while defining attributes inside new classes or points.

Each new class has to be derived from another one, from which it inherits all the attributes. Inside a class definition (or a point instantiation) it is also possible to:

- redefine attributes to assign new initial values
- add new attributes
- overload structured attributes

Methods to be used from C++ applications can be implemented developing a C++ twin class, using specific classes provided in CCS.

7 THE TCS DATABASE

Every TCS module defines its own database branch on the workstation's database, following a standard structure. This is automatically imposed by using the database classes coupled with the C++ classes provided by the event handling application framework: for every C++ class accessing the database, there is a twin database class containing the required points and attributes.

The use of inheritance lets the programmer specialize the database class according to the changes in the C++ class.

The application "owner" of the database branch is the only one with write access. All the other applications can read database values by accessing them directly or, better, through the provided C++ access classes.

The following figure shows a simplified view of a TCS database:

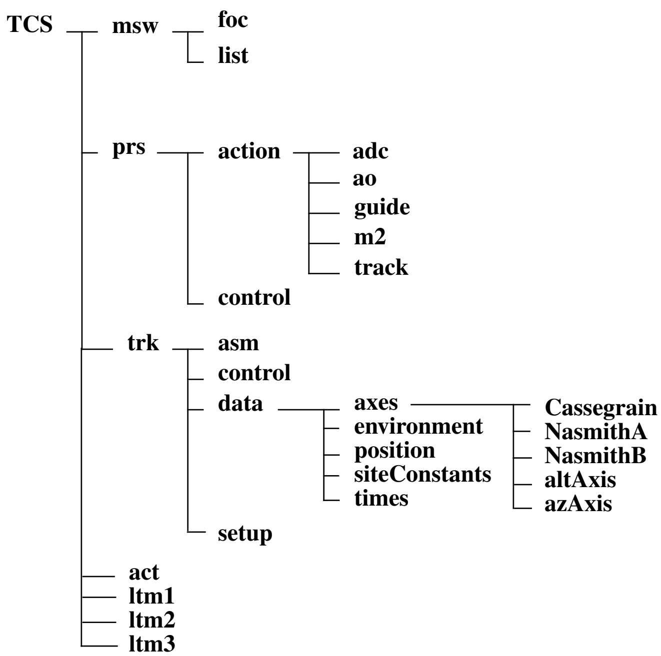

Fig.5 - Structure of a TCS real-time database

8 STATUS

The real-time database preprocessor and the event handling tool-kit have been developed during the second half of ‘94 and the first half of ‘95 and have been extensively used in many VLT applications.

In particular they have been used for the development of TCS part 1 (which includes Presetting, Tracking and Mode Switching modules), released in September ‘95.

TCS part 1 will be tested at the end of this year on the NTT (New Technology Telescope) in LaSilla as part of the upgrade of its control software. The NTT will use as much as possible the same software as the VLT, in order to reduce maintenance resources and to test it before the VLT itself is available. In the case of the TCS, the software has been explicitly designed for this purpose, isolating in specific subclasses all the functionality that is different in the two telescopes. In every case where specific behaviour must be implemented, a base class provides whatever can be in common and two specific sub-classes (one for NTT and one for VLT) implement the part that is unique to each system.

In the first half of ‘96, there will be tests on a pre-assembled telescope at the manufacturer’s site in Milan, Italy. It will be a test of the telescope structure, drive system and encoders and also of TCS part 1.

As a by-product of the development of TSC part 1, a number of general classes have been developed and added to the set provided by CCS. These classes are now used by other applications.

9 CONCLUSION

The adoption of an object oriented design methodology, with the essential support and guide to implementation provided by the C++ application framework, has allowed the development of a Telescope Control System that is intrinsically more maintainable and extendible compared with traditional techniques.

The initial effort spent in designing the general event-driven architecture and implementing the support toolkit has been already paid back by the fact that the software can be easily adapted to other systems (like the NTT and, in perspective, the auxiliary telescopes). Moreover, the same general concepts are now being applied to other components of the VLT control system, like instrument software.

This approach provides a number of benefits:

- Different applications, developed by different teams, share the same architecture, making easier maintenance in the future.
- Standards and conventions are not only stated “on paper” but automatically enforced by the use of standard classes.

- Important code components are better tested because they are extensively used in many applications.

10 ACKNOWLEDGMENTS

The author wishes to thank all colleagues in the VLT Software Group , and any other ESO staff, who have contributed to the concepts, ideas and software reported in this paper.

11 REFERENCES

[1] G.Chiozzi - CCS Event Handling Tool-kit User Manual - VLT-MAN-ESO-17210-0771, European Southern Observatory, 1995

[2] G.Chiozzi - CCS On Line Database Loader User Manual - VLT-MAN-ESO-17210-0707, European Southern Observatory, 1995

[3] B.Gilli - Workstation environment for the VLT - Proceedings of SPIE, vol.2199, pp.1026-1033, 1994

[4] B.Gustafsson - VLT Local Control Unit Real Time Environment - Proceedings of SPIE, vol.2199, pp.1014-1025, 1994

[5] G.Raffi - Status of ESO/VLT Control Software - These proceedings

[6] G.Raffi - Control Software for the ESO VLT - Proc. Int. Conf. on Accelerator and Large Experimental Physics Control Systems, Tsukuba, Japan, 1991, KEK Proc. 92-15

[7] C.O.Pak, S.Kurokawa, T.Katoh - Eds. Proc. Int. Conf. on Accelerator and Large Experimental Physics Control Systems, Tsukuba, Japan, 1991, KEK Proc. 92-15

[8] K.Wirenstrand - VLT Telescope Control Software, an overview - Proceedings of SPIE, vol. 2479, pp.129-139, 1995

NOTE: The postscript files of the ESO documents, included those mentioned in this paper, are available through anonymous ftp at ftp.eso.org.