# The LBT-AdOpt Arbitrator. Coordinating many loosely coupled processes

Luca Fini, Fabio Tosetti, Lorenzo Busoni, Alfio Puglisi, Marco Xompero

INAF, Osservatorio di Arcetri, L.go E.Fermi 5, Firenze, Italia

## ABSTRACT

The LBT-AdOpt Supervisor is a collection of software processes which control the operations on the set of devices which make up the Adaptive Optics subsystem. The Arbitrator is the software component which coordinates the operations of the Supervisor in order to support operations at the telescope in reply to requests issued by the Instrument Control Software.

In this paper we describe the architecture of the Arbitrator, based on an extremely modular, extensible and maintainable approach, designed using object-oriented techniques, that include intensive use of classes, exception handling and design patterns, as well as a clear division of tasks.

**Keywords:** Adaptive Optics, Control Software, LBT

## 1. INTRODUCTION

### 1.1 The LBT-AdOpt system

The Adaptive Optics system for the LBT is a single conjugate AO system which, thanks to the use of a deformable secondary mirror, is integral part of the telescope. Detailed descriptions of the LBT-AdOpt system and follow-ups of its development have been presented many times[1–4] and are updated in this same conference[5,6] .

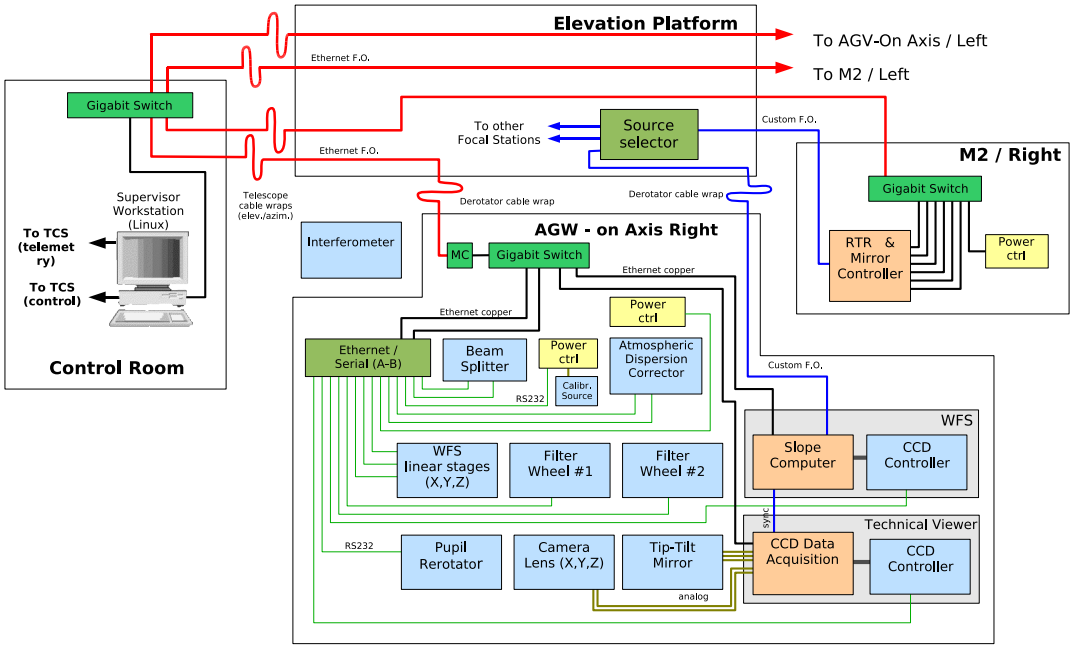


Figure 1. LBT-AdOpt System Components and Communications

Send correspondence to: *lfini@arcetri.astro.it*

The system is essentially subdivided into two largely independent subsystems: a pyramid based Wavefront Sensor[2,3] (WFS), and an Adaptive Secondary mirror[7–11] (AdSec).

From the system point of view the two AO subsystems are an assembly of various hardware components (optical, mechanical, electronics) part of which are off-the-shelf commercial devices and part are custom developed.

A detailed diagram of the hardware components of the system is shown in figure 1, together with the communication paths between the components and the Supervisor Workstation.

The WFS is hosted in a cylindrical cage at one of the elevation platform focal stations together with the auto-guiding system (the combination of the two is named AGW); it's task is to use the image from the CCD of the wavefront sensor to evaluate the wavefront deformation and compute an array of deformation values (slopes) to be sent to the secondary mirror for correction. The slopes computation is performed by a custom developed electronics board hosting two DSP's*. Slope values are sent to the secondary mirror through a fiber optics channel by using a custom developed specialized protocol.

The deformable secondary mirror is equipped with custom developed electronics for the control of the magnetic actuators which modify the mirror shape; moreover it provides enough computational power to implement the wavefront reconstruction, i.e.: to convert slopes into commands for the deformable mirror[12,13] .

Figure 2 shows a diagram of the data communication paths among AO subsystems and the existence of two "software domains" within the system.

The adaptive loop real-time operations are controlled by firmware running on the DSP's embedded into the Wavefront Sensor electronics the Adaptive Secondary electronics. Here are implemented the algorithms for slope computing and for reconstruction and also is managed the transmission of data along the dedicated fiber channel.

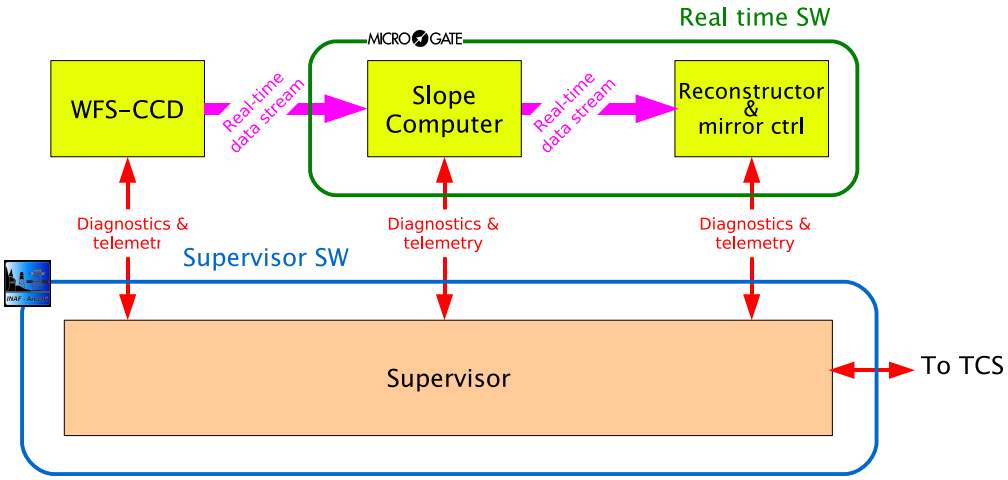


Figure 2. LBT-AdOpt System Blocks

All the housekeeping and diagnostic operations are performed by the **Supervisor**: a collection of programs running on a Linux workstation, which coordinate the operations of the hardware components of the system. The Supervisor tasks includes operations such as: uploading firmware, starting and stopping specific devices, control the sequence of operations, gather diagnostic data and analyze them to discover unsafe conditions, etc.

The Supervisor runs on a Linux based workstation and communicates with the hardware devices through the Ethernet LAN.

*The BCU board developed for the LBT-AdOpt system is the basic building block also for the Secondary Mirror electronics.

### 1.2 Supervisor architecture

The Supervisor's architecture[14] is essentially based on the concept of many loosely coupled processes communicating by means of a message based network protocol. It is essentially a collection of programs, referred to as *components* in the following, each one dedicated to a specific and well defined task. E.g.: we have *controllers* which operate single hardware devices, we have *GUI*s for specific controllers, and so on. Components which are dedicated to the coordination of operation within subsystems have been called *arbitrators* and are the main object of this paper.

The heart of the system is the Message Daemon (`MsgD-RTDB`) which provides message exchange functions, operates as a central repository for variables to allow sharing of data and information between processes, and provides a centralized logging facility.

All the Supervisor components are stand-alone processes which run independently and can cooperate with other components by means of message based services provided by the `MsgD-RTDB`; each component may be started or stopped while the system is running without affecting other processes. The `MsgD-RTDB` also provides mechanisms to synchronize the operations among individual components, when synchronization is needed.

The components can access `MsgD-RTDB` services via an API provided by the "Treading Library" (`thrdlib`) which, as suggested by its name, also includes facilities to simplify the programming of threaded applications which can efficiently manage both synchronous and asynchronous message exchanges. As a further level of abstraction, available to components coded by using the C++ language, we have developed the class `AOApp` which encapsulates all the functionalities provided by the library and is the common building block for most of the Supervisor components.

The strategy adopted in the development of the Supervisor software was based on a few key points:

- The capability to develop and test hardware/software components (E.g.: the CCD and its control program) independently from each other.
- The capability to assemble components into subsystems and operate and test subsystems independently[†].
- The capability to use in the final version of the software (i.e.: the version running at the telescope) exactly the same components used (and thoroughly tested) in the lab.
- The capability to add or remove functionalities without affecting the rest of the system.

The high modularity of the Supervisor is proving to be very valuable during the system test phase when most operations are performed interactively and when there is the need to cope efficiently and promptly to unexpected issues.

On the other side we are aware that a robust and efficient coordination between single subsystems is needed when the system will be integrated into the LBT. When observing with the support of the Adaptive Optics system the observer expects that the system can be set up with a few easy commands and can react automatically to many possible different situations, without the need to understand the complexity of underlying operating sequences. Even during lab tests it is necessary to have procedures to perform sequences of measures in a repeatable way.

To this purpose we have developed the concept of *Arbitrator*: a process whose task is the coordination of operations of various components in order to automate the execution of a complex sequence of operations. In some other contexts a similar process has been called "sequencer", but with the name Arbitrator we would like to stress the fact that the successful completion of some operation is not the result of the execution of a simple sequence of steps, but requires a number of intermediate decisions which depend on many external conditions.

[†]E.g.: both the WFS and the Adaptive Mirror have been separately assembled and tested in the lab[3,4] and during lab tests the related Supervisor components have been debugged and improved against the real hardware. Now the two subsystems are being moved together to the optical bench where they must be integrated for the final test.

## 2. ARBITRATING LBT-ADOPT OPERATIONS

The control of AdOpt operations at the telescope is performed via commands originating from the Instrument Software by means of the related software interface (`IIF`)[15] to the Telescope Control System (`TCS`). AdOpt related commands are managed by a TCS subsystem (the `AOS`) which is the bridge between the TCS and the Supervisor; in this respect AOS is both a TCS Subsystem and a Supervisor component. The AOS main task is to communicate the commands to the Supervisor or, more precisely, to the Arbitrator which is the Supervisor component dealing with coordination among the various parts of the AO System.

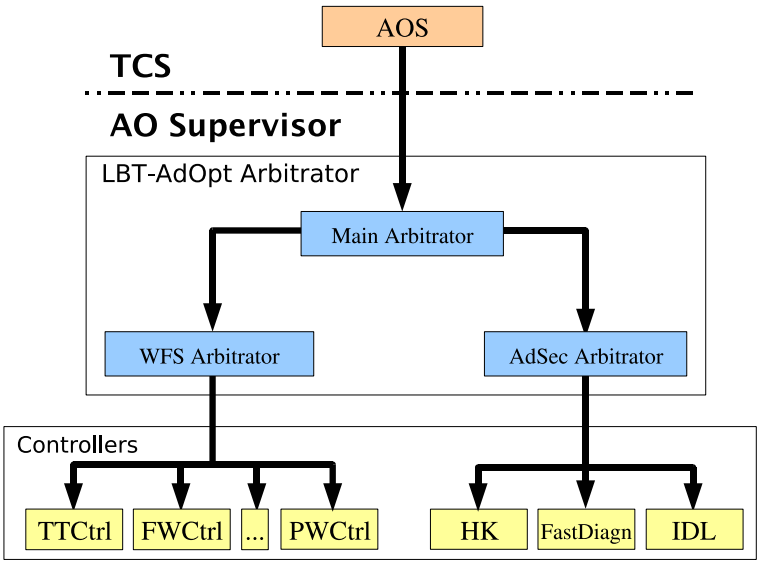


Figure 3. LBT-AdOpt Arbitrator hierarchy

Coordination of tasks within the Supervisor which is needed to orderly execute commands from the TCS is performed by a few processes in a hierarchical structure as shown in figure 3. The three processes as a whole are called the *LBT-AdOpt Arbitrator*. The two lower level processes (the *WFS Arbitrator* and the *AdSec Arbitrator*) have responsibility on their own subset of components and will receive commands from the upper level *Main Arbitrator*. The latter receives commands from the AOS, and converts them into the proper sequence of commands to be issued to its subordinates. The latter will in turn communicate with the components controlling the hardware to perform their own sub-task. Arrows in the figure indicate the main direction of commands, which are executed in sequence and synchronously along each command path. As it will be detailed in a following paragraph, a few messages flow in the opposite direction in order to support notification of asynchronous events. As command paths in figure 3 clearly show, no communication is needed between the two lower arbitrators: they can thus operate (and be tested) separately; when coordinated operation of the two subsystem is needed, this is demanded to the Main Arbitrator.

## 3. THE LBT-ADOPT ARBITRATOR

As shown in the previous section, the LBT-AdOpt Arbitrator is a structure of three processes with very similar functions: their task is to receive external commands, validate them, and execute the corresponding actions. The natural model to be used for such a task is a Finite State Machine (FSM). Each independent arbitrator is thus described and implemented as a FSM, with some added functionalities as we will show next.

As an example let's have a look to the FSM of the Main Arbitrator shown in figure 4, where the state changes correspond to the commands as defined for the LBT-AdOpt system and states to the various steps which must

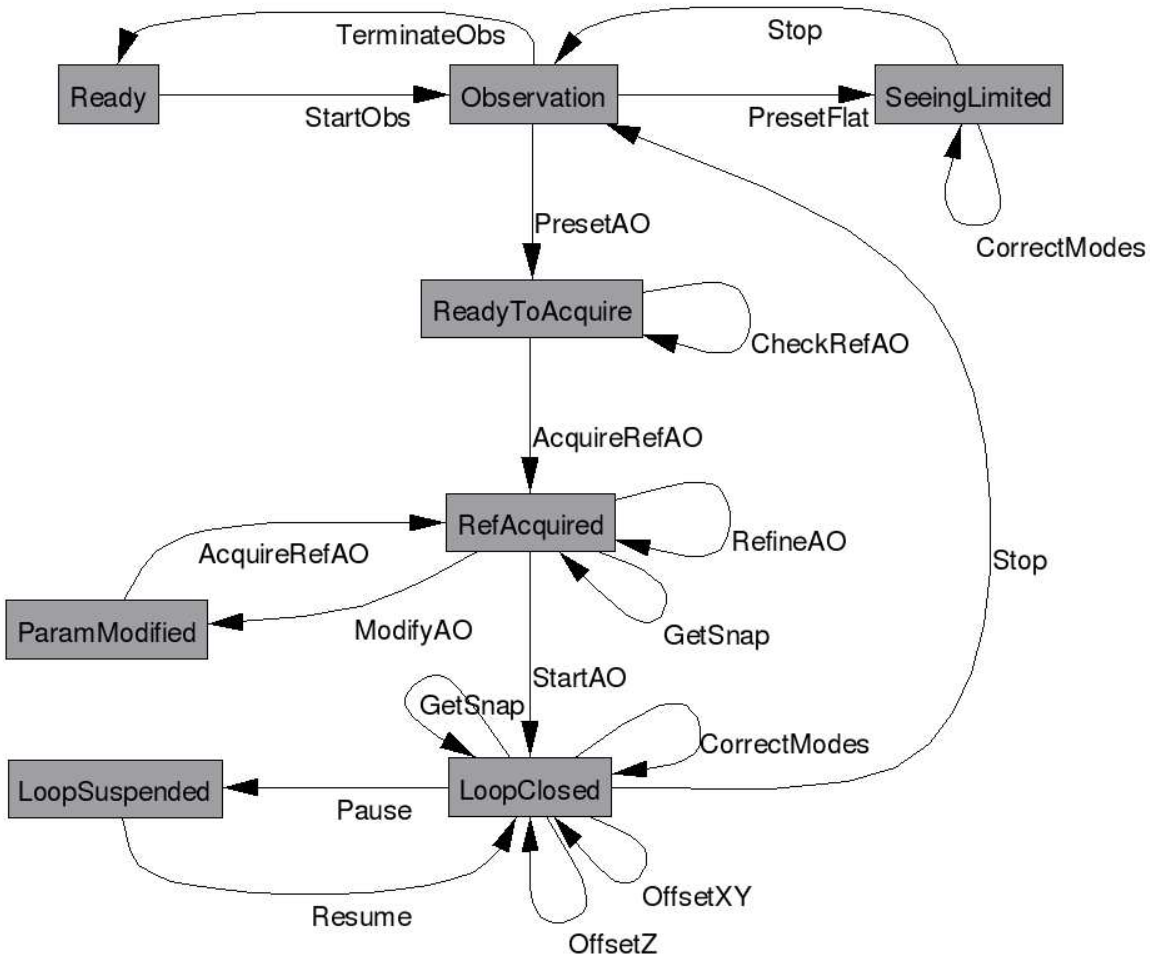


Figure 4. Main arbitrator FSM

be followed during an adaptive optics supported observation at the telescope[‡].

While the FSM is a powerful device to describe a desired sequence of operations, in the real world more functionalities are needed:

- **Error management**. As a result of errors a state transition could not be possible.
- **External events**. Many devices under the control of the FSM may need to asynchronously notify events (environmental changes, hardware failures, etc.).
- **Command cancellation**. Command cancellation may be necessary upon request of the user or as a result of some failure.
- **Command history**. It may be needed if multiple level command undoing is to be implemented.

Of the above points, the management of errors and of external events are key issues upon which the robustness of the entire system depends heavily. Error conditions may arise as the result of commands which cannot be

[‡]The meaning of commands and states as shown in the figure should be clear enough from their names, but it is not essential for the discussion. A detailed description can be found in the specific report[16].

successfully completed for any reason, e.g.: because they will lead to dangerous or otherwise unreachable status of the system or some device[§]. In this cases error notification may be managed synchronously, i.e.: by returning an error condition as reply to the command and the corresponding action should be the return to the status before the command issued.

External events are related to conditions of some device in the system, independent on any command received, but caused by device internal failures, changes in the environment and the like. As an example, the secondary mirror is provided with a complex internal diagnostic system which continuously monitors a large number of electrical, mechanical and environmental parameters in order to detect potentially dangerous conditions. Or possibly the WFS could detect a decrease in total flux from the reference star (due to atmospheric condition variation) which requires adjustments in parameters or even to interrupt the observation. External events may thus have different levels of severity, but must anyway be notified to be properly managed.

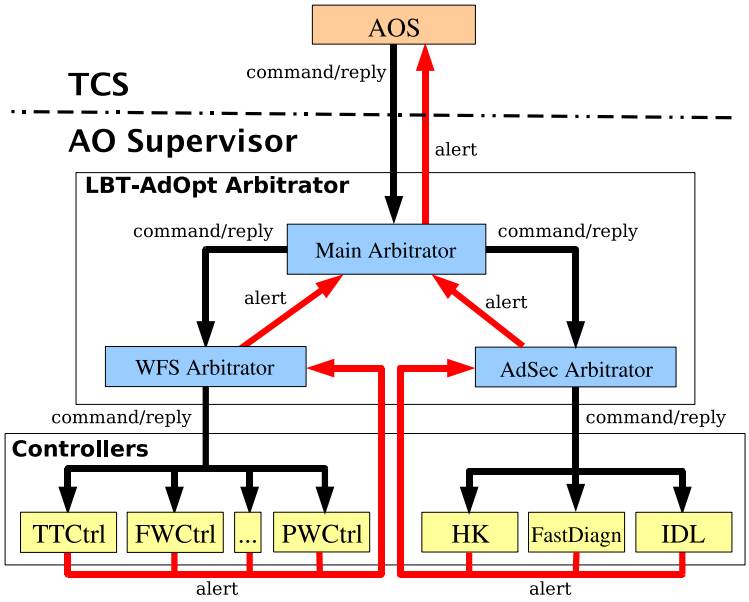


Figure 5. Complete communication path scheme

It is so necessary to add more functionality to the arbitrator as shown in figure 5. Here to the hierarchical scheme of figure 3 a reverse data path is added through which *alerts*, i.e.: messages notifying asynchronous events, can be propagated up to the level where they can be properly managed.

## 4. IMPLEMENTATION ISSUES

In order to efficiently implement the arbitrators which, as we have seen, share a number of common requirements, we have developed the **Arbitrator Framework**, i.e.: a small set of classes from which the actual arbitrators (and their clients) can be derived adding to each only the specific functionalities.

The design is based on two main classes: `AbstractArbitrator` and `ArbitratorInterface` to support, respectively, the derivation of an arbitrator and the implementation of the communication functions in the client.

### 4.1 The Arbitrator

The arbitrator architecture is based on a design scheme named *Command Pattern*[17] and involves a few modules as shown in the UML diagram of figure 6.

[§]Although it is in principle possible to avoid errors depending on illegal requests by an exhaustive validation of command inputs, this is often unpractical for complex systems.

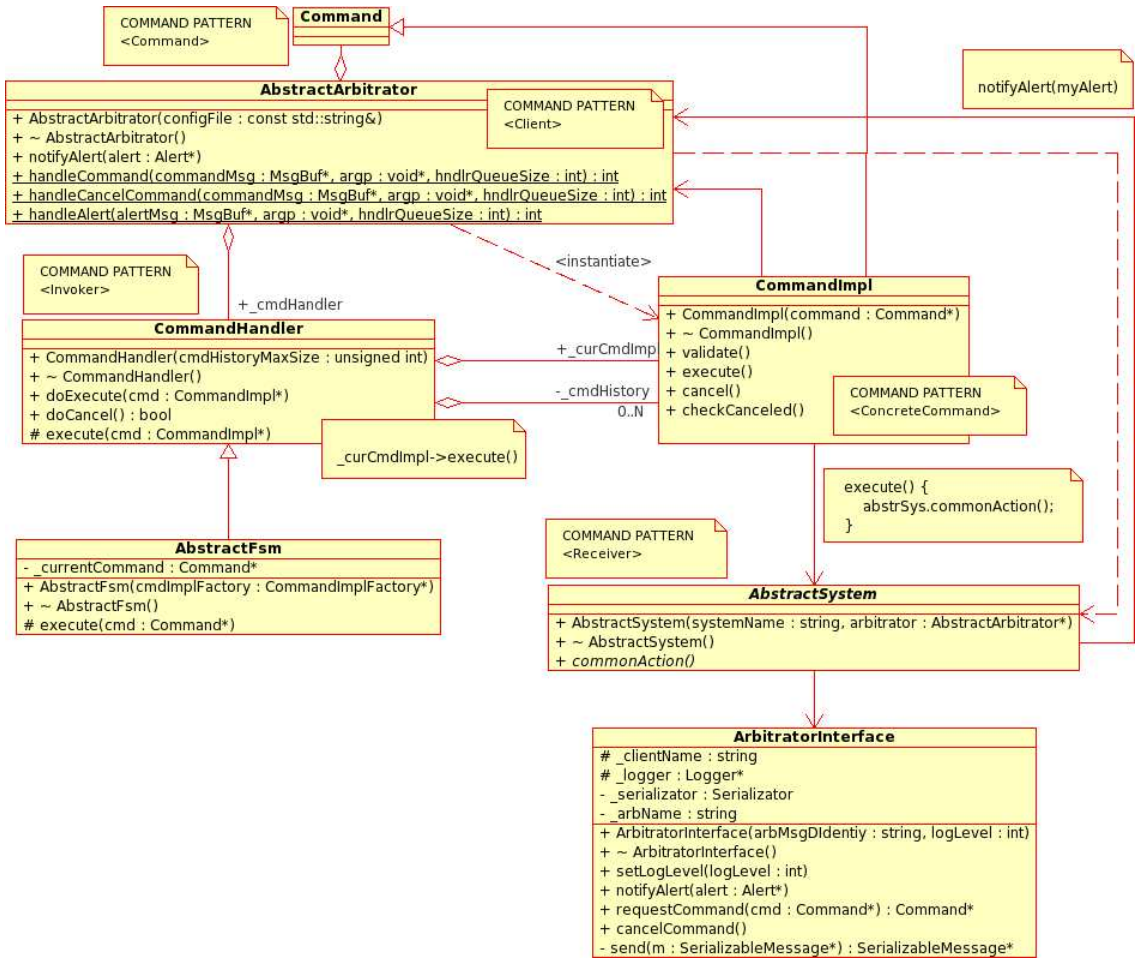


Figure 6. Arbitrator modules and their relations

- **AbstractArbitrator**: is an extension of the class `AOApp` (see sect. 1.2) which provides the support for communication among components through the `MsgD-RTDB`. This allows the implementation of functionalities needed to execute commands in sequence and to reply to asynchronous alerts. It manages initialization of the system and receives messages containing either commands or alerts. Alerts are properly managed by specific code in the derived actual arbitrator and commands are delegated to the `CommandHandler` for execution.
- **CommandHandler**: validates, executes and cancels commands; maintains a command history useful for logging and debugging (and undo, if possible). It doesn't know anything about the implementation of the command it is executing: this is the task of the class `Command` and `CommandImpl`.
- **AbstractFSM**: extends `CommandHandler` to add FSM logic. The specific FSM for the implementation of each arbitrator is designed with FSME[18] , so that the actual code may be generated automatically.
- **AbstractSystem**: defines an abstract class to implement a subsystem. It is essentially a wrapper around a subsystem interface library which, in addition, stores the subsystem's status.
- **Command** and **CommandImpl**: are the two classes used for command implementation. The *decorator* design pattern[17] model has been adopted for the latter which contain the actual code implementing the command.

### 4.2 ArbitratorInterface

The **ArbitratorInterface** class used by clients (e.g.: the AOS) to manage command exchanges with arbitrators has a very simple structure, as shown in figure 7. Command requests are supported by the `Command` class from which actual commands are derived in order to specify details such as the command argument list and, possibly, the return arguments. The `ArbitratorInterface` object uses a `Serializator` to pack command arguments into messages of the underlying protocol defined by the `MsgD-RTDB`. The same `Serializator` class is used by the arbitrator to unpack data from the command message. Return arguments are managed in the same way by messages flowing in the opposite direction. It may be noted that in the implementation of `Serializator` we have used tools provided by the *Boost* library[19] .

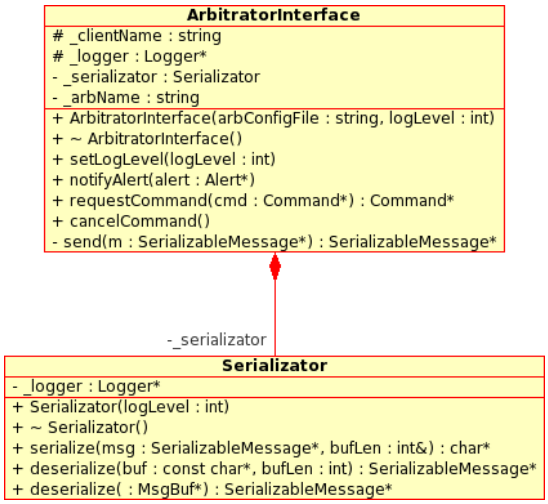


Figure 7. ArbitratorInterface diagram

### 4.3 Building specific arbitrators

Based on the framework described above, the implementation of a specific arbitrator is a 4 step process:

1. Definition of the command set; i.e.: of subclasses of the class `Command`, which define commands with their arguments and the related validation code.
2. Definition of the corresponding set of command implementations (derived from the `CommandImpl` class)
3. Design of the FSM which describes the relations among commands. The implementation of the machine is then done automatically by using the FSMC[18] tool.
4. Implementation of alert management code in the arbitrator class (derived from `AbstractArbitrator`).

The implementation of the corresponding client code is even more straightforward: what is needed is an instance of the `ArbitratorInterface` class to provide the communication path with the related arbitrator and then, whenever a command must be issued, the required `Command` object is instantiated from the set of commands defined at point 2 above and it is passed to the `ArbitratorInterface` instance for delivering and processing.

## 5. CONCLUSIONS

Coordinating a number of loosely coupled processes in a real world application is a fairly complex problem, due to the many aspects which must be taken into account. This could lead to a logic structure so complex to become easily unmanageable.

Moreover when the system as a whole is rapidly evolving[¶] it is essential to be able to extend the design of the arbitrator to follow changes in the requirements.

The development of a framework based on sound software engineering principles derived from powerful OO concepts such as *design-patterns* allowed us to decompose the problem into a hierarchical tree of smallest problems while maintaining a common structure to the modules so that many common problems (communication, error management, event management) could be solved at design level.

The implementation also took advantage from the employ of well tested software tools and libraries, either developed in house or imported from well established projects.

The result is a robust framework on which the LBT-AdOpt Arbitrator has been based.

## REFERENCES


[1] Esposito, S., Tozzi, A., Ferruzzi, D., Carbillet, M., Riccardi, A., Fini, L., Vérinaud, C., Accardo, M., Brusa, G., Gallieni, D., Biasi, R., Baffa, C., Biliotti, V., Foppiani, I., Puglisi, A., Ragazzoni, R., Ranfagni, P., Stefanini, P., Salinari, P., Seifert, W., and Storm, J., "First Light Adaptive Optics System for Large Binocular Telescope," in [*Adaptive Optical System Technologies II. Edited by Wizinowich, Peter L.; Bonaccini, Domenico*], *SPIE Proceedings* **4839**, 164–173 (Feb. 2003).

[2] Esposito, S., Tozzi, A., Puglisi, A., Fini, L., Stefanini, P., Salinari, P., Gallieni, D., and Storm, J., "Development of the first-light AO system for the large binocular telescope," in [*Astronomical Adaptive Optics Systems and Applications. Edited by Tyson, Robert K.; Lloyd-Hart, Michael.*], *SPIE Proceedings* **5169**, 149–158 (Dec. 2003).

[3] Esposito, S., Tozzi, A., Puglisi, A., , Pinna, E., Stefanini, P., Giorgetti, G., Camiciottoli, F., Salinari, P., Bianchi, P., and Storm, J., "Integration and test of the first flight AO system for LBT," in [*Advancements in adaptive optics. Edited by D. Bonaccini, B. Ellerbroek and R. Ragazzoni*], *SPIE Proceedings* **5490**, 228–235 (2004).

[4] Esposito, S., Tozzi, A., Puglisi, A., Pinna, E., Riccardi, A., Busoni, S., Busoni, L., Stefanini, P., Xompero, M., Zanotti, D., and Pieralli, F., "First light AO system for LBT: toward on-sky operation," in [*Advances in Adaptive Optics IIi*], Ellerbroek, B. L. and Bonaccini Calia, D., eds., *SPIE Proceedings* **6272** (July 2006).

[5] Esposito, S., Tozzi, A., Puglisi, A., Riccardi, A., Fini, L., Gori, L., Cavallaro, A., Busoni, L., Pinna, E., Pieralli, F., Quiros, F., Tosetti, F., Xompero, M., Zanotti, D., Ranfagni, P., Brusa Zappellini, G., Brynnel, J., and Salinari, P., "Lbt ao unit #1: integration and testing results," *This Conference* **7015-229** (June 2008).

[6] Riccardi, A., Xompero, M., Zanotti, D., Busoni, L., Del Vecchio, C., Salinari, P., Ranfagni, P., Brusa Zappellini, G., Biasi, R., Andrighettoni, M., Gallieni, D., Anaclerio, V., Martin, H. M., and Miller, S. M., "The adaptive secondary mirror for the large binocular telescope: results of acceptance laboratory test," *This Conference* **7015-37** (June 2008).

[7] Gallieni, D., Anaclerio, E., Lazzarini, P. G., Ripamonti, A., Spairani, R., Del Vecchio, C., Salinari, P., Riccardi, A., Stefanini, P., and Biasi, R., "LBT adaptive secondary units final design and construction," in [*Adaptive Optical System Technologies II. Edited by Wizinowich, Peter L.; Bonaccini, Domenico*], *SPIE Proceedings* **4839**, 765–771 (Feb. 2003).

[8] Riccardi, A., Brusa, G., Salinari, P., Busoni, S., Lardiere, O., Ranfagni, P., Gallieni, D., Biasi, R., Andrighettoni, M., Miller, S., and Mantegazza, P., "Adaptive secondary mirrors for the Large binocular telescope," in [*Astronomical Adaptive Optics Systems and Applications. Edited by Tyson, Robert K.; Lloyd-Hart, Michael*], *SPIE Proceedings* **5169**, 159–168 (Dec. 2003).

[9] Riccardi, A., Brusa, G., Salinari, P., Gallieni, D., Biasi, R., Andrighettoni, M., and Martin, H. M., "Adaptive secondary mirrors for the Large Binocular Telescope," in [*Adaptive Optical System Technologies II. Edited by Wizinowich, Peter L.; Bonaccini, Domenico*], *SPIE Proceedings* **4839**, 721–732 (Feb. 2003).


[¶]The LBT-AdOpt system is actually a prototype and many of its aspects are being discovered during the various phases of lab tests; in some cases details of the design were modified on the fly to cope with unexpected problems.

[10] Riccardi, A., Brusa, G., Xompero, M., Zanotti, D., Del Vecchio, C., Salinari, P., Ranfagni, P., Gallieni, D., Biasi, R., Andrighettoni, M., Miller, S., and Mantegazza, P., "The adaptive secondary mirrors for the Large Binocular Telescope: a progress report," in [*Advancements in Adaptive Optics. Edited by Domenico Bonaccini Calia, Brent L. Ellerbroek, Roberto Ragazzoni*], *SPIE Proceedings* **5490**, 1564–1571 (2004).

[11] Martin, H. M., Brusa Zappellini, G., Cuerden, B., Miller, S. M., Riccardi, A., and Smith, B. K., "Deformable secondary mirrors for the LBT adaptive optics system," in [*Advances in Adaptive Optics II*], Ellerbroek, Brent L.; Bonaccini Calia, D., ed., *SPIE Proceedings* **6272**, 0U (July 2006).

[12] Biasi, R., Andrighettoni, M., Veronese, D., Biliotti, V., Fini, L., Riccardi, A., Mantegazza, P., and Gallieni, D., "LBT adaptive secondary electronics," in [*Adaptive Optical System Technologies II. Edited by Wizinowich, Peter L.; Bonaccini, Domenico*], *SPIE Proceedings* **4839**, 772–782 (Feb. 2003).

[13] Biasi, R., Andrighettoni, M., Riccardi, A., Biliotti, V., Fini, L., Mantegazza, P., and Gallieni, D., "Dedicated flexible electronics for adaptive secondary control," in [*Advancements in Adaptive Optics. Edited by Domenico Bonaccini Calia, Brent L. Ellerbroek, Roberto Ragazzoni*], *SPIE Proceedings* **5490**, 1502–1513 (2004).

[14] Fini, L., Puglisi, A., and Riccardi, A., "LBT-AdOpt control software," in [*Advanced software, control, and communication systems for astronomy. Edited by L. Hilton and G. Raffi*], *SPIE Proceedings* **5496**, 528–537 (2004).

[15] Borelli, J., "Instrument interface for the LBT Telescope Control System. C++ interface control document," Tech. Rep. 481s011b, LBTO (Mar 2007).

[16] Fini, L., Busoni, L., and Puglisi, A., "AOS functional description," Tech. Rep. 486f006b, INAF - Arcetri (Jun 2006).

[17] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., [*Design Patterns: Elements of Reusable Object-Oriented Software*], Addison Wesley (1995).

[18] http://fsme.sourceforge.net.

[19] http://www.boost.org/.