Overview

The AO high-level coordination is implemented by three components, called "arbitrators":

- **Overall AO arbitrator** (simply called "AO Arbitrator"): implements the overall AO finite state machine and talks directly with the AOS. To operate the system, gives commands to the two lower-level arbitrators.
- **AdSec arbitrator**: implements the Adaptive Secondary finite state machine
- **WFS arbitrator**: implements the Wavefront Sensor finite state machine

Arbitrators may be written in any language.

An arbitrator exports a series of commands, which can be called by a higher-level arbitrator, by the AOS or by an external process (e.g. calibration procedures). To allow interoperability from different languages, each arbitrator makes available a library which other processes must use to call arbitrator commands. Programs written in different languages must make their own wrapper around this library.

Hereafter a first draft of the AO Arbitrator structure document:

- arbitrator.pdf: AO Arbitrator: architecture and functionalities
- AoArbitratorFsm: AO Arbitrator FSM and use cases

Requirements

See the attachment *Arbitrator-Requirements.pdf* for a detailed requirement analysis.

Adopted solution

A generic **arbitrator framework** is designed as basis for all arbitrator implementations.
A generic **arbitrator interface** is designed to communicate (send commands and alerts) with the generic arbitrator.
Both "modules" uses object-oriented design and implementation (C++) to allow an easy and safe extension:

- The AOArbitrator extends the framework, providing its own set of commands and FSM.
- The AOArbitrator interface extends the generic interface to provide interaction with the AOArbitrator.

Design 0.9

The design V0.9 UML diagram (for Umbrello tool) is available in *CVS/Documentation/Architecture/AO-Arbitrator-Framework_0_9.xmi*.

Design 1.0 (final)

The UML diagram (for Umbrello tool) is available in *CVS/Documentation/Architecture/AO-Arbitrator-*

Framework.xmi.

Please see code in CVS (*lib/arbitrator* and *Arbitrator/aoarbitrator*).

Arbitrator: how to add a new command

Each arbitrator has its own subdirectory in $ADOPT_SOURCE/lib/arblib and $ADOPT_SOURCE/Arbitrator. The structure is identical apart from name prefixes. Here we will use as an example the main AO Arbitrator (aoArb):

AOArbitrator folder's structure:

$ADOPT_SOURCE/Arbitrator/aoarbitrator	**Contains the extension (implementation) for the AO Arbitrator of the framework arbitrator classes.**
$ADOPT_SOURCE/lib/arblib/aoArb/	Contains the definition for the AO Arbitrator of the Arbitrator framework classes.

The **Arbitrator Framework** base classes to be known are (omissing the root folder):

lib/arblib/aoArb/Command.h	Defines a base command class.
arbitrator/aoarbitrator/CommandImpl.h	Defines a base command implementation class.

To add a new command to **AO Arbitrator**, the interesting files and folders are (omissing the root folder):

lib/arblib/aoarbitrator/MainArbitratorOpCodes.h	Defines the "operation code" for every available command.
lib/arblib/aoarbitrator/Commands.h	Contains the available commands classes.
lib/arblib/aoarbitrator/Commands.cpp	Implements some command methods, at least ::validateImpl() and optionally ::log()
lib/arblib/aoarbitrator//AOCommandsExport.h	Export the available commands for boost serialization.
arbitrator/aoarbitrator/AOCommandsImpl.h	Contains the implementations classes for the available commands.
arbitrator/aoarbitrator/AOCommandsImpl.cpp	Contains the implementations classes for the available commands.
frameworkImpl/AOCommandsImplFactory.h	Bind a Command class (actually its operation code) with the corresponding implementation (CommandImpl class)
arbitrator/aoarbitrator/AOFsm.h	Contains an Enum with the opcode values for the FSM
arbitrator/aoarbitrator/AOFsm.cpp	Implements command execution within the FSM

No other files or directories must be changed.

To defines and implement a new command follows this steps:

- Step 1: defines an operation code (type *OpCode*) in the file *lib/arblib/aoarbitrator/*

MainArbitratorOpCodes.h.

- Step 2: create your *MyCommand* class (deriving from *Command.h*) in the *lib/arblib/aoarbitrator/Commands.h* file:
    - Defines a public constructor with at least the parameter *timeout_ms* (optional, but suggested) and a call to the *Command* class constructor (with the *OpCode* defined in Step 1)
    - Add all needed command parameters to the default constructor.
    - Defines a protected default constructor with no parameters (needed for boost serialization).
    - Defines private fields for needed command parameters.
    - Defines a private *serialize(Archive ar, unsigned int version) method* to serialize the base class and the parameters.
    - Implement all pure virtual method of Command class.
- Step 3: update the file *lib/arblib/aoarbitrator/AOCommandsExport.h* to enable the boost serialization for your command.

Now your command is fully defined: can be sent using the AOArbitratorInterface and received by AOArbitrator. Now you need to define an implementation for the command; otherwise the default EMPTY implementation (*EmptyCommandImpl*) will be used.

- Step 4: create your *MyCommandImpl* class (deriving from *CommandImpl*) in the *arbitrator/aoarbitrator/AOCommandsImpl.h file.*
    - Implement all pure virtual method of the CommandImpl class.
- Step 5: update the file *arbitrator/aoarbitrator/AOCommandsImplFactory.h* to bind *MyCommandImpl* with the corresponding *MyCommand* class.
- Step 6: add the command opcode to the Enum in *arbitrator/aoarbitrator/AOFsm.h*
- Step 7: add the command execution to the FSM code in *arbitrator/aoarbitrator/AOFsm.cpp*

Todo

Framework and interface design next updates

- **Framework and AOArbitrator**: check return values and errors (only exceptions, please!) for methods
- **Framework**: review transition method to failure state

Next steps

- **AOArbitrator**: complete the set of *CommandsImpl*.
-
- **AdSecArbitrator: begin!**

-- FabioTosetti - 09 Jan 2008 -- MarcoXompero - 16 Oct 2007 -- AlfioPuglisi - 18 Feb 2009

Attachments (2)

Attach files   Show options

Arbitrator-Requirements.pdf (723.69K) Version 2 uploaded by Fabio Tosetti on 19 Nov 2007 - 11:41 ... more Arbitrator requirements' analysis	arbitrator.pdf (3135.12K) Version 1 uploaded by Marco Xompero on 16 Oct 2007 - 09:56 AO Arbitrator: architecture and

Select all

Edit | Attach | Print version | History: r36 < r35 < r34 < r33 | Backlinks | View wiki text | Edit wiki text | More topic actions

Topic revision: r36 - 16 Jun 2011, LucaFini