PROCEEDINGS OF SPIE

SPIEDigitalLibrary.org/conference-proceedings-of-spie

Designing and managing software interfaces for the ELT

Gianluca Chiozzi, Luigi Andolfato, Mario Kiekebusch, Nick Kornweibel, Marcus Schilling, et al.

Gianluca Chiozzi, Luigi Andolfato, Mario Kiekebusch, Nick Kornweibel, Marcus Schilling, Michele Zamparelli, "Designing and managing software interfaces for the ELT," Proc. SPIE 10707, Software and Cyberinfrastructure for Astronomy V, 1070725 (6 July 2018); doi: 10.1117/12.2312175

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2018, Austin, Texas, United States

Designing and managing software interfaces for the ELT

Gianluca Chiozzi[1], Luigi Andolfato, Mario Kiekebusch, Nick Kornweibel,
Marcus Schilling, Michele Zamparelli
European Southern Observatory, Garching bei München, Germany

ABSTRACT

The Extremely Large Telescope[1] (ELT) is a 39 meters optical telescope under construction at an altitude of about 3000m in the Chilean Atacama desert. The optical design is based on a novel five-mirror scheme and incorporates adaptive optics mirrors. The primary mirror consists of 798 segments, each 1.4 meters wide.

The architecture of the control system[2] is split in layers and in a high number of subsystems/components developed by different parties. This implies a high number of interfaces that must be designed and maintained under configuration control, to ensure a flawless integration of the different parts.

Having interfaces (and data) definitions in a flexible central place allows us to extract several different artifacts (for example Interface Control Documents (ICDs), Interface Definition Language (IDL) files, tabular spreadsheets, help files, other generated code formats like code stubs or state machine implementations).

In this paper, we explain how selecting a graphical modeling language like SysML and using graphical and tabular editing features made available by state of the art modeling tools presents a number of advantages with respect to other solutions like spreadsheets, a relational database, or a custom textual DSL. Still, using standard export/import formats (EMF XMI), we do not bind ourselves to a specific vendor.

We describe the workflow that we have identified for the definition of interfaces, what artifacts we want to automatically produce and why. We also describe what technologies we are using to reach these objectives.

A key aspect of this work is the selection of interface design patterns that are formal enough to allow automatic generation of the artifacts and, at the same time, pragmatic and simple to gain acceptance from all users and not incur in overhead.

Keywords: ELT, modeling, interface definition

1. INTRODUCTION

ESO has been relying since several years on model driven engineering methodologies and tools for the development of the control systems of telescopes and instruments. Rational, concrete usage and lessons learned have been presented in several papers, like [3],[4] and [6]. The Active Phasing Experiment project (APE) was also used by the OMG Telescope Modeling Challenge Team as a base for writing a cookbook for Model Based System Engineering (MBSE) with SysML[5].

For the ELT we are developing models using these methodologies in the areas where we can get concrete advantages and when we can use them to produce and maintain documentation and other artifacts. This is particularly true for control software design and development.

Selecting a SysML model and a tool like MagicDraw[2] as the master format presents several advantages with respect to other formats, like spreadsheets, relational database or custom textual languages, including:

- Easy visually-assisted handling of interface hierarchies and modularity.

[1] gchiozzi@eso.org; phone +49-89-32006543
[2] https://www.nomagic.com/products/magicdraw

Software and Cyberinfrastructure for Astronomy V, edited by Juan C. Guzman, Jorge Ibsen, Proc. of SPIE
Vol. 10707, 1070725 · © 2018 SPIE · CCC code: 0277-786X/18/$18 · doi: 10.1117/12.2312175

- Reuse across various types of model and checking of consistency. For example, defining queries to identify who is using any of the specified interfaces and if the usage is consistent.
- Native implementation by the tool of several typical system engineering concepts.
- Possibility of keeping together interfaces and their behavioral descriptions.
- Flexibility of reusing the information for further design/verification activities.
- Possibility of implementing validation rules using the tool's validation engine in order to promote model consistency.
- Collaborative work with partial locking of edited interfaces.

Using standard export/import formats (EMF XMI), we do not bind ourselves to a specific tool or to a specific vendor. Model transformations to generate artefacts from EMF XMI exported files do not depend on the tool used to graphically develop the model and can rely on open tools, frameworks and transformation languages, like the EMF and the Xtend tool set[7].

The choice does not preclude to change the master database to a different format at a later time. Operational needs might later require, for instance, to maintain interface information in an ELT Central Configuration Database, rather than in a descriptive SysML model. Reversing the direction of generation and import/export achieves this goal.

One area where we consider modeling particularly useful is in the definition of interfaces, since we need to keep the definitions aligned in the Interface Control Documents and all through requirements analysis, design and implementation. In the ELT we have to keep under configuration control the interfaces between the Local Control Systems that are contracted to industry, and the Central Control System and the interfaces with the instruments. Interfaces evolution is a natural part of the design and implementation life cycles, with new features and details being added or edited. Model assisted interface definition greatly enhances the task of maintaining such definitions in a consistent state, across iterative modifications. Also, the daily usage of API interfaces can be greatly enhanced if both their documentation and derived code stubs can be obtained from the same, consistent source.

We have therefore defined a workflow and elements (terms, and their attributes) in a dedicated ELT SysML profile to support the development of interfaces, described in the following sections.

2. STANDARD ICD STRUCTURE

In the case of ICDs for Local Control Systems, the basic assumption is that we shall follow the structure already used in the ICD documents prepared for the contractors of the ELT subsystems, and exemplified in Figure 1 and Figure 5:

- Every ICD can be hierarchically structured in functional groups
  For example, the interface for Dome control can be split in
  - Dome Azimuth Rotation
  - Wind Screen
  - Slit Doors
  - ...
- Each ICD or functional group have a Control and a Safety part.
- Interface specifications for Control or Safety parts can include:
  - Commands
  - Monitoring (measurements and status)
  - Configuration
- For each interface it shall be possible to specify
  - Documentation
  - Types of parameters and monitor/configuration data items
  - Standard quality of service (Q0S) parameters such as
    - Rate
    - Synchronization
    - Latency

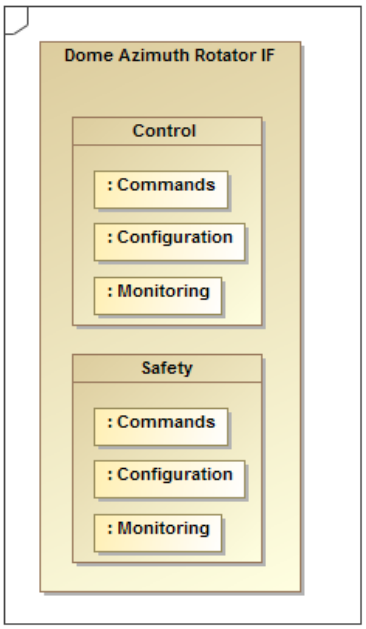

Figure 1. Basic standard interface structure

- Eventually other semantic information

This structure is not binding (in particular for usages different from LCS ICDs) and can be easily adapted to the specific needs of different projects. For example, instead of identifying Control and Safety interfaces at sub-component level, a project can decide to do the other way around and have a Control and Safety interface directly below the subsystem and have a functional structure inside the Control or Safety level. Nothing changes at the level of what is described here, but for the order in which the packages are nested. Consistency inside the project and the usage context naturally remains an important goal for the ELT project.

3. ICS SYSML PROFILE AND TEMPLATES

We follow the MBSE and OOSEM conventions[5], as they have been adapted for the ELT, in terms of package structure and usage of UML and SysML elements and modeling specifications. All conventions were formalized in the mentioned SysML profile.

Figure 2 is a screenshot of the package structure for the Subsystem metamodel in the ELT profile.

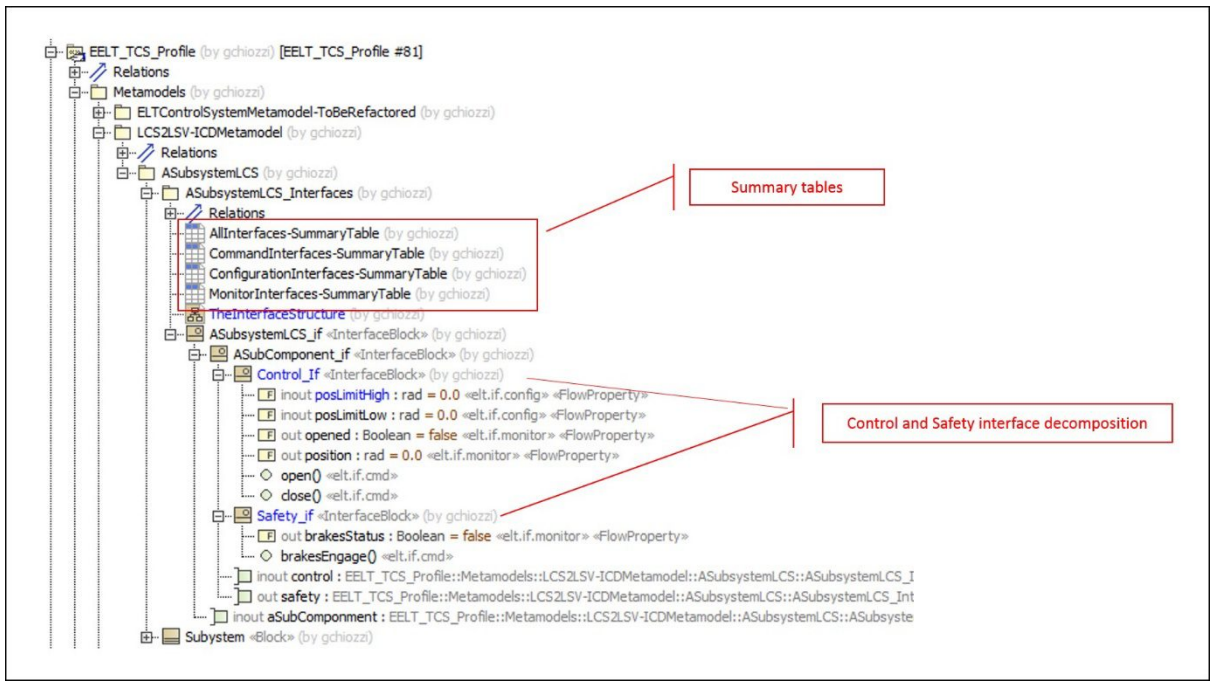

Figure 2. Subsystem model structure

The metamodel contains templates intended to be copied & pasted as a complete sub-tree in the place where the developer wants to use them and doing search & replace operations to customize them. They are therefore as much as possible self-contained and relative to the root package.

Together with the model-based definition of the interfaces, the template contains a set of standardized tables, whose content is generated by querying the model, and that are automatically updated when the model changes. These tables are meant to be directly used in the ICD printable documents and are also very handy for editing the details of the interface elements (otherwise often a drawback of graphical modeling), as shown in Figure 3.

4. DEFINING INTERFACES IN MODELS

The hierarchical containment tree (on the left in Figure 3) is the best place to create/remove interfaces.

Summary tables (on the right in Figure 3) are the best place to edit the details of single interfaces.

In the following description (using UML/SysML notation), as well as in the template, `ASubsystem` is a generic name to be replaced by the name of the actual subsystem being modeled, for example `M4`.

`ASubsystemLCS_Interfaces` is the package grouping together the interfaces of the whole LCS. The package, in its basic form, contains an «`interfaceBlock`» element modeling the whole interface according to the structure defined in section 2 or any other logically defined structure.

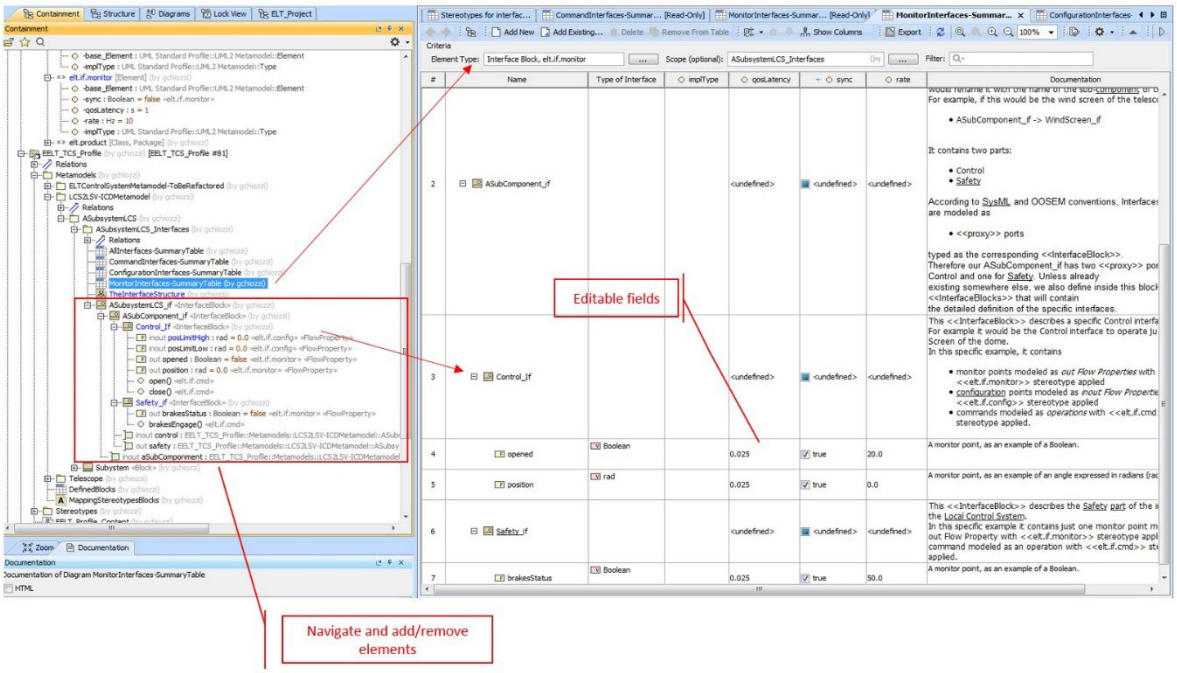

Figure 3. Interface navigation and editing with tables.

According to SysML and OOSEM conventions, interfaces are modeled as «`proxy`» ports typed as the corresponding «`interfaceBlock`»**.** Therefore, in our LCS standard, each sub-component interface `ASubComponent_if` has two «`proxy`» ports, one for Control and one for Safety. Unless reusing interfaces already defined somewhere else, we also define inside this block two «`interfaceBlock`» elements that will contain the detailed definition of the specific interfaces.

Interfaces are modeled by:

- creating an «`interfaceBlock`» model element, typically
- inside an `_Interface` package
- or inside another «`interfaceBlock`»

Interfaces are declared by:

- adding a «`proxy`» port element to the block exposing that interface and assigning as type to the port the corresponding «`interfaceBlock`».

An «`interfaceBlock`» can be structured in sub-interfaces to model finer granularity.

Typically, the «`interfaceBlock`» defining the internal sub-interface is created inside the super-interface (unless reusing something already existing).

An «`interfaceBlock`» can contain the description for interfaces as:

- Commands.
  A Command is defined by:
  - Adding to the «interfaceBlock» an operation with the name of the command.
  - Applying the «`elt.if.command`» stereotype.
  - Defining parameters and return values.
  - Setting the tags for the stereotype, from the specification panel of from the interface table.
- Monitor points (measurements and status).
  A Monitor point is defined by:
  - Adding to the «`interfaceBlock`» a Flow Property with the name of the monitor point.
  - Setting its Direction to out (we can only access the value to read it, we cannot set it).
  - Applying the «`elt.if.monitor`» stereotype.
  - Setting the logical type of the Flow Property to a valid Type (typically the units, for example rad or m).
  - Setting the tags for the stereotype, from the specification panel of from the interface table, for example setting, if necessary and typically just at detailed design level, the implType to the type used for the implementation (like int, int64, float, double).
- Configuration.
  A Configuration point, i.e. an element that is part of the configuration description for a subsystem and that, for example, will be set in its own configuration database, is defined by:
  - Adding to the «`interfaceBlock`» a Flow Property with the name of the configuration point.
  - Setting its Direction to inout (we can both access and change a configuration value).
  - Setting the logical type of the Property Value to a valid Type (typically the units, for example rad or m).
  - Setting the tags for the stereotype, as for Monitor points.

Notice that, once the corresponding operations and value properties (see the description of the interface elements) have been placed inside the «`interfaceBlock`»(s), it is possible and very convenient to edit the values directly in the tables. Tables provide a compact view of the details of operations and properties of the interfaces.

Figure 4 shows an excerpt from the stereotypes typically used to model the interfaces, and their documentation, as drawn

#	Name	Documentation
1	«» elt.if.cmd	This stereotype shall be applied to all elementary ELT command interface elements (to be represented as "operations"). It adds a number of tags used to qualify the characteristics of the command interfaces. It is a specialization of DirectedFeature and therefore featureDirection shall be used to specify if the command is provided/required interfaceElements shall be owned by SysML InterfaceBlocks.
2	«» elt.if.config	This stereotype shall be applied to all elementary ELT configuration interface elements (to be represented as "inout Flow Properties", i.e. Flow Properties with inout drection, since we can both read and change the value). It adds a number of tags used to qualify the characteristics of the configuration interfaces. The multiplicity of the Flow Port can be used to specify how many properties are specified (for example for arrays); a multiplicity of [0..1] make the Property optional. The tag implType (if set) is used to specify ad design level the specific implementation type. For example an **angle** can have has **type rad** (radians, the physical units specified at an abstract level) and has **implType** a **float** or a **double** or some other implementation specific representation. Notice that also the implType shall be a valid Type object. interfaceElements shall be owned by SysML InterfaceBlocks.
3	«» elt.hasAsPrecondition	This stereotype is applied to relations between Use Cases, to mark the execution of a Use Case as a pre-condition for the execution of another one.

Figure 4. Example of terms (Stereotypes) defined in the ELT Profile

from the ELT SysML Profile.

The interfaces can also be represented effectively using a SysML Block Definition Diagram (BDD), as shown in Figure 5, that gives an intuitive representation of the hierarchical structure of the interfaces.

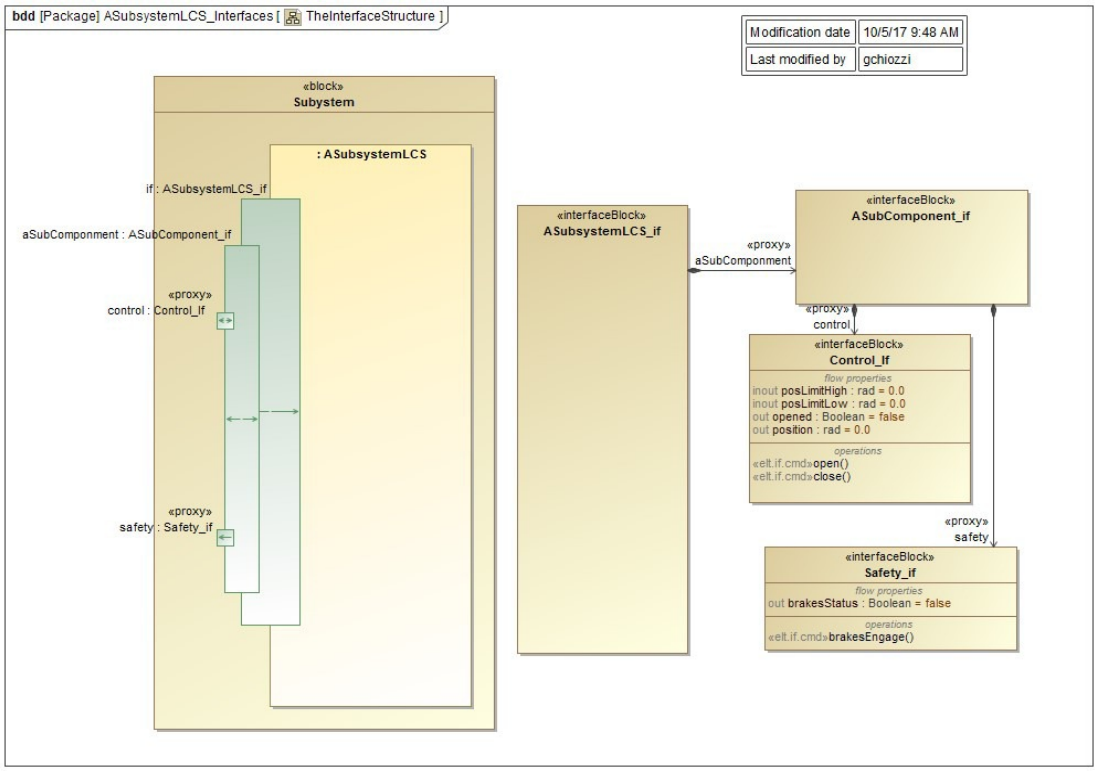

Figure 5. Interface structure in a block definition diagram

5. USING INTERFACES IN MODELS

There are different options to model the usage of an interface by a client.

The simplest way, that we are suggesting here, is to:

1. add a «`proxy`» port element to the block representing the client
2. assign as type to the port the «`interfaceBlock`» representing the used interface
3. "*conjugate*" the port (think about the male/female interfaces) to model the fact that you are a user of the interface and therefore output becomes input and the other way around. Conjugation is displayed with the ~ symbol (see e.g. Figure 6).
4. draw a connector between the port of the server and the one of the client (Figure 7).

Most modeling tools will perform some validation when you draw the connector and will markup the connection in red color if the ports are not compatible.

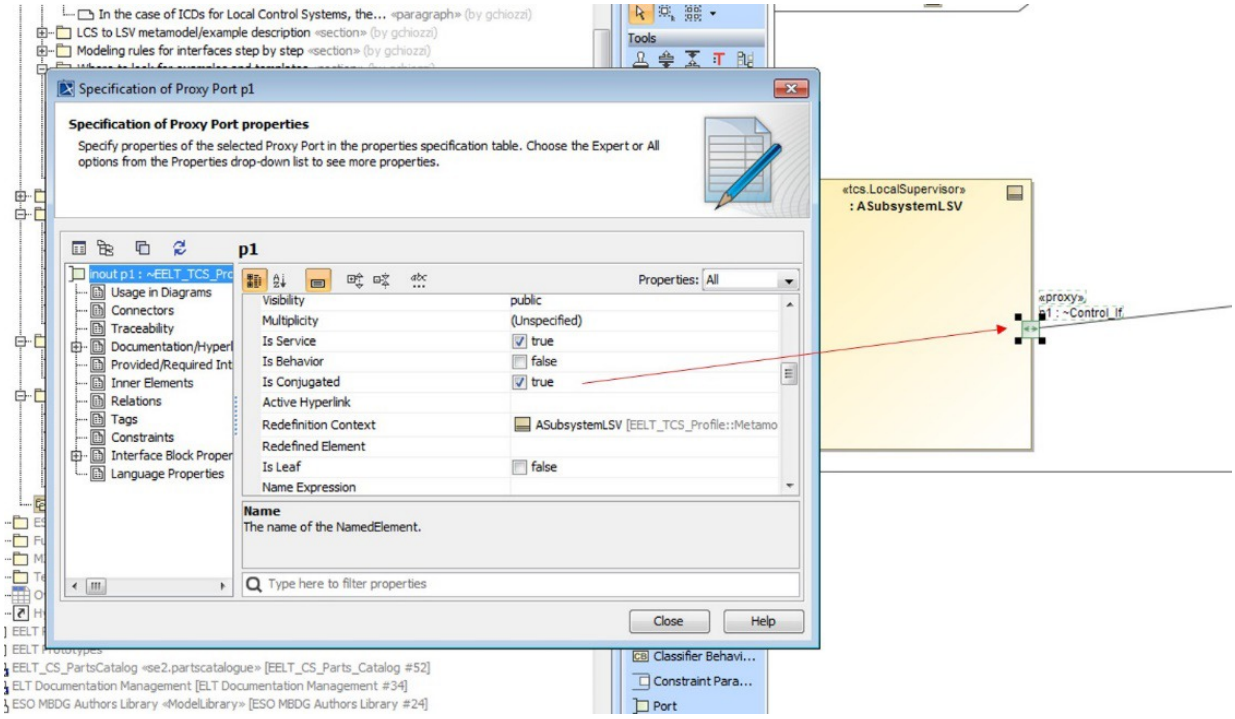

Figure 6. Adding to a client a port with conjugate interface

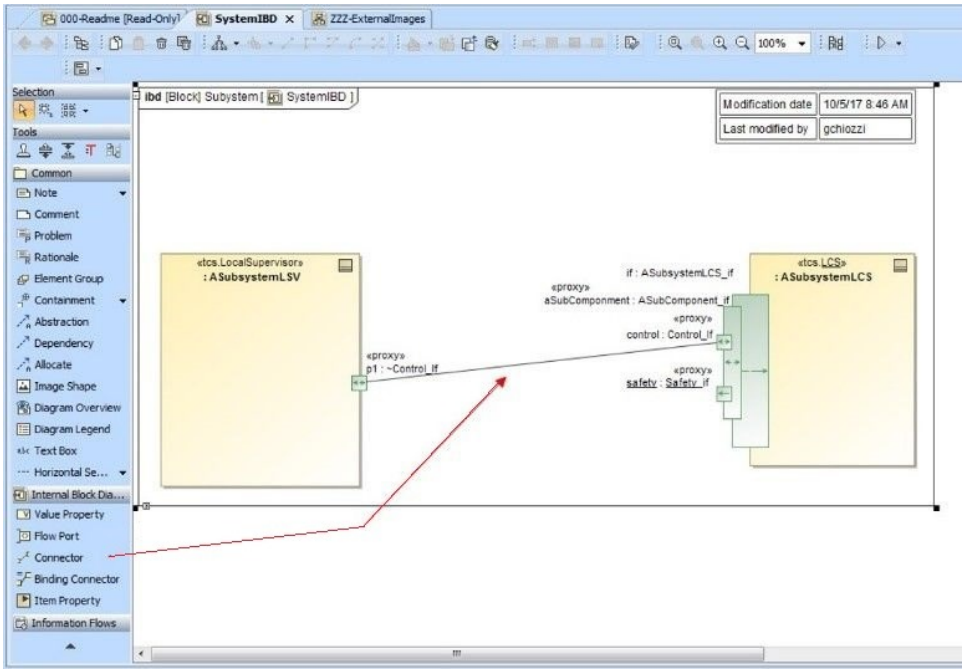

Figure 7. Connect client and server interfaces

6. ICD EXAMPLES

As a concrete example, Figure 8 shows the BDD for the interface of the of the M4 LCS. The interface is split in 3 functional groups (Adaptive Mirror, Cooling System and Power Supply), each with a control and a safety interface.

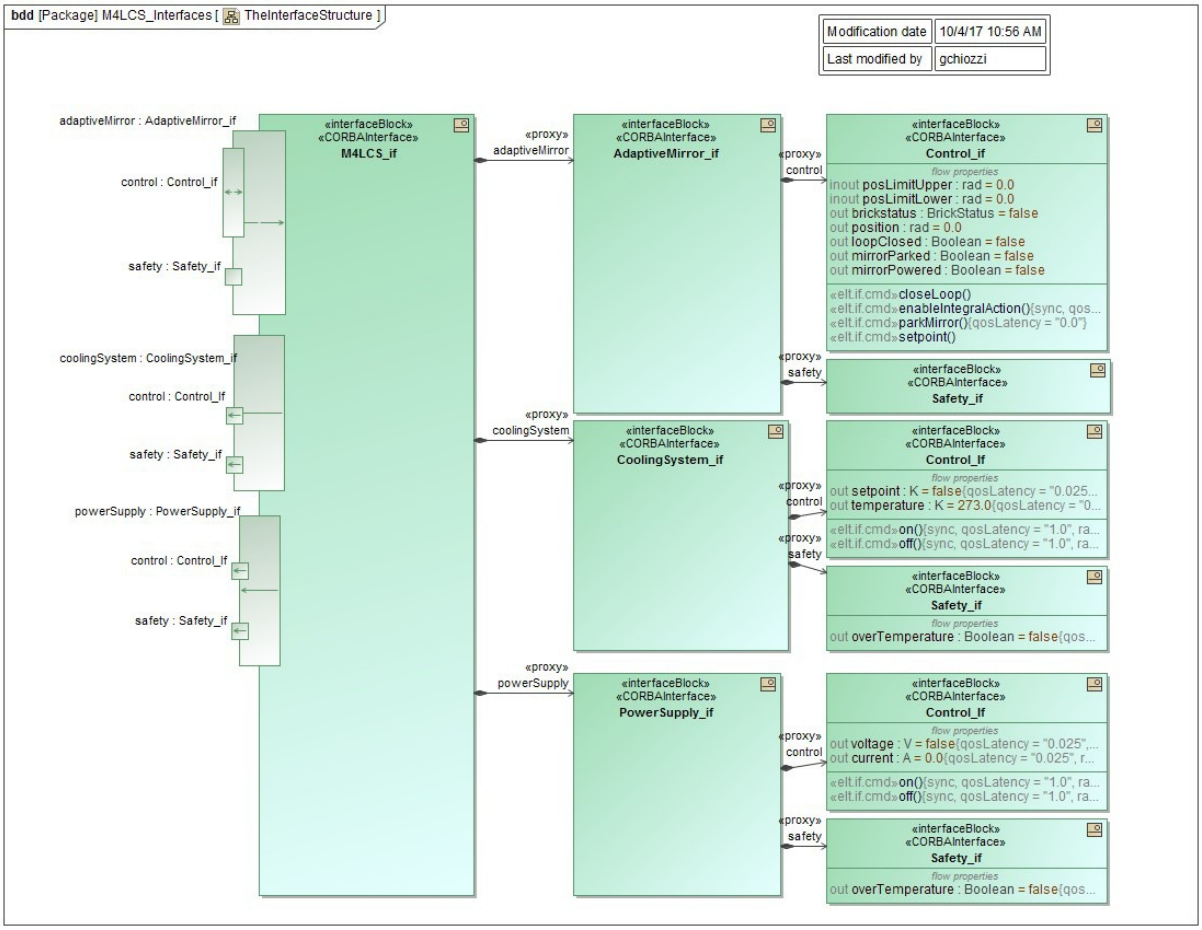

Figure 8. M4 LCS Interface

Figure 9 shows instead the standard interface defined in the Instrumentation Framework for shutter devices.

The ELT Instrumentation Framework defines standard interfaces for the devices commonly used in instruments, like Lamps, Motors, Shutters, Piezo.

The interfaces defined for these devices are at detailed design level and take into account specific implementation details, like the fact that communication from the PLCs to the applications using them goes through OPCUA.

The controller for each device has two interfaces:

- an Opcua interface, toward the users, implemented through OPCUA, and
- a Mapping interface directly communicating with the connected hardware through digital, analog or other types of I/O ports on the PLC.

For the Opcua interfaces it has been decided to separate explicitly control, configuration and monitoring (here called status) in three separate «`interfaceBlock`» definitions.

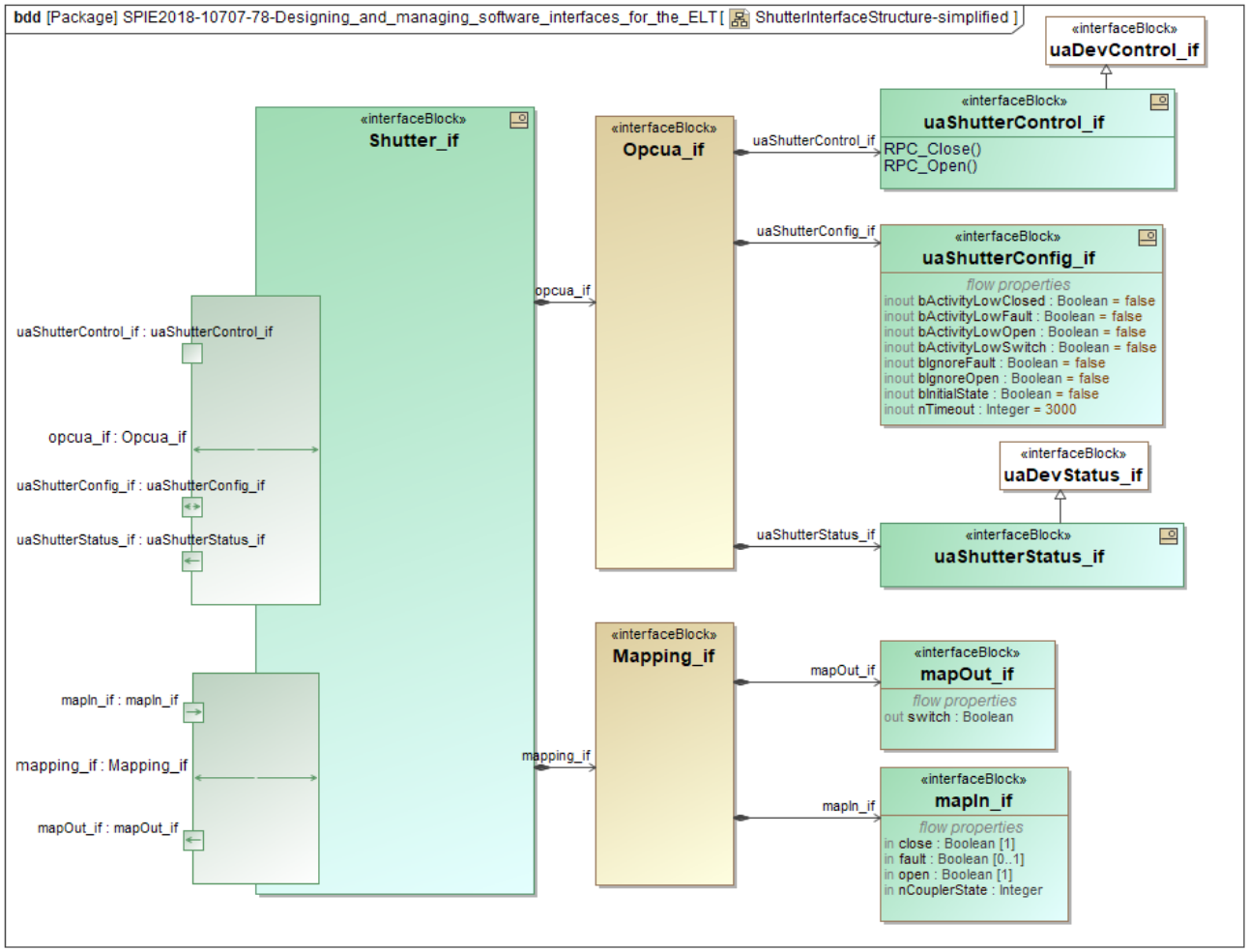

Figure 9. Instrumentation Framework Shutter standard interface

7. MODEL TRANSFORMATIONS

At the moment, the primary usage of the interfaces defined in the model is for analysis and design inside the models themselves and for producing documentation. A documentation engine executed from inside MagicDraw based on a document generation UML profile is responsible for generating documents in DocBOOK and PDF format[8], but also the tool's native reporting engine for PDF can be used to generate customary ICD documentation,

We aim at developing model transformations to generate from XMI exports of the models:

- Stubs and skeletons for the interfaces in the form of IDL files for the CII infrastructure, in the supported languages.
- Google Protocol Buffer definitions.
- Schema for the configuration database and configuration files.

In parallel to interfaces, we will be using the structural model of the system to generate skeletons of the complete applications and we are generating already control state machines as described in [3] and [4].

8. CONCLUSION

Until now we have defined the standard structure, modeling profile and workflow for the definition of interfaces.

We are also generating documentation and we have started prototyping the generation of other artefacts, following the path adopted with our previous projects. This work will proceed in parallel to the consolidation of the design of CCS supervisory applications, of the Instrumentation Frameworks and of CII, that will define the targets of the model transformations.

The process of managing interfaces as being used now still involves manual steps, but most of them can be simplified or made completely automatized by customizing the elements of the profile or by writing MagicDraw macros. It will have to be evaluated up to which level automation pays off in pragmatic terms.

We want to be sure that the effort spent in writing each model transformations or modeling support tool will be abundantly compensated by the saved effort on the side of the users and in maintaining and keeping aligned the information/documentation and the actual implementation along the lifetime of the project.

ACKNOWLEDGMENTS

We would like to thank here many colleagues who have provided requirements and feedback for the definition of the ELT control system architecture and in particular R.Karban who played a major role in laying down the foundations of the concept already several years ago before moving to JPL.

REFERENCES

[1] Tamai, R., et al., "The E-ELT program status", These proc. SPIE 10700, paper 10700-36 (2018)
[2] Chiozzi, G. et al., "The ELT Control System", These proc. SPIE 10707, paper 10707-31 (2018)
[3] Andolfato, L. et al., "Behavioural Models for Device Control", Proc. ICALEPCS 2017, Barcelona, Spain (2017)
[4] Chiozzi, G. et al., "A UML Profile for Code Generation of Component Based Distributed Systems", Proc. ICALEPCS 2011, Grenoble, France (2011)
[5] Karban, R. et al., **Cookbook for MBSE with SysML,** MBSE Initiative - SE2 Challenge Team (2011)
[6] Andolfato, L. et al., "Experiences in Applying Model Driven Engineering to the Telescope and Instrument Control System Domain", Lecture Notes in Computer Science Volume 8767, 2014, pp 403-419 - Springer International Publishing Switzerland
[7] Klatt B., "Xpand: A Closer Look at the model2text Transformation Language" 12th European Conference on Software Maintenance and Reengineering, (2008).
[8] Karban, R., Zamparelli, M., Bauvir, B., Chiozzi, G., "Three years of MBSE for a large scientific programme: Report from the Trenches of Telescope Modeling", INCOSE 2012 (Rome, 09-12 July 2012)