MBSE for the ELT Control Software

Michele Zamparelli[1], Luigi Andolfato[2], Gianluca Chiozzi[3]

[1]European Southern Observatory, Karl-Schwarzschild Strasse 2,
85748 Garching bei München, Deutschland, mzamparelli@eso.org;
[2]ESO, landolfa@eso.org; [3]HYPERLINK "mailto:gchiozzi@eso.org"EHYPERLINK "mailto:gchiozzi@eso.org"SO, HYPERLINK "mailto:gchiozzi@eso.org"gchiozzi@eso.org

Abstract: This paper summarizes how Model Based System Engineering (MBSE) methodologies and tools are used at the European Southern Observatory (ESO) to support in particular software development but also other SE areas for the construction of our world class ground based astronomical observatories. In the paper we describe how and based on which considerations we have decided to integrate different tools in the project development process.

Introduction

ESO is an International Organization building and managing astronomical observatories on the Chilean Andes. The Extremely Large Telescope (ELT) [Tam18], currently under construction, is the biggest and more complex optical telescope to be ever built, with a 39m diameter primary mirror, composed of almost 800 segments and controlled by thousands of high precision actuators and sensors.

The ELT is a project presenting many system engineering challenges and a general introduction about system engineering at ESO is given in [Eg19] in this conference.

The community involved in the design and construction of the ELT is varied, including engineers of diverse disciplines (system engineering, mechanics, optics, control, electronics, software), scientists as well as people responsible for contract procurement and management, integration and verification or maintenance.

It is natural in this context that different stakeholders show a strong preference for producing or receiving artefacts using tools or formats they are comfortable with. Imposing unfamiliar formats has proved to create (at least in our environment) strong friction among different users.

While consensus is sought on a common tool for each specific activity, we strive at the same time to have a seamless and transparent synchronization of information across tools used for different purposes.

Failing to do so would likely produce inconsistent information, and rapidly lead to divergence.

The strategy followed for the ELT and other ongoing projects is the result of several years of applying model driven engineering methodologies and tools for the development of the

control systems of telescopes and their instruments. Rationale, concrete usage and lessons learned have been presented in several papers, like [And14] and [And17]. The Active Phasing Experiment project (APE) was also used by the OMG Telescope Modeling Challenge Team as a base for writing a cookbook for MBSE with SysML [Kar11].

1 Systems engineering process for large scientific facilities

In a typical cascade process, system requirements are extracted in a formalized way from the customers, architecture and design are derived from these and finally the system is implemented, integrated, tested and deployed in operation.

As we all know very well from experience, this process is in principle very good to formalize contracts with external suppliers, but does not really work in practice for complex systems: while we proceed with analysis, design and implementation we discover that the requirements originally written were incomplete or ambiguous and it is therefore necessary to start over from them.

An iterative process is therefore necessary when developing not trivial systems and very often requirements can be understood and clarified only by discussing with stakeholders at intermediate development stages.

This is even more important in domains like our, where we implement scientific facilities with each new generation consisting of very few instances worldwide and exploring a much wider parameter space than the previous one, thanks to the adoption of new technologies: these are “unique machines” with a high margin of uncertainty with respect to final operational usage and internal efficiency, and a lot of fine-tuning at commissioning time (in contrast, the development of industrial consumer products like cars or phones proceeds in much smaller steps and functionality is much more predictable from one generation to the next).

Our strategy to manage these difficulties consists in:

- Splitting the system hierarchically in subsystems and parts that can be developed independently
- Contracting out subsystems and parts that we are confident are understood well enough to be specified in a contract that will not require drastic changes
- Keeping in house, for easy management of tight iterations:
    - All coordination functions involving multiple subsystems
    - The development of subsystems that are critical with respect to novel technologies or unexplored functionality
    - The components strictly related to the astronomical domain, for which we have a specific knowhow difficult to find in industry.

This strategy is very well supported by the adoption of MBSE principles and tools. In the following, general considerations and the chosen tools for MBSE are introduced. Some

usage examples are briefly described: requirements import and analysis, documentation generation, application code generation from state machines, and reliability management.

2 Requirements specifications using IBM Doors

It has been decided at ELT Project level to adopt the IBM Rational DOORS™ [Go16] tool for managing requirements specification across the whole project, from top level scientific requirements down to the requirements of specific subsystems or part.

Adopting a tool like DOORS™ allows to manage requirements and their relations without departing too drastically from the "traditional requirements document" paradigm with which our stakeholders are comfortable:

- The concept of modules collecting atomic objects that identify requirements can be easily understood by all stakeholders.
- Editing requirements in modules is very similar also structurally to writing a document with a word processor or a spreadsheet.
- "Paper documents" can be easily extracted from the Doors database for reviews, archiving in document-based systems and contractual purposes.
- Dependencies between requirements are managed using the intuitive concept of "link" and the tool identifies changes in linked requirements.

But this tool cannot and is not meant to support us beyond pure requirements management.

3 SysML modelling and tool

The team responsible for the development of the ELT Control Software has adopted the System Modelling Language (SysML) [Omg17] for analysis and design; the model is a hub around which several activities revolve and the ultimate source for automatically generated artefacts. Still, we do not try to produce a complete model of the system, but we model only the elements and their details that are necessary for our specific purposes of documentation, analysis and generation of artefacts. This means that different parts of the model have less details if these are not necessary for a specific purpose: we want to avoid "*modelling for the sake of modelling*".

The adoption of a modelling language goes side by side with the adoption of a powerful modelling tool: it is the tool that allows validation, simulation and the integration with other tools. We have adopted MagicDraw (*https://www.nomagic.com*), because easily extensible, open to integration with other tools and with proven standards conformance.

To ensure that the modelling is consistent across the project, we discuss, formalize and maintain modelling guidelines, starting from the guidelines defined in [Kar11]. We support the modelling of entities in our domain (control system, telescope and instrumentation) with SysML profiles that we have developed. In many cases it is then

possible to implement validation rules to check the conformance of the models to the guidelines, but this is not always possible and it does not replace discussions and peer reviews across the team.

The diagram in Figure 1 describes the dataflow of some of the modelling tools we use to develop SW. In blue are the tools we have developed in-house, in red the third-party tools. Some of these will be briefly introduced in the following sections.

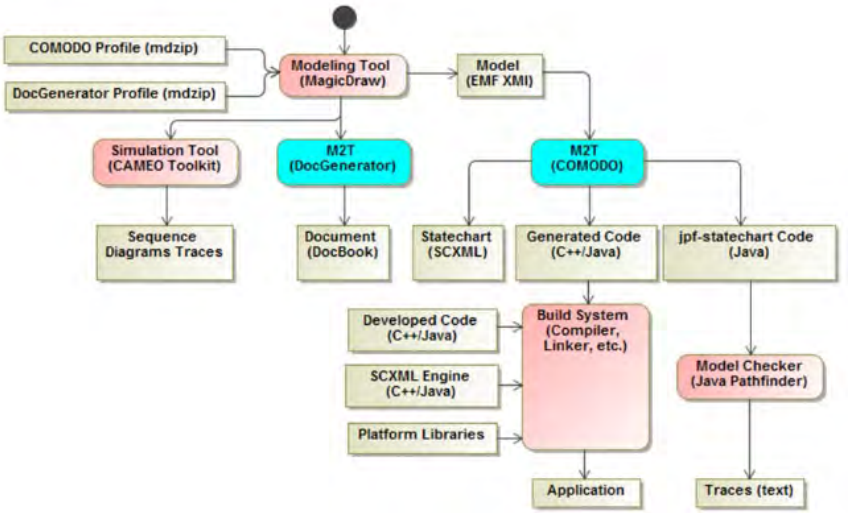

Figure 1- Toolchain and code generation.

4 Requirements management beyond specification

In order model the relations between requirements and implementation of the system, we import them from DOORS™ in MagicDraw using the ReqIF format. The tool is very good in managing synchronization, and using hyperlinks we can seamlessly navigate between the tools, so that we do not duplicate information (bi-directional synchronization is possible, but we have decided to import from DOORS™ unidirectionally).

Once requirements are in the SysML environment we can trace them through all other phases of development (analysis, design, testing, verification & validation).

5 Document generation

In order to produce “traditional documents” consistent with the model, we have developed a MagicDraw plug-in based on the DocBook standard [Oas16] that allows modelling a

document using the entities in our system model and generating a PDF or word out of it. We have also agreed on guidelines and a workflow to export from the model diagrams and tabular information that can be automatically embedded and updated in documents of various types and in other tools. This is essential to avoid “information rotting”, i.e. to ensure that documents do not become dangerously obsolete and inconsistent with time.

In general, one of the strengths of MagicDraw is in the ability to collect information from the model in tables in very powerful ways and to export this information in various formats: for many stakeholders, tables are more comprehensible than SysML diagrams.

6 Code Generation

To generate code for our SW platforms we have developed a tool called COMODO based on EMF and Xpand/Xtend[Chi11]. COMODO takes a UML model based on COMODO Profile exported from MagicDraw in the EMF XMI format and transforms it into a SCXML model (in case there are UML State Machines) and in source code for one of the selected platforms (ELT, VLT, ALMA, etc). Generated code is then compiled together with manually written code linked with platform specific libraries including the SCXML interpreter.

In addition, COMODO can be used to translate UML state machines into Java code compliant with jpf-statechart that can be analysed by the JPF model checker.

7 Reliability Modelling

The specification of functional and structural decomposition in a descriptive model using SysML can be further leveraged to address reliability aspects. In our projects, reliability plays a significant role throughout the long lifespan of a telescope or instrument.

The language extension capabilities of SysML and the possibility to plug-in external computational engines may provide a cost-effective alternative to dedicated commercial tools. A probability-based transition execution in state machines may be simulated to analyze system behavior, also un-attended. Crucially though, as specified in [Bpd09], all the known failure modes may be managed in the model at the appropriate level of decomposition (like in FMECA) and combined to provide system level failures (FTA). The computational engine allows basic faults with a given probability distribution to be used in series, parallel or redundancy calculations. While this solution does not offer the reliability catalogs provided by dedicated applications, the tool’s reporting capabilities may be used successfully to highlight which faults do not have a matching detection capability or require architectural change for further mitigation. To this end, a SysML profile and a corresponding plug-in for the tool were implemented as a proof of concept.

Conclusion

ESO's selective adoption of descriptive SysML/UML models has largely benefitted software development activities and the choice of the particular tool, with a rich and open programming interface, has permitted us to demonstrate the inter-operability with other commercial tools in the wider systems engineering domain.

The learning curve is steep and cultural resistance may only be overcome when adequate resources are available for mentoring and policing. In the future, we intend to look at how exploiting the tools' own engine for customizable validation can provide guidance through immediate visual cues when modeling guidelines are violated.

Bibliography

[And14] 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

[And17] Andolfato, L. et al.: Behavioural Models for Device Control, Proc. ICALEPCS 2017, Barcelona

[Bpd09] Douglass, P., Analyze system safety using UML within the IBM Rational Rhapsody environment, June 2009

[Chi11] Chiozzi, G. et al.: A UML profile for code generation of component based distributed systems, ICALEPCS 2011, Grenoble, France.

[Chi18] Chiozzi, G. et al.: Designing and managing software interfaces for the ELT – Proceedings SPIE 10707-31, Software and Cyberinfrastructure for Astronomy, June 2018

[Egn19] Egner, S. et al.: Systems Engineering bei der Europäischen Südsternwarte, TdSE 2019

[Go16] González-Herrera, J.C. et al.: E-ELT requirements flow down, SPIE 991101, 2016

[Kar11] Karban, R. et al.: Cookbook for MBSE with SysML, MBSE Initiative - SE2 Challenge Team, 2011

[Oas16] OASIS: DocBook Version 5.1, November 2016

[Omg17] OMG SysML v1.5, 2017

[Tam18] Tamai, R., et al., "The E-ELT program status", Proc. SPIE 10700, paper 10700-36, 2018