Model-Driven Software Development of Safety-Critical Avionics Systems: an Experience Report

Aram Hovsepyan[1], Dimitri Van Landuyt[1], Steven Op de beeck[1], Sam Michiels[1], Wouter Joosen[1], Gustavo Rangel[2], Javier Fernandez Briones[2], Jan Depauw[2],

[1] iMinds-DistriNet, KULeuven
first.last@cs.kuleuven.be
[2] Space Applications Services N.V./S.A.
gustavoenriquerangel@spaceapplications.com, jfb@spaceapplications.com, jan.depauw@spaceapplications.com

Keywords: dependability and safety, model-driven development process, early verification and validation

Abstract. The model-driven software development (MDSD) vision has booked significant advances in the past decades. MDSD is said to be very promising in tackling the "wicked" problems of software engineering including development of safety-critical software. However, MDSD technologies are fragmented as these are typically limited to a single phase in the software development lifecycle. It seems unclear how to practically combine the various approaches into an integrated model-driven software development process.
In this experience report, we present an end-to-end MDSD process that supports safety-critical software development from the point of view of Space Applications Services, an industrial aerospace company. The proposed development process is a bottom-up solution based on the state of the practice and the needs of Space Applications Services. The process integrates every software development activity starting from requirements definition all the way to the verification and validation activities. Furthermore, we have created an integrated toolchain that supports the presented MDSD process. We have performed an initial evaluation of both the process and the toolset on a case study of an On-Board Control Procedure Engine.

1 Introduction

Given the advances in the hardware technologies software development in general is becoming an increasingly complex activity. Building software for avionics systems is posing an even bigger challenge as dependability and safety are concerns of paramount importance. Dependability refers to how software failures

can result in a degradation of the system performance or even in loss of mission or material. Safety, on the other hand, is defined as a system property that is concerned with failures that can result in hazards to people or systems. For safety-critical systems it is often compulsory to perform various safety-related analyses as part of the software development lifecycle.

The model-driven software development (MDSD) vision seems very promising in efficiently tackling the essential complexities (including safety concerns) of the software development process [1]. The MDSD vision, primarily focused on the vertical separation of concerns, aims at reducing the gap between problem and software implementation domains through the use of models that describe complex systems at different abstraction levels and from a variety of perspectives. Various standards, tools and techniques that are well-aligned with the MDSD vision are currently becoming widely accepted by the industry. The Architecture Analysis & Design Language (AADL) is a de-facto standard in the domain of avionics and automotive software systems. The use of AADL enables various types of analyses that link to dependability and safety aspects (e.g., schedulability analysis). SysML is a standard general-purpose language for systems engineering. SysML could be used to plug-in certain safety-related analyses, such as Software Failure Mode, Effects & Criticality Analysis (SFMECA) and Software Fault Tree Analysis (SFTA) [2]. While these techniques contribute to the aspect of safety, they are all focused on a specific phase of the software development lifecycle. As a consequence, these tools and approaches are fragmented and it remains unclear how these approaches can be chained together to form a complete MDSD development process and toolchain. There is a lack of a pragmatic model-based software development process that provides a complete software lifecycle and transparently integrates the various building blocks. The required process should enable model-based software development starting from requirements analysis all the way to the verification and validation activities of the final implementation. Finally, the transitions and traceability links between the different phases in the development lifecycle should be automatically managed.

In this paper, we present our experiences with designing a complete MDSD process in collaboration with Space Applications Services (an independent Belgian space technology company). Our contributions are twofold. Firstly, we have created an end-to-end MDSD process that covers all phases of software development lifecycle and focuses explicitly on safety and dependability aspects. The proposed end-to-end process is conform with a set of guidelines for embedded and real-time software development prescribed by European Space Agency (ESA) [3]. The end-to-end development process leverages the V-model and the DSDM Atern agile framework [4]. We use design models for incremental skeleton code generation. Moreover, the proposed integrated process provides the necessary mechanisms to perform several critical architectural analyses, i.e., SFMECA/SFTA and various analyses enabled by the use of AADL. Secondly, we have successfully integrated a number of tools that enable the proposed MDSD process. The presented approach is currently being validated by Space Applica-

tions Services in the development of a spacecraft On-Board Control Procedure Engine (OBCP) [5].

The remainder of the paper is structured as follows. In section 2, we provide background information on relevant dependability- and safety-related standards and techniques. We also describe the problem statement in detail. In section 3, we present our solution in detail and discuss its advantages and drawbacks. Section 4 presents a number of related works. Finally, section 5 concludes this paper.

2 Background

To develop software in the avionics domain, software engineers must not only develop complex real-time software, but also place the safety and dependability qualities in the driver seat. Furthermore, certification plays an essential role in avionics software otherwise not present in many other domains. The dependability, safety and certification concerns pose a significant challenge as they affect each phase of the software development lifecycle. In this section, we provide an overview of these concepts. Then we briefly outline a number of methodologies and techniques that focus on specific aspects related to dependability and safety. In addition, we summarise how these activities are typically performed within Space Applications Services. Finally, we present the problem statement in detail.

2.1 Dependability and Safety

Dependability and safety are key concerns in the development and operations of avionics systems. The contribution of software to system dependability and safety is a key factor given the growing complexity and applicability of software in avionics applications. *Dependability* is concerned with software reliability, availability and maintainability. Software reliability is the property of software of being "free from faults" [6]. A fault can be a result of human mistake made in requirements specification, design specification, coding or even, mistakes made while executing the software development process. In general, faults can lead to errors that can lead to failures, i.e., an unexpected/unintended behaviour of the system. Software maintainability relates to the ease with which the software can be modified and put back into operation. Finally, software availability is the capability of the software to perform a required function at a given instant of time (or time interval). *Safety* is concerned with those failures that can result in actual system hazards (as opposed to software reliability that is concerned with all software failures). Safety is a system property. Nonetheless, software is a main component of a system, and therefore contributes to its safety. As opposed to typical software development, avionics software must undergo a certification process before its utilisation. Safety *certification* assures that deployment of a given system does not pose an unacceptable risk of harm. Furthermore, safety certification is also concerned with the quality of the development process and all its intermediary artifacts, such as requirements, architecture, etc.

2.2 Safety Analysis Activities

A number of tools and techniques exist that focus on dependability and safety. These techniques are typically applied in very different stages of the software development process. This section briefly describes three methodologies that are highly relevant within the domain of applications developed by Space Applications Services. It is not our intention to be exhaustive in listing the relevant approaches.

Software Failure Modes, Effects and Criticality Analysis (SFMECA) is an iterative activity, intended to analyse the effects and criticality of failure modes of the software within a system [7]. The analysis provides an essential contribution to the development of the product architecture and to the definition of the test and operation procedure. The result of the SFMECA analysis is a table that contains the failure, function, failure mode, effect, criticality, impact, action and mitigation. This analysis method can reveal failures not detected by system level analysis. Furthermore, SFMECA analysis can identify critical components, support design and verification decisions. It is essential that such decisions are easily traced back from the latter software development phases to their original artifacts.

Software Fault Tree Analysis (SFTA) is a deductive, top down method for analysing system design and performance [7]. It involves specifying a top event (also referred to as "feared event") to analyse, followed by identifying all of the associated elements in the system that could cause that top event to occur. SFTA is a logical and structured process that helps identify potential causes of system failure before the failure actually occurs. The resulting output of SFTA is a fault tree, describing the potential faults in the software.

2.3 Problem Statement

From the Space Applications Services point of view the current state of practice in MDSD suffers from three drawbacks that play a significant role in MDSD adoption.

Lack of an Integrated Process/Toolchain. Despite the clear advances in the state-of-the-art, MDSD research methodologies and techniques typically stay focused on a specific phase in the software development lifecycle. It remains quite unclear how to produce a software system starting from customer requirements all the way to a validated and verified implementation. While a one-size-fits-all approach is unlikely to provide a systematic solution, we believe that a collection of pragmatic bottom-up solutions is essential for the mainstream adoption of MDSD. This problem relates to both an end-to-end process as well as a toolchain that supports this process in a MDSD context.

Lack of Safety Engineering Methodology Integration. Even if a UML-centric end-to-end process seems feasible given the MDSD tools, avionics software systems must adhere to stringent safety standards. In the previous section, we have briefly described a number of safety analyses and methodologies that tackle a specific dependability and/or safety related aspect of the system. Space Applications Services currently performs both SFMECA/SFTA analyses manually by leveraging Office-like (e.g., Powerpoint/Excel) applications. Ideally, architecture and design models (with some additional annotation for feared events and causing/contributing factors) could be used to run SFMECA/SFTA analyses. Real-time performance analyses, such as schedulability analysis, end-to-end flow latency analysis, are automated, but performed only at the implementation level. The use of architecture-level analyses enabled by AADL could allow Space Applications Services to early verify and validate all design decisions. The integration of all these activities in a hypothesised UML-centric end-to-end MDSD process is not obvious.

Lack of End-to-End Process Traceability. Traceability plays an essential role in the domain of avionics systems especially in the context of the certification process. Indeed, it is crucial to have the necessary abstractions to trace each code-level entity back to a set of requirements. An end-to-end MDSD process introduces additional challenges as traces should ideally provide complete information regarding the code, model and requirements interrelations.

3 Space Applications Services Development Process

In this section, we present a prototype end-to-end development process that provides a pragmatic answer to the challenges outlined in the previous section. The proposed approach is inspired by the engineering process proposed by the European Space Agency (ESA) for the development of embedded and real-time on-board software. We also briefly mention a number of tools that provide the backbone of the proposed process.

3.1 Software Development Process

ESA has introduced a standard engineering process relevant to all space elements of a space system [3]. The phases covered by the standard are as follows. **Requirements Baseline** corresponds to the complete specification provided by the end-user regarding the software product expectations. **Technical Requirements** correspond to all technical aspects that the software shall fulfil with respect to the end-user requirements. **Software Architecture Design** corresponds to the overall architecture that is created and refined based on the technical requirements. **Software Component Design** corresponds to a more detailed description of the elements described by the software architecture. **Implementation** corresponds to the development of the various software components described in the software component design phase. **Verification**

corresponds to the testing of produced implementation in order to verify the correctness of the product performance. **Validation** corresponds to the testing of the software components as well as the complete software in order to validate the correctness of product performance.

We have created an integrated development process that is based on the notion of the V-Model and the Agile Dynamic System Development Method Atern framework. Figure 1 illustrates a structural view of the development process that presents each development activity along with their structural connections to other activities. This process is in line with the V-Model. Our contribution is represented in grey by the two additional activities, i.e., SFMECA/SFTA and AADL analyses, that cut across multiple development phases. The lines between each activity schematically represent not only the process flow, but also the artifact exchange between activities. For instance, technical requirements analysis is preceded by the software architecture design. Ideally, each requirement is known and accessible at the architectural level. This enables the creation of traces (or the traceability information) that link architectural elements to their corresponding requirements. The traceability information between different development phases is essential as it enables early requirements validation. Note that the process is inherently iterative and one can always go back to an earlier activity. This information was dropped from figure 1 for readability purposes.

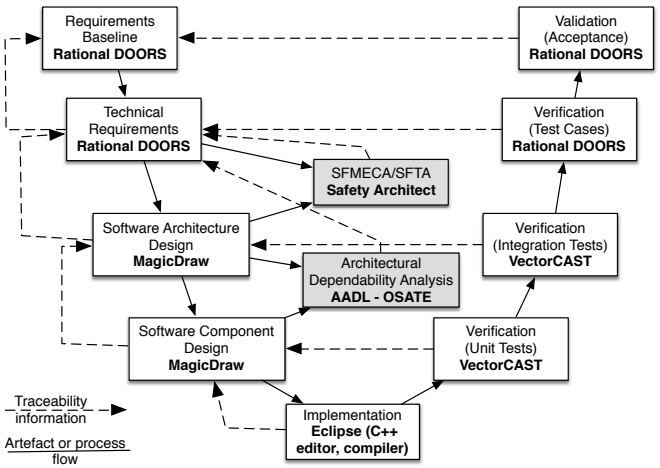

Fig. 1. Space Apps V-Model Software Development Process

For the dynamics of this process we leverage the Dynamic System Development Method (DSDM) Atern agile project delivery framework used for software development. The idea behind DSDM is to develop a solution iteratively starting from global view of the product. For a detailed description of DSDM Atern, we refer the interested readers to [4].

3.2 Integrated Toolchain

We further briefly outline the tools currently utilised within Space Applications Services that support the presented structural process. We also provide information how software development artifacts are interchanged between the tools. Figure 2 presents the tools for each activity. Note that some Space Applications Services' customers require the use of Rational DOORS during the software development. However, all other tools can be freely replaced by alternatives.

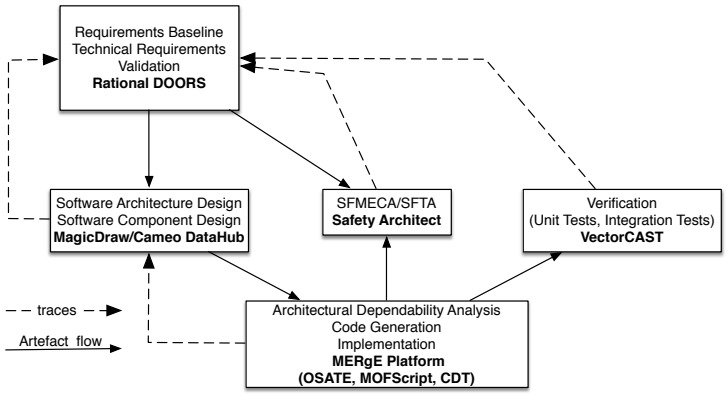

Fig. 2. Structural Software Development Process Toolchain

IBM Rational DOORS tool is used for the the *Requirement Baseline*, *Technical Requirements* and *Validation Activities*.
MagicDraw tool is used for the *Software Architecture Design* and *Software Component Design* phases. The data interchange between Rational DOORS and MagicDraw is realised by the Cameo DataHub tool that features a complete synchronisation of requirements as well as traceability links between the tools. Note that MagicDraw features a built-in functionality to both export and import all modelling artifacts to the Eclipse Modelling Framework (EMF). EMF [8] provides the common infrastructure and a de facto UML standard implementation for the model interchange between various tools.
Safety Architect is used for the *SFMECA/SFTA Analyses* activities [2]. Eclipse EMF is used as a common language to interchange models.
MERgE Platform is an Eclipse-based toolset that provides an integrated collection of plug-ins. We leverage the **MOFscript** [9] plug-in for transforming the UML models into their AADL representation. We use the UML MARTE profile for annotating the UML model elements with real-time and embedded properties [10]. The AADL model is used by the **OSATE** tool for performing *Architectural Dependability Analysis*. We also use MOFScript to generate skeleton C code that is further incrementally refined into a working implementation.
Eclipse CDT plug-in provides the necessary tools for the C code implementa-

tion. **VectorCAST** is used for the three *Verification* phases it automates unit and integration testing activities.

3.3 Evaluation

In collaboration with Space Applications Services, we have performed an initial evaluation of the proposed process and toolchain on the case study of an On-Board Control Procedure Engine. The MDSD process is considered to be complete in the sense that it covers all phases from a software development lifecycle required from the Space Applications Services point of view. The software artefacts integration throughout the various phases is either provided by the tool (e.g., Requirements and Technical requirements in Rational DOORS, or Software Architecture and Software component Design in MagicDraw) or automatically transformed by additional tools (e.g., MOFScript). Moreover, all the transitions support the incremental nature of the complete process. This is essential as existing artefacts shall not break by subsequent iterations (e.g., manual refinements to the AADL models or the generated code must be preserved). We have successfully integrated a number of AADL analyses by implementing ideas presented in [10]. The integration of safety-related analyses (i.e., SFMECA and SFTA) are currently work in progress. Finally, we have provided an initial implementation towards tackling the traceability challenge. The traceability of various elements between Rational DOORS and MagicDraw are actually provided by the tools. The traceability between MagicDraw and the MERgE Platform is implemented by incorporating references to the MagicDraw modelling elements as comments both in the generated AADL model as well as in the generated code.

While the proposed approach seems pragmatic and effective in tackling the challenges described in section 2.3 we still face a number of challenges that are work in progress. At the toolchain level, we are still working towards integrating the SFMECA/SFTA analyses in the Safety Architect tool. At the process level, we are facing the challenge of having a process that contains many implicit constraints. If these constraints are not correctly followed the process may become completely useless. For instance, source code should not be manually refined without encoding that information into the detailed design as subsequent iterations may break the manual code. This problem can be efficiently tackled by using a Process Modelling Language (PML), such as OMG SPEM [11]. However, even if such constraints are made explicit it is impossible to capture all possible situations. Furthermore, developers always deviate from the process model either because of lack of experience or the imperfections of the proposed process. Da Silva et al [12] present a systematic approach agnostic to a particular PML selection to deal with such deviations. Our end-to-end MDSD approach could substantially benefit from an integration with such a systematic framework. Finally, a challenge both at the process and at the toolchain level remains the management of the traceability information between various entities across different abstraction levels, which is currently somewhat implicit. Ideally, a systematic approach should allow the transparent management of all traces within a separate view.

4 Related Work

A number of research efforts focused on architecture optimisation are complementary to our work as they would enhance the Software Architecture Design and Architectural Dependability Analysis phases. Etemaadi et al have presented an approach with a supporting toolkit named AQOSA to support architecture optimisation with respect to quality attributes [13]. Meedeniya et al have addressed the problem of evaluating reliability based on software architectures in the presence of uncertainty [14]. Brosch et al have introduced a reliability modeling and prediction technique that considers the relevant architectural factors of software systems by explicitly modeling the system usage profile and execution environment and automatically deriving component usage profiles [15]. As opposed to these research initiatives our approach is focused more on the overall development process rather than a specific development phase.
Several research initiatives are focused on providing a systematic methodology for tool integration. Balogh et al have proposed a workflow-driven tool integration framework using model transformations that allows one to formally specify contracts for each transition between tools in the tool chain [16]. Klar et al have created a meta-model-driven environment that allows to integrate tools by focusing on traceability links between dependent tool artifacts [17]. The approach we have proposed in this work could be seen as a case study for the tool integration frameworks.

5 Conclusion

Developing software for the avionics domain is an extremely challenging task given the strict dependability and safety requirements. The advent of Model-Driven Software Development (MDSD) standards and tools has substantially improved the current state-of-the-art by introducing a number of systematic disciplines throughout various stages of the software development lifecycle. Unfortunately, these techniques are currently fragmented and it remains unclear how these could be combined into an integrated end-to-end software development process. In this experience paper, we have proposed a complete MDSD process inspired by the V-model that integrates the various standards and tools into a single integrated software development process. The proposed process is based on the needs and experiences within Space Applications Services, an industrial aerospace company. The MDSD process integrates transparently a number of standards from the aeronautics domain, such as architectural analysis and design language (AADL), software failure modes, effects and criticality analysis (SFMECA) and software fault tree analysis (SFTA). Finally, the end-to-end MDSD provides an initial answer to the traceability requirements within Space Applications Services. In the future we plan to integrate the MDSD process with a systematic process modelling and deviation detection and resolution framework. We are also looking to improve the integration of traceability information. Finally, we plan to further validate the proposed process on the case study of an On-Board Control Procedure Engine.

Acknowledgements

The presented research is partially funded by the Research Fund KU Leuven and the Flemish agency for Innovation by Science and Technology (IWT 120085). The research activities were conducted in the context of ITEA2-MERgE (Multi-Concerns Interactions System Engineering, ITEA2 11011), a European collaborative project with a focus on safety and security [18].

References

1. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: Proceedings of the 29th International Conference on Software Engineering, IEEE Computer Society (2007) 37–54
2. All4Tec: Safety architect. (http://all4tec.net/index.php/en/model-based-safety-analysis)
3. ECSS Space Engineering: Safety. ECSS-E-ST-40C. Misc (2009)
4. Consortium, D.: The DSDM Atern Handbook. DSDM Consortium (2008)
5. ECSS Space Engineering: Spacecraft on-board control procedures. ECSS-E-ST-70-01C. Misc (2010)
6. ECSS Space Engineering: Software dependability and safety. ECSS-Q-HB-80-03A. Misc (2012)
7. Jet Propulsion Laboratory: Software Fault Analysis Handbook. (2005)
8. Eclipse: Eclipse modeling framework (EMF). (http://www.eclipse.org/emf/)
9. SINTEF: MOFScript. (http://modelbased.net/mofscript/)
10. Faugére, M., Bourbeau, T., de Simone, R., Gérard, S.: MARTE: Also an uml profile for modeling AADL applications. In: ICECCS, IEEE Computer Society (2007)
11. OMG: Software Process Engineering Metamodell SPEM 2.0. Technical report, OMG (2006)
12. da Silva, M.A.A., Bendraou, R., Blanc, X., Gervais, M.P.: Early deviation detection in modeling activities of mde processes. In: MoDELS. (2010) 303–317
13. Etemaadi, R., Lind, K., Heldal, R., Chaudron, M.R.V.: Quality-driven optimization of system architecture: Industrial case study on an automotive sub-system. Journal of Systems and Software (2013) 2559–2573
14. Meedeniya, I., Moser, I., Aleti, A., Grunske, L.: Architecture-based reliability evaluation under uncertainty. In: Proceedings of the Joint ACM SIGSOFT Conference. QoSA-ISARCS '11, New York, NY, USA, ACM (2011) 85–94
15. Brosch, F., Koziolek, H., Buhnova, B., Reussner, R.: Architecture-based reliability prediction with the palladio component model. Transactions on Software Engineering (2011)
16. Balogh, A., Bergmann, G., Csertán, G., Gönczy, L., Horváth, Á., Majzik, I., Pataricza, A., Polgár, B., Ráth, I., Varró, D., Varró, G.: Workflow-driven tool integration using model transformations. In: Graph Transformations and Model-Driven Engineering. Lecture Notes in Computer Science, Springer (2010) 224–248
17. Klar, F., Rose, S., Schürr, A.: TiE - a tool integration environment. In: Proceedings of the 5th ECMDA Traceability Workshop. Volume WP09-09 of CTIT Workshop Proceedings. (2009) 39–48
18. MERgE Consortium: MERgE: Multi-concerns interactions system engineering. (http://www.merge-project.eu)