ACM/IEEE 17th International Conference on
Model Driven Engineering Languages and Systems

September 28 – October 3, 2014 • Valencia (Spain)

MD²P² 2014 – Model-Driven Development Processes and Practices

Workshop Proceedings

Regina Hebig, Reda Bendraou, Markus Völter, and Michel Chaudron (Eds.)

Published on Sept 2014 v1.0

Editors' addresses:

Regina Hebig, Reda Bendraou
Sorbonne Universités, UPMC Univ. Paris 06, UMR 7606, LIP6, F-75005, Paris (France)

Markus Völter
independent/itemis (Germany)

Michel Chaudron
Chalmers Technical University and University of Gothenburg (Sweden)

Organizers

Regina Hebig (co-chair)	Université Pierre et Marie Curie, LIP6, Paris (France)
Reda Bendraou (co-chair)	Université Pierre et Marie Curie, LIP6, Paris (France)
Markus Völter (co-chair)	independent/itemis (Germany)
Michel Chaudron (co-chair)	Chalmers Technical University and University of Gothenburg (Sweden)

Program Committee

João Paulo Almeida	University of Espírito Santo (Brazil)
Reda Bendraou	Université Pierre et Marie Curie, Paris (France)
Gregor Berg	BIOTRONIK SE & Co. KG (Germany)
Michel Chaudron	Chalmers Technical University and University of Gothenburg (Sweden)
Bernard Coulette	Universite de Toulouse II-Le Mirail (France)
Brian Elvesæter	SINTEF (Norway)
Regina Hebig	Université Pierre et Marie Curie, Paris (France)
Jochen Kuester	Bielefeld University of Applied Sciences (Germany)
Bran Selic	Malina Software Corp. (Canada)
Florian Stallmann	SAP AG (Germany)
Jim Steel	University of Queensland (Australia)
Harald Störrle	Danmarks Tekniske Universitet (Dänemark)
Matthias Tichy	Chalmers Technical University and University of Gothenburg (Sweden)
Markus Völter	independent/ itemis (Germany)
Ingo Weisemöller	Carmeq GmbH (Germany)

Additional Reviewers

Grischa Liebel
Tewfik Ziadi

Table of Contents

Model-Driven Development Processes and Practices: Foundations and Research Perspectives 2
Regina Hebig, Reda Bendraou, Markus Völter, Michel Chaudron

Approaches to Automation and Interoperability in Systems Engineering 7
Tom Ritter

Introducing Usability Concerns Early in the DSL Development Cycle: FlowSL Experience Report 8
Ankica Barišić, Vasco Amaral, Miguel Goulão, Ademar Aguiar

A Qualitative Study of Model Transformation Development Approaches: Supporting Novice Developers 18
Gabriel Costa Silva, Louis M. Rose, Radu Calinescu

Model-Driven Software Development of Safety-Critical Avionics Systems: an Experience Report 28
Aram Hovsepyan, Dimitri Van Landuyt, Steven Op de beeck, Sam Michiels, Wouter Joosen, Gustavo Rangel, Javier Fernandez Briones, Jan Depauw

Towards Enabling Cross-Organizational Modeling in Automotive Ecosystems .. 38
Eric Knauss, Daniela Damian

Model-Driven Development Processes and Practices: Foundations and Research Perspectives

Regina Hebig[1], Reda Bendraou[1], Markus Völter[2], Michel Chaudron[3]

[1] Université Pierre et Marie Curie, LIP6, Paris (France), forename.surname@lip6.fr
[2] independent/itemis (Germany), voelter@acm.org
[3] Chalmers Technical University and University of Gothenburg (Sweden), michel.chaudron@cse.gu.se

Abstract. MD^{2}P^2 is a workshop about the interrelation of model-driven development (MDD) and development processes. The workshop provides a forum for researchers and practitioners to exchange experiences on the questions how processes need to adapt or can be adapted when model-driven techniques are applied. We argue that the interrelation between MDD and development processes can be crucial for the success of MDD. For example, the need to adapt a process when introducing MDD can be a reason to decide against an MDD adoption. Further, we aim to give an introduction to foundations and research perspectives. MD^{2}P^2 is co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages & Systems.

1 Introduction

Model-driven development, which includes the synthesis of executable systems from models or the use of abstract languages, e.g. UML, Simulink, or DSLs, and software development processes are used to reach similar targets, such as increasing the quality of software or the efficiency of software development. Although, these goals are approached differently, MDD and software development processes are not totally independent.

In fact, the literature has a multitude of proposals for MDD specific development processes. Proposals for the use of MDD in context of established development processes, such as the V-Modell XT or SCRUM, however, sometimes include significant adaptations of the development process. Stakeholders' responsibilities might be as affected as quality assurance activities, which need to respect the structure of the artifacts used. Partly, even process phases and sprints are adapted to enable a combination with MDD. It seems that most adaptations have pragmatic reasons, and aim at supporting a fruitful combination.

However, it is also possible that MDD and a particular development process do not fit together well. For example, the benefit of an agile approach might be reduced, if long running transformation chains enforce long running sprints.

Further, an MDD approach that requires to build languages and transformations first before "business value" can be delivered to the customer, conflicts superficially with some agile processes.

Thus, companies that aim at adopting MDD also have to face the questions, how appropriate development processes have to look like as well as whether and how existing development processes can be adapted. It is important to consider that development process adaptation can be cost intensive (e.g. due to required training of developers or changes in the involvement of stakeholders).

In summary, the questions whether and how an established development process must be adapted when MDD is introduced are crucial, since they can impact the efficiency of the development processes as well as the costs of the MDD introduction. In face of the growing number of MDD techniques and the variety of MDD approaches applied in practice, there is a need for systematic guidelines or best practices that help with adjusting or tailoring of development processes, when MDD is introduced.

In the following we give an introduction into foundations of the workshop's topic and present various research perspectives. We then provide an overview on this first edition of the **MD^{2}P^2** workshop.

2 Foundations

Software processes define various aspects of development, such as phases or tasks, but also documents that need to be created at different points in time. Furthermore, software processes are concerned with "soft" aspects, e.g. teams, skills, communication and roles. Similarly in MDD there are technical aspects, such as automation, through e.g. generators or model transformations, that can predefine fine-grained sequences of activities between manual and automated tasks. Further, just as processes, also MDD can, due to concepts such as abstraction, affect soft aspects as the skill set of developers. Thus, both MDD and software development process can affect similar and related aspects of development. However, the question how these interrelations can lead to mutual constraints or to synergy effects is only rarely studied.

Research on the keyword *software process tailoring* (i.e. on the question how processes need to be customized due to environmental factors) is rarely concerned with MDD directly. However, as summarized in the survey of Kalus et al., programming languages and tool infrastructure are known to be criteria for process tailoring [4].

A bit more attention was drawn to this topic in the modeling community [2]. For example, Aranda et al. found out that the division of labor changed within General Motors when MDD was introduced [1]. Further, Heijstek et al. learned in a case study with at an international IT service provider that MDD usage can lead to an increased need for collective code ownership. Further they observed changes in communication, required skills, and tooling [3].

As our survey on processes that had been adapted for MDD has shown, influences between both MDD and software process can be most diverse [2]. While in some cases mainly roles had been adapted within the processes, in other cases the structure of phases or sprints of the process changed. For example,

Loniewski et al. describe an adaptation of OpenUP (a variant of RUP), where new roles, such as "model analyst" had been introduced [6]. In contrast, Kulkarni et al. adapt SCRUM by adding a meta sprint for long running tasks [5].

Summing up, there is an awareness that software processes might need or have the potential to adapt, when MDD is applied. However, there is still much research to be done before we fully understand this interrelation.

3 Research Perspectives

This first workshop on model-driven development processes and practices aims at calling attention to the question how development processes can be integrated with an MDD approach.

First of all, this is an empirical research question. Therefore, the **MD^{2}P^2** workshop provides a forum for researchers and practitioners to exchange and discuss experiences on how the use of MDD affects the development process in practice. For example, there is so far not much knowledge about how MDD is affected by a maturing development processes. Further, the workshop tries to uncover what happens to the development processes in practice, when MDD is introduced. Empirical data or case studies from practice can help to approach diverse questions:

- Which stakeholders are involved in modeling tasks & which stakeholders are not affected by the integration of MDD? – These questions are interesting for two reasons. On the short term, it might be less expensive to train a smaller set of developers and stakeholders to the new technologies. On the long term, however, only stakeholders who are involved in modeling also have potential to benefit from the higher level of abstraction and the improved automation.
- Which (modeling) artifacts are subject to quality assurance activities, e.g. reviews or testing? – MDD defines the set of artifacts that represent the system under development. This includes the potential for quality assurance, when certain checks can be performed earlier in development. However, it can also change the skills that are required for e.g. reviews and with it the roles in the process.
- Are development process phases adapted? Does the number or frequency of iterations change? – The structure of process phases or the iterations are essential for many processes. Changing them might have a major impact on the characteristics of the process. Since there are hints that such changes sometimes occur due to MDD, the question becomes pressing, when and why these adaptations happen.
- Is there empirical evidence that the intended MDD effects occur, e.g. does front-loading actually reduce the number of errors in later phases? – As indicated above, MDD is associated with certain hopes. However, only empirical evidence can prove whether these goals are reached.

In addition to the empirical perspective, it is also necessary to approach the topic more theoretically. Such research can cover systematic investigations

of the mechanisms that drive impacts from MDD approaches on development processes or in turn define constraints on MDD approaches that are implicitly defined by development processes. Based on these investigations, researchers might foster the success of MDD, e.g. by providing guidelines, methods, or tools that support practitioners in reusing or adapting development processes when MDD is introduced are required.

Investigations on what aspects of a process are affected by MDD can concern diverse aspects, e.g.: How are different stakeholders integrated in the modeling activities? Can modeling tasks be split over multiple roles and phases? What is the effect of automated verification methods on testing methodologies and philosophies defined in development processes (e.g. in test-driven development processes)? Is there a need to adapt test and quality assurance activities in development processes, such that the various modeling artifacts are covered appropriately? When is it necessary or beneficial to adapt the number of development process phases or to change the frequency of iterations?

One motivation to answer these questions, is to identify combinations of MDD and processes that do not fit together, i.e. where the benefits of the process or of the MDD approach cannot fully be used. This is a first step towards tailoring MDD or processes, such that a better fit can be reached. A second motivation is that it might be possible to achieve synergies between MDD and development processes. Associated questions are: How can the combination with an MDD approach increase (or decrease) the benefits of a process? How can the choice or adaptation of a process increase (or decrease) the benefits of an MDD approach?

Ideally this research leads to guidelines and methods that can support practitioners in reusing or adapting development processes when MDD is used. Furthermore, researchers might come up with tool support for the integration of MDD into a given process. Finally, as both the development processes and MDD evolve and mature, there is the question how synergistic effects can be maintained over time, i.e. how co-evolution or co-maturation of MDD and development processes can be supported.

To summarize, there are many open questions related to the combination of MDD and software processes. With this workshop we want to strongly encourage more researchers to contribute to the investigation of these questions.

4 First Edition of MD²P²

For this first edition of the workshop, we received eight papers, of which four have been accepted for inclusion in the proceedings. The accepted papers can be split in two groups:

The first group of accepted papers deals with the question how processes look like that are used for the development of MDD technologies, such as transformations or DSLs. Barisic et al. present of the case study of FlowSL, how a usability concerns can be considered throughout during DSL development throughout an agile development approach where MDD tools are used [7]. Silva et al. present a survey on model transformation development approaches, discuss what phases of the development process are supported by the different approaches and present several lessons learned [10].

The second group of accepted papers provide an empirical view on processes that are used in combination with MDD. Hovsepyan et al. present an experience report on the use of MDD in development of safety-critical avionic systems [8]. Knauss et al. investigated a case study in the automotive domain and identify challenges for the use of MDD in multi-tier automotive ecosystems [9].

We hope that the workshop will help researchers and practitioners to build up a community that shares data and experience. In closing, we would like to thank all authors papers and reviewers, for their contributions, the effort they invested, and for making this workshop possible.

References

1. Aranda, J., Borici, A., Damian, D.: Transitioning to model-driven development: What is revolutionary, what remains the same? In: Model Driven Engineering Languages and Systems, LNCS, vol. 7590, pp. 692-708, Springer, (2012)
2. Hebig, R., Bendraou, R.: On the need to study the impact of model driven engineering on software processes. In: Proceedings of the 2014 International Conference on Software and System Process, pp. 164-168. ACM, New York (2014)
3. Heijstek, W., Chaudron, M. R.: The Impact of Model Driven Development on the Software Architecture Process. In: Software Engineering and Advanced Applications (SEAA) (2010)
4. Kalus, G., Kuhrmann, M.: Criteria for software process tailoring: a systematic review. In: Proceedings of the 2013 International Conference on Software and System Process, pp. 171-180, ACM, (2013)
5. Kulkarni, V., Barat, S., Ramteerthkar, U.: Early experience with agile methodology in a model-driven approach. In: Proceedings of the 14th international conference on Model driven engineering languages and systems (MODELS'11), eds. Whittle, J., Clark, T., Kühne, T., pp. 578 -590, Springer, Berlin, Heidelberg (2011)
6. Loniewski, G., Armesto, A., Insfran, E.: An agile method for model-driven requirements engineering. In: The Sixth International Conference on Software Engineering Advances (ICSEA 2011), pp. 570-575 (2011)
7. Barisic, A., Amaral, V., Goulo, M., Aguiar, A.: Introducing usability concerns early in the DSL development cycle: FlowSL experience report. In: Proceedings of the 1st International Workshop in Model-Driven Development Processes and Practices (MD^2P^2), pp. 8-17 (2014)
8. Hovsepyan, A., Van Landuyt, D., Op de Beeck, S., Michiels, S., Joosen, W., Rangel, G., Fernandez Briones, J., Depauw, J.: Model-Driven Software Development of Safety-Critical Avionics Systems: an Experience Report. In: Proceedings of the 1st International Workshop in Model-Driven Development Processes and Practices (MD^2P^2), pp. 28-37 (2014)
9. Knauss, E., Damian, D.: Towards Enabling Cross-Organizational Modeling in Automotive Ecosystems. In: Proceedings of the 1st International Workshop in Model-Driven Development Processes and Practices (MD^2P^2), pp. 38-47 (2014)
10. Silva, G. C., Rose, L., Calinescu, R.: A Qualitative Study of Model Transformation Development Approaches: Supporting Novice Developers. In: Proceedings of the 1st International Workshop in Model-Driven Development Processes and Practices (MD^2P^2), pp. 18-27 (2014)

Keynote

Approaches to Automation and Interoperability in Systems Engineering

Tom Ritter

Fraunhofer FOKUS, Berlin, Germany

Creating large and complex systems typically involves a number of potentially geographically separated development teams and a number of various different tools. These two aspects are two important dimensions of complexity, which should be considered when planning large system engineering efforts. The importance of these aspects has become increasingly eminent and recent approaches try to handle these issues. Among them are OSLC (Open Services for Lifecycle Collaboration) and ModelBus®. Those approaches address the platform aspect of system engineering on complementary level of abstraction and are a step forward with respect to integration and interoperability challenges. These approaches set de facto standards, which is important to increase efficiency in systems engineering. Based on these considerations big European initiatives like the ARTMIS Joint Undertaking projects CRYSTAL and VARIES started to work on a Reference Technology Platform with the goal to improve interoperability. The talk will present challenges and solutions in large-scale system engineering efforts and focuses on the aspect of collaboration and standardisation.

Dr. Tom Ritter *graduated with a Masters degree in Computer Science from the Technical University of Berlin and did his PhD in 2011 at Humboldt University Berlin in modeling quality of service of component oriented systems. Since 1998 he worked at Fraunhofer Institute FOKUS in the area of tool development and distributed systems. Since 2006 he was heading the Model-Driven Engineering group at FOKUS and from 2010 he was the Deputy Head of the competence center SQC (formerly MOTION). As of December 2013 he is Head of the Competence Center "System Quality Center" (SQC). His major interest is the model-driven software engineering, the development of software tools, software development processes, tool integration infrastructure and the consideration of non-functional properties and QoS at design and execution time. Tom is one of the Co-Authors of a CORBA Component based Middleware Platform (Qedo). Since 2004 Tom participated to the design of the tool integration infrastructure ModelBus and he is heading the ModelBus development team. Tom is involved in different standardization activities at the Object Management Group, he is co-author on books about components and services and contributes continuously to workshops and conferences.*

Introducing Usability Concerns Early in the DSL Development Cycle: FlowSL Experience Report

Ankica Barišić[1], Vasco Amaral[1], Miguel Goulão[1], and Ademar Aguiar[2]

[1] CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Campus de Caparica, 2829-516 Caparica, Portugal
[2] Departamento de Engenharia Informática, Faculdade de Engenharia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
a.barisic@campus.fct.unl.pt, vma@fct.unl.pt, mgoul@fct.unl.pt, ademar.aguiar@fe.up.pt

Abstract. Domain-Specific Languages (DSLs) developers aim to narrow the gap between the level of abstraction used by domain users and the one provided by the DSL, in order to help taming the increased complexity of computer systems and real-world problems. The *quality in use* of a DSL is essential for its successful adoption. We illustrate how a usability evaluation process can be weaved into the development process of a concrete DSL - FlowSL - used for specifying humanitarian campaign processes lead by an international Non-Governmental Organization. FlowSL is being developed following an agile process using Model-Driven Development (MDD) tools, to cope with vague and poorly understood requirements in the beginning of the development process.

Keywords: Domain-Specific Languages, Usability Evaluation, Agile Development, Language Evaluation, Software Language Engineering

1 Introduction

Domain-Specific Languages (DSLs) and Models (DSMs) are used to raise the level of abstraction, while at the same time narrowing down the design space [1]. This shift of developers' focus to using domain abstractions, rather than general purpose abstractions closer to the computation world, is said to bring important productivity gains when compared to software development using general purpose languages (GPLs) [2]. As developers no longer need to make error-prone mappings from domain concepts to computation concepts, they can understand, validate, and modify the produced software, by adapting the domain-specific specifications [3]. This approach relies on the existence of appropriate DSLs, which have to be built for each particular domain. Building such languages is usually a key challenge for software language engineers. Although the phases of a typical DSL life cycle have been systematically discussed (e.g. [4, 5]), a crucial step is often kept implicit: the language evaluation.

DSLs are usually built by language developers in cooperation with domain *experts* [6]. In practice the DSL will be used by domain *users*. These domain *users*

are the real target audience for the DSL. Although domain *users* are familiar with the domain, they are not necessarily as experienced as the domain *experts* helping in the language definition. Neglecting domain *users* in the development process may lead to a DSL they are not really able to work with.

In this paper we apply action research to the development of a DSL, named FlowSL, designed to support managers in specifying and controlling the business processes supporting humanitarian campaigns. FlowSL is targeted to non-programmers. Their ability to use this language was identified as one of the highest concerns, so discovering usability issues in early development iterations, facilitated the achievement of an acceptable usability, while tracking the design decisions and their impact.

Usability has two complementary roles in design: as an *attribute that must be designed into the product*, and as the *highest level quality objective* which should be the overall objective of design [7].

This paper is organized as follows: Section 2 discusses related work; Section 3 provides a description of the evaluation approach; Section 4 discusses the language and evaluation goals and its development and evaluation plan; Section 5 discusses the lessons learned from the application of the described approach; finally, Section 6 concludes by highlighting lessons learnt and future work.

2 Related work

The need for assessing the impact of introducing a DSL in a development process has been discussed in the literature, often with a focus on the business value that DSL can bring (see, e.g. [8]). This business value often translates into productivity gains resulting from improved efficiency and accuracy in using a DSL [6], when compared to using a general-purpose baseline solution [9]. The quality in use of a DSL is, therefore, extremely important. In general, these assessments are performed with a final version of a DSL, when potential problems with the DSL are expensive to fix. A key difference in the work described in this paper is that we introduce language evaluation *early* in the DSL development process, so that problems can be found 'on-time' and fixed at a fraction of the cost it would take to fix them, if detected only in the deployment phase.

The term *quality in use* is often referred to more simply as *usability* [7], and includes dimensions such as *efficiency*, *effectiveness*, *satisfaction*, *context coverage* and *freedom of risk* (ISO 25010 2011). Usability evaluation investments have brought an interesting return on investment in software development [10]. Usability evaluation benefits span from a reduction of development and maintenance costs, to increased revenues brought by an improved effectiveness and efficiency by the product users [11].

Two important issues are *how* and *when* to assess DSL usability.

Concerning the *how*, we have argued that we can think of DSLs and their supporting editors as communication interfaces between DSL users and a computing platform, making DSL usability evaluation a special case of evaluating User Interfaces (UIs) [12] . This implies identifying the key quality criteria from

the perspective of the most relevant stakeholders, in order to instantiate an evaluation model for that particular DSL [13, 14]. These criteria are the evaluation goals, for which a set of relevant quantitative and qualitative measurements must be identified and collected. We borrow from UI evaluation several practices, including obtaining these measurements by observing, or interviewing, users [15]. In general, it is crucial that the evaluation of human-computer interactions includes real users [16], for the sake of its validity. In the context of DSLs, the "real users" are the domain users.

Concerning the *when*, we argued that we should adopt a systematic approach to obtain a timely frequent usability feedback, while developing the DSL, to better monitor its impact [17]. This implies the integration of two complementary processes: language development and evaluation. Software language engineers should be aware of usability concerns during language development, in order to minimize rework caused by unforeseen DSL usability shortcomings. In turn, usability designers should have enough understanding of the DSMs involved in software language development to be able to properly design the evaluation sessions, gather, interpret, and synthesize meaningful results that can help language developers improving the DSL in a timely way. This requirement is in line with agile practices, making them a good fit for this combined DSL building (i.e.software development) and evaluation process (i.e. usability design) [18].

3 Building usability into a DSL development process

Building a DSL may have a rather exploratory nature, with respect to the DSL requirements, particularly when the DSL is aimed for users with limited computational skills or poorly understood, or evolving domains. To build up a cost-effective and high quality process, we defined an agile and user centered DSL evaluation process [17, 13].

By placing DSL users as a focal point of DSLs' design and conception, the goal was to ensure that the language satisfies the user expectations. Besides involving Domain Experts and Language Engineers, as typically happens in the development of a DSL, we add the role of the Usability Engineer to the development team. Usability engineers are professionals skilled in assessing and making usability recommendations upon a given product (in this case, the DSL) and gathering unbiased systematic feedback from stakeholders [18].

Each development iteration focuses on a different increment or level of abstraction to be evaluated or refined. In the early phases it is important to study existing guidelines or standards for a particular domain and interview current or potential users about their current system or tools they are using to help them in accomplishing their tasks. This *context of use* study of a particular situation is intended to elicit the strengths and weaknesses of the baseline approach as well as the user expectations for the DSL.

Finally, once the language is deployed to users, an evaluation of its use in real contexts should be conducted, reusing the methods and metrics that were validated in the previous iterations.

4 Flow Specification Language (FlowSL)

The generic process described in the previous section was instantiated to the development of a concrete DSL — the FlowSL. FlowSL is a DSL for specifying humanitarian campaigns to be conducted by a non-governmental organization. FlowSL is integrated in MOVERCADO[3] (MVC), a mobile-based messaging platform at the core of an ecosystem that enables real-time and a more efficient impact, by facilitating interactions among beneficiaries, health workers and facilities, e-money and mobile operators. The platform is meant to allow data mining in the search of insights that can be used to improve the effects of the campaigns while supporting a high degree of transparency and accountability.

A first version of the system (MVC1) was developed as a proof-of-concept to validate the key underlying principles. The second version of the system (MVC2) was developed in the form of a platform easily customizable by managers and extensible by developers of the organization's team. An important goal was to develop a language, FlowSL, to empower the Campaign Managers to define new kinds of campaign flows taking advantage of their domain knowledge.

Without FlowSL, managers needed to specify the flows orchestrating their campaigns exclusively by means of presentations and verbal explanations. The implementation and evolution of campaigns often resulted in rework and unexpected behavior, usually due to vague specifications, incorrect interpretations, and difficulties in validating the implementation, a phenomenon known as *impedance mismatch* [19]. Therefore, the primary goal was to evolve the system to enable new users to easily create new campaigns and underlying flows. FlowSL is expected to enable the organization to streamline the process of defining campaigns and their base workflows, namely participants, activities, interaction logic, and messages.

4.1 FlowSL development process

In order to balance the development effort with effective reusability (e.g. while envisioning new marketing solutions), MVC2 was developed in a fast-paced way, iteratively, along six two-weeks sprints, following an agile development process based on Scrum[4] and best practices of evolving reusable software systems [20]. In the process of FlowSL development, the Domain Experts were part of the Product Owners team, while the Language Engineers were part of the Scrum Team. The DSL evaluation process was guided by the FlowSL development stages, as different effort was estimated in each sprint for its development.

The problem analysis was performed by mutual interaction and brainstorming between Domain Experts and Language Engineers in each sprint planning. Usability Engineers, in this case the researchers, had the role of observing and guiding the analysis outputs, while preparing the evaluation plan, without being directly involved in the language specification. To better understand and define

[3] http://enter.movercado.org/ (accessed in July 19, 2014)
[4] http://www.scrum.org/ (accessed in July 18, 2014)

the problem, the required functionalities were described in terms of small user stories. Also, the new description of the user roles was introduced as the FlowSL is expected to change existing organizational workflows. To improve interaction between the development team and the users, all the produced results from the analysis were continuously documented in a wiki. As Scrum suggests, the project management was based on a product backlog maintained and shared on-line.

The relationship between the MVC system, FlowSL development, and relevant language users and expected workflow is presented in Fig.1. The original MVC1 system was developed in a GPL (Ruby). FlowSL was first developed as a Ruby-based internal DSL. This approach allowed an optimal use of resources while keeping the existing system running. The second phase of language development was intended to support the managers to design the campaign flow specifications by themselves, using simple and understandable visual language constructs. In the planned third phase (future work), the focus will be on evolving the language's editor to be collaborative and web-based. It will also be an opportunity to work on language's optimizations in the generation process.

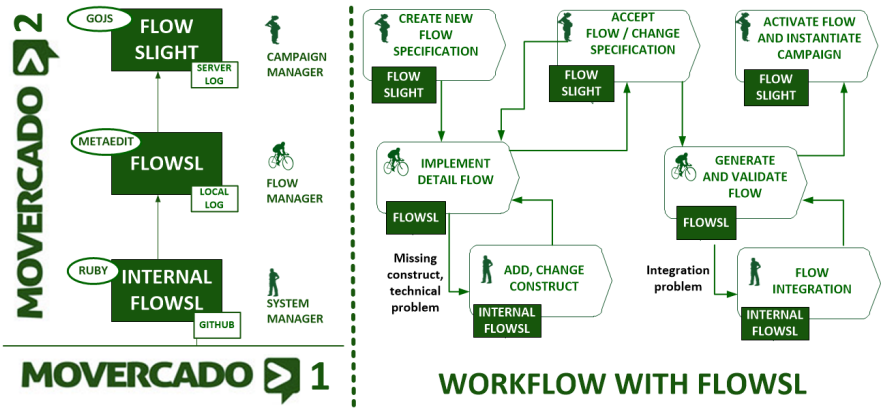

Fig. 1. FlowSL development and relevant language users with expected workflow

After defining the evaluation plan, the Usability Engineer prepared the usability requirements, using a goal-question-metric approach presented in Table 1, where goals conform to the Quality in Use model. These requirements were detailed and related to the right metrics and measurement instruments to perform appropriate usability tests in each development cycle. The validation of some of these requirements in earlier stages (e.g. understandability, readability) are stepping stones to achieve other soft requirements that cannot be evaluated in early phases (e.g. learnability). Multiple evaluations helped in validating and improving the set of identified metrics.

Table 1. Usability requirements

Requirement	*Metric*
Understandability: Does the user understand the different concepts and relations, and when and why to use each one of the concepts?	NCon - number of concepts, NRel - number of relationships NErrSpec - incorrect verbal definitions of total NCon and Nrel given in language NErrMod - incorrect interpretations of presented NCon and NRel given in modeled solution
Readability: How accurately is the user able to read the specified flows and interpret their meaning?	NConInst - number of concept instances in the model (flow), NReInst - number of relationship instances in the model NErrInst - number of incorrect verbal interpretation of NConInst and NRelInst given in language
Efficiency: How much time is needed for a user to read existing or specify a new flow?	TModInst - time necessary to read existing model instance (flow) TModSpec - time necessary to implement a new model instance (flow)
Effectiveness: Is the user able to correctly implement a flow from a given high-level description of the required flow?	NErrModInst - number of misinterpretation while reading existing model instance (flow) NErrModSpec - number of errors while implementing new model instance (flow)
Learnability: How much time is needed for users to learn the FlowSL language?	TLearNov - training time necessary to learn novice users to use language TLearExp - training time necessary to learn domain experts to use language
Flexibility How long does it take to quickly change or correct existing flow specifications?	TModEvol - time necessary to evolve model instance (flow) TModCorr - time necessary to correct incorrect implementation of model instance (flow)
Reusability How often user reuse existing flow specifications?	NModReuse - number of reusing existing model instance (flow) NModEvol - number of evolving existing model instance (flow)
Expressiveness Is the user able to specify all parts of flow?	NErrCon - number of concept, or its property that user is missing to implement model instance (flow) NErrRel - number of relationships, or its appropriate role that user is missing to implement model instance (flow)
Freadom of Risk Is the user able to implement the specifications in a way that can lead to unexpected or unwanted system behavior?	NEconDem - number of occurrence of economic damage due to incorrect flow specification NSofCor - number of occurrence of software corruption due to incorrect flow generation to system (flow)
Satisfaction How much is the user satisfied with FlowSL?	ConfLevel - self rated confidence score in a Likert scale LikeLevel - self rated likability score in a Likert scale

5 FlowSL evaluation and lessons learned

5.1 First FlowSL iteration: bottom-up approach (MVC2.1)

The *language goal* of the first iteration was to find the differences and commonalities in the Ruby code relevant for visual FlowSL and then do a corresponding mapping into a graphical representation, which would define the first draft of the concrete visual syntax of FlowSL. This is considered as a way to describe appropriate activities step by step by mapping relevant fragments of extracted code to a visual representation and to identify repetitive patterns that represent

reusable code artifacts. The *evaluation goal* was to assess whether this representation would be good enough to enhance the *understandability* and *readability* of flows from the perspective of Campaign Managers. It was expected that with the flow abstraction, the Domain Experts could describe more concrete requirements for the visual flow concepts.

The *evaluation intervention* was conducted when all existing flows of the MVC1 system were migrated to MVC2. This was the moment when the stakeholders could more clearly express the language purpose by distinguishing campaign processes from the flows underneath. The intervention was followed by an interview conducted with one representative *subject*: the Domain Expert with the role of Campaign Manager that was involved in specifying flows using the MVC1 system and who was also involved in the MVC2 Scrum development assuming, in that case, the role of Product Owner.

The *evaluation document* was prepared by the Usability Engineer containing 4 tasks: *Task 1* and *Task 2* describing user scenarios by roles and a global organization scenario that evaluator was asked to clarify and improve by placing him in organization workflow; *Task 3* presenting alternative feature models of FlowSL that are reviewed and redefined with a goal of separating campaign instantiation data and improving a vague language definition; *Task 4* presenting campaign flow based on simple and complex level of specification of the flow example (*IPC Validation*) that was found to be the most representative to describe. This task used metrics from the GQM table, which showed that the considered solution is very hard to understand.

The two major threats to validity of this evaluation were that it was subjective and only one user surrogate was involved. However, as the intended solution was seen as a step that helped to understand and to model the domain better, the guided interview helped to redefine the technical concepts using domain terms. Evaluation resulted in a clearer plan for the next development cycles as well as clarifying usability requirements and appropriate tasks. The textual FlowSL makes explicit all relevant domain concepts, but also many extra more. considered more technical, The performed evaluation helped the DSL developers to adjust the level of abstraction to the needs of the DSL end users. The language at this phase, could be used by the System Managers (knowledgeable of the concepts of the baseline system), but not by Campaign Managers.

5.2 Second FlowSL iteration: top-down approach (MVC 2.2)

The *language goal* of this iteration was to develop a visual FlowSL prototype using the MetaEdit[5] language workbench, that was selected for its support to top-down development. The *evaluation's goal* was to assess whether both the campaign managers and novice system managers were able to validate the specified flows using the newly proposed visual language and editor. These evaluations covered also the effectiveness and expressiveness of the target language.

[5] http://www.metacase.com/ (accessed in July 19, 2014)

The *First evaluation intervention* was organized very quickly and involved interviewing two *subjects*: the campaign manager from the first development iteration and the system manager who was involved in the DSL development. The intervention consisted of one task where the subjects had the opportunity to compare two alternative concrete flow representations for the same ongoing example.

Based on the evaluation results the Usability Engineer produced designs of the concrete syntax for the DSL development team.

The *second evaluation intervention* involved the same subjects. The *evaluation document* had three tasks: Task 1 focused in assessing the *understandability* and *expressiveness* of the individual symbols; Tasks 2 and Task 3 meant to measure the *readability* and *efficiency* of the designed solution of the simple and complex flow. In addition to that, the Domain Expert was asked to describe the use of the symbols from Task 1 to produce the presented flow solutions and to describe the situations in which the existing flows can be reused. The evaluation session with the System Manager made it possible to identify important missing relationships between FlowSL concepts, as well as their connection points (hot spots) with the MVC system underneath.

For the *third evaluation intervention* the usability engineer introduced the design improvements motivated by the feedback obtained the previous evaluation. The new notations were designed and implemented, to be again compared. The tasks were similar to the previous intervention, although more elaborated. Here, the same subjects from the previous interventions were involved, as well as a member of the Scrum team.

For this third intervention the rules related to the usage of a certain activity were discussed. The usability engineer evaluated the cases where the system manager would have the need to hack the existing campaign flows, in order to customize certain functionality or rule. The goal was to use an example-based approach to identify improvements in the language.

It became clear that the evaluation materials prepared earlier helped to speed up the following evaluation phases and reduced their implementation costs. Besides, they became templates for the corresponding learning materials. Also, it was possible to abstract the language one level further, so that an online visual editor was built to support rapid high level specifications of flows. To better deal with the increasing complexity of the specified models, rather than presenting all the concepts related to the flow definition visually, a better option would be to present just high level concepts that are reused often, while others are hidden and based on predefined rules that can be eventually reconfigured textually. This approach empowered both the domain experts and the product owners to better control the design decisions.

6 Conclusions and future work

In this paper, we presented an experience report on how to integrate top-down usability engineering practices into a bottom-up agile development of a DSL

from its beginning. While playing the role of Usability Engineers, we experienced that small iterations involving Domain Experts, Product Owners and End Users can help us to clarify the meaning and the definition of the relevant language concepts. This enables an early identification of possible language usability shortcomings and helps reshaping the DSL accordingly.

Early evaluations can be executed with a relatively low cost thanks to model-driven tools that support production of rapid prototypes and presenting the idea. These evaluations support well-informed trade-offs among the strategy and design of the DSL under development, and its technical implementation, by improving communication. Besides, they improve the traceability of decisions, and of the solution progress. These iterations also help to capture and clarify contractual details of the most relevant language aspects that need to be considered during DSL development, and are a key element to improve the End Users experience while working with FlowSL.

We plan to validate our decisions, metrics, and the overall merit of the developed DSL, by performing experimental evaluations with both expert and novice users, by making comparisons to the baseline approach in Ruby, as well as to other process modelling languages that are natural candidates to serve for similar purposes (e.g. BPMN, JWL).

An additional step is to conceptualize the traceability model of design changes and evaluate its impact on the decision making process. We expect that in each iterative evaluation step we will not only identify opportunities to improve the usability of the DSL, but also to improve the evaluation process itself (e.g. through the validation, in this context, of the chosen metrics).

Weaving usability concerns into agile process is helping us to continuously evolve FlowSL, improving the cost-effectiveness of DSL usage in specifying campaigns, and supporting a clearer assessment of which language concepts are more relevant to the different kinds of language users, which in turn helps finding the right level of abstraction and granularity of concepts. All these benefits come with the cost of adding usability skills and of introducing new practices in the agile process, namely the introduction of lightweight metamodeling tools. The balance however, seems to be very positive, but ROI should be calculated precisely to support this claim.

References

1. Gray, J., Rossi, M., Tolvanen, J.P.: Preface. Journal of Visual Languages and Computing, Elsevier **15** (2004) 207–209
2. Kelly, S., Tolvanen, J.P.: Visual domain-specific modelling: benefits and experiences of using metacase tools. In Bézivin, J., Ernst, J., eds.: International Workshop on Model Engineering, at ECOOP'2000. (2000)
3. Deursen, A.V., Klint, P.: Little languages: Little maintenance? Journal of Software Maintenance: Research and Practice **10**(2) (1998) 75–92
4. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Computing Surveys **37**(4) (2005) 316–344

5. Visser, E.: WebDSL: A case study in domain-specific language engineering. In Generative and Transformational Techniques in Software Engineering II, Ralf Lämmel, Joost Visser, and João Saraiva (Eds.). Lecture Notes In Computer Science **5235** (2007)
6. Voelter, M., Dietrich, C., Engelmann, B., Helander, M., Kats, L., Visser, E., Wachsmuth: DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. CreateSpace Independent Publishing Platform (2013)
7. Petrie, H., Bevan, N.: The evaluation of accessibility, usability and user experience. Human Factors and Ergonomics. CRC Press (2009)
8. Kelly, S., Tolvanen, J.P.: Domain-specific modeling: enabling full code generation. John Wiley & Sons (2008)
9. Kosar, T., Mernik, M., Carver, J.: Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. Empirical Software Engineering **17**(3) (2012) 276–304
10. Nielsen, J., Gilutz, S.: Usability return on investment. Technical report, Nielsen Norman Group (2003)
11. Marcus, A.: The ROI of usability. In Bias, Mayhew, eds.: Cost-Justifying Usability. North- Holland: Elsevier (2004)
12. Barišić, A., Amaral, V., Goulão, M., Barroca, B.: Quality in use of domain-specific languages: a case study. In: Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools. PLATEAU '11, New York, NY, USA, ACM (2011) 65–72
13. Barišić, A., Monteiro, P., Amaral, V., Goulão, M., Monteiro, M.: Patterns for evaluating usability of domain-specific languages. Proceedings of the 19th Conference on Pattern Languages of Programs (PLoP), SPLASH 2012 (October 2012)
14. Kahraman, G., Bilgen, S.: A framework for qualitative assessment of domain-specific languages. Software & Systems Modeling (2013) 1–22
15. Rubin, J., Chisnell, D.: Handbook of Usability Testing: How to plan, design and conduct effective tests. Wiley-India (2008)
16. Dix, A.: Human computer interaction. Pearson Education (2004)
17. Barišić, A., Amaral, V., Goulão, M., Barroca, B.: How to reach a usable DSL? moving toward a systematic evaluation. Electronic Communications of the EASST **50** (2011)
18. Lárusdóttir, M., Cajander, Å., Gulliksen, J.: Informal feedback rather than performance measurements–user-centred evaluation in scrum projects. Behaviour & Information Technology (ahead-of-print) (2013) 1–18
19. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the Tropos project. Information systems **27**(6) (2002) 365–389
20. Roberts, D., Johnson, R.: Evolving frameworks: A pattern language for developing object-oriented frameworks. In: Proceedings of the Third Conference on Pattern Languages and Programming, Addison-Wesley (1996)

A Qualitative Study of Model Transformation Development Approaches: Supporting Novice Developers

Gabriel Costa Silva, Louis M. Rose, and Radu Calinescu

Department of Computer Science, University of York,
Deramore Lane, York YO10 5GH, UK
gabriel@cs.york.ac.uk,
{louis.rose,radu.calinescu}@york.ac.uk

Abstract. Developing model transformations is not a straightforward task. It is particularly challenging when the developer has limited or no experience in this area. This not only impedes the adoption of model transformations, but also prevents companies from the benefits of Model-Driven Engineering. We qualitatively analyse eight of the most relevant approaches to developing model transformations cited in the literature. Different from most studies in this area, we focus on life-cycle activities other than implementation. By highlighting the strengths and weaknesses of these approaches, we help new developers in selecting an approach and complement existing studies in this area.

Keywords: Model transformation; Development; Analysis; Comparison

1 Introduction

The use of Model Transformations (MT) in software engineering leads to several benefits, such as complexity reduction and portability [3], [4]. However, developing MT is a challenge. As Siikarla et al. state in [12], "*An application developer with proper domain knowledge can manually create a target model based on a source model. He can probably even come up with some of the most often used rules of thumb for the transformations. However, defining a complete mapping from all possible source models to target models is remarkably harder.*" The state-of-the-art for developing MT consists of describing the MT definition informally in natural language [5], based on an "educated guess" [12], and implementing the transformation definition using a transformation language [6], [18]. This informal approach leads to several drawbacks [6], [18], [20], such as inconsistency between specification and implementation [5] and cost increase [8].

Since MT are software [12], they need to be engineered as software [6], using a systematic process of:

(i) specifying problems and requirements [5], [6], [7];
(ii) acquiring knowledge about semantic correspondences [19], [20];

(iii) modelling source and target models to outline a transformation [7], [12], [19];
(iv) creating mappings between meta-models [6], [19], [20];
(v) setting up constraints and rules for model mappings [6], [19], [20];
(vi) defining transformation architecture [6];
(vii) implementing the transformation definition using a transformation language [5], [6], [12], [20]; and
(vii) validating the transformation definition [5], [6], [12].

Furthermore, analogous to software development, MT can take advantage of further techniques to support its development, such as transformation patterns [7], [18]. As Guerra et al. advocate in [6], developing MT "*requires systematic engineering processes, notations, methods and tools*". Although there are several tools for implementing MT definitions [20], there is: (i) little work addressing the whole life-cycle of MT [12], [19]; (ii) lack of guidelines for the systematic development of MT, in particular, to support novice MT developers; and (iii) different perspectives regarding the best way to develop MT.

The analysis in this paper aims to contribute to the adoption of Model-Driven Engineering by providing a qualitative analysis of state-of-the-art approaches in developing model-to-model (M2M) transformations, investigating their strengths and weaknesses, and highlighting lessons learned for supporting MT developers starting in this area. Our analysis was prompted by the following research question: Which methods and techniques should a novice MT developer use? Unlike most work in MT, such as that reported in [10], our focus lies in activities of MT life-cycle other than implementation. In particular, we investigate processes, a modelling language, and transformation patterns to support MT development.

Note that this is not the only challenge associated with MDE adoption. However, challenges such as model and meta-model development are outside the scope of our paper. By a literature review, we identify approaches to support MT life-cycle. Then, we qualitatively analyse recent approaches according to three aspects (Section 2): (i) model transformation foundations; (ii) features; and (iii) applicability. Finally, we summarise lessons learned during our experience with analysing and adopting MT in the cloud computing domain (Section 3).

2 Analysis

In our investigation of M2M transformation development approaches, we identified no study, either qualitative or quantitative, comparing approaches to MT development in phases other than implementation. Furthermore, taking into consideration approaches we analysed in this paper, apart from the approach presented in [6], no approach was extensively evaluated, providing only worked examples to support a feasibility analysis. Moreover, so far, no approach for MT development has been widely adopted. These three facts make hard to choose a suitable approach when developing MT, fostering the concept of "*ad hoc*" MT development [6]. It becomes even harder when the MT developer has limited experience in this activity. In order to support novices in the task of selecting an

approach for MT development, we provide in this section a qualitative analysis of approaches present in the MT literature. Table 1 summarises our analysis.

2.1 Study Design

As Sjoberg, Dyba & Jorgensen define in [16], "*qualitative methods collect material in the form of text, images or sounds drawn from observations, interviews and documentary evidence, and analyze it using methods that do not rely on precise measurement to yield their conclusions.*" To this end, we used 13 criteria for analysing the approaches covered in our analysis, classifying the approaches into three groups: MT foundations, main features, and applicability of each approach.

The first group consists of foundation concepts of MT, as presented in [3] and [4]. The second group is based on the most relevant features implemented by approaches. Finally, the last group consists of our evaluation of the applicability of these approaches, taking into consideration the perspective of a novice MT developer. The evaluation of these approaches was exclusively based on information presented in their respective papers. As Seaman advocates in [11], by conducting this qualitative analysis we aim at providing rich and informative analysis to support novice MT developers. In addition, we aim at increasing the knowledge on how these approaches can contribute to designing MT.

Investigating model transformation literature, we identified eight research papers presenting approaches supporting M2M transformation development. As in our previous investigation [14] the IEEE digital library provided the most significant set of primary studies for our research, we decided to use this library as our primary source. Our search identified 121 papers, from which we critically analysed both title and abstract, resulting in two papers selected for full reading [7], [8]. In addition, we were supported by an MDE expert, who suggested another paper [6]. Finally, by analysing references and citations of previously selected papers, we found five more relevant papers [5], [12], [18], [19], [20].

Based on an analogy with software development, we classified the identified approaches as:

(i) *traditional* [6], [12] when they advocate an iterative process of refinements, from requirements to MT implementation and test;
(ii) *by example* [18], [19], [20] when they concentrate on simplifying the transformation development, focusing on model mappings [9];
(iii) *emergent* [5] when based on emergent software development approaches (e.g., agile), which proposes innovative ways to develop software; and
(iv) *transformation patterns* [7], [8] which enable the identification of recurrent relations in a model transformation, supporting the definition of transformation rules in a reusable way [8].

Table 1. Summary of key characteristics of MT approaches

		Traditional		By-Example			Emergent	Transformation Pattern	
		Siikarla et al. (1)	Guerra et al. (2)	Varró (3)	Wimmer et al. (4)	Sun, White & Gray (5)	Giner & Pelechano (6)	Jin & Guisheng (7)	Iacob, Steen & Heerink (8)
MT Foundations	Can it have multiple sources/targets?	No	Yes	No	No	No	No	No	No
	Can it support exogenous/ endogenous transformations?	Exogeneous	Both	Exogeneous	Exogeneous	Endogeneous	Exogeneous	Exogeneous	Both
	Can it support to mappings between source and target elements?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Can it support rule definitions?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Features	Is it language independent?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
	Which phases of transformation life cycle does it cover?	3, 4, 5, 7, 8	1, 2, 3, 4, 5, 6, 8	3, 4, 5, 7, 8	3, 4, 5, 7	3, 4, 5, 7, 8	1, 4, 5, 7, 8	4, 5	4, 5
	Where does the focus lies in?	Model	Both	Model	Model	Model	Model	Meta-model	Both
	How many types of artefacts does it specify?	4	9	1	3	3	1	1	1
	Does it requires tooling support?	No	Yes	Yes	Yes	Yes	Yes	No	Yes
Applicability	How sound is the approach?	Minimal	Excellent	Minimal	Minimal	Minimal	Minimal	Minimal	Good
	Which type of evaluation was carried out?	Feasibility	Case study	None	None	None	Feasibility	Feasibility	Feasibility
	How much detail is provided?	Good	Excellent	Good	Excellent	Excellent	Minimal	Minimal	Good
	Can support other approach?	No	Yes	No	No	No	No	Yes	Yes

2.2 Model Transformation Foundations

The first group of criteria consists of: multiple source/ target, exogenous/ endogenous MT, mappings, and rules. It is critical to note that the papers we analysed provided little, or no clear information about the two first criteria. For those cases, we analysed the examples provided throughout the paper as the main reference to infer answers for this question. To simplify the reference to each approach, Table 1 enumerates approaches using numbers from (1) to (8). These numbers will identify their respective approaches from now on.

Taking into consideration the support for multiple source/ target, only (2) explicitly stated the support for multiple source and targets by their mapping diagram. All other approaches suggested support for only single source/ target models. For example, (3) made clear the decision taken by a set of stated assumptions. Siikarla, (1), suggested his decision in a figure, used to explain his approach. Analysing the transformation patterns presented, none of them consider multiple source/ targets. For all other approaches, they suggested their decision by examples presented throughout the paper.

Regarding the type of transformation supported, most of approaches support only exogenous transformations. The only exception is (5), which presented only endogenous examples, suggesting that this approach aims at also supporting endogenous transformations. Examples presented by (2) suggested that it supports

both types of transformations. Likewise, the patterns flattening and mapping, presented by (8), indicated that it supports both types of transformations.

As mappings and rules are core concepts in MT, both concepts are considered by all analysed approaches. The main difference in this regard lies in how these concepts are implemented by different approaches. For example, whereas (1) set up correspondence examples to map models, and transformation patterns to map meta-model, (6) specified mappings by test cases. Regarding rules, whereas (2) defined it in their low-level design, (3) proposed their automatic generation.

2.3 Features

The second group of criteria consists of: language independence, phases covered, focus, artefacts, and tooling support. In the context of this paper, language independence means that the approach is not specific to a particular language for the source and target models/meta-models. Apart from (8), that defines its transformation patterns in the context of QVT, all other approaches are language independent – though the examples presented are based on a particular language, such as (1), that adopted UML. To analyse the phases of MT life cycle covered by approaches, we took into consideration the general process for developing MT, introduced in Section 1. The set of phases we included in this process is the result of an analysis of MT literature. Table 2 summarises the phases covered by each approach, and how each approach covers such phase.

Regarding the focus of each approach, although MT is defined at the meta-model level, most approaches focus on the model rather than the meta-model level. As Wimmer et al. explain in [20], meta-models might not define all language concepts explicitly. Regarding the approaches we analysed, the exception is (7), which focuses on the meta-model, and both (2) and (8), which address both model and meta-model. It is important to note that (8) described their patterns in terms of model, but the definition is made at meta-model level.

Approaches proposed a number of artefacts to support the developer. Table 1 shows the number of artefact types though it is central to note that this information is unclear in some papers, such as those which present (6), (4), and (5). Finally, regarding tool support, (1) and (7) do not require a particular tool. In the context of this analysis, requiring tool support means that the approach cannot be wholly applied without a particular tool. For example, (7) does not require a particular tool although it defines several automatic activities. Likewise, although (2) defines a family of languages, its MT development process is language independent. Unlike the other approaches, (2) generates automatically only test and traceability artefacts. In particular, by-example approaches define semi-automatic tasks, that require specific tools. Approach (6) also defined the use of specific tools, such as HUTN, EVL, and an adapted version of JUnit.

2.4 Applicability

To evaluate the applicability of each approach, we considered four criteria: soundness, evaluation, level of technical detail, and the ability of the analysed approach

Table 2. MT phases covered by each approach analysed

	Traditional		By-Example			Emergent	Transformation Pattern	
	Siikarla et al. (1)	Guerra et al. (2)	Varró (3)	Wimmer et al. (4)	Sun, White & Gray (5)	Giner & Pelechano (6)	Jin & Guisheng (7)	Iacob, Steen & Heerink (8)
1 - Specifying problems and requirements		Requirements diagram				Test case		
2 - Acquiring knowledge about semantic correspondences		Formal specification/ Scenario definition						
3 - Modelling source and target models to outline a transformation	Correspondence examples	Scenarios definition	Step 1 - Mapping models	Step 1 - Creating models	Demonstration			
4 - Creating mappings between meta-models	Transformation pattern/ definition	Mapping diagram	Step 1, 2 - Derive transformation rules	Step 2 - Creating correspondence	Automatically infered	Test case	Specified by patterns	Specified by patterns
5 - Setting up constraints and rules for model mappings	Transformation pattern/ definition	Rule diagram	Step 2, 3 - Rules adjustment	Step 2, 3 - Deriving correspondences	Automatically infered	Test case	Specified by patterns	Specified by patterns
6 - Defining transformation architecture		Architecture diagram						
7 - Implementing the transformation definition by a transformation language	Transformation implementation		Step 4 - Rules execution	Step 3 - Deriving correspondences	Step 5 - Replaying transformations and checking	Transformation language		
8 - Validating the transformation definition	Evaluation	Test language	By process iteration		Step 5 - Replaying transformations and checking	Comparing result of transformation with test case		

to support another — like design patterns can support software development process. We define the soundness of an approach, using the following scale: (i) *minimal* — the paper introducing the approach provides very limited information about approach theoretical foundations; (ii) *good* — the paper justifies the decisions taken and reasons to underpin their decisions, even the information is limited; and (iii) *excellent*, meaning that the text not only describes decisions and their reasons, but also provides empirical evidence and basis from literature.

Taking into consideration this scale, only (2) was classified as having an *excellent* soundness whereas (8) was classified as having a *good*. This result becomes worse when analysed along with the next criterion, evaluation. None of by-example approaches were evaluated in the papers analysed. Apart from (2), which conducted two case studies, other approaches conducted feasibility analyses, taking into consideration worked examples. These two results highlight the need for empirical evidence to support future work in this area. This is particularly important for the industrial adoption of these approaches. However, note that these results do not invalidate the approaches. In fact, as discussed in Section 3, many of these ideas are valid and useful.

Aiming at the application of these approaches, we evaluated the level of technical detail provided in the paper. There are three possible classifications for this criterion: (i) *minimal*, when the text does not provide enough information to support the application of this approach; (ii) *good*, when the text provides enough information to support the application of this approach, though detailed information might be missing; and (iii) *excellent*, when the text not only provide

enough information, but also further details about activities, such as examples. In this regard, (7) and (6) were classed as *minimal*. Approaches (2), (4), and (5) provide *excellent* information, supported by examples that complement the understanding. However, it is critical to point out that these three approaches rely on tooling support, and the knowledge regarding the tools might be not enough. The other three approaches were classified as *good*.

Finally, we analysed whether these approaches could support other approach. For example, a traditional software development process might be supported by several approaches, such as design patterns and graphical modelling languages. Apart from transformation patterns and the MT language (approach (2)), which by definition could be applied to support other approaches, all other approaches defined their particular, non-extendable, life-cycle. In some cases, such as that of (5), the need for tooling support creates additional dependency.

3 Lessons Learned

In this section, we summarise lessons learned while transforming a cloud model [13], defined as part of our approach to support cloud portability [15], into a TOSCA definition. Cloud computing is a computing model in which resources, such as computing, are provided as services through the Internet [1]. Topology and Orchestration Specification for Cloud Applications (TOSCA) is a standard specification supported by OASIS to enable cloud portability [2]. This section is based on our attempt to systematically apply the approaches previously presented. Lessons reported in this section complement the analysis in Section 2.

Developing model transformations is as complex as traditional software development

Comparing the generic MT process introduced in Section 1 with the traditional software development (TSD) life-cycle, such as waterfall [17], we can conclude that both processes are similar. A developer needs requirements to define the objectives, artefacts to guide the development, and tests to check for errors. Like the TSD, the code is the main artefact, which represents the MT definition. However, a number of questions arise when starting the requirement definition for a MT. In contrast to TSD, in which one defines several requirements, in MT, inicially, there is one single requirement: transform model A into model B.

In MT, the single requirement proposed must be broken down into several others, specifying that the entity X, in the meta-model A, will become the entity Z, in the meta-model B. To this end, the requirements diagram presented in [6] is a relevant contribution since it enables requirement decomposition and the creation of links between requirements to set up dependencies. However, to define such a mapping, it is necessary to have a clear understanding of semantic correspondences between the meta-models involved. Furthermore, unless these semantic correspondences have been tested before, establishing the requirements would be an error-prone task. For example, when we defined the requirements

for our cloud-to-TOSCA MT, we did not know which TOSCA entity a cloud entity will become – though we knew very well both domains.

Thus, we had to analyse semantic correspondences of both meta-models before defining the requirements. In addition, we had to test if these correspondences made sense. In this regard, the test-driven approach proposed in [5] was useful. A test-case is an artefact which outlines these semantic correspondences. Then, by implementing the transformation, this initial assumption is confirmed or refuted. However, from a novice MT developer, the complexity involved in defining requirements for MT is far harder than in TSD. In addition, requirement definition and mapping definition are two close activities. Finally, M2M transformations might involve model-to-text transformations as well, making the process even more complex. Thus, despite the similarity with TSD, in our perspective, MT development requires extra artefacts and activities.

Models work well as examples, but not as the main transformation drivers

By-example approaches, in particular, advocate the use of models rather than meta-models as the main driver of a MT. Indeed, having two models, one representing a domain A, and another representing a domain B, aids the design of an A-to-B MT. However, creating these models is not trivial. Although it might be an intuitive process when creating a transformation from a UML class diagram to an E-R diagram – a common example used in the literature, other domains require a huge effort. In our case, we found it to be impossible to devise a TOSCA model based on our cloud model without an in-depth preliminary analysis of semantic correspondences. The reason for that is quite simple: a model conforms to a meta-model. Therefore, one cannot create a representation of model A, which conforms to meta-model A, in conformance with meta-model B unless the semantic correspondences between meta-models A and B are known beforehand.

For example, the by-example approaches proposed in [19] and [20], define in their first activities the creation of source and target models, and the mapping of entities between these models. In our case, we already had the cloud model, however, we expected to follow a well-defined MT process in deriving the TOSCA model (target). At that moment, we could infer that a cloud *Service* is similar to a TOSCA *TNodeType*, but we did not know correspondences for other entities, such as a cloud *Region*, and *User*. Thus, a process advocating the mapping of models to achieve meta-models correspondences was not applicable because without the meta-model correspondences it was impossible to derive the models.

In this regard, we learned that by-example approaches could be useful when meta-model correspondences are already known, and two well-known meta-models are given. Thus, models can be mapped and meta-model correspondences can be automatically generated using these approaches. On the other hand, the creation of two models is quite useful when designing MT as a way to validate meta-model mappings. For example, after identifying that a cloud *Resource* is equivalent to a TOSCA *TNodeType*, we created a TOSCA model representing this mapping. Then, we could validate the model in two ways: checking in the general context

of TOSCA whether this transformation makes sense, and submitting the generated model to a TOSCA runtime environment. If the environment could process the specification, it meant that the transformation succeeded.

Other lessons

- Although not yet addressing all MT development concerns, the analysed approaches provide very useful contributions. Overall, we concluded that, like MT area, the identified approaches are still maturing. As shown by Table 1, most of them were neither extensively evaluated nor sound enough. However, they provide several insights about performing MT, such as correspondence examples [12], requirements diagram [6], and test-cases [5];
- Testing is critical in MT as it is in TSD. It is important to carry out several tests when developing MT. In our experience, we identified problems with different datatypes (e.g., conversion of *String* to *xs:string*), names (space between nouns versus no space), and one entity in the source meta-model being mapped to several others in the target meta-model;
- Capturing trace links between source and target model elements is a good practice for MT, particularly if a MT becomes complex. In our experience, one single entity in the cloud meta-model became three entities in the TOSCA meta-model, which in turn gave rise to several others. At the end, the set of dependencies created was so complex, that it was hard to validate them. Inspecting the trace model can help considerably in cases like this, as it enables to identify source and target entities.

4 Conclusion

In our investigation of M2M transformation development, we identified no study, either qualitative or quantitative, comparing approaches for MT development in phases other than implementation. This complicates the selection process of a suitable approach when developing MT, fostering the concept of "ad hoc" MT development. It becomes even harder when the MT developer has limited experience in this activity. To support the selection of an approach for MT development, we provided in this paper a qualitative analysis of eight state-of-the-art approaches for MT development. This analysis took into consideration 13 criteria, classified in three groups: model transformation foundations, features and applicability. We complemented this analysis presenting the lessons learned from our own experience with developing a MT for cloud domain.

Acknowledgments. This work was funded in part by CNPq - Brazil.

References

1. Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A.: A view of cloud computing. Communications of the ACM 53(4), 50–58 (Apr 2010)

2. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications. In: Advanced Web Services, chap. TOSCA}: Po, pp. 527–549. Springer, New York (2014)
3. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Morgan & Claypool Publishers (2012)
4. France, R., Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. In: Future of Software Engineering (FOSE '07). pp. 37–54. IEEE, Minneapolis, MN (May 2007)
5. Giner, P., Pelechano, V.: Test-Driven Development of Model Transformations. In: MDE Languages and Systems, pp. 748–752. Springer, Berlin (2009)
6. Guerra, E., de Lara, J., Kolovos, D.S., Paige, R.F., dos Santos, O.M.: Engineering model transformations with transML. Software & Systems Modeling 12(3), 555–577 (2013)
7. Iacob, M.E., Steen, M.W.A., Heerink, L.: Reusable Model Transformation Patterns. In: 2008 12th Enterprise Distributed Object Computing Conference Workshops. pp. 1–10. IEEE, Munich (Sep 2008)
8. Jin, L., Guisheng, Y.: Method of constructing model transformation rule based on reusable pattern. In: 2010 International Conference on Computer Application and System Modeling (ICCASM 2010). pp. 519–524. IEEE, Taiyuan (Oct 2010)
9. Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., Wimmer, M.: Model Transformation By-Example: A Survey of the First Wave. In: Düsterhöft, A., Klettke, M., Schewe, K.D. (eds.) Conceptual Modelling and Its Theoretical Foundations, pp. 197–215. Springer Berlin Heidelberg, Berlin (2012)
10. Lano, K., Kolahdouz-Rahimi, S., Poernomo, I.: Comparative Evaluation of Model Transformation Specification Approaches. International Journal of Software and Informatics 6(2), 233–269 (2012)
11. Seaman, C.B.: Qualitative Methods. In: Guide to Advanced Empirical Software Engineering, pp. 35–62. Springer London, London (2008)
12. Siikarla, M., Laitkorpi, M., Selonen, P., Systä, T.: Transformations Have to be Developed ReST Assured. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) Theory and Practice of Model Transformations, pp. 1–15. Springer, Berlin (2008)
13. Silva, G.C., Rose, L., Calinescu, R.: Cloud DSL: A Language for Supporting Cloud Portability by Describing Cloud Entities. In: 2014 CloudMDE Workshop. p. To be published. Valencia (2014)
14. Silva, G.C., Rose, L.M., Calinescu, R.: A Systematic Review of Cloud Lock-In Solutions. In: 2013 IEEE CloudCom. pp. 363–368. IEEE, Bristol (Dec 2013)
15. Silva, G.C., Rose, L.M., Calinescu, R.: Towards a Model-Driven Solution to the Vendor Lock-In Problem in Cloud Computing. In: 2013 IEEE CloudCom. pp. 711–716. IEEE, Bristol, UK (Dec 2013)
16. Sjoberg, D.I.K., Dyba, T., Jorgensen, M.: The Future of Empirical Methods in Software Engineering Research. In: FOSE '07. pp. 358–378. IEEE, Minneapolis, MN (May 2007)
17. Sommerville, I.: Software Engineering. Addison Wesley, Harlow, 8 edn. (2007)
18. Sun, Y., White, J., Gray, J.: Model Transformation by Demonstration. In: Model Driven Engineering Languages and Systems, pp. 712–726. Springer, Berlin (2009)
19. Varró, D.: Model Transformation by Example. In: Model Driven Engineering Languages and Systems, pp. 410–424. Springer, Berlin (2006)
20. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards Model Transformation Generation By-Example. In: HICSS'07. pp. 285–294. IEEE, Waikoloa (2007)

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.

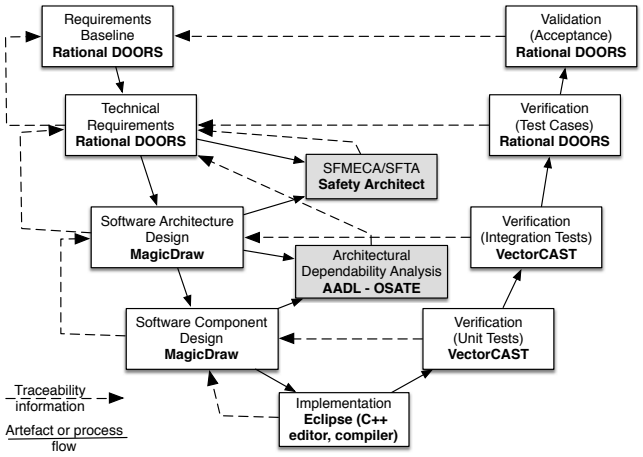

Fig. 1. Space Apps V-Model Software Development Process

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.

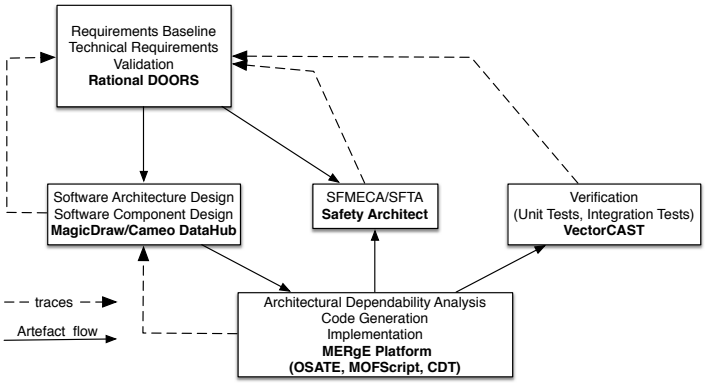

Fig. 2. Structural Software Development Process Toolchain

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)

Towards Enabling Cross-Organizational Modeling in Automotive Ecosystems

Eric Knauss[1] and Daniela Damian[2]

[1] Department of Computer Science and Engineering
Chalmers | University of Gothenburg, Sweden
eric.knauss@cse.gu.se
[2] Department of Computer Science
University of Victoria, Victoria BC, Canada
danielad@uvic.ca

Abstract. Automotive engineering is characterized by relying heavily on complex supplier networks as well as by strong dependence from hardware and software component development. OEMs (original equipment manufacturers) coordinate and integrate the work of hardware and software component suppliers and to an increasing amount develop application software themselves (component suppliers can be internal). For OEMs the transition to model-driven development promises potential reduction in the development lead-time of complex in-vehicle automotive features, such as semi-autonomous driving, but it is not without challenges. For example, the verification of such features is still performed using mainly physical properties such as hardware benches and mule vehicles. While this step is necessary, it is not sufficient, because it does not allow early verification of design decisions to the required extent. In addition, the development speed of hardware and software components is (a) limited by hardware development cycles as well as (b) slowed down by unsynchronized software development cycles of key suppliers. This prevents detailed information from being available early and potentially resulting in expensive and late changes. Understanding this situation as an ecosystem of cross-organizational collaborations allows us to reason about challenges and opportunities of the interaction between the OEM and different component- as well as tool-providers. In this paper, we report first results from an exploratory study that involved interviews with one of our industrial partners, General Motors (GM). First, we describe our understanding of the automotive ecosystem. Second, we explore interactions and roles of different ecosystem actors based on workshops and interviews with engineers at GM.

Keywords: automotive, cross-organizational modelling, software ecosystem

1 Introduction

Automotive engineering is characterized by the complexity of the system under construction as well as the required supply chain. In this environment, development is market driven. If the market motivates development of a new feature,

the automaker (OEM, original equipment manufacturer) starts with high level design of the feature, maps it to the overall system architecture, and derives hardware and software requirements, which are most often given to suppliers for implementation. Once the suppliers deliver hardware and software components, the OEM starts integrating them into a final product. The complexity of design artifacts and supplier networks lead to the need to coordinate and verify design decisions across organizational borders.

With few exceptions, OEMs address feature development in a way that resembles the waterfall process, characterized by upfront requirements analysis, early design decisions with limited knowledge, and by being able to verify the success of the development only later in the process after integration of the components into a mule vehicle (a complete, often drivable vehicle with experimental and prototype components). This situation can lead to sub-optimal decisions, which often result in late changes and rework. Lean software development considers this to be waste and suggests to change the process in a way that allows to make design decisions later, based on knowledge about the performance of a number of candidate solutions [14]. In the automotive domain, this would require the ability to verify non-functional properties (e.g. task performance) of such candidate solutions very early.

In this paper, we report first results from an exploratory study in which we aim at understanding coordination needs between actors in the automotive value chain by interpreting it as an ecosystem. Our contributions are (i) a characterization of the automotive ecosystem based on related work in software ecosystems, (ii) a first sketch of crucial interactions and roles of different ecosystem actors based on workshops and interviews with engineers at GM, and (iii) a discussion of challenges and opportunities of changing the ecosystem in a way that supports to make *binding decisions* (decisions that would be expensive to reverse such as contractual or architectural decisions, e.g. about a certain microcontroller) later and to do early non-functional verification across organizational borders earlier.

2 Research Questions and Method

This paper provides a first step towards describing automotive engineering as an automotive ecosystem of interacting organizations and is based on our collaboration with engineers and technical leaders at GM based on the following research questions:

RQ1: Who are the actors and what relationships exist among actors in the automotive ecosystem?

RQ2: What are examples of challenges and opportunities that currently exist in the automotive ecosystem?

To answer our research questions, we start by characterizing and defining the scope of the automotive ecosystem based on related work in the field of software ecosystems. We then follow up with a qualitative case study based upon worked done at GM R&D to further our understanding of such an ecosystem. At this

stage, our research is exploratory in nature and the preliminary results we present here are based on aggregating discussions during workshops and unstructured interviews. In future research, we want to engage with representatives of potential internal and external actors in the automotive ecosystem in semi-structured interviews, with the goal to elicit a clear understanding of actor relationships and their coordination needs as well as measurable objectives to measure and continuously monitor success, health, and gains of the automotive ecosystem.

3 Roles and Relationships of Automotive Ecosystem Actors

In this section, we explore relationships and interactions of actors in the automotive ecosystem based on literature on software ecosystem and interviews as well as workshops. We start with an example of how actors collaborate in automotive supplier networks and characterize this ecosystem based on related work in software ecosystems, before we report preliminary results from our ongoing interviews with practitioners on how these actors interact to create value. This is a first step towards a more formal assessment of the automotive ecosystem (e.g. based on [9]).

3.1 Characterization of the Automotive Ecosystem

In understanding ecosystems, one would draw on three fields in software engineering: open source software [15], modelling and architecture (e.g. software evolution, architecture, and product lines [2]), and managerial perspectives (e.g. business aspects and co-innovation [6]). Different strategies exist in software ecosystems, varying from widely proprietary ecosystems based on a semi-open partnership program to pure open source ecosystems [1,5] and literature discusses almost as many proprietary ecosystems as free-and-open-source ecosystems [12].

Previous research proposed [2] to characterize ecosystems based on their category (end-user programming, application, or operating system) and the scope of the ecosystem's operating environment (Desktop, Web, or Mobile). While we see application software and operating systems for embedded systems in the automotive ecosystem, it does not clearly fit into one of the suggested scope categories. Instead, we propose to see automotive components as cyber-physical systems to emphasize the increasing degree of connectivity between components.

Automotive engineering depends to a large degree on collaboration across a large supplier network, which generates a significant coordination need. It is characterized by the integration of different hardware and software components, thus it is touching on various levels in the ecosystem stack [4], including hardware, systems software, middleware, and application software. In a given electronic control unit (ECU) at least three components can be distinguished: (i) The hardware component, e.g. the microcontroller and peripherals, (ii) the middleware component, providing drivers, communication facilities, and other

enabling routines, and (iii) the application software component, which provides (parts) of functionality for user-facing features (e.g. semi-autonomous driving).

All of these components can be provided by different suppliers or be developed in house at the OEM, who is also responsible for integrating the different inputs. We would thus see the OEM in the role of an ecosystem coordinator and characterize it in terms of Jansen et al. [5] as privately owned and participation to be based on a list of screened actors. Actors of the ecosystem can be keystones (which have a huge impact on the ecosystem and provide room for niche players), niche players offering complementary services to what keystone players already provide, and the orchestrator/coordinator who is coordinating the ecosystem [6]. Those actors have various relationships among each other, including selling software, providing services, providing software and assets, endorse software, train consultants, and contribute to the artifacts produced by the ecosystem [13].

This integration requires to make binding decisions which typically, in automotive engineering, need to be made before the application problem is completely understood. Making such early binding decisions can lead to late changes (e.g. if the hardware is too weak to support a given algorithm), or to waste (e.g. if the hardware is more powerful and more expensive than required). Both problems can only be discovered late, during non-functional verification and validation through testing. Based on Farbey et al.'s classification of inter-organizational relationships, we classify the automotive ecosystem to operate on Rung 1 (market relationships with dominant focal firm) or, in some cases, where embryonic networking among actors occurs, as Rung 2 [3]. Other works by Manikas et al. and Schultis et al. about characterizing actor relationships in internal or non-FOSS (Free and Open Source Software) ecosystems [10,16] will allow a comparison of our data in future work. We assume that network analyzis as proposed by Manikas et al. [10] will reveal more dependencies and interrelations of actors than in their case study because of the integrated development efforts.

Consider for example a case, where the OEM is developing the application software component in house. R&D Engineers would develop a new control algorithm, using Simulink for model-driven development and simulation to allow functional testing, before hardware is in place. System designers would then decompose the Simulink models and generate software from Simulink blocks. By using for example *software-in-the-loop* based functional verification, this development is independent from the ECU hardware development, which can be started in parallel as soon as the hardware requirements are sufficiently understood. However, non-functional verification on the integration or system level can only be done once the hardware is developed and the execution time of the application tasks on the target hardware can be measured.

3.2 Actors and their Relationships in the Automotive Ecosystem.

To identify and discuss the coordination needs of the ecosystem actors, we analyze one typical scenario in the automotive domain that uses MDD (Model-Driven Development) to provide a situation where software can be developed

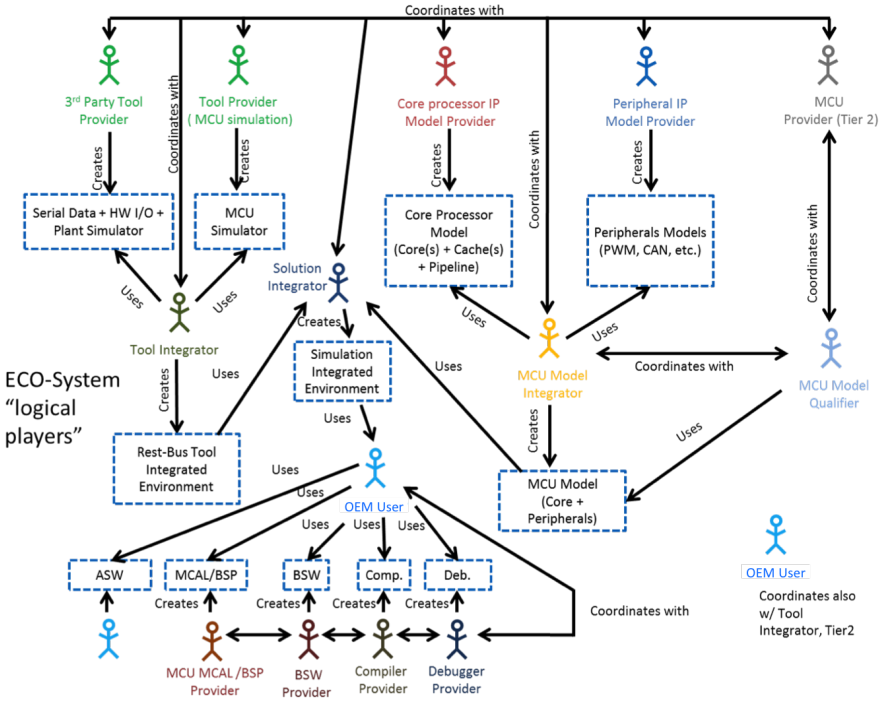

Fig. 1: Actors and relationships in a typical engineering example in the automotive ecosystem [17].

prior to the availability of MCU (Micro-Control Unit, part of the ECU) hardware. The same target code compiled for the actual MCU can be run on the simulated MCU at close to real time speeds and high degree of timing fidelity. Fig. 1 shows the actors and their relationships, as provided by one of our GM interviewees [17]. The figure emphasizes the strong coordination needs among actors and shows artifacts as well as services these actors exchange. The Figure shows logical roles of the actors, i.e. one organization could choose to fulfil several of the roles, e.g. both the model provider or the OEM user could also become the Model Qualifier for a given development project. We discuss these roles in the following.

The 3^{rd} *Party Tool Provider* contributes a model of the rest of the system, the plant model, which typically includes both a simulated controlled plant (e.g. an engine) or at least a simulation of the digital interface to it, as well as the simulated incoming messages from other MCUs in the system. This model allows the target code to interact with other parts of the system before these have been finished. Such models are usually exchanged using a third part tool format. For example, for an engine model in Simulink, a MDL file is exchanged, while for the network message model as Vector DBC file is included.

The *Tool Provider* contributes a standardised simulation environment (e.g. based on SystemC, a set of C++ classes that provide event-driven simulation) to (1) run the simulated MCU (2) execute the target code for the actual MCU on the simulated MCU.

The *Tool Integrator* combines both tools into the Rest-Bus Tool Integrated Environment, which allows OEM users to do Rest-Bus Simulation. Rest-Bus Simulation is a technique used to validate ECU functionality by simulating parts of an in-vehicle bus like the controller area network (CAN).

The *Core Processor IP Model Provider* provides a core processor model of the intended hardware (usually an MCU). The *Peripheral IP Model Provider* provides a model of the peripherals in which the core processor is embedded. The *MCU Provider* will provide the hardware (typically, this is done by an automotive Tier 2 supplier).

The *MCU Model Qualifier* ensures the requested level of accuracy of the model in representing the MCU hardware. Currently there is no formal definition of this role or default processes to qualify or certify the accuracy of models, yet this would be crucial to overcome challenges as discussed later in this paper.

The *MCU Model Integrator* uses Core Processor and Periphery Models to create a MCU Model. The *Solution integrator* integrates all models and tools into an integrated simulation environment.

The *OEM user* develops the application software component, which relies on the MCU hardware. It is important, that the development can start before the availability of hardware, but also that design decisions can be non-functionally verified early in the process. As development proceeds, uncertainty is reduced and more knowledge about constraints becomes available. Also, a higher accuracy of verification becomes necessary. While for later verification, a hardware board is crucial, the availability of a virtual board early can be an asset, if adequate accuracy is provided.

4 Opportunities and Challenges of GM's Automotive Ecosystem

Based upon the work done in GM R&D, our GM interviewees have stated that one of the biggest opportunities of the automotive ecosystem is to enable its actors to share accurate models of hardware, periphery, and middleware software *early and continuously* in order to facilitate later binding decisions and early non-functional verification. These models could then be partly refined as more knowledge becomes available. For this, development partners (e.g. OEM, Tier 1 and 2 suppliers) need synchronization points where they share their current level of knowledge. Examples of these include:

- OEM shares current version of Simulink prototypes and models with other actors in the ecosystem.
- Tier 1 regularly shares information about the task composition and the characterization of task execution time as it becomes available. This allows the OEM early simulation and non-functionally verification.

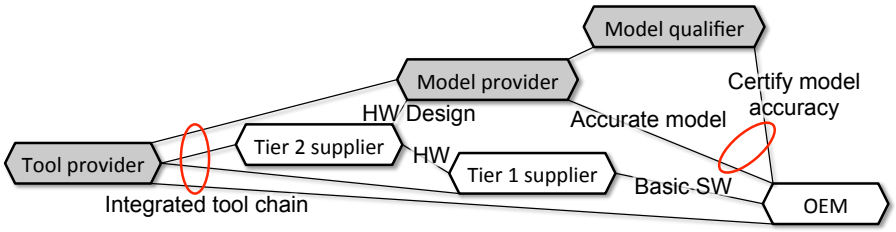

Fig. 2: Schematic view of abstract roles and their relationship in the automotive ecosystem. Actors to the left provide services and artifacts to actors on the right.

– Tier 2 shares information about the microcontroller design and its capabilities. This allows the Tier 1 supplier to estimate task execution times.

To increase the efficiency of this information exchange, our interviewees suggested to consider tool providers as part of the ecosystem, as they enable interoperability and exchange of relevant models. Also, while Tier 1 and 2 suppliers can be model providers, it could be preferable for some to rely on external modeling experts to create accurate models of their hardware in development.

From these observations, we extracted a more generic view on the automotive ecosystem (see Fig. 2). Basically, our study so far revealed two value chains relevant to automotive ecosystems. First, we identified the classic automotive value chains, where the OEM relies on delivery of HW and SW components by Tier 1 and 2 suppliers. Secondly, in order to support early non-functional testing (and consequently late design decisions based on accurate knowledge), a second value chain needs to be introduced to provide and certify accurate models of the HW in parallel to the main development stream (gray actors in Fig. 2). A Tier 2 Supplier might provide such models themselves or rely on an external Model Provider. Further, the accuracy of the models with respect of non-functional properties needs to be certified to create reliable models that allow testing *target-code-in-the-loop* (TCIL, a new simulation paradigm where application code compiled for the actual MCU runs on a simulated MCU).

Opportunities in the Automotive Ecosystem. The envisioned automotive ecosystem as sketched in Fig. 2 provides opportunities for win-win sscenarios, because actors are not competitors. Tier 2 suppliers for example would gain a competitive advantage if they can more easily integrate into such an ecosystem, e.g. by collaborating with model providers (or by taking that role themselves) in order to provide models of their hardware early and update them regularly so that their accuracy continuously grows until the hardware is completely developed. Tool providers will benefit from a larger market of their integrated tools, which enable the TCIL cycle. Tier 1 suppliers would benefit from the ability to run regression tests on virtual boards quickly.

Challenges in the Automotive Ecosystem. As of today, OEM users may decide against sophisticated models of hardware as they can be more expensive than a hardware evaluation board, especially when considering that currently there is no formal and independent certification of the accuracy of models. Potential time-saving opportunities are typically not exploited because non-functional verification can only start when the evaluation board becomes available.

Acceptance of modeling can depend on availability of certification processes. Our interviews identify the lack of certification as one of the main technical challenges. A clear process for the qualification of models needs to be in place and best practices as well as standard collaboration models need to be defined. By this, the MCU Model Qualifier actor in Fig. 1 appears as one of the key roles in the ecosystem to allow for transparent and reliable assessment of model accuracy.

Introduction of MDD might impact ecosystem health. To enable a healthy ecosystem, it should be avoided that a keystone player becomes a dominator [11]. This could e.g. happen, if a specific Tier 2 supplier would be the only supplier that offers models with sufficient quality, thus becoming a monopolist in the example.

Effective cross-organizational modeling depends on new solutions for legal concerns. Providing accurate HW models to other ecosystem partners early may require Tier 2 suppliers to define provisions to protect against potential disclosure of their intellectual property. While such openness is required to leverage the opportunities our interviewees see in the automotive ecosystem, adequate licenses and contract models need to be introduced to offer sufficient protection.

Cross-organizational modeling impacts local processes. To decrease development lead-time by offering early non-functional testing as well as later binding decisions, actors may need to adjust their internal processes, such as those in sales and purchasing to include provisions that cover models of IPs in the contracts.

5 Discussion and Outlook on Future Work

In this paper, we analyzed the automotive value chain and supplier network as an ecosystem and explored the extent to which this allows us understanding actor relationships (e.g. information flows), challenges (e.g. coordination needs) and potential for optimization. An accurate charting of the automotive ecosystem landscape requires more interviews with technical leaders at OEMs, Tier 1 and Tier 2 suppliers, tool providers, and potential model providers and qualifiers. We report now on this ongoing work to gather feedback on our intended approach to understand the automotive ecosystem.

The ecosystem perspective offers a unique chance to analyze actors and their relationships in the ecosystem and to uncover challenges as well as opportunities, as shown in the example of the TCIL case. In previous work, we found that navigating the tradeoffs between *protecting intellectual properties* $\longleftrightarrow$ *openness*

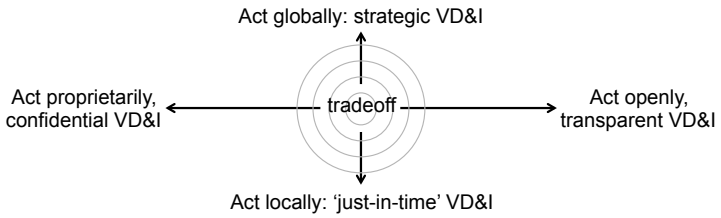

Fig. 3: Tradeoffs in software ecosystems, here with respect to VD&I (Virtual Development and Integration).

as well as between *global top-down approach* $\longleftrightarrow$ *Responsive bottom-up approach* generates opportunities for shared understanding of requirements between actors in an ecosystem [8]. In Fig. 3 we represent such tradeoffs with respect to the Virtual Development and Integration process and challenges and impediments for leveraging the true potential of the automotive ecosystem. For example, an increased information exchange among actors in the ecosystem benefits the overall ecosystem, but requires that the intellectual property of partners is protected. Enabling actors to make their own design decisions and to share them with relevant other actors increases the responsiveness of the ecosystem as well as the potential to allocate resources where they are needed most, but still a coordinator needs to guarantee that such engineering efforts lead to the intended performance of the integrated system and its user-facing features.

Future research should engage in a more systematic analysis of the actors' information and coordination needs as well as of how to optimize the information flow between them. This is a unique chance for the modeling community, which is called to provide mechanisms for efficient exchange of requirements, design decisions, and models between different organizations. We plan on furthering our discussions with more industrial partners in the automotive industry and broaden as well as validate our understanding of their ecosystem and its challenges, e.g. in the context of Autosar (www.autosar.org). Our long term research goal is propose and develop, in collaboration with industrial partners, solutions towards these challenges.

Acknowledgements

This work was sponsored by NECSIS and Software Center (an industry-academia partnership, hosted by Dept. of Computer Science and Engineering, Chalmers | University of Gothenburg). We would like to express our special gratitude and thanks to our contacts and interview partners at GM – this work would not have been possible without you!

References

1. van Angeren, J., Kabbedijk, J., Popp, K.M., Jansen, S.: Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry, chap. 5: Managing software ecosystems through partnering, 85–102. In: [7] (2012)
2. Bosch, J.: From Software Product Lines to Software Ecosystems. In: Proc. of Int'l Conf. on Softw. Product Lines (2009)
3. Farbey, B., Finkelstein, A.: Software acquisition: a business strategy analysis. In: Proceedings of Requirements Engineering (RE '01). pp. 76–83 (2001)
4. Gao, L.S., Iyer, B.: Analyzing complementarities using software stacks for software industry acquisitions. Journal of Management Information Systems 23(2), 119–147 (2006)
5. Jansen, S., Cusumano, M.: Defining software ecosystems: A survey of software platforms and business network governance. In: Int'l WS on SW Ecos. (2012)
6. Jansen, S., Brinkkemper, S., Finkelstein, A.: Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry, chap. 2: Business Network Management as a Survival Strategy, pp. 29–42. In: Jansen et al. [7] (2012)
7. Jansen, S., Cusumano, M.A., Brinkkemper, S. (eds.): Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry. Edward Elgar, Cheltenham, UK (2012)
8. Knauss, E., Damian, D., Knauss, A., Borici, A.: Openness and Requirements: Opportunities and Tradeoffs in Software Ecosystems. In: Proc. of 22nd Int'l Requirements Engineering Conf. (RE '14). Karlskrona, Sweden (2014)
9. Knauss, E., Hammouda, I.: EAM: Ecosystemability Assessment Method. In: Proc. of 22nd Int'l Requirements Engineering Conf. (RE '14). Karlskrona, Sweden (2014)
10. Manikas, K., Hansen, K.M.: Characterizing the danish telemedicine ecosystem: Making sense of actor relationships. In: Proc. of MEDES'13. pp. 211–218. Neumnster Abbey, Luxembourg (2013)
11. Manikas, K., Hansen, K.M.: Reviewing the Health of Software Ecosystems - A Conceptual Framework Proposal. In: Proc. of Int'l Wksp on Softw. Ecosys. pp. 33–44. Potsdam, Germany (2013)
12. Manikas, K., Hansen, K.M.: Software ecosystems: A systematic literature review. Systems and Software 86, 1294–1306 (2013)
13. Popp, K.M.: Definition of supplier relationships in software ecosystems as a basis for future research. Tech. rep. (2010), `http://www.drkarlpopp.com/resources/ICSOBSubmission2.pdf`
14. Poppendieck, M., Poppendieck, T.: Lean Software Development. Addison Wesley (2003)
15. Scacchi, W.: Understanding requirements for open source software. In: Proc. of Design Reqts. Wksp. pp. 467–494. Springer LNBIP 14 (2009)
16. Schultis, K.B., Elsner, C., Lohmann, D.: Architecture Challenges for Internal Software Ecosystems: A Large-Scale Industry Case Study. In: Proc. of 22nd ACM SIGSOFT Int'l Symp. on the Foundations of Software Engineering (FSE '14) (2014)
17. Yantchev, J., Serughetti, M., Lapides, L., Giusto, P.: Session ID #6P17I(Panel) - Intermediate, Simulation: Expert Insights Into Modelling Microcontrollers. Panel, Renesas DevCon (2012), `http://renesasdevcon.com/archives/course.html`, meet the expert, Session ID #6P17I