Tool-Support of Socio-Technical Coordination in the Context of Heterogeneous Modeling

A Research Statement and Associated Roadmap

Francis Bordeleau
Ecole de Technologie Superieur,
Universite du Quebec
Montreal, Canada
francis.bordeleau@etsmtl.ca

Benoit Combemale
University of Toulouse, CNRS IRIT
Toulouse, France
benoit.combemale@irit.fr

Romina Eramo
University of L'Aquila
L'Aquila, Italy
romina.eramo@univaq.it

Mark van den Brand
Eindhoven University of Technology
Eindhoven, The Netherlands
m.g.j.v.d.brand@tue.nl

Manuel Wimmer
TU Wien
Vienna, Austria
wimmer@big.tuwien.ac.at

CEUR-WS.org/Vol-2245/gemoc_paper_3.pdf

ABSTRACT

The growing complexity of everyday life systems (and devices) over the last decades has forced the industry to use and investigate different development techniques to manage the many different aspects of the systems. In this context, the use of model driven engineering (MDE) has emerged and is now common practice for many engineering disciplines. However, this comes with important challenges. As set of main challenges relates to the fact that different modeling techniques, languages, and tools are required to deal with the different system aspects, and that support is required to ensure consistence and coherence between the different models. This paper identifies a number of the challenges and paints a roadmap on how tooling can support a multi-model integrated way of working.

CCS CONCEPTS

• **Software and its engineering** → **Software notations and tools**;

KEYWORDS

Heterogenous modeling, model consistency, domain specific languages

1 INTRODUCTION

The fast growing, ever increasing, complexity of everyday life systems (and devices) over the last decades has forced the industry to use and investigate different development techniques to manage the many different aspects of the systems. In this context, model driven engineering (MDE) has emerged as an effective solution [10] as it allows leveraging abstraction and automation. Among other things, it provides automated transformation/generation techniques, which allow increasing productivity and reduce time to market, and analysis/validation/simulation techniques, which allow increasing system quality. The use of MDE has been steadily increasing and is now common practice for many engineering disciplines [13].

However, this comes with important challenges. A set of important challenges are related to the fact the development of the different aspects of a system require engineers from different disciplines, with different skills and expertise, and that these aspects are typically developed using different modeling techniques, languages, and tools. Using the automotive industry as an example, the development of cars has evolved in the last decades from pure mechanical engineering to multidisciplinary engineering where engineers from different domains (including software engineering, electrical engineering, safety engineering, and mechanical engineering) are involved. From an MDE perspective, tools like Matlab/Simulink are used to describe continuous behavior of a component or system, while modeling languages like UML or SysML are used to describe system architecture and discrete behavior (using state machines or activity diagrams), and other modeling techniques, languages and tools are used for other aspects. Over the years, the MDE community has placed the main focus on the development of techniques, languages and tools to support specific modelling context and domains, but little focus has been placed on the development of proper methodologies and tooling to support the development, maintenance, and evolution of a set of related models in the context of cross- or multi-disciplinary modeling [4]. Important issues include:

- How can we ensure coherence and consistency between models defined using different modeling languages (that are based on different metamodels)?
- How can we maintain coherence and consistency between models that are independently defined and evolved throughout the development process by different engineers and teams?
- How can we govern the evolution of the system and associated models?

To complement previous general discussions about the globalization of modeling languages[1] [3] or multilingual programming environments [12], we focus in this discussion paper on the importance of the tool-support for the socio-technical coordination within the development of complex software-intensive systems. Indeed, since languages are pivot in between the various engineers and the technical artefacts they have to build with, we envision that the tool-support of the relationships between those languages helps both to support the seamless coordination of the stakeholders, and to automate the coordination of the technical artefacts.

[1]See [3] for a comprehensive description of the related Grand Challenge of the *Globalization of Modeling Languages*.

In the following section, we motivate the language-support of both stakeholders and technical artefacts built during the development process of complex systems. Then, in Section 3, we look ahead and provide a research and technology roadmap to meet the requirements.

2 LAYERING IN HETEROGENEOUS MODELING

The excavation of domain concepts depends on the engineers working in different technological spaces when developing (complex) systems. The dependencies and relations between the models used by these engineers are defined by the relations between the domain concepts at the domain (i.e., metamodel) level. The model driven technology space is defined by the architectural layering of M0 through M3. We will take a slightly different viewpoint. We are not interested in the level M3 (meta-metamodel) level, but are interested in the organisational relations or the relations created via workflow dependencies in product development.

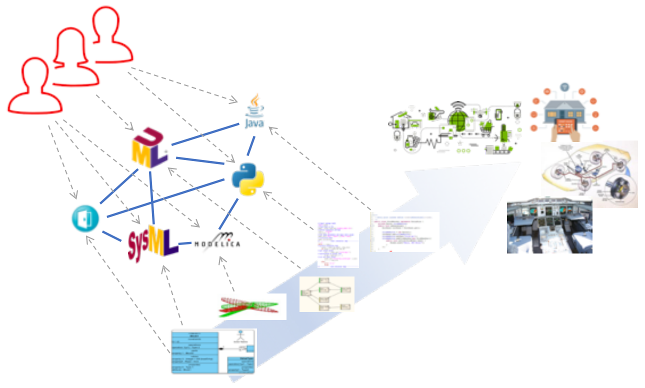

Figure 1: Language-Oriented Socio-Technical Coordination

To answer the questions mentioned in Section 1, we propose to focus in this paper about the language and the tooling issues. As a first step, we need to focus on the definition of the overall domain model that defines the top level system concepts independently of the specific modeling techniques, languages and tools that will be used in the development process. At this stage the goal is to define the overarching conceptual framework that will govern the development of the system. As Melvin Conway [2] had observed, how organizations were structured would have a strong impact on any systems they created. In this task the (system) architect plays an important role, the architect has the overall view on the project and the technical and non-technical dependencies between the domain engineering disciplines. The excavation of domain concepts needs to be done with care and involves a close investigation whether the definition of domain concepts used in different disciplines can be mapped onto each other. The identification of common domain concepts is necessary in order to establish relations and dependencies on the model level and to be administrated and maintained via tooling. Adapting existing domain specific tooling is costly; an alternative is to develop generic tooling that communicates with existing tools, via APIs, repositories, etc., to administrate and maintain the relations and dependencies. We envision that the relations and dependencies on the model level represent the organisation structure and/or the workflow of product development.

As a second step, we can proceed with the definition of the role of the different modeling techniques, languages and tools that will be used to model the different aspects of the systems and the definition of the semantic relationships between the different models and model elements. The identification of relations can be done in two directions (see Fig. 1), from models upwards and from people downwards. The optimal results is obtained when both directions are taken into consideration.

2.1 From Models to People

We restrict ourselves to the relationships between models across disciplines. The relationships within models and across models within a specific discipline are often maintained via the current tooling used by the domain engineers.

The construction of inter-relations among models from different disciplines can only be done if the models use clearly related domain concepts. If both domains use the same name for a concept then this can be a strong indication that the objects adhering to this concept are related. Of course, a sanity check on the semantic level is still required. The corresponding metamodels should use the same name as well. These common concepts have to be acknowledged by the domain engineers or the (system) architect.

2.2 From People to Models

Objects that adhere to domain concepts with implicit relationships are much harder to establish. There could be a temporal dependency based on the fact that objects are created, updated or deleted more or less in sync, but advanced repository mining techniques are needed for this. These implicit relationships have to be made explicit on the level of the domain engineers and (system) architects. Engineers and architects have to map the domain concepts and have to administer the relationships explicitly. These relationships have to be established in the supporting tooling and maintained during the development, maintenance and destruction of the models.

This type of functionality is typically tool agnostic. We can not expect that existing modeling environments offer this type of functionality out-of-the-box. Overarching tooling is needed to deal with multidisciplinary models and their relationships.

3 ROADMAP

For realizing overarching tooling for multidisciplinary modeling projects, several challenges have to be tackled. First, more explicit connection points are needed for models which in turn allow for a more systematic interlinking and integration of models. Second, rich linking support is needed which is capable to work across discipline boundaries. Third, governance for evolving linked models is required to ensure effective and efficient engineering processes.

3.1 Model Interfaces

Current approaches to connect models assume to define links potentially between every model element available in the models. While this allows for general solutions to connect models, more pragmatic solutions are needed for large-scale projects. The interface(s) of a

[2]Conway's law: https://en.wikipedia.org/wiki/Conway%27s_law

model to other discipline(s) have to be explicitly defined in order to coordinate work without having to explore and interpret full models. First work going in this direction is emerging [11] where the interaction points between metamodels and models are explicated. However, a strong requirement to be further explored in the future is how much control can be enforced on the existing tools to manage such interfaces on both, the provider and customer side. Finally, model interfaces must support IP management, and correspond to the integration concerns for supporting functional chains.

3.2 Model Linking

Traceability relationships may help stakeholders to understand the associations and dependencies that exist among heterogeneous models and their correspondences [8, 14]. In MDE, the definition of traceability focuses on models as the primary artifacts and refers to traceability (or trace) links: in particular, a *trace link* is a relationship between one or more source model elements and one or more target model elements, whereas a *trace model* is a structured set of trace links, e.g., between source and target models. Trace links may be defined between entire artefacts (e.g., a requirements document and a design model) or between parts of artefacts. We propose the use of trace links as the basis for defining and informing related stakeholders about correspondences between models. Furthermore, trace links incorporate consistency relations between the connected models. As it is difficult to keep a software system consistent at all times, tools need to have different policies for consistency enforcement. Thus, traceability may support consistency restoration policies. In MDE, this can be implemented by updating unidirectional transformations and synchronizing bidirectional transformations, and it can be also supported by means of constraints [2, 7]. Among the existing approaches that work with correspondences, it is worth mentioning multi-view modeling [9], that supports distinct views on a shared model, and megamodeling [1], that proposes a technique for managing dependencies among models in a heterogeneous system.

The framework we are outlining aims at supporting stakeholders that need to coordinate their activities without impacting their own current way to work. Typically, when designers work with heterogeneous models, they considered employing different modeling languages that are tailored to specific system aspects or a specific application area, i.e., domain specific languages (DSLs) [5]. Stakeholders coordination does not require to know any languages that are used, but rather the concepts that are involved in their activities. To this end, we need to create an external infrastructure to relate heterogeneous models in different tools. If heterogeneous software artifacts, tools, stakeholders are to interoperate effectively, they must have a common understanding of each other's information structures, which requires a common language (i.e., a *common metamodel* for traceability) describing how common concepts are related one with the other.

3.3 Model Governance

Besides the technical aspects discussed in the previous sections, the development and evolution of a set of related models also require the establishment of a clear governance that defines who is responsible for updating the different models and how model changes must be managed to ensure coherence and consistency between the models. This requires a tooling infrastructure that provides a cross-model change impact capability that accounts for the maintenance and evolution of the trace links. This means, that trace links have to be versioned and that they may need meta-information on current validity (invalid trace links may point out inconsistencies between the models) and who is maintaining the links [6]. Based on this meta-information, a life-cycle model for inconsistencies may be established. One may tolerate, ignore, or resolve them, but awareness is needed all the time as inconsistencies have to live with the evolving models and co-evolve.

4 CONCLUSION

In this paper, we elaborated on our vision about the language-support of the socio-technical coordination required in the development of modern complex software-intensive systems. More specifically, we first discuss the need for explicit relationships between the domain models of the various languages involved. Then, we argue for more focus in the community on tool-support for the coordination at the levels of language users and the actual heterogeneous models. For such a purpose, we derive a roadmap of concrete and actionable challenges to be further addressed by the community.

We hope this paper raises interesting discussions, and initiates and federates various related research activities. Collections of benchmarks and case studies are also required for further evaluating and positioning the different approaches and intents.

REFERENCES

[1] M. Barbero, F. Jouault, and J. Bézivin. 2008. Model Driven Management of Complex Systems: Implementing the Macroscope's Vision. In *IEEE Int. Conference and Workshop on Engineering of Computer Based Systems (ECBS 2008).* 277–286.

[2] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. [n. d.]. JTL: A Bidirectional and Change Propagating Transformation Language. In *International Conference on Software Language Engineering (SLE).*

[3] B. Combemale, B.H.C. Cheng, R.B. France, J.M. Jézéquel, and year=2015 B. Rumpe, series=Lecture Notes in Computer Science. [n. d.]. *Globalizing Domain-Specific Languages: International Dagstuhl Seminar.*

[4] B. Combemale, J. Deantoni, B. Baudry, R.B. France, J.M. Jézéquel, and J. Gray. 2014. Globalizing Modeling Languages. *Computer* (2014), 10–13.

[5] A. Van Deursen, P. Klint, and J. Visser. 2000. Domain-specific languages: An annotated bibliography. 35, 6 (2000), 26–36.

[6] S. Feldmann, M. Wimmer, K. Kernschmidt, and B. Vogel-Heuser. 2016. A comprehensive approach for managing inter-model inconsistencies in automated production systems engineering. In *IEEE International Conference on Automation Science and Engineering (CASE).* 1120–1127.

[7] S. Hidaka, M. Tisi, J. Cabot, and Z. Hu. 2016. Feature-based classification of bidirectional transformation approaches. *Software & Systems Modeling* 15, 3 (2016), 907–928.

[8] R.F. Paige, N. Drivalos, D.S. Kolovos, K.J. Fernandes, C. Power, G.K. Olsen, and S. Zschaler. 2011. Rigorous identification and encoding of trace-links in model-driven engineering. *Software and System Modeling* 10, 4 (2011), 469–487.

[9] J.E. Rivera, J.R. Romero, and A. Vallecillo. 2008. Behavior, Time and Viewpoint Consistency: Three Challenges for MDE. In *Models in Software Engineering, Workshops and Symposia at MODELS 2008. Reports and Revised Selected Papers.* 60–65.

[10] D.C. Schmidt. 2006. Guest Editor's Introduction: Model-Driven Engineering. *Computer* 39, 2 (2006), 25–31.

[11] D. Strüber, S. Jurack, T. Schäfer, S. Schulz, and G. Taentzer. 2016. Managing Model and Meta-Model Components with Export and Import Interfaces. In *Workshop on Scalable Model Driven Engineering at STAF 2016.* 31–36.

[12] T. van der Storm and J.J. Vinju. 2013. Towards Multilingual Programming Environments. *Science of Computer Programming* (2013).

[13] J. Whittle, J. Hutchinson, and M. Rouncefield. 2014. The State of Practice in Model-Driven Engineering. *Software, IEEE* 31, 3 (2014), 79–85.

[14] S. Winkler and J. Pilgrim. [n. d.]. A Survey of Traceability in Requirements Engineering and Model-driven Development. 9, 4 ([n. d.]), 529–565.