Some Narrow and Broad Challenges in MDD

Martin Gogolla, Frank Hilken, and Andreas Kästner

University of Bremen, Germany
{gogolla|fhilken|andreask}@informatik.uni-bremen.de

Abstract. This contribution describes a number of challenges in the context of Model-Driven Development for systems and software. The context of the work are formal descriptions in terms of UML and OCL. One focus point is on making such formal models more approachable to standard developers.

1 Introduction

Model-driven development (MDD) is regarded today as a promising approach to system and software design, based on the idea that expressive, abstract models and not concrete code is in the focus of the deployment process. The terms 'narrow' and 'broad challenges' refer to our view that for bringing the MDD vision into practice, short term and long term goals have to be considered.

As shown in Fig. 1, in our view on the development process we distinguish between development artifacts (in the left lane) and property artifacts for quality assurance (in the right lane). Our work concentrates on formal UML and OCL models and offers on the basis of the design tool USE [5,7] various quality assurance approaches based on testing [11], validation [6] and verification [8] for structural and behavioral [3,10] aspects.

2 Challenges for Model and Transformation Properties

Inspired by [1] this contribution is designed to formulate some ideas for possible research in Model-Driven Development (MDD). Fig. 1 shows our view for structuring and arranging challenges in MDD. The left lane sketches a (traditional) waterfall process (with feedback) using *core development artifacts* (e.g., models, code): Starting from a high-level, descriptive model various model transformations lead to an efficiently realized program of the initial model. The right lane emphasizes the role of *property artifacts* (e.g., validation scenarios, proofs, tests) that quality check the core development artifacts. The middle lane puts emphasis on involving human developers stressing the need for *human-oriented techniques* in the development process.

As there are several kinds of models (e.g., descriptive or prescriptive ones), different properties will be of interest. For example: (a) global properties valid in the complete model or local properties for model parts must be separated; (b) invariants and contracts have to be checked against implementations.

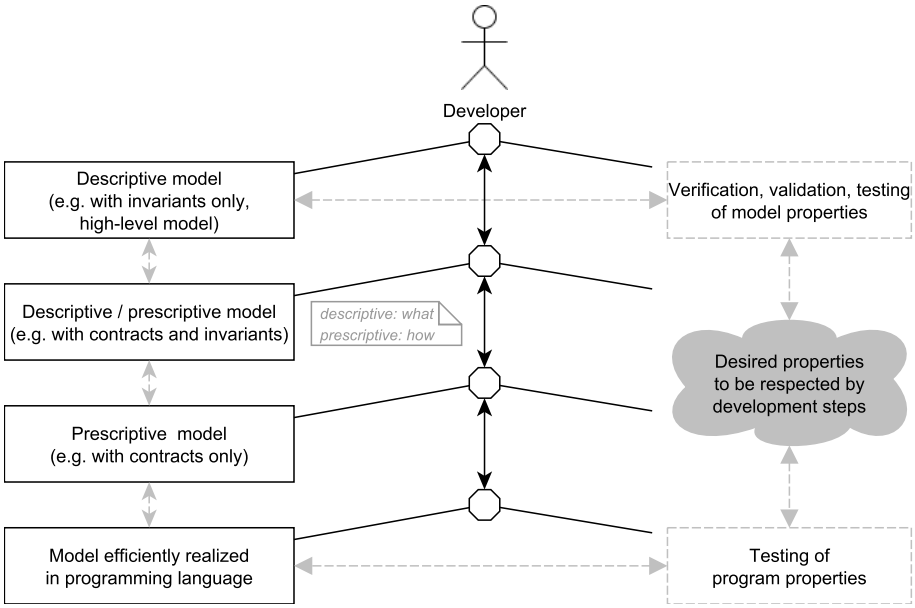

Fig. 1. Context for MDD challenges.

This implies finding and fixing appropriate techniques in the development process. For example: (a) global invariants have to be transformed into local contracts; (b) conceptual modeling features (as e.g., associations) must be turned into programming language like features (e.g., class fields); (c) platform-independent features must be specialized into platform-dependent features; (d) large models must be split into manageable small model slices; (e) generally, descriptive high-level features must be transformed into efficient low-level features.

We suggest the handling of models and model transformations with lightweight model finders and model provers. Different proving machineries have been suggested and are already employed for checking model qualities, e.g., relational logic, rewriting, description logics, logic programming, SAT, or SMT. These different approaches have all their own advantages, allow to inspect different model qualities and can coexist in the MDD world. A general strategy for the formulation of properties (of models and transformations) in an approach-independent way is however still missing. For example, we have previously developed rather particular model-to-model transformations in order to map (a) behavioral properties to efficient structural properties (filmstripping) [12], (b) multi-level models into two-level models [4], (c) complex model features into simpler ones (e.g., transformation of composition and aggregation into class diagrams with constraints) [13], or (d) linear temporal logic into UML and OCL for validation and verification purposes [10].

A continuously high priority in the field of model finders is to increase the performance in order to keep up with the ever increasing comlexity of systems

and their models. New techniques for the validation and verification of (partial) models are still coming up regularly [2,16]. However, none of the existing approaches to date has a full coverage of the modeling elements and it is difficult to find – or choose – the right verification engine. A major part of the problem is the lack of benchmarks for these verification engines to compare feature sets and performance of existing tools. Without common criteria that can be compared it is difficult to judge the effectiveness of the approaches in comparison to each other. Such benchmark needs to be flexible enough to account for tools that can only handle specific validations or only support a restricted set of modeling elements.

Further challenges are located at the more detailed levels. They include the handling of (a) arithmetic or non-classical logics, (b) more data collections like tuples, (c) advanced behavioral features like state machines.

Finally, there exists no tool that can handle most types of modeling paradigms to be considered suitable for everyday use in most situations. The solution to multiple modeling problems is often scattered among multiple approaches with different tools. There exists no integration between these tools and the paradigms have to be covored individually, meaning that every paradigm has to be solved with a completely different approach with ever changing details, i.e. required artifacts and their usage.

3 Challenges for the Development Process

Formal techniques as the ones advocated by us are usually machine-oriented, not human-oriented descriptions. We see a need to make formal techniques more approachable to everyday developers and to allow for a development style that mixes formal and informal techniques.

Taking Bran Selic's slogan "Objects before classes" seriously, we want to offer the option to start modeling with objects and to create classes based on objects. Object diagrams are less abstract than class diagrams, they represent a specific moment of time in a system. When the whole system is not known during the start of the development process, it might be easier to model such a specific moment. A class is an abstract concept, objects are more familiar, they can represent a real entity and are easier to grasp. When the goal is to model a whole system, a single object diagram will probably not be enough. But it can be used as a starting point to make educated guesses and transform it into a first version of a class diagram. Similarly, developing formal behavioral models from behavioral scenarios remains a challenging task.

To explore the possibilities of starting modeling with objects, we developed a plugin for the design tool USE. As shown in Fig. 2, it is possible to create objects without corresponding classes and links without corresponding associations, an approach that shows many similarities to partial models [15]. Most parts can be missing, to allow for more freedom in creating the object diagram. In the given input example, some role names are missing. The second diagram then shows the result of the transformation from objects to classes. Looking at the

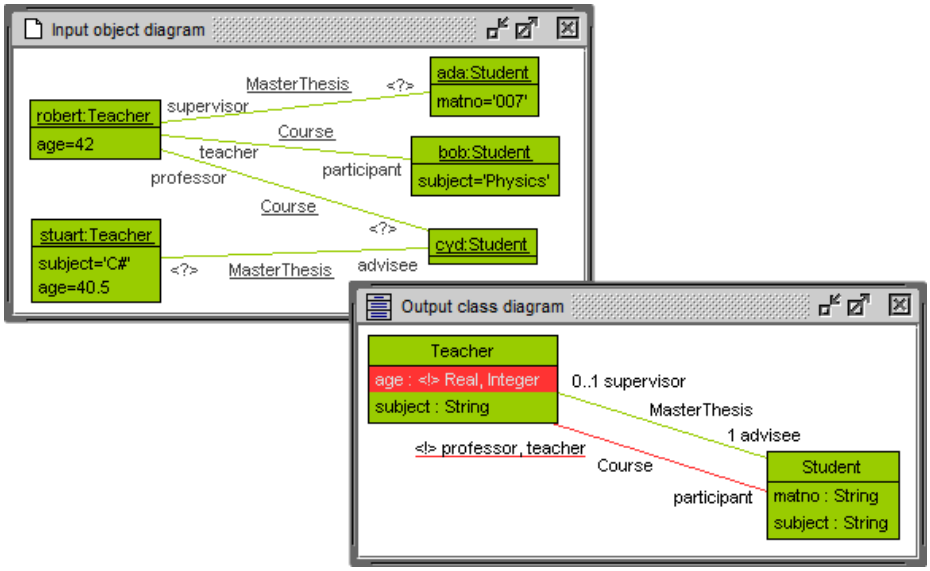

Fig. 2. Transformation example: Object diagram to class diagram

`MasterThesis` association, the links in the object diagram are merged and a completely labeled association is the result. The `Course` association however, results in a conflict, because different role names are used for the same end in the source diagram. Now that it is highlighted in the output diagram, it can be fixed in the input diagram as a next step in the iterative modeling process. Another conflict is shown in the `age` attribute of the `Teacher` class. The plugin detects different types of the attribute, which again gets highlighted. The highlighting is done using an informal notation, utilizing color and symbols. With the help of the color, the problems can be easily found, even in bigger diagrams. The symbols then highlight the specific problem. In this case, the exclamation mark highlights a conflict and the question mark highlights missing information.

This prototypical version of the plugin can already be used to get acceptable results. However, it is planned to expand the functionality. The multiplicities for example are currently given directly based on the exemplary object diagram, which is of course wrong most of the time. Instead, it is planned to make the iterative creation process more interactive and allow the user to directly input the wanted multiplicities. User input should be stored for all further iterations and only be replaced by new user input. Another future task includes the order of attributes in the classes. A concept has to be developed, how the order of the object-attributes can somehow be preserved, even though different objects might have different orders. Another limitation of the current version is the handling of attribute types. Right now, only four different types are allowed, this needs to be expanded. Also up to further discussion is the conflict between the specific types Real and Integer, like in Fig. 2. It might be better to merge the types instead.

Further ideas for extensions include the implementation of generalization, higher order associations, composition and aggregation.

Apart from considering structural aspects in taking object diagrams as the basis for the design of class diagrams, the same principle 'from concrete to universal descriptions' can be considered for behavioral aspects. Instance-level sequence diagrams and object diagram sequences can be taken as the starting point for behavioral descriptions like pre- and postconditions, protocol state machines or operation implementations, as this has been done to a certain extent already in [9,14].

4 Conclusion

This contribution has shortly discussed some narrow and broad challenges for model-driven development. Our focus is and will be on formal system descriptions, however we believe that much work has to be done in order to make the many existing methods and tools approachable to software and system developers who do not have expertise on formal approaches.

References

1. Assmann, U., Bézivin, J., Paige, R., Rumpe, B., Schmidt, D., eds.: Perspectives Workshop: Model Engineering of Complex Systems (MECS). Dagstuhl Seminar Proceedings 08331 (2008)
2. Dania, C., Clavel, M.: Ocl2msfol: A mapping to many-sorted first-order logic for efficiently checking the satisfiability of ocl constraints. In: International Conference on Model Driven Engineering Languages and Systems. MODELS'16, ACM (2016) 65–75
3. Doan, K.H., Gogolla, M., Hilken, F.: Towards a Developer-Oriented Process for Verifying Behavioral Properties in UML and OCL Models. In Milazzo, P., Varro, D., Wimmer, M., eds.: Proc. STAF 2016 Workshops, Workshop Human-Oriented Formal Methods, Springer, LNCS 9946 (2016) 207–220
4. Gogolla, M.: Experimenting with Multi-Level Models in a Two-Level Modeling Tool. In Atkinson, C., Grossmann, G., Kühne, T., de Lara, J., eds.: Proc. 2nd Int. Workshop on Multi-Level Modelling (MULTI'2015), http://ceur-ws.org/Vol-1505/, CEUR Proceedings, Vol. 1505 (2015) 3–11
5. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming **69** (2007) 27–34
6. Gogolla, M., Hamann, L., Hilken, F., Kuhlmann, M., France, R.B.: From Application Models to Filmstrip Models: An Approach to Automatic Validation of Model Dynamics. In Fill, H., Karagiannis, D., Reimer, U., eds.: Proc. Modellierung (MODELLIERUNG'2014), GI, LNI 225 (2014) 273–288
7. Gogolla, M., Hilken, F.: Model Validation and Verification Options in a Contemporary UML and OCL Analysis Tool. In Oberweis, A., Reussner, R., eds.: Proc. Modellierung (MODELLIERUNG'2016), GI, LNI 254 (2016) 203–218

8. Gogolla, M., Hilken, F., Niemann, P., Wille, R.: Formulating Model Verification Tasks Prover-Independently as UML Diagrams. In Anjorin, A., Espinoza, H., eds.: Proc. 13th European Conf. Modeling Foundations and Application (ECMFA 2017), Springer, LNCS 10376 (2017) 232–247
9. Grønmo, R., Møller-Pedersen, B.: From UML 2 sequence diagrams to state machines by graph transformation. Journal of Object Technology **10** (2011) 8: 1–22
10. Hilken, F., Gogolla, M.: Verifying Linear Temporal Logic Properties in UML/OCL Class Diagrams Using Filmstripping. In Kitsos, P., ed.: Proc. Digital System Design (DSD'2016), IEEE (2016) 708–713
11. Hilken, F., Gogolla, M., Burgueno, L., Vallecillo, A.: Testing models and model transformations using classifying terms. Software and Systems Modeling (2016) DOI: 10.1007/s10270-016-0568-3, Online 2016-12-09.
12. Hilken, F., Hamann, L., Gogolla, M.: Transformation of UML and OCL Models into Filmstrip Models. In Ruscio, D.D., Varró, D., eds.: Proc. 7th Int. Conf. Model Transformation (ICMT 2014), Springer, LNCS 8568 (2014) 170–185
13. Hilken, F., Niemann, P., Gogolla, M., Wille, R.: From UML/OCL to Base Models: Transformation Concepts for Generic Validation and Verification. In Kolovos, D., Wimmer, M., eds.: Proc. 8th Int. Conf. Model Transformation (ICMT 2015), Springer, LNCS 9152 (2015) 1–17
14. Kaufmann, P., Kronegger, M., Pfandler, A., Seidl, M., Widl, M.: A sat-based debugging tool for state machines and sequence diagrams. In Combemale, B., Pearce, D.J., Barais, O., Vinju, J.J., eds.: Proc. 7th Int. Conf Software Language Engineering SLE, Springer, LNCS 8706 (2014) 21–40
15. Salay, R., Chechik, M., Famelis, M., Gorzny, J.: A methodology for verifying refinements of partial models. Journal of Object Technology **14**(3) (2015) 3:1–31
16. Semeráth, O., Varró, D.: Graph constraint evaluation over partial models by constraint rewriting. In Guerra, E., van den Brand, M., eds.: Theory and Practice of Model Transformation, ICMT. Volume 10374 of LNCS., Springer (2017) 138–154