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

September 29, 2014 – Valencia (Spain)

XM 2014 – Extreme Modeling Workshop Proceedings

Davide Di Ruscio, Juan de Lara, Alfonso Pierantonio (Eds.)

Editors' addresses:

Juan de Lara
Escuela Politécnica Superior
Departamento de Ingeniería Informática
Universidad Autónoma de Madrid (Spain)

Davide Di Ruscio
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica
Università degli Studi dell'Aquila (Italy)

Alfonso Pierantonio
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica
Università degli Studi dell'Aquila (Italy)

Preface

Increasingly, models are starting to become commonplace and Model-Driven Engineering is gaining acceptance in many domains, including

- Automotive Software Engineering
- Business applications and financial organizations
- Defense / aerodynamics / avionic systems
- Telecommunications domain

Raising the level of abstraction and using concepts closer to the problem and application domain rather than the solution and technical domain, requires models to be written with a certain agility. This is partly in contrast with MDE whose conformance relation is analogous to a very strong and static typing system in a current programming language. For instance EMF does not permit to enter models which are not conforming to a metamodel: on one hand it allows only valid models to be defined, on the other hand it makes the corresponding pragmatics more difficult. In this respect, there is a wide range of equally useful artefacts between the following extremes

- diagrams informally sketched on paper with a pencil
- models entered in a given format into a generic modeling platform, e.g., Ecore/EMF

At the moment MDE encompasses only the latter possibility, while depending on the stage of process it might make sense to start with something closer to the former to eventually end up with the latter. For instance, this clearly requires different notions of conformance and the possibility to even have a method for user-defined conformance relations depending on the scope. In other words, we do need different forms of agility in terms of both artefacts (the way they are conforming to metamodels) and processes (the way they are created and whose subsequent versions linked together in a consistent and uniform framework).

The third edition of the Extreme Modeling Workshop (http://www.di.univaq.it/XM2014/) has been co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages & Systems. It provided a forum for researchers and practitioners where to discuss different forms of agility as demonstrated by the technical program, including

- EMF modularity
- agile ways to assign semantics to graphical languages
- scalable modeling approaches
- agile development of model transformations

as well as empirical studies related to model-driven agile development.

Many people contributed to the success of XM 2014. We would like to truly acknowledge the work of all Program Committee members, and reviewers for the timely delivery of reviews and constructive discussions given the very tight review schedule. Finally, we would like to thank the authors, without them the workshop simply would not exist.

September 2014

Davide Di Ruscio, Juan de Lara, and Alfonso Pierantonio

Organizers

Davide Di Ruscio (co-chair)	Università degli Studi dell'Aquila (Italy)
Juan De Lara (co-chair)	Universidad Autonoma de Madrid (Spain)
Alfonso Pierantonio (co-chair)	Università degli Studi dell'Aquila (Italy)

Program Committee

Colin Atkinson	University of Mannheim (Germany)
Paolo Bottoni	Sapienza University of Rome(Italy)
Antonio Cicchetti	Maalardalen University (Sweden)
Tony Clark	Middlesex University (UK)
Jean-Marie Favre	University of Grenoble (France)
Cesar Gonzalez-Perez	Incipit CSIC (Spain)
Jeff Gray	University of Alabama (USA)
Robert Hirschfeld	Hasso-Plattner-Institut (Germany)
Gerti Kappel	Vienna University of Technology (Austria)
Philipp Kutter	Montages AG (Switzerland)
Stephen Mellor	Freeter (UK)
Mark Minas	Universität der Bundeswehr Mnchen (Germany)
Richard Paige	University of York (UK)
Jesus Sanchez Cuadrado	Universidad Autonoma de Madrid (Spain)
Bran Selic	Malina Software Corp. (Canada)
Jim Steel	University of Queensland (Australia)
Bernhard Rumpe	(Germany)
Antonio Vallecillo	Universidad de Malaga (Spain)
Vadim Zaytsev	Universiteit van Amsterdam (NL)

Table of Contents

Keynote

Putting Engineering into MDE: Components and contracts for models and transformations

Steffen Zschaler

King's College London

Models and model transformations are at the heart of MDE. To truly enable model-driven engineering at scale, we need to ensure that we have the right technology in place for creating, reasoning about, and maintaining models and model transformations. From other areas of software engineering—such as Component-Based Software Engineering—we can learn that two principles are key for scalable engineering: modularity for dividing big problems into smaller ones and a strong notion of contracts to enable independent development and modular reasoning. In this talk, I will explore some of the work done over the past years in developing notions of modularity and contracts for models and model transformations. I will argue that the overall research agenda needs to aim towards a theory of MDE, including a calculus of model management and sound but flexible notions of typing for models and transformations..

Steffen Zschaler *is a lecturer in software engineering at King's College London, UK. His research interests are in MDE, with a particular focus on modularity and reuse as well as the modelling and analysis of non-functional properties. Most recently, he has worked on composition of transformation-based DSL semantics and flexible notions of typing for model-management operations. He has published around 90 scientific publications and has co-founded two series of workshops: MiSE at ICSE and NfC at MODELS. He received his Dr. rer. nat. from Technische Universitt Dresden, Germany, in 2007.*

How MAD are we? Empirical Evidence for Model-driven Agile Development

Sebastian Hansson, Yu Zhao, and Håkan Burden

Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Sweden
gussebasha@student.gu.se, guszhaoyu@student.gu.se, burden@cse.gu.se

Abstract. Since the launch of the Agile Manifesto there has been numerous propositions in how to combine agile practices with Model-driven Development. Ideally the combination would give the benefits of agile – e.g. rapid response to changes and shorter lead times – with the promises of Model-driven development – such as high-level designs and automation. A commonality among the proposals is that they lack in empirical evaluation. Our contribution is a systematic literature review to find out to what experiences there are of Model-driven Agile Development, MAD, from an empirical context. Among our conslusions is that MAD is sitll an immature research area and that more experience reports from industry are needed before we can claim to have understood the possibilities and drawbacks of MAD.

Keywords: Systematic literature review, Agile practices, Model-driven development

1 Introduction

Model-driven Agile Development, MAD, which aims to combine the benefits of agile practices with the positives of Model-driven development, MDD, has been a dream for many years. Mellor et. al. [15, 14] discuss the possibilities of combining agile and MDA[16], proposing that many of the agile practices are just as suitable for MDA. For MDA, executable models is a key feature for successful MAD since they can replace code and use expressions closer to the language of the customer. The idea that executable models serve as a better communication media than code is supported by R. Gomes et al. [7] and Stahl and Völter [21]. In addition, the latter claim that software is developed faster since tedious, recurring implementation tasks are automated. Transformations will also ensure that changes in the problem formulation are consistently propagated through the solution much faster than in a code-centric context. Selic proposes the usage of heterogeneous models by combining high-level modeling languages with detail-level action languages, so that the benefits of MDD can be extended "*to the full*

development cycle, including its use in agile development techniques based on multiple iterations" [20]. Rumpe argues that MAD will be successful since the models will enable static analysis, rapid prototyping, code generation, automated tests, refactoring and transformation as well as documentation [19]. Kaim, et al. also argue that efficient model transformations are cruical if MAD is to be achieved [21]. All contributions have in common that there is no concrete empirical data backing the claims.

1.1 Research Topic

To see what empirical evidence there is for MAD we decided to conduct a systematic literature review [10]. The resulting contribution delivers an initial analysis from the collected publications in order to answer our research questions

RQ1: What is the state of the art of MAD from an empirical point of view?
RQ2: What is lacking in the empirical literature regarding MAD?

Our results tell us that MAD is still too immature to claim wide-spread success or a state of the art over another and that we need more reports on industrial experiences of MAD in order to close the current gap in the literature.

1.2 Overview

The rest of our contribution is structured as follows; in the next section we will explain our methodology, including the identification and analysis of relevant publications. In section 3 we present our findings based on the papers that passed both our inclusion and exclusion criteria as well as the quality criteria. Section 4 synthesises the findings in relation to the research questions while threats to validity are given in section 5. Finally, in section 6 we conclude and propose future work.

2 Collecting and Analysing the Publications

Following the guidelines presented by Kitchenham et al. [10] we have performed a systematic literature review in order to answer our two research questions.

2.1 Collecting the Relevant Publications

We used three different digital libraries – IEEE explore, ACM digital library and the SpringerLink library applying the search string ("agile" AND "model") to the listed titles. Publications explicitly mentioning MAD in an empirical setting where included while publications discussing the combination of agile development and MDD as a theoretical contribution were excluded. Only contributions published after 2001, after the Agile manifesto was launched, were included while those not written in English were excluded. If a publication was published in multiple forms we selected the most comprehensive one, meaning that journals

Table 1. Included papers mapped to bibliographic reference and publication venue – **J**ournal, **C**onference and **W**orkshop.

Paper Nr	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13
Bibl. ref.	[25]	[3]	[11]	[18]	[12]	[4]	[9]	[5]	[17]	[22]	[8]	[23]	[24]
Venue	J	C	C	J	C	C	C	C	C	W	C	C	J

had precedence over book chapters, chapters over conference preceedings which in turn had precedence over workshops. Only peer reviewed publications were considered.

The publications were then included or excluded in a three-step process. First, titles and abstracts were analysed according to our inclusion/exclusion criteria. A total of 291 publications were found – 86 publications originated from Springerlink, ACM Digital Library yielded 84 publications while IEEE Xplore contained 121 publications. Second, the introduction and conclusion sections of the included papers were studied to further refine the selection. This resulted in 78 included publications. Third, the criteria were applied to the full publications. After removing eleven duplicates and five contributions that were published at multiple venues we ended up with thirteen papers that mention empirically documented cases of MAD. We then went through all references in the thirteen included papers, without finding any secondary studies that matched our inclusion and exclusion criteria. The included publications are presented in Table 1 together with their respective bibliographic reference number and type of publication venue.

2.2 Quality Assessment of the Selected Publications

To ensure that we can get quality data from the selected papers we need sufficient contextual and methodological information [6]. We therefore defined seven quality criteria that represented 1) the aim of using MAD, 2) the strategy for achieving MAD, 3) which agile practices that were used, 4) which MDD practices that were used, 5) details regarding the team and the project, 6) in what kind of domain MAD was adopted, and 7) the impact in terms of MAD leading to success or failure. The outcome of applying the quality criteria to the selected papers is found in Table 2. Papers P4, P6, P7, P8, P10, and P12 fail to report sufficient information about how MAD was applied, making it difficult to draw parallels between the publications or synthesise empirical evidence of MAD. The publication with insufficent context will not be a part of the results.

3 Results

In this section we present our findings based on the seven publications that passed both the inclusion, exclusion and quality criteria. The focus in this section is on the strategies that were employed, the challenges in adopting MAD and

Table 2. Results of applying the quality criteria to the included publications

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13
Aim of MAD	X	X	X	X	X	X	X	X	X	X	X	X	X
MAD strategy	X	X	X	X	X	X	X	X	X	X	X	X	X
Agile practice	X	X	X	X	X	X	X	–	X	X	X	X	X
MDD practice	X	X	X	X	X	X	X	X	X	X	X	X	X
Team/project	X	X	X	–	X	–	–	–	X	–	X	–	X
Domain	X	X	X	X	X	X	X	X	X	X	X	X	X
Impact	X	X	X	X	X	X	–	–	X	X	X	X	X

the impact of MAD while an overview of the quality criteria for Aim, Agile and MDD practices, Team and Project as well as Domain is found in Table 3.

3.1 Overview

As seen in Table 3, publication P9 deploys Scrum, Model-Driven Architecture (MDA [16]) as well as feature and mockup models to improve the handling of variability in their accounting system. The actual implementaion was done by an existing web design team.

Publication P1 is interesting in two different comparisons; First, P1 and P13 have both applied MAD within the telecom industry – which is not surprising since Zhang is an author of both publications. However, the publications differ in their motivations for MAD as well as how MAD was implemented in terms of agile and MDD practices. The second comparison is between publications P1 and P3 which are similar in context – aim and chosen practices – but not in domain. From the publications that passed the three sets of inclusion, exclusion and quality criteria there is no common theme in how and why MAD was adapted in relation to contextual factors.

3.2 MAD Strategies

The strategy used to achieve agile model-driven development differs among the selected publications. P9 and P11 suggest to build mockup models, as a mean of communication and for easy requirement gathering and handling requirement changes. P11 promotes an agile way of building models, with specific tools to easily transform the models under development. P1 suggests using iterations for MDD and P5 wants sprints and iterations but to still have a high level design. P13's strategy is to have both agility and quality built into their development process and P3 suggests a modified agile method and presents a tailored approach to address the need of managing evolution using model-based techniques. P11 and P13 use a Test-driven development approach combined with modeling. Our findings tell us that the common trend among the reviewed papers is to include a more agile way into existing model-driven engineering practices. Especially because of agile development's advantages regarding rapid response to change

Table 3. The collected data for the quality criteria

	Aim	Agile practices	MDD practices	Team/Project	Domain
P1	Shorten lead time	Scrum, XP, Iterations or incremental development	UML, Code generation	New team, no prior experience of agile or MDD	Telecom
P2	Respond to change	Iterations or incremental development	UML	Changed over time	Legacy system
P3	Shorten lead time	Scrum, XP, Iterations or incremental development, Feature-driven	Feature models	New team	Database
P5	Demanded by domain	Scrum, XP	UML	Architect, domain expert, coders, tester and client	Web application
P9	Improve variability	Scrum	MDA, Feature models, Mockup models	Web design team	Accounting system
P11	Involve stakeholders	Scrum, XP, Test-driven development	Mockup models	10 developers	Customer satisfaction system
P13	Respond to change, Involve stakeholders, Improve productivity and quality	Test-driven development	UML	60 people divided into sub-teams	Telecom

and close stakeholder involvement – a claim of MDD but where the included publications see agile more successful.

The most interesting and comprehensive strategy we found is in P1. They use a methodology called System Level Agile Process (SLAP). SLAP is a Scrum-based agile methodology, constructed by Motorola that includes XP practices. In SLAP the software lifecycle is split into short iterations, where each iteration includes three sprints – requirements, architecture and development – and then system integration feature testing. They conclude that "*From MDD perspective the key to success is to maximize automation using the MDD tools chain to enable mistake-free (high-quality) development and significant productivity in-*

crease" and "*From the Agile perspective, the key is to efficiently achieve end-to-end iterations, from system engineering all the way to system testing. This requires streamlining different process activities such as system engineering, development, and testing*".

3.3 Challenges

Zhang et al. state in P1 that MAD "*is still relatively new in real software development. The learning curve is sharp for any new organization to adopt due to process, culture, methodology, and other related changes. Thus, adopting a new agile MDD process is not likely to produce a short-term benefit. But, for the long-term, it's ultimately worth it for large projects with multiple releases*". P12 also mentions the steep learning curve, this time for web modelling tools.

To provide lightweight-agility in development seems to be a solution for Kulkarni et al. who in P3 argues that traditional agile development is not suited for larger teams or projects.

3.4 Impact

When it comes to impact, publications P1, P2, P3 and P9 state that they successfully combined agile practices and MDD. P1 and P2 argue that the combination could be useful for future development teams, due to the fact that the learning curve is steep it is not likely to produce short-time benefits. P2 and P13 noticed an increase in commitment, quality and productivity from the developers – despite having different aims and strategies – while P11 report on improvements regarding both time and satisfaction. P5 state that their approach would not be effective on large projects while P13 states that their approach could be beneficial and possible for any-size projects. Still, the overall assessment is that the authors fail to provide detailed descriptions on what was successful and why.

4 Synthesis

The data drawn from seven publications can now be further refined into a synthesis answering the research questions that motivated our systematic literature review.

4.1 What is the State of the Art of MAD from an Empirical Point of View?

Based on our findings, we can conclude that the area around MAD is immature when it comes to empirical evidence and industrial experiences. To answer this research question, it is not enough to look at the theoretical part, we need evidence to prove any best practice or state of the art, though the evidence seems to be lacking. The strategies used among the papers we found is in many cases also contradictory to each other – providing information about different

domains, goals and strategies, making it difficult to claim any best practice or success over someone else. Though the most common goals seem to be the agile value of rapid response to change and contact to external stakeholders. Another commonality is that a majority of the included publications want to include agility into their MDD practices – and not extend agile practices with MDD features – a situation that is seen in other areas since agile is the trend in current software development.

Only when the area is more mature will it be possible to claim a state of the art regarding the combinations of agile practices and MDD. Though claiming that the area is not yet mature does not mean it is not possible to adapt. As Zhang and Patel say in P1, MAD requires investments in both learning and technology and the success will not be immediate. Other authors such as Lee et al. argue in P5 that a pure agile methodology or a pure MDD approach is not longer enough "*we can not apply agile processes to web application development directly. UML is not sufficient for modeling the navigation of Web applications*".

4.2 What is Lacking in the Empirical Literature Regarding MAD?

After removing articles based on our inclusion and exclusion criteria we only had thirteen papers left mentioning any concrete empirical evidence. This is a rather low value for an systematic literature review, which in turn suggests that the area is still immature. According to the publication venues, this conclusion is supported by the fact that there are only three articles published in journals, suggesting again that the more mature and empirical research contributions regarding MAD are still to be written. The search for secondary studies did not result in any additional publications that matched our inclusion, exclusion nor quality criteria – again suggesting that more research in the area is needed.

There are authors (e.g. Ambler [1, 2]) who discuss MAD from a more abstract empirical setting, drawing from their own experiences without giving specific details about a certain project or case and therefore lacking information how a certain company or a specific team adopted MAD into their existing context. Six out of the seven papers that passed our quality criteria also fail to deliver detailed information how they performed MAD. They barely mention their team-setup, what practices they combined and what tools they used. Only in P5 is the team described in detail with information about team members and roles.

Future empirical publications would contribute substantially to our understanding of MAD if they provided information about:

Teams/project: What was the size of the team? Which responsibilities did the team members have? Did the team members ahve any earlier experience of agile or MDD? For how long did the project run?

Used practices: Which agile and MDD practices were used?

Strategy: How were the agile and MDD practices combined? Were the practices easy to combine? Which where the challenges of introducing MAD?

Tools: Which tools were used to achieve MAD? Where specific tools for MAD develop or could off-the-shelf tools be used?

Impact: Was MAD a success or a failure? What succeeded/failed and why?

As Ambler himself indicates "*The use of Agile methodology in model-driven development is not prevalent yet, except tailored Agile approaches, such as Agile model driven development*" [1]. Reza Matinnejad comes to the conclusion that MAD "*is a promising research context and a great practical concept, but it is not mature enough and is still in its infancy*" [13].

5 Threats to Validity

Our results may face validity issues as our search was restricted to ACM digital library, Springerlink and IEEE explore. There are other digital libraries which are widely used in the software engineering field. However, these three libraries include the journals and conferences with the highest impact factor which should imply that they represent the most mature research. Another threat to the validity is our search string, as we did a title search on the words *agile* and *model*, it is possible that we have missed publications discussing the combination of agile and model-driven development while not explicitly mentioning the fact in their title. We tried to mitigate this by searching for secondary studies among the references in our primary studies but after applying our inclusion and exclusion criteria we could not find any additional publications. This gives us some confidence that we could not have missed many articles in our initial search. Our inclusion and exclusion criteria and our quality assessment can also be a threat to validity as we were searching for actual empirical cases where the authors mention certain key information that we as researchers see as empirical evidence, such as teams, project success, MDD and agile practices among others.

There are many other papers that discuss how and why you should combine the two development methods but they lack the key information, thus not being identified as relevant publications for our systematic literature review. The publications not passing our criteria might have important information regarding MAD, but could not be included in our data extraction and synthesis. Also, the publications that passed all three sets of criteria could still contribute data of poor quality as they might pass our criteria but fail to deliver detailed information.

6 Conclusions and Future Work

In this paper we have presented the results of a systematic literature review about empirical evidence on Model-Driven Agile Development. In the seven papers which passed all our quality criteria, the authors wanted to combine agile methods and MDD in a way that could draw benefits from both worlds and at the same time avoid their respective shortcomings. In the result section, we have presented different integrations of agile and MDD practices for various purposes. There are multiple authors discussing different theoretical approaches for MAD but just a handful publications describing detailed information how a company

or a team adopted such a practice into their existing context. Six out of the seven publications that passed our quality criteria also fail to deliver comprehensive information how they performed MAD. They barely mention their team-setup, what practices they combined and what tools they used.

Discovering that the empirical contributions regarding MAD are immature gives plenty of opportunities for future research, especially when it comes to detailed experience reports. The empirical experiences regarding MAD are still few and lacking details. We aim to extend this study by including more publication databases and by providing a more comprehensive analysis of the found publications. Such an anlsyis would also seek to map the theoretical proposals to the empirical evidence for MAD.

References

1. Ambler, S.: Agile/Lean Documentation: Strategies for Agile Software Development. www.agilemodeling.com/essays/agileDocumentation.htm, accessed June 19th 2013
2. Ambler, S.: Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. John Wiley & Sons, Inc., New York, NY, USA (2002)
3. Burkhardt, R., Gruhn, V.: Agile Software Engineering: A New System for an Expanding Business Model at SCHUFA. In: 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts and Applications for a Networked World. pp. 201–215. Lecture Notes in Computer Science, Springer Berlin Heidelberg, Erfurt, Germany (September 2004)
4. Cardoso, N., Rodrigues, P., Ribeiro, O., Cabral, J., Monteiro, J., Mendes, J., Tavares, A.: An agile software product line model-driven design environment for video surveillance systems. In: 2012 IEEE 17th Conference on Emerging Technologies Factory Automation (ETFA). pp. 1–8 (Sept 2012)
5. Djanatliev, A., Dulz, W., German, R., Schneider, V.: Veritas - a versatile modeling environment for test-driven agile simulation. In: Proceedings of the 2011 Winter Simulation Conference (WSC). pp. 3657–3666 (December 2011)
6. Dybå, T., Sjøberg, D.I.K., Cruzes, D.S.: What works for whom, where, when, and why?: on the role of context in empirical software engineering. In: Runeson, P., Höst, M., Mendes, E., Andrews, A.A., Harrison, R. (eds.) ESEM. pp. 19–28. ACM (2012)
7. Gomes, R., Rivera, G., Willrich, R., Lima, C., Courtiat, J.: A Loosely Coupled Integration Environment for Collaborative Applications. Systems and Humans, IEEE Transactions on Systems, Man and Cybernetics, Part A 41(5), 905–916 (September 2011)
8. Grigera, J., Rivero, J.M., Robles Luna, E., Giacosa, F., Rossi, G.: From Requirements to Web Applications in an Agile Model-Driven Approach. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012, 12th International Conference on Web Engineering, Lecture Notes in Computer Science, vol. 7387, pp. 200–214. Springer Berlin Heidelberg (2012)
9. Huang, Y.C., Chu, C.P.: Legacy System User Interface Reengineering Based on the Agile Model Driven Approach. In: Qian, Z., Cao, L., Su, W., Wang, T., Yang, H. (eds.) Recent Advances in Computer Science and Information Engineering, Lecture Notes in Electrical Engineering, vol. 125, pp. 309–314. Springer Berlin Heidelberg (2012)

10. Kitchenham, B., Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering. Tech. Rep. EBSE 2007-001, Keele University and Durham University Joint Report (2007)
11. Kulkarni, V., Barat, S., Ramteerthkar, U.: Early Experience with Agile Methodology in a Model-Driven Approach. In: MODELS 2011, 14th International Conference on Model-Driven Engineering Languages and Systems. pp. 578–590. Lecture Notes in Computer Science, Springer Berlin Heidelberg, Wellington, New Zealand (October 2011)
12. Lee, W., Park, S., Lee, K., Lee, C., Lee, B., Jung, W., Kim, T., Kim, H., Wu, C.: Agile development of Web application by supporting process execution and extended UML model. In: APSEC '05, 12th Asia-Pacific Software Engineering Conference (December 2005)
13. Matinnejad, R.: Agile Model Driven Development: An Intelligent Compromise. In: SERA 2011, 9th International Conference on Software Engineering Research, Management and Applications. pp. 197–202. IEEE Computer Society, Baltimore, MD, USA (2011)
14. Mellor, S.J., Balcer, M.: Executable UML: A Foundation for Model-Driven Architectures. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
15. Mellor, S.J., Kendall, S., Uhl, A., Weise, D.: MDA Distilled. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA (2004)
16. Miller, J., Mukerji, J.: MDA Guide Version 1.0.1. Tech. rep., Object Management Group (2003)
17. Pohjalainen, P.: Bottom-up Modeling for a Software Product Line: An Experience Report on Agile Modeling of Governmental Mobile Networks. In: 15th International Software Product Line Conference (SPLC). pp. 323–332 (August 2011)
18. Ramos, A., Ferreira, J., Barcelo, J.: LITHE: An Agile Methodology for Human-Centric Model-Based Systems Engineering. Systems, Man, and Cybernetics: Systems, IEEE Transactions on 43(3), 504–521 (May 2013)
19. Rumpe, B.: Agile modeling with the UML. In: Wirsing, M., Knapp, A., Balsamo, S. (eds.) Radical Innovations of Software and Systems Engineering in the Future, pp. 297–309. No. 2941 in Lecture Notes in Computer Science, Springer Berlin Heidelberg (Jan 2004)
20. Selic, B.: Model-Driven Development: Its Essence and Opportunities. In: ISORC'06 Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing. pp. 313–319. IEEE Computer Society, Gyeongju, South Korea (April 2006)
21. Stahl, T., Völter, M.: Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons Inc. (2005)
22. Urli, S., Blay-Fornarino, M., Collet, P., Mosser, S.: Using Composite Feature Models to Support Agile Software Product Line Evolution. In: Proceedings of the 6th International Workshop on Models and Evolution. pp. 21–26. ME'12, ACM (2012)
23. Xufeng, L., Marmaridis, I., Ginige, A.: Facilitating Agile Model Driven Development and End-User Development for EvolvingWeb-based Workflow Applications. In: ICEBE 2007, IEEE International Conference on e-Business Engineering. pp. 231–238 (October 2007)
24. Zhang, Y.: Test-driven modeling for model-driven development. Software, IEEE 21(5), 80–86 (September 2004)
25. Zhang, Y., Patel, S.: Agile Model-Driven Development in Practice. IEEE Software (2011)

Assigning Semantics to Graphical Concrete Syntaxes

Athanasios Zolotas, Dimitrios S. Kolovos,
Nicholas Matragkas and Richard F. Paige

Department of Computer Science
University of York, York, UK
Email: {amz502, dimitris.kolovos, nicholas.matragkas, richard.paige}@york.ac.uk

Abstract. Graphical editors that are used in the domain of Model-Driven Engineering (MDE) follow specific conventions to denote relations between elements such as edges and containments. However, existing research suggests that there are other visual aspects that can better encode these relations such as the shape, the position and colour of the elements. In this paper, we propose the use of these physical variables to denote information regarding attributes and relations between elements. Running examples of DSLs in which such paradigms can be of benefit are presented.

1 Introduction

In the majority of graphical model editors, nodes are used to represent different types and edges to denote relations between them. Some editors also allow the use of containments to group elements that conceptually belong to the same container while others allow the replacement of geometrical shapes with icons.

The importance of visual notation in diagrams and the impact on understanding them has been confirmed by different empirical studies [1], [2], [3]. Visual characteristics of the notation can help to better understand the modelled concepts while changes to them affect the final understanding, even if the semantics of the concepts have not changed. However, research on the design of graphical concrete syntaxes for modelling languages and Domain-Specific Languages (DSLs) in general suggests that diagram-based information related to the colour, size, location of model elements is ignored by the metamodel tools as the majority of effort is put on the semantics of the notation rather that the visual representation. [3]

In the emerging community of bottom-up and flexible modelling, where domain experts are invited to create example models of the envisioned DSL, the traditional graphical MDE conventions are not always easy to follow. For instance, in a DSL that will be used to graphically design the seating plan for an event, a domain expert will likely place each guest close to the table he/she belongs to. A person not familiar with the traditional MDE conventions is not likely to use edges to connect guests to the tables or place the guests on the

tables to express the notion of containment nor would they create an attribute for each guest to hold the ID of the table she belongs to. The former notation arguably looks more natural than the latter.

Even in cases where bottom-up modelling is not used or where users are familiar with the traditional MDE conventions, the use of such physical characteristics could be beneficial. In the same example above, it would be more natural to change the table of a guest by just moving her around different tables, rather than changing the value of the appropriate attribute, or by deleting an existing node and drawing another node to the new table.

In this paper we argue that some of the physical characteristics of diagram elements can be used to extract useful information about their underlying model elements. We present examples where such information is useful. In the examples we use a flexible modelling technique, called Muddles [4], to represent our models benefiting from its model querying capabilities to evaluate our claims.

The rest of this paper is structured as follows. In Section 2 related work in the field of notation design is discussed and bottom-up flexible modelling techniques are presented. Section 3 includes a brief presentation of the Muddles approach. In Section 4 we present an number of physical attributes and illustrate via running-examples how can assist in extracting useful information like the types of elements and relations between them. In Section 5 we conclude the paper and outline plans for future work.

2 Related Work

In [3], Moody proposes a set of rules that should be followed when creating graphical notation for a modelling language. He highlights the importance of the physics of notations in the development of DSLs and the fact that this is a neglected issue so far. The *theory of communication* by Shannon and Weaver [5], is adapted by Moody for the domain of graphical notations: the effectiveness of the communication of a diagram can be increased by choosing the most appropriate notation conventions of these that the human mind can process. In [6], Bertin identified a set of 8 visual variables that can encode and reveal information about the elements they represent or their relations in a graphical way. These variables are: the *horizontal position*, the *vertical position*, the *shape*, the *size*, the *colour*, the *value* (referred as "brightness" [3]), the *orientation* and the *texture*.

In [7], [8] the authors propose a set of metamodels that can be used in the classification of visual languages taking into account spatial information. In [9] the authors present a parsing technique that can incorporated into freehand editors and turn them into syntax-aware, using different criteria such as spatial relationships. Finally, Baar [10], proposes the formal definition of the concrete syntax of modelling languages.

In the field of bottom-up metamodelling, [11] proposes the use of example models to semi-automatically infer the metamodel. In [12], the example models created by domain experts using drawing tools can be used to construct the

metamodel. In [13], models that do not conform to their metamodel because the latter evolved, can be used to recover the metamodel they are instances of. Finally, in [4], users, using a simple drawing tool, define example models which are then amenable to programmatic model management (validation, transformation, etc.).

In this work we implement the examples using an extended version of the flexible modelling technique in [4]. The same process could be followed using any other editor for flexible or traditional metamodel-based modelling.

3 Muddles

In this section, we present the basic details that will help the reader understand how the Muddles approach [4] works. In addition, we present the work carried out to extend the Muddles to keep information about the visual properties of the models.

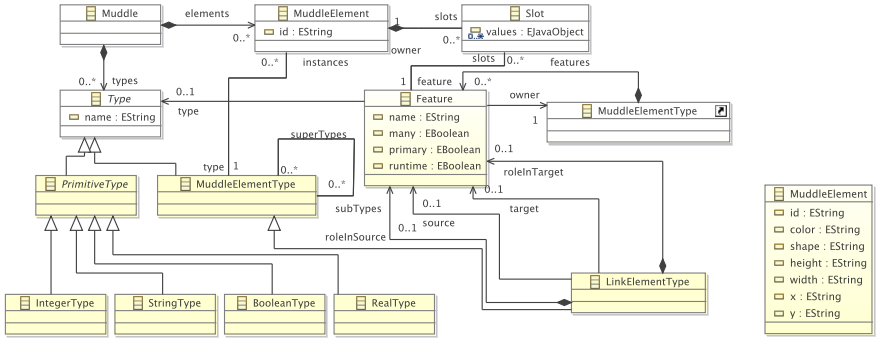
3.1 Overview

The Muddles approach [4] proposes the use of general drawing tools, for the construction of diagrams that are amenable to programmatic model management. More specifically, domain experts use a GraphML-compliant drawing tool (the yEd Editor[1] in their work) to express the example models, which conform to their envisioned metamodel. Engineers annotate these drawings to specify types and attributes for each element. The annotated diagram is then automatically transformed to an intermediate Muddle model (the Muddle metamodel is shown in Figure 1a). The Epsilon platform [14] provides an abstraction layer (the Epsilon Model Connectivity - EMC[2]) that allows access to models that conform to a range of technologies. A driver that implements EMC's interfaces and allows Epsilon to consume muddles was developed. Using the driver, model management programs (M2T transformations, Validation, etc.) can be written and executed on the muddles.

For a better understanding of the above process the authors in [4] provided an example which is presented here. In their example, the goal is to create a new flowchart-like language.

The process starts with the creation of a drawing of an example flowchart (see Figure 2). The next step is the annotation of the diagram elements with information to allow programmatic management. For instance, in this case one needs to declare that the type of the rectangles as an *Action* and the type of the directed edges as a *Transition*. The types are not bound with the shape but with each element (in another example one rectangle can be of type *Action* while another one can be of type *Process*). Types and type-related information like properties (attributes of the type), roles and multiplicity of edges can be provided using the fields in the yEd's custom properties dialog (see Figure 3). More details about these properties are presented in [4].

[1] `http://www.yworks.com/en/products_yed_about.html`
[2] `http://eclipse.org/epsilon/doc/emc/`



(a) The Muddle metamodel (b) Changes

Fig. 1.

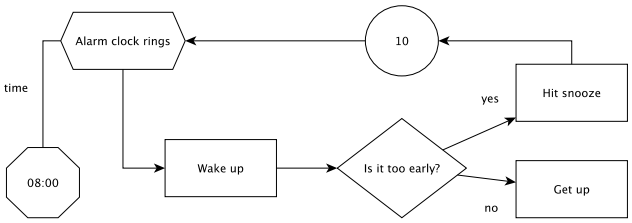


Fig. 2. An example diagram

This type-related information are keywords that will be used by the model management programs to elements of the diagram. For example, writing the following Epsilon Object Language (EOL) [15] script will return the names of all the elements of Type *Action*. (In this case, *name* was declared as a property of the *Action* node by writing *String name = "..."* in the *Properties* field of the node.)

```
var actions = Action.all();
for (a in actions) {
  ("Action: " + a.name).println();
}
```

Listing 1.1. EOL commands executed on the drawing

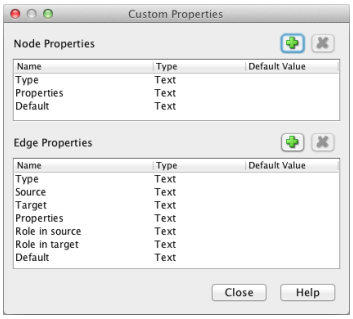


Fig. 3. Custom properties dialog

3.2 Extending Muddles

The current Muddles metamodel and the implementation of the EMC driver for muddles discard information regarding graphical and spatial properties of each element. These properties are the: x and y *coordinates*, the *width* and *height*, the *shape* and the *colour* of each Muddle Element.

Firstly, we extended the muddles metamodel to allow Muddle Elements hold the above information. The changes are shown in Figure 1b. The graphical and spatial properties of each element are now stored as attributes in the MuddleElement class.

Secondly, we implemented the required functionality in the EMC Muddles driver to be able to parse the GraphML file (the drawing), retrieve the information from it and store them in the muddle model instance. Further technical details about the new features of the EMC driver will not be discussed as they are beyond the scope of this paper.

4 Physical Attributes and Application Scenarios

In our running examples we demonstrate how 5 physical characteristics of the elements of graphical models can be used to extract relations and attributes. These are:

- Proximity: the distance between two or more elements.
- Colour: the colour of the element.
- Shape: the shape of the node.
- Size: the area of the node.
- Overlap: the intersection between two or more elements.

For each example, we implemented a set of functions to calculate the desired characteristic and executed it on the diagram using the querying capabilities of the Epsilon platform.

4.1 Proximity

The fact that an element is closer to another than a third one may infer that it is related to the former rather than the latter. In our scenario, we designed an example model of an envisioned DSL where the organiser of an event needs to assign guests to the tables and later perform model management actions on them (e.g. M2T transformations to generate invitation letters).

A possible example model could be the one shown in Figure 4 where 24 nodes of type *Guest* are assigned to 3 different nodes of type *Table*. Naturally, each guest belongs to the table that he is closest to. In a traditional graphical modelling editor, this relationship could be specified by creating an edge from each guest to the table it belongs, or by placing guests inside "table containments", or by assigning an attribute for each guest that declares his/her table. In our approach, this assignment is done by placing them closest to the table they belong to.

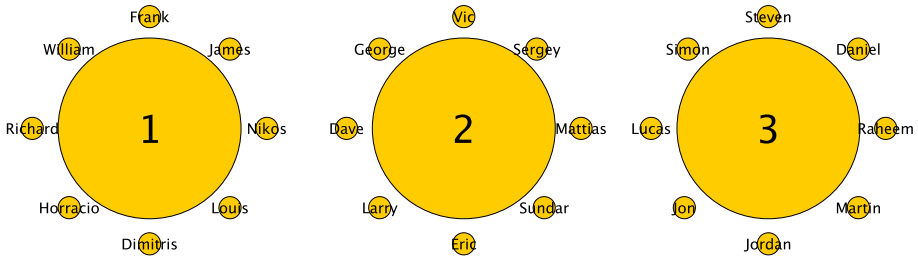


Fig. 4. Tables and Guests

To achieve this, an EOL function to calculate the proximity between two elements was implemented. Using this re-usable function (calculateProximity) we can query the models and get the relation of interest. The code for calculating and returning the closest table is illustrated in Listing 1.2.

```
function Guest getTable() {
  var minDistance = calculateProximity(getCircleCenterX(self), getCircleCenterX(
      tables.get(0)), getCircleCenterY(self), getCircleCenterY(tables.get(0)));
  theTable = tables.get(0);
  for (t in tables) {
    var candidateDistance = calculateProximity(getCircleCenterX(self),
        getCircleCenterX(t), getCircleCenterY(self), getCircleCenterY(t));
    if (candidateDistance < minDistance) {
      minDistance = candidateDistance; theTable = t;
    }
  }
  return theTable;
}
```

Listing 1.2. Get closest node of type Table

Indeed, if we query the model using the EOL statement of Listing 1.3 the correct table is returned.

```
var james = Guest.all.selectOne(p|p.name = "James");
("James belongs to table " + james.getTable().number + ".").println();
Output:
James belongs to table 1.
```

Listing 1.3. Query Guest's Table and output

We should note that the *proximity* characteristic may be error prone, as there might be cases that the user believes that a node is closer to the desired node while in reality it is closer to another.

4.2 Colour

In some cases, the colour of nodes or edges can declare that they belong to the same group or that they are of the same type.

In this scenario, we create an example model of an DSL that can be used to described football line-ups (see Figure 5). Each player belongs to a team illustrated by the colour of the node that represents each player. In a traditional MDE manner, this property could be defined in many different ways. Among others, one could use a string attribute for the name of the player's team or connect players of the same team with edges declaring a "team-mates" relation.

For this category, the function that returns the colour of the node is already implemented as part of our extended Muddles metamodel and driver (the extended Muddle metamodel stores the colour of the Element as an attribute - see

Fig. 5. Players and Teams

Figure 1(b)). We can create a mapping of colour with team and then query a Player using EOL to get his team. This is shown in Listing 1.4.

```
var teamsMap = new Map;
teamsMap.put("#FF6600", "Netherlands");
teamsMap.put("#333333", "Germany");
var vanPersie = Player.all.selectOne(f|f.name = "Van Persie");
("Van Persie plays for " + vanPersie.getTeam()).println();

function Player getTeam() {
  var color = self.getColor();
  return tMap.get(color);
}
Output:
Van Persie plays for Netherlands
```

Listing 1.4. Get Player's team implementation and output

4.3 Shape and Size

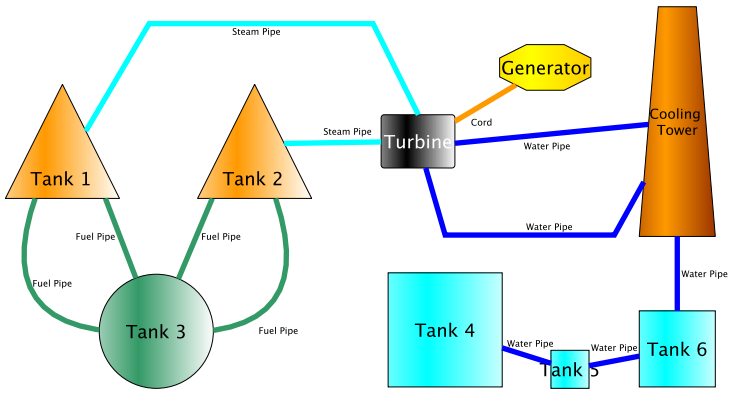


Fig. 6. Nuclear energy production

In some DSLs, the shape or the size of a node may encode information about its type or its attributes. We demonstrate that with an example DSL that can be used to design liquid tank configurations. In this scenario, the shape that is used to describe a tank, declares the subtype of the tank (Water, Uranium, etc.) By creating a mapping as in the previous example, we can query the model and identify the subtype of each tank.

In addition, the size (and the area) that each tank has can give us information about two other attributes of each tank like the "Size Category" and "Capacity". We can calculate the area of the tank to find its capacity and assign it to a predefined size category (small, medium, large). The querying code to get the type, the size category and the capacity is given in Listing 6.

```
...
for (t in tanks) {
  (t.name + " is a " + t.getSizeCategory() + " (" + t.getArea() + " litres) " + t.
      getTankType()).println();
}

function Tank getTankType() {
  return shapesMapping.get(self.getShape());
}

function Tank getSizeCategory() {
  if(self.getArea()<5000.0){
    return "Small";
  } else if (self.getArea()<20000.0) {
    return "Medium";
  } else {
    return "Large";
  }
}
Output:
Tank 4 is a Large (22500.0 litres) Water Tank
Tank 5 is a Small (2500.0 litres) Water Tank
Tank 1 is a Medium (11250.0 litres) Steam Tank
...
```

Listing 1.5. Get tank's type, size category and capacity implementation and output

The *size* characteristic can be error-prone. Mistakes can be made if the shape's area is close to the thresholds that defines different size categories. For instance, one tank may look like a small tank, but in reality it is medium.

4.4 Overlap

An overlap between two or more elements can provide us with information regarding their types and attributes. In a DSL that allows the creation of Venn diagrams this can be very useful. For instance, the Venn diagram of Figure 7 is an example of a model that would be an instance of a Venn DSL. In this case, the overlap between a node of type "Person" (yellow rectangles) with a circle denotes that the Person belongs to that set.

For this category, we can define a function to calculate whether two elements overlap or not. We can then re-use this function to query the model and receive, for instance the signatures of all the members of the department as seen in Listing 1.6.

```
var persons = Person.all;
var raBox = RA.all.first();
var rsBox = RS.all.first();
var esBox = ES.all.first();

for (p in persons) {
```

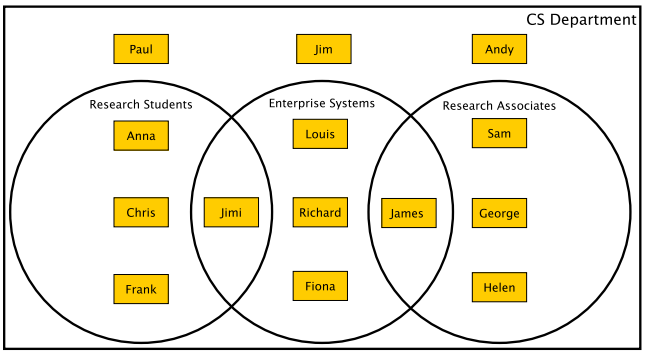


Fig. 7. Computer Science department Venn diagram

```
  p.getSignature().println();
}

function Person getSignature() {
  if((self.overlaps(raBox)) and (not (self.overlaps(esBox)))) {
    return self.name + " is a RA in the CS Department.";
  } else if ((self.overlaps(raBox)) and (self.overlaps(esBox))) {
    return self.name + " is a RA in the CS Department and member of the ES group.";
  } else if ((self.overlaps(rsBox)) and (self.overlaps(esBox))) {
    return self.name + " is a RS in the CS Department and member of the ES group.";
  } else if ((self.overlaps(rsBox)) and (not (self.overlaps(esBox)))) {
    return self.name + " is a RS in the CS Department.";
  } else if ((not(self.overlaps(rsBox))) and (not (self.overlaps(esBox))) and (not (
      self.overlaps(raBox)))) {
    return self.name + " is member of the CS Department.";
  } else if ((not(self.overlaps(rsBox))) and (self.overlaps(esBox)) and (not (self.
      overlaps(raBox)))) {
    return self.name + " is member of the CS Department and member of the ES group.";
  }
}
Output:
James is a RA in the CS Department and member of the ES group.
Andy is member of the CS Department.
Chris is a RS in the CS Department.
...
```

Listing 1.6. Get person's signature implementation and output

5 Conclusions and Future Work

Physical characteristics included in graphical models can be used to extract meaningful information about the models and their elements. In this work, we presented examples demonstrating how they can be utilised to extend the current conventions for representing relations and attributes of model elements.

We believe that such an approach can be useful especially in the flexible modelling area were the involvement of stakeholders who are unfamiliar with the traditional conventions is common.

In the future, we plan to investigate how other physical attributes (like texture or orientation) that can, according to the literature, encode information about the diagram be used in MDE to help us represent better relations and attributes of elements.

Acknowledgments

This work was carried out in cooperation with Digital Lightspeed Solutions Ltd, and was part supported by the Engineering and Physical Sciences Research Council (EPSRC) through the Large Scale Complex IT Systems (LSCITS) initiative, and by the EU, through the MONDO FP7 STREP project (#611125).

References

1. Nordbotten, J.C., Crosby, M.E.: The effect of graphic style on data model interpretation. Information Systems Journal **9**(2) (1999) 139–155
2. Hitchman, S.: The details of conceptual modelling notations are important-a comparison of relationship normative language. Communications of the Association for Information Systems **9**(1) (2002) 10
3. Moody, D.L.: The physics of notations: toward a scientific basis for constructing visual notations in software engineering. Software Engineering, IEEE Transactions on **35**(6) (2009) 756–779
4. Kolovos, D.S., Matragkas, N., Rodríguez, H.H., Paige, R.F.: Programmatic muddle management. XM 2013–Extreme Modeling Workshop (2013) 2
5. Shannon, C.E., Weaver, W.: The mathematical theory of communication. (2002)
6. Bertin, J.: Semiology of graphics: diagrams, networks, maps. (1983)
7. Bottoni, P., Grau, A.: A suite of metamodels as a basis for a classification of visual languages. In: Visual Languages and Human Centric Computing, 2004 IEEE Symposium on, IEEE (2004) 83–90
8. Bottoni, P., Costagliola, G.: On the definition of visual languages and their editors. In: Diagrammatic Representation and Inference. Springer (2002) 305–319
9. Costagliola, G., Deufemia, V., Polese, G., Risi, M.: Building syntax-aware editors for visual languages. Journal of Visual Languages & Computing **16**(6) (2005) 508–540
10. Baar, T.: Correctly defined concrete syntax for visual modeling languages. In: Model Driven Engineering Languages and Systems. Springer (2006) 111–125
11. Cho, H., Gray, J., Syriani, E.: Creating visual domain-specific modeling languages from end-user demonstration. In: Modeling in Software Engineering (MISE), 2012 ICSE Workshop on, IEEE (2012) 22–28
12. Sánchez-Cuadrado, J., De Lara, J., Guerra, E.: Bottom-up meta-modelling: An interactive approach. Springer (2012)
13. Javed, F., Mernik, M., Gray, J., Bryant, B.R.: Mars: A metamodel recovery system using grammar inference. Information and Software Technology **50**(9) (2008) 948–968
14. Paige, R.F., Kolovos, D.S., Rose, L.M., Drivalos, N., Polack, F.A.: The design of a conceptual framework and technical infrastructure for model management language engineering. In: Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on, IEEE (2009) 162–171
15. Kolovos, D.S., Paige, R.F., Polack, F.A.: The epsilon object language (eol). In: Model Driven Architecture–Foundations and Applications, Springer (2006) 128–142

EMF Splitter: A Structured Approach to EMF Modularity

Antonio Garmendia[1], Esther Guerra[1], Dimitrios S. Kolovos[2], and Juan de Lara[1]

[1] Modelling and Software Engineering Group (miso, htpp://www.miso.es)
Computer Science Department
Universidad Autónoma de Madrid (Spain)
{Antonio.Garmendia, Esther.Guerra, Juan.deLara}@uam.es
[2] Enterprise Systems Group (http://www.enterprise-systems.org/)
Computer Science Department
University of York (United Kingdom)
dimitris.kolovos@york.ac.uk

Abstract. Model-Driven Engineering aims at reducing the cost of system development by raising the level of abstraction at which developers work. Thus, models become the main assets in this paradigm, guiding the development until code for the final application is obtained.
However, even though domain-specific, models may become large and complex, becoming cumbersome to edit and manipulate. In this scenario, mechanisms helping in the agile definition and management of models in the large are crucial. Modularity is one of such mechanisms.
In this paper, we describe a novel approach to the construction of EMF models in a structured way. It is based on the annotation of the Ecore meta-models with modularity concepts (like project, package and unit), from which we generate an Eclipse plug-in that enables the editing of models according to that structure (i.e., organized in projects and decomposed into folders and files). The paper presents our supporting tool and discusses benefits and future challenges.

Keywords: Meta-modelling, Modularity, Agile modelling, Eclipse Modeling Framework

1 Introduction

Nowadays, many organisations use Model-Driven Engineering (MDE) [11] to develop their systems or migrate legacy code. MDE proposes the use of models as primary artefacts to construct software, being supported by model management tools [2].

MDE-based solutions frequently involve the creation of Domain-Specific Languages (DSLs), which are defined by a meta-model that describes the set of models considered valid. When a new DSL is created, there is the need to build the corresponding modelling environment as well, in order to facilitate the construction of valid models of the DSL and provide basic functionalities like model persistence or model consistency checking.

In this respect, the Eclipse Modeling Framework (EMF) [12] is a well-known and widely-used framework that allows the definition of meta-models and models. Starting from the definition of a meta-model, the framework generates a basic tree-editor to edit the instance models. However, models defined with these editors are monolithic. As the requirements of a system grow over time, the models tend to become complex and large, which results in poor comprehensibility and maintainability, while their editing becomes a tedious task. Moreover, the manipulation of large models may also affect performance in terms of model persistence, loading, querying and transformation.

Software design and programming languages provide mechanisms to simplify the creation of complex systems. One of these mechanisms is modularity [8], which promotes a scalable approach to the construction of software systems through the composition of smaller subsystems which can be implemented separately in a simpler way. Other benefits of modularity include increased flexibility and reuse possibilities, facilitating distributed teamwork and version control.

In MDE, models are the main assets to create software. However, models frequently lack native modularization mechanisms, unless they are explicitly encoded in the modelling language and implemented in the supporting modelling environments. Thus, we propose to apply modularity mechanisms to the construction of models, allowing the definition of complex models from submodels which are easier to process and reuse. Our approach is based on the annotation of the meta-model elements that will play some role in the structuring and modularity of the models. We propose several structures, based on the concepts of project, package and unit. Starting from this definition, we automatically generate a modelling environment (an Eclipse plug-in) which permits editing models in modular way, following a similar philosophy to the Java Development Tools[3] (JDT). In this way, each model corresponds to an Eclipse project, and the model content can be organized in folders and files, with a direct mapping to the file system. Altogether, our plug-in enables:

- the editing of models according to a given modularity structure.
- the splitting of monolithic models according to a given modularity strategy.
- the composition of parts of a model to build a monolithic one.

The remaining of this paper is organized as follows. First, Section 2 describes our approach to incorporate modularity mechanisms to models, as well as the modular structures we support. Then, Section 3 presents the implementation of our approach for Eclipse and EMF. Next, Section 4 discusses related work. Finally, Section 5 concludes the paper with some conclusions and lines of future work.

2 Specifying the Modular Structure of Models

We propose a modular structure for models, based on the philosophy of the Eclipse JDT and how Eclipse organizes Java projects. Eclipse projects are orga-

[3] `http://www.eclipse.org/jdt/`

nized hierarchically, defining a root node which contains a tree view of the project content. In this way, the compilation units of the project (e.g., Java classes) are organized into different types of folders (e.g., source folders) and packages, which can be nested. This modular organization facilitates the structuring of projects, the provision of scoping mechanisms, and the use of indexes to enable load-on-demand, incremental builds and efficient resolution of cross-references, among other advantages.

Inspired by this framework, we propose a notion of modularity for models based on the concepts of project, package and (compilation) unit. The upper frame in Figure 1 shows a simplified version of the pattern that formalizes these concepts, as well as their relations. Thus, the main modularity concepts in our approach are `Project`, `Package` and `Unit`. `Project` is the root that contains the rest of the elements, which can be of any of the other types. Objects of type `Package` can contain units as well as other packages (i.e., it implements the *Composite* object-oriented design pattern [3]). Finally, objects of type `Unit` can be defined either inside of packages, or directly inside a project.

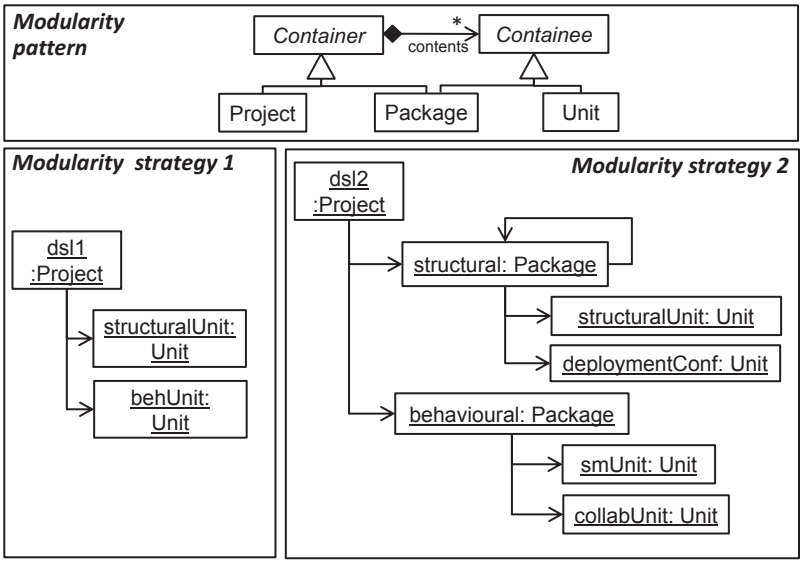


Fig. 1: Pattern to describe the modular structure of a meta-model (top). Two possible modularity strategies, as instances of the pattern (bottom).

This pattern allows for the configuration of different structures or ways of organizing a model. For instance, we can have projects that do not contain packages but units are directly placed in the project root node, projects that consider different types of packages containing different types of units and where package nesting might be allowed or not, or a mix of both. Conceptually, if we interpret the pattern in Figure 1 as a meta-model, then the possible structures

that can be applied to a particular DSL can be seen as instances of this meta-model, as shown to the bottom of the same figure.

While the modularity pattern allows configuring a particular modularity strategy, in addition, the strategy needs to be mapped to a particular meta-model. That is, we need to select the class playing the role of `Project`, and the classes for the different `Package`s and `Unit`s. Conceptually, it is natural to consider the application of the pattern to the meta-model of a DSL as a case of multi-level modelling [1], where the modularity pattern meta-model is interpreted as a partial type for the DSL meta-model, as Figure 2 shows. We refrain from introducing the full details of multi-level modelling here, which can be found in [1], and only present the information needed to understand how the pattern application works. The complete modularity pattern is shown in this figure. In particular, projects, packages and units have a descriptive `name`, as well as an `icon` which will be used in the modelling environment to identify them. On their side, units have an `extension` that will be used by the corresponding system files.

The particular structure to be used for the models of a DSL is determined by annotating which elements of the DSL meta-model will play the roles of project, package and root of a unit. Conceptually, this is equivalent to typing some elements of the DSL meta-model with the types offered by the modularity pattern. This typing is *partial*, because some elements of the DSL meta-model may be not typed by any pattern element. However, as shown in Figure 1, it is useful to consider the relation between the meta-models of the DSL and the pattern as a typing relation (and not just as annotations) because the typing rules ensure a correct annotation of the DSL meta-model.

The middle frame in Figure 2 shows the annotation (represented with dotted arrows) of an example meta-model with our modularity concepts. The annotation must respect the modular structure of the pattern, given by instantiation rules. In this example, the `ComponentSystem` class is annotated as `Project`, and the attribute `sysID` is bound to the project `name`. The `SubSystem` class is annotated as `Package`; this is possible because there is a composition relation from `ComponentSystem` (the project) to `SubSystem`, as the pattern demands by means of the relation `contents`. While this annotation permits creating `SubSystem` packages inside projects, it does not allow the nesting of subsystems inside subsystems, for which the meta-model would need to define an appropriate containment relation. Finally, both classes `Component` and `Behaviour` are annotated as `Unit` (i.e., we instantiate `Unit` twice). This means that the instances of `Component` and `Behaviour`, as well as the objects they contain through containment relationships (e.g., `Port` in the case of `Components`) will be stored in the same unit (i.e., in the same file).

A remark is interesting here. Some attributes in the modularity pattern, like the `icon`s to be used in the modelling environment or the file `extension` for units, need to receive a value in the DSL meta-model. Using multi-level terminology, we say that such attributes have potency 1, as they receive a value in the meta-level right below the pattern. Other attributes, like `name`, need to be bound to

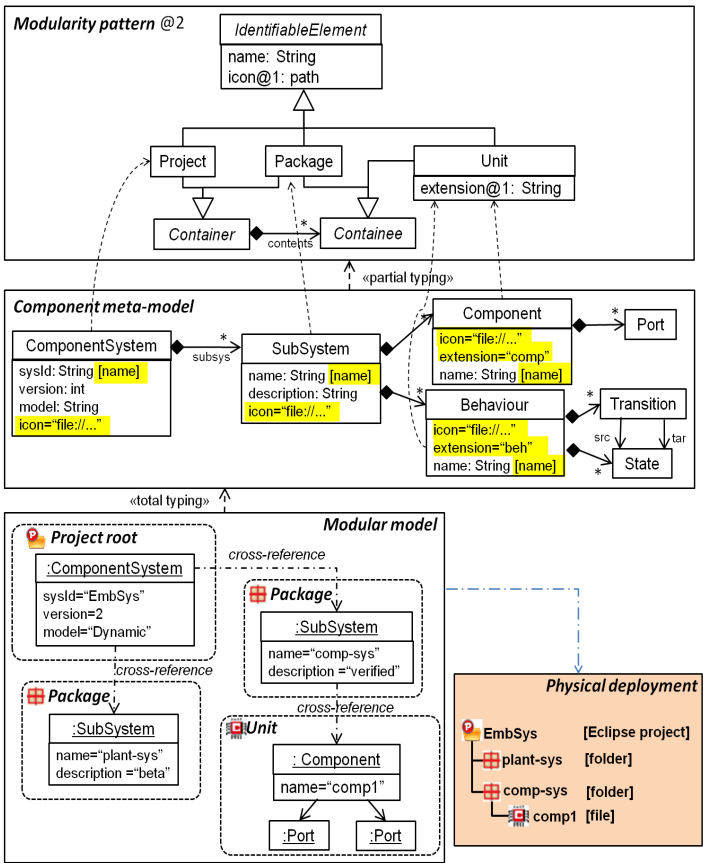


Fig. 2: Pattern to describe the modular structure of a meta-model (top). Application of pattern to a meta-model (middle). A structured model and its physical deployment (bottom).

some class attribute of the DSL meta-model, and only take a value at the model level (i.e., two levels below from the pattern point of view). Hence, we say that such attributes have potency 2. In the modularity pattern, every element has potency 2 (indicated by the `@2` of the meta-model), except attributes `icon` and `extension`, which have an explicit potency 1.

Once the DSL meta-model is annotated with the desired structure, an automated process generates the machinery to split existing monolithic models according to the chosen structure, and it also generates a modelling environment that permits building models according to that structure. The bottom of Figure 2 shows an illustration of a modular model, the corresponding Eclipse project structure, and how the modular structure is physically mapped to the file system. Hence, the model root (the instance of `ComponentSys`) is mapped to the project root, the two instances of `SubSystem` are mapped to two folders, and the

`Component` instance is mapped into a file. As discussed in the following section, hidden XMI (XML Metadata Interchange) files are also created for the project root and the packages, storing the properties of the corresponding objects, and allowing their manipulation by means of the *Property* view of Eclipse, so that the user has the impression of manipulating folders with different properties. Next, we explain the tool support implementing this modularity machinery.

3 Tool Support: EMF Splitter

This section describes our tool, called *EMF Splitter*[4], supporting the modular structure for EMF models proposed in the previous section. The main functionality of this tool is to, given an annotated Ecore model, generate an Eclipse plug-in that allows: (a) the creation of instance models according to the specified modularity strategy, (b) the decomposition of an existing monolithic model according to the modularity strategy, and (c) the composition of a single model out of a project consisting of folders and units. We have implemented this tool using Acceleo[5], a code generation language based on the Object Management Group (OMG) MOF Model to Text Language (MTL) standard.

Figure 3 shows the steps to create an Eclipse modular project for editing the instances of an Ecore meta-model. The first step is to annotate the meta-model classes with the concepts of project, package and unit. To facilitate this task, we have developed a graphical environment that allows building and annotating meta-models graphically according to different predefined patterns (in this case, according to our modularity pattern). In a second step, the annotated meta-model is fed into *EMF Splitter*. This tool automatically produces a *genmodel* configuration file, which is used to generate code implementing the chosen modularity strategy, and that is distributed as an Eclipse plug-in. Finally, tool users may use the generated plug-in to create models as dedicated Eclipse projects.

The generated plug-in provides functionalities to structure a model in several XMI files. This organization is transparent to the user, and can be useful for reusing parts of the models which can be nested. Each instance of a class annotated with project or package has an associated XMI file with the value of its attributes and cross-references to the objects it contains. These files are filtered and hidden to the user, who can edit the properties of such objects using the *Property* view of Eclipse when the corresponding folder or the root of the project is selected. Each instance of a class annotated as `unit` is stored in an XMI file, together with the objects it contains. These files can be modified through the generated standard tree editors.

Next, we present an example that makes use of the annotated meta-model from Section 2 and the corresponding plug-in generated using *EMF Splitter*. To show the benefits of the generated plug-in, we start from a synthetic, monolithic model which is shown in Figure 4(a). In practice, the model could be much larger, so that once decomposed into folders and smaller units, its parts would be easier

[4] See its web page at `http://www.miso.es/tools/EMFSplitter.html`

[5] `http://www.eclipse.org/acceleo/`

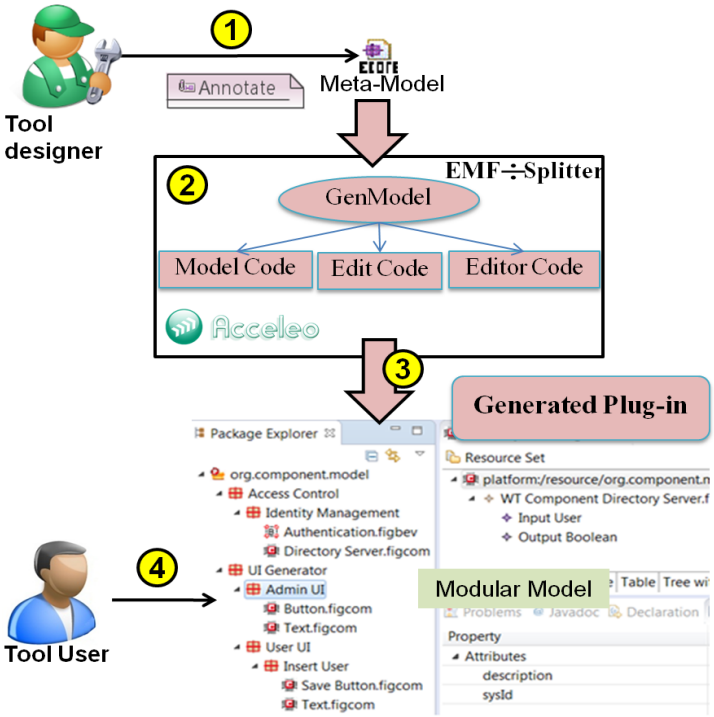
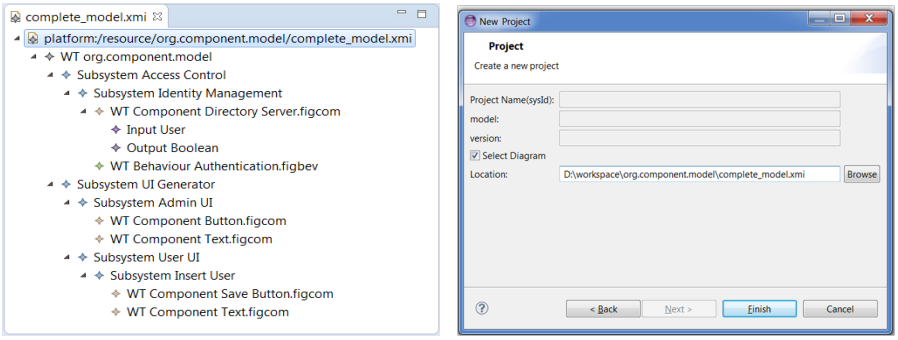


Fig. 3: Process overview of *EMF Splitter*.

to edit, navigate and comprehend. Thus, we first create a modular project using the plug-in. The plug-in offers a creation wizard with two alternatives, as shown in Figure 4(b). In the first one, we can create a model from scratch, giving values to the root class (`Project`) attributes. The wizard has input controls to introduce values for the attributes of the root object, which in this case is of type `ComponentSystem`. As the `sysId` attribute was mapped to the `name` of the project, the value introduced in this field is used as project name.

The second option of the wizard is to provide the path of an existing model. In that case, the wizard creates a new modular project, where the initial model is decomposed according to the selected modularization strategy. For the development of this example, we use the second option. As a result, we obtain the modelling environment shown in Figure 5. The *Package Explorer* view shows the structure of the containment hierarchy, made of all the objects that belong to the model, represented physically as folders and files within the project. In the top-right, a submodel (i.e., file) of the project is being edited using the tree-based editor. As we have explained previously, the root of the tree can be an instance of a class annotated with unit. At the bottom, the figure shows the edition of the attributes of a package using the *Property* view. While these attributes are actu-



(a) Example of model. (b) Wizard to create the Model Project.

Fig. 4: Generating a modular project from a model.

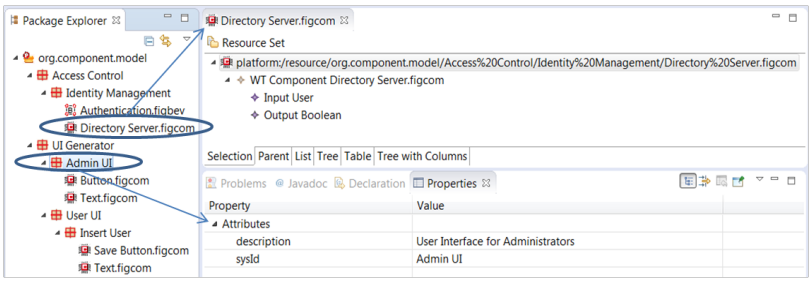


Fig. 5: Modelling environment in action.

ally stored in an XMI file, this is transparent to the user, who has the impression of editing properties of a folder.

The decomposition of a model into an Eclipse project offers many advantages. For example, we can include any kind of artefact (like documentation or source code) inside the folders, facilitating, e.g., the traceability from model elements to external artefacts. Moreover, we no longer have a monolithic model, but the division and organization into folders permits shorter loading times (of a fragment w.r.t. loading the complete monolithic model) and facilitates comprehensibility, reusability, distributed teamwork and version control.

4 Related Work

Our main goal is to provide a tool that allows building models in a structured way. For this purpose, we start by annotating the meta-model with the modularity concepts formalised in a pattern. Next, we compare with related works addressing model modularity, fragmentation and model slicing.

Although introducing modularity in ad-hoc ways into existing DSLs can deliver benefits, it is also costly [7]. Hence, it is desirable to have mechanisms to achieve modularity in a generic, automated way.

Due to the need to process large models, some authors have proposed to split models for solving different tasks. For instance, Scheidgen and Zubow [10] propose a persistence framework that allows automatic and transparent fragmentation to add, edit and update EMF models. This process is executed at runtime, with considerable performance gains. However, the user does not have a view of the different fragments as we have in *EMF Splitter*, which could help improving the comprehensibility of the fragments.

Other works [6, 13] decompose models into submodels for enhancing their comprehensibility. For example, in [6], the authors propose an algorithm to fragment a model into submodels (actually they can build a lattice of submodels), where each submodel is conformant to the original meta-model. The algorithm considers cardinality constraints but not general OCL constraints, and there is no tool support. Other works use Information Retrieval (IR) algorithms to split a model based on the relevance of its elements [13]. Therefore, splitting models that belong to the same meta-model can produce different structures.

Other works directed to define model composition mechanisms [4, 5, 14] are intrusive. These papers [4, 14] present techniques for model composition and realize the importance of modularity in models as a research topic to minimise the effort. Strüber et. al [5] present a structured process for model-driven distributed software development which is based on split, edit and merge models for code generation.

Hence, altogether, while techniques for model modularization have been proposed in the context of MDE, to the best of our knowledge, *EMF Splitter* is unique in its way to generate structured model editors from meta-models.

5 Conclusions and Future Work

The MDE paradigm is gradually being established for the production of software, giving rise to the problem that if the systems are complex, they may lead to large models, making their management more difficult and increasing the development costs. Our goal is to provide developers with tools that help in defining the way to structure models, facilitating distributed development through division into layers, which improve comprehension. To achieve these objectives, *EMF Splitter* permits defining a modularity strategy and generates an Eclipse plug-in that allows developers to build their models in a structured way, as well as split existing models according to the defined modularity strategy.

In the near future, we plan to assess the performance of the tool when handling large models, compared with using a monolithic model. We are working on the idea of using Concordance[6], an indexing mechanism to manage and reconcile EMF references when models are updated or deleted [9]. We are also working on heuristics to propose good modularization strategies for a meta-model and a set of (large) models. In the long term, we plan to generate more advanced editors, e.g., graphical ones, as an alternative to the tree-editors for the units. Moreover,

[6] `http://www.eclipse.org/epsilon/doc/concordance/`

we would like to enhance our modularity pattern, e.g., by including scoping and access rules, to allow enabling or disabling references between model elements in units that belong to different packages, or to define visibility rules for elements inside units.

Acknowledgements. Work supported by the Spanish Ministry of Economy and Competitivity with project Go-Lite (TIN2011-24139) and the EU commission with project MONDO (FP7-ICT-2013-10, #611125).

References

1. C. Atkinson and T. Kühne. Rearchitecting the UML infrastructure. *ACM Trans. Model. Comput. Simul.*, 12(4):290–321, 2002.
2. P. Baker, S. Loh, and F. Weil. Model-driven engineering in a large industrial context - Motorola case study. In *Proceedings of MoDELS'05*, volume 3713 of *Lecture Notes in Computer Science*, pages 476–491. Springer, 2005.
3. E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison Wesley, 1994.
4. F. Heidenreich, J. Henriksson, J. Johannes, and S. Zschaler. On language-independent model modularisation. *T. Aspect-Oriented Software Development VI*, 6:39–82, 2009.
5. P. Kelsen and Q. Ma. A modular model composition technique. In *Proceedings of FASE'10*, volume 6013 of *Lecture Notes in Computer Science*, pages 173–187. Springer, 2010.
6. P. Kelsen, Q. Ma, and C. Glodt. Models within models: Taming model complexity using the sub-model lattice. In *Proceedings of FASE'11*, volume 6603 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 2011.
7. J. L. Lawall, H. Duchesne, G. Muller, and A.-F. L. Meur. Bossa nova: Introducing modularity into the bossa domain-specific language. In *Proceedings of GPCE'05*, volume 3676 of *Lecture Notes in Computer Science*, pages 78–93. Springer, 2005.
8. D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.
9. L. M. Rose, D. S. Kolovos, N. Drivalos, J. R. Williams, R. F. Paige, F. A. C. Polack, and K. J. Fernandes. Concordance: A framework for managing model integrity. In *Proceedings of ECMFA'10*, volume 6138 of *Lecture Notes in Computer Science*, pages 245–260. Springer, 2010.
10. M. Scheidgen, A. Zubow, J. Fischer, and T. H. Kolbe. Automated and transparent model fragmentation for persisting large models. In *Proceedings of MoDELS'12*, volume 7590 of *Lecture Notes in Computer Science*, pages 102–118. Springer, 2012.
11. T. Stahl, M. Voelter, and K. Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management.* John Wiley & Sons, 2006.
12. D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework, 2^{nd} Edition.* Addison-Wesley Professional, 2008. See also `http://www.eclipse.org/modeling/emf/`.
13. D. Strüber, J. Rubin, G. Taentzer, and M. Chechik. Splitting models using information retrieval and model crawling techniques. In *Proceedings of FASE'14*, volume 8411 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2014.
14. D. Strüber, G. Taentzer, S. Jurack, and T. Schäfer. Towards a distributed modeling process based on composite models. In *Proceedings of FASE'13*, volume 7793 of *Lecture Notes in Computer Science*, pages 6–20. Springer, 2013.

Polymorphic Templates

A design pattern for implementing agile model-to-text transformations

Gábor Kövesdán, Márk Asztalos and László Lengyel

Budapest University of Technology and Economics, Budapest, Hungary
{gabor.kovesdan, asztalos, lengyel}@aut.bme.hu

Abstract. Model-to-text transformations are often used to produce source code, documentation or other textual artefacts from models. A common way of implementing them is using template languages. Templates are easy to read and write, however, they tend to become long and complex as the complexity of the metamodel grows. This paper proposes a design pattern that allows for the decomposition of complex templates with branching and conditions inside into simpler ones. Its main idea is that the code generator does not know about the concrete templates that are called: they are determined by the objects of the model being traversed. The concrete template is selected through object-oriented polymorphism. The pattern results in a flexible code generator with simple templates, good extensibility and separation of concerns. This agility facilitates the design for extension and changes, which is paramount nowadays.

Keywords: Modeling ▪ Domain-Specific Modeling ▪ Model Transformation ▪ Code Generation ▪ Design Pattern

1 Introduction

This paper presents a design pattern that can be used to create flexible model-to-text (M2T) transformation. The pattern is applicable on complex M2T templates and proposes an object-oriented decomposition for the generator, the model objects and the templates. This decomposition achieves reduced complexity, separation of concerns, improved readability and most importantly improved maintainability and flexibility. We believe that the *Polymorphic Templates* pattern will greatly help developers of all kinds of M2T transformations in designing robust code generators that are easy to maintain and extend.

The rest of this paper is organized as follows. Section 2 briefly explains the foundations of M2T transformations using template languages. Section 3 lists existing work available on the subject. Section 4 describes the design pattern in a format that is similar to those that are used in design pattern catalogs. Section 5 concludes.

2 Background

M2T transformations are often used for generating source code, documentation or other artefacts in order to speed up software development. Models can be easily produced and validated by the proper tooling so we can make sure they convey the correct information. By using models and code generation techniques, the resulting source code can be produced more quickly. Besides, if they are validated and the code generator is correct, we can assure that the generated code is correct as well. When using *Domain-Specific Modeling (DSM)* and model processing [1-2], these benefits apply at a much higher extent. DSMs raise the abstraction and allow for expressing the problem from a more human-friendly viewpoint that does not require thinking in programming notions. This makes the process less error-prone.

Because of the above reasons, the code generator is an especially important component in model processing. To facilitate the development of code generators, so called template languages have been developed. Generating code from conventional programming languages is difficult because substitution and print instructions are intermixed with literal fragments of the output, thus it becomes hard to read. Template languages reverse the logic: everything written in the template goes to the output by default, only value substitutions and conditional instructions or loops must be written with special markup. These template languages highly simplify the development of code generators and make the generated code easier to read and write. However, as the complexity of the model grows, templates also become longer and more complex and these advantages can be only achieved at a limited extent. From different model objects, usually different kinds of code fragments are generated and this requires branching instructions in the template. If there is a high number of them, the template becomes hard to read and maintain.

Despite that the explanation above mentions mostly code generation, other kinds of M2T transformations, like generating reports, documentation etc. are very similar in nature and the pattern is also applicable to them. For the sake of simplicity, throughout the paper we will simply refer to the M2T transformations as code generators.

3 Related work

The first well-known work that proposed the reuse of working solutions to common software engineering problems and their description in design pattern catalogs was the one published by Gamma et al. [3]. This work was followed by the *Pattern-Oriented Software Architecture (POSA)* series [4,5,6,7,8]. Apart from these general object-oriented design patterns, some more specialized patterns have also been described. In the field of *Domain-Specific Languages (DSLs)*, [1] provides a pattern catalog, covering several different aspects of DSLs and code generation. This is a rich source of information but it has a more general view than this paper and does not include the pattern described herein. Apart from this, [9] provides some practical uses of general object-oriented design patterns in recursive descent parsers and [10] describes how a parser generator uses object-oriented design patterns. These are specific uses of general design

patterns and these papers do not include more specialized patterns specific to DSLs and code generation. A pattern catalog [11] of architectural design patterns that can be used in language parsers has also been published. This is relevant for implementing DSLs.

Beside the movement of collecting solutions of common problems in design pattern catalogs, Yu and Mylopoulos emphasized [12] that research of software engineering had focused more on the what and the how rather than on the why. Their contribution justifies the need for more work that deals with understanding the requirements. There are also publications that collect the intents of using specific software techniques in so called intent catalogs. These are similar in nature to design patterns but they describe common motivations behind applying a specific solution. The intents behind DSLs have been described in [13]. Amrani et al. has published an intent catalog behind using model transformations [14].

There are several existing template engines that allow for the decomposition of templates into smaller units. These make it possible to organize template code into separate methods and files. By using these tools, each model class can have its own template associated and template code can be further cut down to methods that generate a specific feature from the model class. For example, such template implementations are *Xtend* [15] and *Microsoft T4* [16]. The pattern presented in this paper provides a method for using these tools efficiently.

The technique of incremental model transformation [17] is related to this work in that it also deals with changes in the code generation process. However, this approach handles changes in the input model and is able to update parts of the generated model based on the changes in the input model. By not having to rerun the entire transformation, it saves computational time. In contrast, the design pattern presented herein facilitates the evolution of the M2T transformation itself. As the tool evolves and more features are supported by the code generator, the templates also becoming more complex. The proposed solution decomposes the templates into highly cohesive, flexible units to facilitate changes and extension. So the two techniques address different issues and are not mutually exclusive.

4 The Polymorphic Templates Design Pattern

This section describes the design pattern in catalog format similar to what is used in the *POSA* series. Namely, the following sections are applied:

- *Example:* a concrete use case in which the pattern has been applied.
- *Context:* the context in which the design pattern is applicable.
- *Problem:* the challenges that suggest the application of the pattern.
- *Solution:* the way how the pattern solves or mitigates the problems.
- *Structure:* the main participants and their relationships and responsibilities in the pattern.
- *Dynamics:* the interaction of the participants of the pattern.
- *Implementation:* techniques and considerations for implementing the pattern.
- *Consequences:* advantages and disadvantages that the application of the pattern implies.

- *Example Resolved:* the short description of how the initially presented example has been resolved by using the pattern.

The *Known Uses* and the *See Also* sections are omitted due to lack of space and related patterns.

Example

The *ProtoKit* tool [18] is a DSL and an accompanying code generator for describing the message structure of application-level binary network protocols. Object-oriented general purpose programming languages (GPLs) can represent messages as classes, being the member variables the different fields encompassed in the message. However, several features of these protocols are difficult to support in this way, such as bitfields, encoded fields or length fields of variable-length fields. *ProtoKit* generates the classes, member variables and accessors with the boilerplate code to support the above mentioned features. In the generated code, different fields of the message will result in different variable definitions, initialization code snippets and accessor methods. Because of this, the template of the generated code contains several loops that iterate over the fields and each iteration includes several *instanceof* checks. Because of the looping and branching markup in the template, the actual output is hard to read among the lines. Besides, the template is rapidly growing as new features and model classes are added. Parts of the template are not decomposed according to what feature they generate or what model object they process. This makes it difficult to locate and modify the generated code of a specific feature.

Context

Model-to-text transformations that use templates to produce textual output and have a type hierarchy in the input model.

Problem

When complex models are processed from template languages a number of problems arise:

- *High complexity.* The template class can be decomposed into several methods that are responsible for generating different features but the template that processes the model remains highly complex.
- *Lack of encapsulation and separation of concerns.* Code fragments for generating different features from the same model class are separated by branching instructions and thus are scattered through the template. Logically coherent code fragments are not encapsulated into highly cohesive classes.
- *Poor readability.* Because the output code is intermixed with conditions and branching instructions that generate the proper code fragment from each model object that

is traversed, the main goal of using a template language – good readability – does not apply.
- *Poor maintainability and extensibility.* Because of the lack of encapsulation and the branching instructions that add syntactic noise, several isolated parts of the generator must be modified in order to modify the behavior. Similarly, extension requires adding new code fragments to several places inside the same template.

Solution

Decompose templates on a per model class and per feature basis. The code fragment generated for a specific feature and from a specific model class will be encapsulated in its own template. The generator accesses templates via their common interfaces and does not know about concrete template types. Determining the template to call for a specific feature is the responsibility of the model object being traversed.

Structure

A possible structure of the pattern is depicted in Figure 1. To keep the diagram comprehensible, only one feature, *feature A* is depicted. Of course, the pattern supports multiple features by having multiple feature hierarchies. The pattern has the following participants:

- *ModelClass:* an abstract type of the model elements processed by the generator.
- *ConcreteModelClass1* and *ConcreteModelClass2:* concrete types with different semantics that are instantiated in the model. They usually result in different output.
- *FeatureATemplate:* an interface for the polymorphic templates that generate *feature A* for concrete instances of *ModelClass.*
- *ConcreteFeatureATemplate1* and *ConcreteFeatureATemplate2:* concrete templates that generate "*feature A*" for *ConcreteModelClass1* and *ConcreteModelClass2,* respectively.
- *Generator:* the entry point of the code generator. This component traverses concrete instances of *ModelClass* in the model and calls the corresponding templates.
- *Application:* the main application that obtains the model (depicted as a set of aggregated *ModelClass* instances) and calls the code generator.

Dynamics

The Generator component has knowledge of what features must be generated and in what order. This is specific to the domain. Generating features involves the traversal of model objects in the input model. When a specific model object is visited, the Generator first calls the *getATemplate()* method on the model object to obtain an instance of the template to use. Then the Generator calls the *generateA()* method on the template instance obtained from the visited model object. This is depicted in Figure 2.

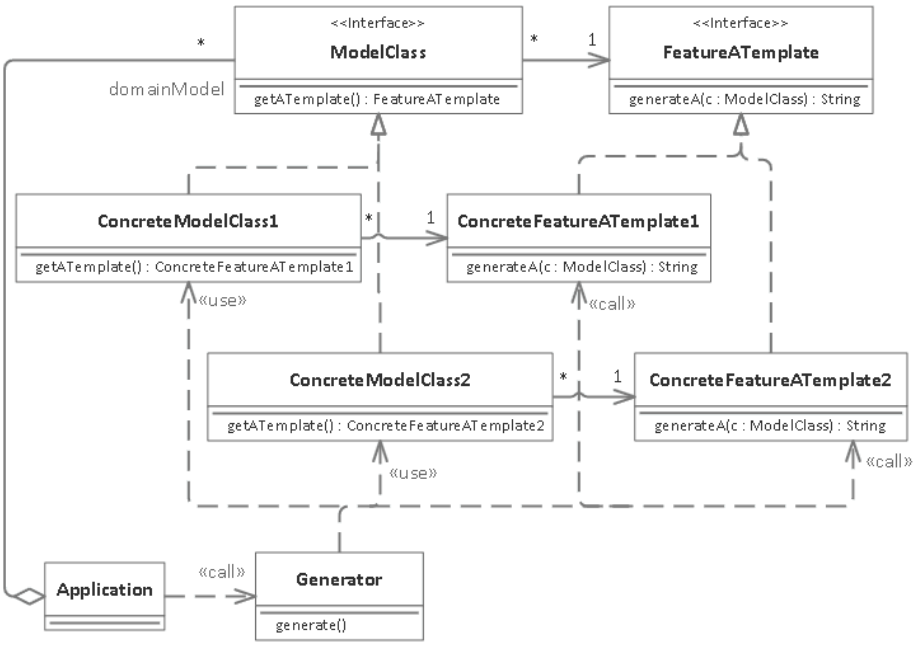


Fig. 1. A possible structure of the participants in the *Polymorphic Templates* pattern

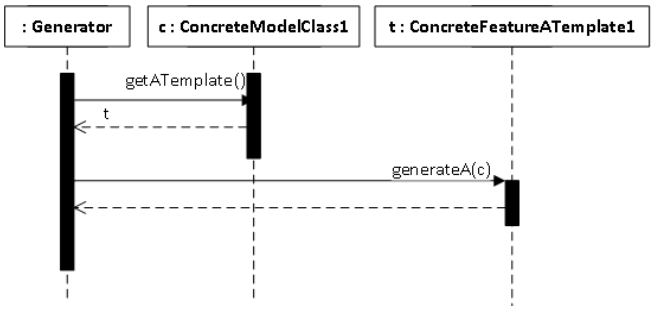


Fig. 2. The interaction of the participants in the *Polymorphic Templates* pattern

Implementation

The following techniques should be considered for the implementation of the pattern:

- Model classes may return template references in three different forms:

  (a) *As object references.* This is the most object-oriented, the most efficient and the safest option. The code generator directly retrieves a reference and uses it to call

the template. To apply this solution, the modeling framework must support references to classes that are not part of the model. Several modeling frameworks support this by importing external data types.

(b) *As the class name.* The class name is returned as a string to the code generator and before calling the template, the code generator must instantiate the template class using reflection. Reflection has some performance hit and the class name is easy to mistype. However, this is a viable option if the modeling framework does not support external data types.

(c) *As an identifier.* An arbitrary identifier that is used to obtain a reference for the concrete template class. For example, it may be a key for a hash table. This option eliminates the performance hit of reflection but the validity of the identifiers still must be guaranteed by the implementor. This solution requires an extra effort to implement the lookup mechanism.

- Deciding what is an independent feature is a crucial point in applying this patterns. This determines the number of templates and how cohesive individual templates will be. This latter greatly affects flexibility and extensibility. It does not only include identifying features per a single class hierarchy but deciding on cases like whether a single template will be provided for aggregating elements or it will be chunked down to separate ones, one per each aggregate.
- If the template references are obtained by instance methods, template inheritance can be leveraged. In case a parent and a child model class generate the same code for a specific feature, the template can be inherited. It is not necessary to define a new template for the child, nor to repeat the template reference.
- The pattern can be implemented through model refinement if the modeling environment supports it. In this scenario there are two layers of models. The lower layer contains information that is strictly the model without template associations. The upper layer refines the lower layer, adding references to templates. If there are several target languages, it is a viable solution to define a separate upper layer for each of them.

Consequences

The pattern achieves the following advantages:

- *Reduced complexity of the templates.* The complexity of templates is reduced by decomposing long and complex ones that contain conditions and branching markup into several shorter “flat” templates.
- *Separation of concerns in the code generator.* Each template is responsible for the generation of a specific feature using a specific model class.
- *Improved readability.* The eliminated branching instructions make templates more readable. Code is not intermixed with control statements therefore the actual code to be emitted is easier to understand. A good abstraction of features and template hierarchy helps to factor templates in a way that each of them contains an intuitively comprehensible unit of code fragment. This further improves readability.

- *Improved extensibility.* The implementation of a concrete feature regarding a concrete model element is encapsulated into a concrete template class. This, combined with an intuitive and consistent naming convention, makes it easy to locate the code that must be modified. As a result, modifications of a specific feature are constrained to a small set of templates or even a single template if it only affects a single concrete model class. The generator itself or templates of other features or unaffected concrete model classes never need modifications. Extensions are similarly straightforward. The only necessary steps are creating the templates for features of the newly added concrete model class and associating them with the new class. Since the generator only has knowledge of the common interface and calls the concrete templates through polymorphism, it is not necessary to modify the generator in any way.

The application of the pattern also has some disadvantages:

- *Mixing model and implementation details.* Although the pattern achieves good separation of concerns in the code generator, it mixes some implementation details with the model. The pattern associates templates with model classes despite that the latter should be part of the code generator itself since it conveys implementation details for the generator. On the other hand, it can be argued that the features generated from model classes has to do with their behavior, which in turn, fits into the model. Developers that apply the pattern should consider whether it is a problem for them that such details are stored in the model. If there is no benefit in separating them from the model, this issue should not be blindly considered as a severe problem just because best practices warrant of separating model and implementation. However, if there is a high number of features or there are several supported target languages, models may become flooded by template references.
- *Increased number of template classes.* Since templates are decomposed on a per feature and a per concrete model class basis, their number increases significantly. This is a direct consequence of decomposition so it is not considered as a real disadvantage. Besides, the number of classes depends on the choice of feature abstraction and the use of template inheritance, therefore it can be slightly adjusted by choosing the right abstractions.
- *Difficult to deal with cross-cutting features.* It is not trivial how to apply the pattern with cross-cutting features that are not associated to a single model class but to several ones. If there is an aggregate that references the involved model classes, it can be considered to associate the features to that model class.

Example Resolved

In the *ProtoKit*[1] tool, several features have been identified: (1) variable definition, (2) initialization code, (3) accessor methods, (4) equals expression to compare the generated variables, (5) hashcode expression to generate a hash for the variable and (5) clone code for the variable. For the implementation of the tool, the *Eclipse Modeling Framework (EMF)* [19] has been used. EMF leverages round-trip code generation and allows

[1] Source code available at https://github.com/gaborbsd/ProtoKit.

for defining methods on model classes by writing their Java code. The abstract *Field* model class is the superclass of all fields that can be used in a message definition and it defines the methods that return an instance of the template to use for the concrete *Field* instance. The templates are implementations of the *FieldGenerator* interface. It is imported to the EMF model as a data type so that the methods can be modelled. The interface only defines a *generate()* method that takes the a *Field* instance as an argument. This generates the code fragment of a specific feature from the specified *Field.* The implementations of the interface implement this method and do not store state so they use the *Singleton* [3] pattern to facilitate the reuse of instances. The templates have been decomposed on a per field per feature basis as the *Polymorphic Templates* pattern suggests. Not all of the Field types require their own template, some templates are reused. For example, Fields that generate a primitive Java variable all share an empty template for the cloning feature since Java's *clone()* method inherited from *Object* takes care of them. Referential variables all cloned in the same way, so they also share a template.

The application of the design pattern has highly improved the readability and the flexibility of the tool. It became easier to add new *Field* types and to modify existing functionality. The increased number of the templates do not cause a problem and they have been grouped to Java packages based on the feature they generate. Because of EMF allowing method definitions on model classes and the moderate number of features on the type hierarchy of *Fields,* mixing models and implementation did not mean any disadvantage either. In the tool, there were no cross-cutting features so the pattern was easy to apply.

5 Conclusion

The paper has presented a novel design pattern for implementing agile M2T transformations. This solution has been identified in our DSL-based tools that leverage code generation to help software development. It has been chosen to publish this solution as a design pattern to facilitate its reuse. The catalog format enables developers to easily understand the context of the application, the problems that arise in this context and how the application of the design pattern addresses these issues. Implementation ideas are also provided. These help developers to decide, which variant fits better their needs. A real-life application of the pattern will be published in our future papers.

We believe that the *Polymorphic Templates* design pattern will be of great use for other software developers who use M2T transformations. It is a potential tool for creating agile code generators and thus is highly demanded in nowadays' software environments.

Acknowledgments. This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013) organized by VIKING Zrt. Balatonfüred. This work was partially supported by the Hungarian Government, managed by the National Development

Agency, and financed by the Research and Technology Innovation Fund (grant no.: KMR_12-1-2012-0441).

References

1. Fowler, M.: Domain-Specific Languages, Addison-Wesley (2010)
2. Kelly, S., Tolvanen, J.: Domain-Specific Modeling: Enabling Full Code Generation, Wiley - IEEE Computer Society Publications (2008)
3. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley (1995)
4. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture Volume 1: A System of Patterns, Wiley (1996)
5. Schmidt, D.C., Stal, M., Rohnert, H., Buschmann, F.: Pattern-oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects, John Wiley & Sons (2000)
6. Kircher, M., Jain, P.: Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management, Wiley (2004)
7. Buschmann, F., Henney, K., Schmidt, D.C.: Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing, Wiley (2007)
8. Buschmann, F., Henney, K., Schmidt, D.C.: Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages, Wiley (2007)
9. Nguyen, D., Ricken, M., Wong, S.: Design Patterns for Parsing, In: 36th SIGCSE Technical Symposium on Computer Science Education, pp. 477–48, ACM, New York (2005)
10. Schreiner, A.T., Heliotis, J.E.: Design Patterns in Parsing, In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181–184, IEEE Press, New York (2001)
11. Kövesdán, G., Asztalos, M., Lengyel, L.: Architectural Design Patterns for Language Parsers, Acta Polytechnica Hungarica, vol. 11, no. 5, pp. 39–57 (2014)
12. Yu, E.S.K., Mylopoulos, J.: Understanding 'why' in software process modelling, analysis, and design, In: Proceedings of 16th International Conference on Software Engineering, pp. 159–168., IEEE Computer Society Press (1994)
13. Kövesdán, G., Asztalos, M., Lengyel, L.: A classification of domain-specific language intents, International Journal of Modeling and Optimization, vol. 1, no. 4, pp. 67–73 (2014)
14. Amrani, M., Dingel, J., Lambers, L., Lúcio, L., Salay, R., Selim, G., Syriani, E., Wimmer, M.: Towards a model transformation intent catalog, In: Proceedings of the First Workshop on the Analysis of Model Transformations, pp. 3-8, ACM (2012)
15. Bettini, L.: Implementing Domain-Specific Languages with Xtext and Xtend, Packt Publishing (2013)
16. Vogel, P.: Practical Code Generation in .NET, Addison-Wesley (2010)
17. Hearnden, D., Lawley, M., Raymond, K.: Incremental Model Transformation for the Evolution of Model-Driven Systems. In: Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science, vol. 4199, pp. 321-335 (2006)
18. Kövesdán, G., Asztalos, M., Lengyel, L.: Modeling Cloud Messaging with a Domain-Specific Modeling Language, In: CloudMDE, A Workshop to Explore Combining Model-Driven Engineering and Cloud Computing. In conjunction with MoDELS 2014. In press.
19. Steinberg D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework, 2nd Edition, Addison-Wesley Professional (2008)

Flexible and Scalable Modelling in the MONDO Project: Industrial Case Studies

Alessandra Bagnato[1], Pedro Malo[2], Salvador Trujillo[3], Xabier Mendialdua[3], Xabier de Carlos[3], Etienne Brosse[1], Andrey Sadovykh[1].

[1] SOFTEAM, 8 Parc Ariane 78284 Guyancourt, France
{alessandra.bagnato, etienne.brosse, andrey.sadovykh}@softeam.fr,
[2] UNINOVA, Pedro Maló, Campus da FCT/UNL, Caparica, Portugal
pmm@uninova.pt,
[3] IKERLAN Paseo J.M. Arizmendiarrieta 2, 20500 Mondragon, Spain
{ STrujillo, XMendialdua, XDeCarlos}@ ikerlan.es

Abstract. Today, system designs and their management are crucial parts of most systems development processes. To stay competitive engineers from several expertise domains use Model-Based engineering (MBE) to design the systems they intend to implement in order to specify, test, simulate, validate and iterate their design as soon as possible. System designs are living and evolving artefacts this imply to be able to manage them in an efficient and agile way. The MONDO FP7 EU project aims to comprehensively tackle the challenge of scalability in system design and management by developing the theoretical foundations and an open-source implementation of a platform and will offer to Model-Driven Engineering (MDE) users advanced flexibility in their different modeling approaches. This paper describes three different industrial demonstrators and three different modelling approaches that will be utilised to evaluate the capabilities of the MONDO technologies. For each demonstrator the interests of the industrial user partners are described along with their current and desired improvements in technologies to support MBE in a much more flexible way. Specific evaluation scenarios are specified for each of the targeted industrial domains as well.

Keywords: scalable modelling, Model-Driven Engineering, software engineering, qualitative evaluation..

1 Introduction

Model-Driven Engineering (MDE) has been shown to increase productivity and reuse and it has significantly enhanced important aspects of software engineering development such as consistency, maintainability and traceability [5]. MDE is therefore increasingly applied to larger and more complex systems. However, the current generation of modelling and model management technologies is being stressed to their limits in terms of their capacity to accommodate, in an agile way, collaborative development, efficient management and persistence of large models (larger than a few hundreds of megabytes in size) . Thus, recent research has focused

on scalability and flexibility across the MDE technical space to enable MDE to remain relevant and to continue delivering its widely recognized productivity, quality and maintainability benefits. The MONDO [1] project proposes to look into the scalability issues for MDE in order to give more flexibility in processes and to help into the agile definition and management of large models. The purpose of this paper is to provide a description of the challenges targeted by three of the industrial demonstrators that will be utilised to evaluate the capabilities of the MONDO technologies. The selected demonstrators are depicting typical MDE issues found in industry and describe the the benefits to be achieved in the selected scenarios.

The Use Cases that will be described herewith are the following:

- Use Case for Modelling Tool domain.
- Use Case for the Offshore Wind Power domain.
- Use Case for Open-BIM Construction domain.

In the next sections we briefly outline the challenges tackled by the MONDO project. We then describe three of the MONDO case studies and summarize the general feedback received from the three industries on the expected benefits to be put in place within the different contexts to achieve better flexibility during engineering and storage of large models. Finally, the paper is concluded along with plans on our future work.

2 The MONDO project

Typically, achieving scalability involves being able to construct large models and associated DSLs or meta-models in a systematic manner; enabling large teams of modellers to construct and refine large models in a collaborative manner; advancing the state of the art in model querying and transformations tools so that they can cope with large models (of the order millions of model elements); and providing an infrastructure for efficient storage, indexing and retrieval of such models [5]. In particular, the maintenance and manipulation of large models unique scenarios addressed by a novel class of model transformations (MT) able to extend the capabilities of traditional MT approaches is under study within the MONDO project [13] and a prototype tool supporting scalable concrete visual syntax enabling the rapid construction of Domain Specific Languages (DSLs) with built-in capabilities to navigate, explore and abstract large models is foreseen and under study by the project.

The MONDO research roadmap towards achieving scalability in model driven engineering is described in [4] and briefly summarized in Figure 1. The project builds on MDE research issues aiming at dealing with industrial scale models that need to be persisted in a way that allows for their seamless and efficient manipulation [6][7], often by multiple stakeholders simultaneously and will tackle scalable queries and transformations, scalable DSLs, scalable collaborative modelling [10][11][12] within scalable model persistence [6]. All the project advances will be reported within the main project web site at [1] for the whole project duration and all technologies developed by research partners will be released as open-source software under EPL [9]. As such, industrial partners within the project and outside it will be able develop

proprietary extensions on top of this infrastructure without having to open-source them.

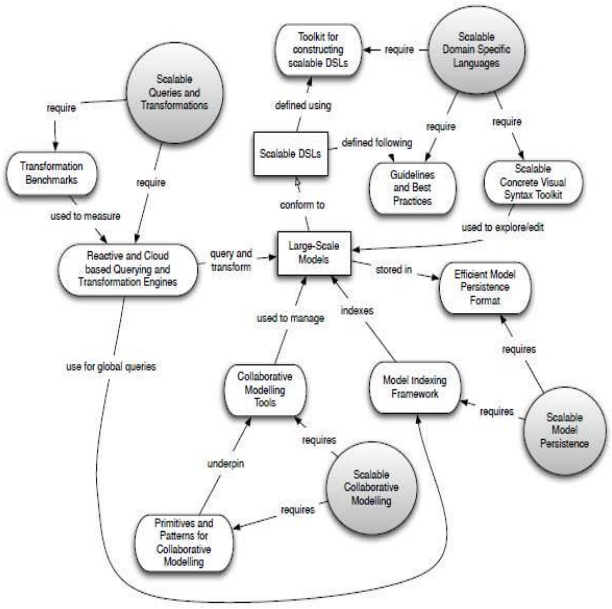


Fig. 1. Tackling the challenge of scalability in MDE

As part of the MONDO project, this paper looks closely into the needed efficient and flexible engineering, storage, indexing and retrieval of large models within three different industrial case studies selected to evaluate the project results.

2 The Modelling Tool domain case study

SOFTEAM aims at applying scalability in MDE technologies within the SOFTEAM Modelio [2] modelling tool. Modelio is an open source modelling tool, providing support for many kind of modelling e.g. UML2 modelling, Enterprise Architecture modelling, Business Process modelling, and SOA modelling. The MONDO technologies are expected to make possible to enhance Modelio capabilities in an agile way and to allow it to manage very large models by gaining speed in the overall design carried out. In particular the production chain that includes, the DSL engineering, the DSL persistence, the model (conforms to a given DSL) engineering and the model persistence, that is currently in place within Modelio and the Eclipse Modeling Framework (EMF)[14] world is considered too much rigid and constrained by MDE users e.g. modellers, architect and developers.

The MONDO research and results will be key to meet the increasing need from SOFTEAM customers to support a more agile production chain that considers larger and larger models, and larger teams of developers.

SOFTEAM will evaluate the MONDO improvements through an modelled application called Voyages Discount. This modelled application uses TOGAF modelling [3] which aims at improving business efficiency ensuring consistent standards, methods, and communication among enterprise architecture professionals.

The model has been chosen since it provides an industrial business process that required multiple stockholders communications to be completed where the modelling flexibility provided within MONDO will be helpful.

Currently the Modelio Teamwork Manager feature endows Modelio with a distributed collaborative modelling environment through a Subversion (SVN) repository. The model fragments [8], composing the Modelio project, are managed by Modelio Teamwork Manager and the enhancement of an improved collaboration environment will be very helpful in increasing productivity.

In **Table 1** we present and summarize the expected targeted flexibility benefits and we consider in particular the need to reduce the time to complete the production chain above mentioned by means of structured EMF modularity and collaborations.

Table 1. Overview of the scenarios within Modelling Tool use case of MONDO.

Scenario	**Purpose**	**Specific expected MONDO/Extreme Modelling benefits**
Scenario 1: MONDO framework querying facility	This scenario focusses on the comparison of the query facilities and performances provided by both Modelio and MONDO frameworks. In the MBE approach, queries are often defined and performed to extract specific and relevant information from model. Modelio store offers a set of predefined scripts among thus the "Simple Statistics Reporting" script provides model statistics on objects present under selected element. This latter will be used as reference to compare both technologies in terms of querying.	MONDO technology will improve the Modelio querying facilities to better meet end user's needs in terms of performance (time and memory). A methodology whereby the performance of model queries can be systematically evaluated, and relevant performance predictor metrics can be identified is under study within MONDO [15]. More advances in this direction are also under study within [10][11][12].
Scenario 2: MONDO framework collaborating modelling	The actual model usage across several domains, several teams and several peoples implies to be able to collaborate during the model specification. This scenario aims at integrating the MONDO technologies within the Modelio modelling tool for supporting large and complex models and large collaborating teams.	MONDO technology will improve end users' experience of modelling gaining speed in the overall design in large team and large model context. MONDO will provide management techniques for collaborative modelling adapted from version control systems including locking, transaction handling with commit, conflict management and resolution [10][11][12].
Scenario 3:	Specification and execution of	MONDO technology will

MONDO framework support in transforming model to text.	ModelToText transformations are the aim of this third evaluation scenario. Capability provided by both environment (MONDO and Modelio tool) will be evaluated in term of possibility, performance and easiness to use. Modelio document publisher extension provides by default a set of existing transformation including the analysis and design one's which will be used as benchmark.	overcome the lack of documentation by improving the scalability and performances of Modelio ModelToText transformations in a large model context. First results in this direction can be found at [13].
Scenario 4: MONDO framework support in transforming model to another model.	The purpose of this scenario is to compare the ModelToModel transformation support by both frameworks. Among all existing ModelToModel transformation specified inside Modelio or its extension, the performance analysis will take EMF UML2 XMI import/export facilities as reference.	MONDO technology will improve the scalability and performances of Modelio ModelToModel transformations in a large model context by reducing the amount of needed time and ressources . An EMF Splitter with a structured Approach to EMF modularity is under study within the project to bring more agility in the overall production chain and will be evaluated.

3 The Offshore Wind Power domain case study

MONDO technology is designed to support collaborative domain specific modelling for offshore wind turbines. Modelling tools will enable engineers to specify concurrently control systems of wind turbines and to share different specifications (model versions). This is expected to raise productivity during the development and customization of software for wind turbine control systems.

Wind turbines are complex systems composed by a set of physical subsystems. Different subsystems must work in a coordinated manner to transform wind energy into electrical energy. The purpose of a wind turbine control system is to supervise and control all those subsystems for their correct operation.

The Wind Turbine Control System Modelling Tool is a tool that the engineers use for specifying behaviour of the system that will control the wind turbine. Software code responsible for turbine control will be generated automatically from models.
Control system development entails collaboration between engineers of different disciplines. Engineers of each discipline have different skills and they develop specific-parts of the wind turbine subsystems. Consequently, model specification implies the use of collaborative modelling tools. Moreover, since wind turbines' control models are large and complex, modelling tools should be scalable and provide agile mechanisms for model development (i.e. advanced queries, load-on-demand, partial load of models, etc.).

Wind Turbine Control System Modeling Tool has been developed based on existing Open-source Eclipse modeling technology. Unfortunately, existing modeling

frameworks in which tools are based are not conceived to be used in an agile and collaborative manner. This is the case of the design of a wind turbine in which several engineers collaborate together step by step to define, integrate and validate models incrementally that form subsystems and then complete systems.
In contrast to the work process described above, existing tools are conceived for a single engineer to work on a complete specification, preventing collaboration beyond the tool and the engineer. MONDO pursues to expand existing tools to support such collaborative modeling of teams of engineers working together. MONDO collaborative tools are expected to foster agility and collaboration in the modeling of wind turbines.

Results obtained on the MONDO Project will be the key to provide collaborative modelling tools enabling to turn the wind turbine design process into a more agile and flexible process. MONDO Project will enable to add new features to the modelling tools such as concurrent model edition, partial load of models, advanced querying capabilities, etc. Additionally, technology developed within the MONDO Project will support modelling from mobile devices. It will allow performing modelling activities also in environments where the conditions are not bests (*in-situ* maintenance of the wind turbine).

Table 2. Overview of the scenarios within Offshore Wind Power use case of MONDO.

Scenario	**Purpose**	**Specific expected MONDO/Extreme Modelling benefits**
Scenario 1: Wind turbine control system collaborative modelling	This scenario is focused on the design of wind turbines' control systems. Several system engineers participate at the same time on the specification of wind turbine control system. Generally, each system engineer focuses in one subsystem and specifies the control of the subsystem. All those parts together specify the behaviour of the control system that is responsible to control and supervise the wind turbine.	Technology provided by MONDO will provide mechanisms that allow working concurrently on model elements. This will provide the flexibility for different modeling engineers to work at the same time on different subsystems or even on the same subsystem. As a result, this will ease teamwork when modeling a complete system. Apart from improving communication among engineers, it will favor an early identification of potential design flaws.
Scenario 2: Partial load and load-on-demand of models related to subsystems	This scenario is focused on the wind turbine commissioning process in which different physical subsystems (or sets of subsystems) are commissioned separately. This entails a scenario that from the view-point of system control, only a fragment of the whole model must be taken into account. System engineer only	MONDO technology will allow system engineers to work only with specific parts of the model, providing features that allow partial load and load/unload on demand of parts of the model where different subsystems are specified. MONDO will provide

	needs the part of the model concerning to the subsystem. Therefore, system engineers involved on the subsystem commissioning must be able to work with a partial view of the whole model, having the ability to load and unload models fragments concerning to subsystems or parts of a subsystem.	technology to enable a team of engineers to work on parts of models that can be validated partially. Enabling partial work will make the process more agile and flexible.
Scenario 3: Modelling from mobile devices	This scenario is located on the installation and maintenance activities performed within a wind farm. Often during these activities, wind turbine control systems require small adjustments (i.e. modify parameters' values or activation/deactivation of non-critical control algorithms). These adjustments will be performed using handy devices. However, these changes imply also (i) re-generate code for the control system and (ii) add the new model version to the repository.	MONDO technology will provide engineers solutions to perform installation and maintenance activities of the wind turbines within a wind farm. These solutions should support modelling from lightweight devices such as mobiles or tablets. In this way, these solutions will facilitate fieldwork where commonly conditions are not good.

Besides Offshore Wind Power domain, IKERLAN-IK4 also plans to use technology developed within the MONDO Project on other domains such as transportation and capital goods.

4 The Open-BIM Construction domain case study

A Building Information Model (BIM) is an instance of a populated data model of buildings that contains multi-disciplinary data specific to a particular building, which it describes unambiguously. BIM offers easier use of interoperable industry software tools, fewer errors and omissions, and time and cost savings that can cumulatively result in earlier building delivery. It can facilitate discussion, checking, analysis and communication about a project much earlier and in much clearer and precise terms than standard practice.

The IFC (Industry Foundation Classes aka Information For Construction) is the neutral and freely available specification (model) to describe, exchange and share information typically used within the building and facility management industry sector. As such, the IFC provides an Open-BIM as any single vendor or group of vendors does not control it, is openly accessible and used free of charge, and it is a standard de facto (in industry) and de jure (ISO 16739). BIM is a key enabler for Model-Based Engineering in construction-related ICT industries. ICT solutions for construction industries take advantage of MBE methods and practices to properly manage and exploit BIM models. The key solution to take advantage of MBE in BIM is the so-called Model Servers. Model Servers are a technological concept that was coined by the AECO software industry that designates a specialised ICT solution

capable of providing BIM capabilities and services. A Model Server provides supports for modelling, storing, sharing, inspect, visualise, and operate BIM models.

Model Servers of today are generically using the Open-BIM IFC model. An Open-BIM/IFC Model Server is thus a data repository/store with supporting services that provide multi-user access, storage and management, and allow the use of the IFC data model as the underlying representation structure (schema).

A BIM-IFC data model of a not-so-complex building comprehends a huge number of modelling elements (to the millions scale). A comprehensive building information model enclosing all the building disciplines can rapidly go from few millions to many dozens of million modelling elements. Moreover, BIM-IFC data models are shared (serialized) using the text-based ISO10303 STEP Part#21 representations that rapidly escalate file sizes to many Megabytes to Gigabytes.

The trouble is that today's breed of Model Servers exhibits problems in presence of big-to-huge BIM data models. Tools considerably degrade performance in presence of large-size BIM models (i.e. with some few millions of modelling elements) or simply "break" in presence of huge-size BIM models (in the scale of tens of millions of modelling elements). This led to very inefficient solutions for BIM-based collaboration like sharing partial models done by construction project designers and engineers that then requires time consuming model aggregation by some person ("model coordinator") and make it difficult for model users (model designers/engineers but also model clients) to get a comprehensive integral view of the model (building or set of buildings in a landscape).

The idea is for MONDO technologies to be able to take charge of addressing the performance and scalability issues of BIM data models that are presently dealt directly by its users (designers, engineers). The vision is that of a solution that enables an efficient management and exploitation of BIM large-to-huge-scale data models by both using best-of-breed MBE solutions and incorporating AECO domain knowledge for the best possibly experience and performance.

Table 3. Overview of the scenarios within Open-BIM Construction case study.

Scenario	**Purpose**	**Specific expected MONDO/Extreme Modelling benefits**
Scenario 1: File- (huge-) based collaboration	The file-based collaboration scenario is one where model designers work on separate models and share models (as huge files) with a model coordinator (that aggregates models altogether) and then with some model clients. Here the Heterogeneous Model Server exists especially at the Model Coordinator that can "upload" models and merge models together as well as doing some validations and checks to provided models. Data models are then exported in designated formats to be shared with model clients.	Technology provided by MONDO will allow designers & engineers to work on a specific part of the model, but with the added flexibility of having all parts merged together within a single model. Technology is to allow users to perform off-line work and then be able to readily reconvene for review and development.
Scenario 2: Shared-	The shared-model collaboration scenario is one where all users (model designers, model,	MONDO technology will allow system engineers to

(huge-) model collaboration	model coordinator, model clients) interact using a share-model hosted in a Model Server. Users are provided with data models views of the huge data models in the Model Server and can check-in/check-off model parts for off-line manipulation.	work only with specific parts of the model, with the added flexibility of providing features that allow partial load and load/unload on demand of parts of the model where different subsystems are specified.
Scenario 3: Quantity Take-Off (QTO) in huge IFC models	Quantity take-off's (QTO) is a key process in construction. QTO are a detailed measurement of the materials needed to complete a construction project. These measurements are used to format a bid on the scope of construction. Estimators review drawings and specifications to find these quantities, which is a very time consuming, and erroneous process. BIM provides a direct way to extract the quantities of a building. It is done using a complex query to the BIM model.	On huge BIM data models performing QTO is a non-trivial task due to the complexity of the query and need to traverse the whole of the data model. The scenario tests then the ability of MONDO to report out of large-to-huge data models using complex queries situations.[10] [11] [12]

5 Conclusion and future work

The case studies and evaluation scenarios selection presented herein was performed during the first ten months of the MONDO project (with a total duration of 30 months). As a general conclusion from the use case and scenario descriptions and the subsequent requirement gathering and prioritization phase form industry, it is believed that adopting MONDO technologies will bring benefits to the current software development processes used by all the involved industries, helping designers to work with large models.

In particular the MONDO technologies are expected to enable the Modelio tool to provide scalability in modelling, being able to construct large models and to enable large teams of modelers to construct and refine large models in a collaborative manner. Moreover, MONDO technologies will provide a solution for collaborative modelling within modelling tools to specify control systems for wind turbines. In this way these modelling tools will support features that ease collaboration. Finally MONDO technologies will be utilized within all the three case studies in the context of large model transformations. More specifically being able to generate documentation and to apply transformation on large models have been found as key aspects of scalable Model-Based Engineering.

The MONDO technologies evaluations within is planned for the end of second year of MONDO Project (October 2015). These will be both a qualitative and quantitative evaluations aimed at assessing the usefulness and ease of use when actually performing scalable modelling on real case studies through supporting tools.

Acknowledgements

The research leading to these results has received funding from the European Community Seventh Framework Programme (FP7/2007-2013) under grant agreement no FP7-611125. We would also like to acknowledge all the members of the MONDO Consortium for their valuable help and in particular Scott Hansen (The Open Group), and Dimitris Kolovos (University of York) for their support in the case studies scenarios and expected benefits definition.

References

1. MONDO Project Consortium: MONDO Project Homepage, http://www.mondo-project.org
2. Modelio Modelling Tool: Homepage, http://www.modeliosoft.com
3. TOGAF® Version 9.1 Enterprise Edition Web Site: http://www.opengroup.org/togaf/
4. Dimitrios S. Kolovos, Louis M. Rose, Nicholas Matragkas, Richard F. Paige, Esther Guerra, Jesús Sánchez Cuadrado, Juan De Lara, István Ráth, Dániel Varró, Massimo Tisi, and Jordi Cabot. 2013. A research roadmap towards achieving scalability in model driven engineering. In Proceedings of the Workshop on Scalability in Model Driven Engineering (BigMDE '13)
5. Mohagheghi, P., Fernandez, M., Martell, J., Fritzsche, M., Gilani, W.: MDE Adoption in Industry: Challenges and Success Criteria. In: Models in Software Engineering. Volume 5421 of Lecture Notes in Computer Science. Springer (2009) 54–59
6. Comparative analysis of data persistence technologies for large-scale models, K. Barmpis, Dimitrios S. Kolovos, Proceeding XM '12 Proceedings of the 2012 Extreme Modeling Workshop Pages 33-38 ACM New York, NY, USA ©2012
7. Hawk: towards a scalable model indexing architecture, K. Barmpis, D. Kolovos, BigMDE '13 Proceedings of the Workshop on Scalability in Model Driven Engineering Article No. 6 ACM New York, NY, USA ©2013
8. OMG: World Wide Modeling: The Agility of the Web Applied to Model Repositories SOFTEAM-Modelio: http://www.omg.org/news/member-news/SOFTEAM-ModelioWhite-Paper-WorldWideModeling.pdf
9. Eclipse Public License - v 1.0 - http://www.eclipse.org/org/documents/epl-v10.php
10. Ujhelyi, Z., Bergmann, G., Hegedüs, Á., Horváth, Á., Izsó, B., Ráth, I., Szatmári, Z., and Varró, D., "EMF-IncQuery: An Integrated Development Environment for Live Model Queries", Science of Computer Programming, 2014.
11. Debreceni, C., Horváth, Á., Hegedüs, Á., Ujhelyi, Z., Ráth, I., and Varró, D., "Query-driven incremental synchronization of view models", Proceedings of the 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling, York, ACM, pp. 31, 07/2014.
12. Szárnyas, G., Ráth, I., and Varró, D., "Scalable Query Evaluation in the Cloud", STAF Doctoral Symposium, 07/2014.
13. Dávid, I., Ráth, I., and Varró, D., "Streaming Model Transformations By Complex Event Processing", ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, MODELS 2014, Valencia, Spain, Springer, 2014.
14. The Eclipse Project, “Eclipse Modeling Framework,” http://www.eclipse.org/emf/.
15. Izsó, B., Szatmári, Z., Bergmann, G., Horváth, Á., and Ráth, I., "Towards Precise Metrics for Predicting Graph Query Performance", 2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), Silicon Valley, CA, USA, IEEE, pp. 412--431, 11/2013.

Configurable Formal Methods for Extreme Modeling

Position Paper

Uli Fahrenberg and Axel Legay

IRISA / Inria Rennes

Abstract. Reliable model transformations are essential for agile modeling. We propose to employ a configurable-semantics approach to develop automatic model transformations which are correct by design and can be integrated smoothly into existing tools and work flows.

Model management is an essential activity in a model-driven development process. This is especially true in an agile context where models are continuously refined, iterated and combined.

It is a common misunderstanding that development by stepwise refinement, or use of component algebras, requires using a highly planned and waterfall-like development process. See for example the following quote:

> An important variant of the waterfall model is formal system development, where a mathematical model of a system specification is created. This model is then refined, using mathematical transformations that preserve its consistency, into executable code. Based on the assumption that your mathematical transformations are correct, you can therefore make a strong argument that a program generated in this way is consistent with its specification. [30, p.32]

We will argue below that methods from model management and formal system development, when combined with and inspired by approaches originating in the area of formal interface and specification theories, can play an important role also in an agile context.

Within the context of model-driven engineering, automated model transformations such as merging, differencing and slicing are of great importance. This is especially true in the industrial context where models can easily get so large that the engineers only see them through viewpoints, or slices; indeed, an explicit system model may not even exist, so that the general model is only implicitly given as a collection of viewpoints.

When the system model is so complex that no single engineer has a comprehensive view, it can be very challenging to ensure correctness of an applied model transformation by inspection. (Even when the system model is less complex, ensuring correctness by inspection may be a difficult and error-prone process and

require advanced tooling.) The use of model transformations which are correct-by-design, or at least checkable-by-design, hence becomes increasingly important. This point of view has also been argued in [1, 6, 24, 27].

A good example is given by one of the case studies in the MERgE ITEA2 project [28]. This consists of a large system model with multiple (more than twenty) viewpoints, each detailing a different aspect of the model. The involved engineers are only working with the model through these viewpoints, as the whole model is too complex to be worked with directly. Now when a model transformation is applied to the model (*e.g.* a new subsystem is added through model merging), the engineers can inspect at their respective viewpoints whether, *locally*, the model transformation has been applied correctly. But can we be sure that this implies that the transformation is also *globally* correct? Can we design a procedure which allows such kind of local-to-global reasoning?

To put it succinctly, it is an important problem in model-driven engineering to ensure that model transformations are *semantically correct* or that, at least, their semantic correctness can be inferred by a combination of slicing and local inspection.

One basic model transformation is the one of *differencing*, *i.e.* assessing differences between models. This is an important ingredient in version control and essential in three-way merging, but can also be applied, more elementarily, to inspect related models for their common points and differences. Semantically correct differencing procedures, for class diagrams and activity diagrams respectively, have been proposed in [24–26]. However, these procedures rely on a complete, formal semantics of the modeling formalism in question, which for most cases is unrealistic: in practice, engineers use modeling tools which do not have a formal semantics, or where an existing formal semantics is too complex to be practically useful.

It is our point of view that any "bottom-up" approach to correct model transformations which uses a complete formal semantics as a starting point, such as the above-cited, is of doubtful use in practice. In practice, engineers develop models according to an intuitive understanding of how things work, and not according to a complete formal semantics (if indeed it exists at all).

Apart from being correct-by-design or checkable-by-design, we have also argued in [18] that it is important that model transformations return an object *of the same type* as the inputs. Hence, the merge of two class diagrams, for instance, should again be a class diagram, the difference of two feature diagrams should again be a feature diagram, etc. This allows developers to visualize the transformation and to manipulate it using the usual tools for working with models; the transformation integrates smoothly into existing tools and work flows. Figure 1 shows an example of such a work flow which necessitates that the difference between two models is again a model of the same kind.

[24–26] propose semantic difference operators for class diagrams and activity diagrams, respectively, however, their approaches are *enumerative* in nature: their output is a (potentially infinite) list of object models which witness the difference between the input diagrams. The output language is thus different

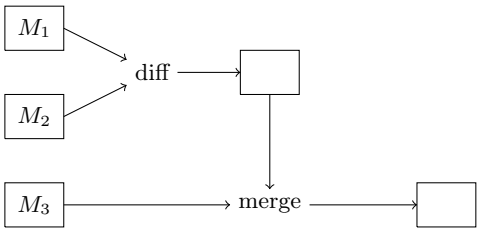


Fig. 1. Example of a work flow where the changes ("diff") between models M_1 and M_2 are automatically applied ("merge") to M_3. Note that a simplistic "diff" is not always the correct approach; it can be important to *e.g.*, tell the difference between renaming an entity and a deletion followed by an addition.

from the input language, which makes it impossible to integrate their tool into a standard work flow without additional processing steps.

Within the subject of interface and specification theories in formal methods, semantic model transformations exist for many types of low-level behavioral models [3–5, 7–9, 11–14, 20, 22, 23]. In recent work, we have developed a generalization of these approaches in which the semantics of models is *configurable* [2,16,17]: When models and specifications contain quantities such as timing information or resource use, the precise behavior of model transformations depends on the type of quantities and on the application. Hence, the precise definitions and properties of model transformations depend on the quantitative semantics, and our generalization offers a generic way of configuring the transformations and properties according to the semantics.

Similarly in spirit, Maoz *et.al.* have in [10, 27] introduced techniques for semantically configurable analysis of some high-level models, *viz.* class diagrams and scenario specifications. They use feature diagrams [21,29] for configurability, so that the analysis depends on the selection of features.

We have in [15, 18] introduced semantically correct model transformations for feature diagrams and class diagrams, see Figures 2 and 3 for some examples. These operators' return types are the same as their input types, so that they can be integrated smoothly into existing tools and work flows, but they rely on a complete formal semantics. We believe that these approaches can be combined with the configurability of [10, 27] to yield model transformations which are automatic and correct-by-design, yet flexible enough to be practically useful.

Conclusion

We propose to employ a configurable-semantics approach, using feature diagrams, to develop automatic model transformations which are correct by design and can be integrated smoothly into existing tools and work flows. Such model transformations are important for agile modeling methods.

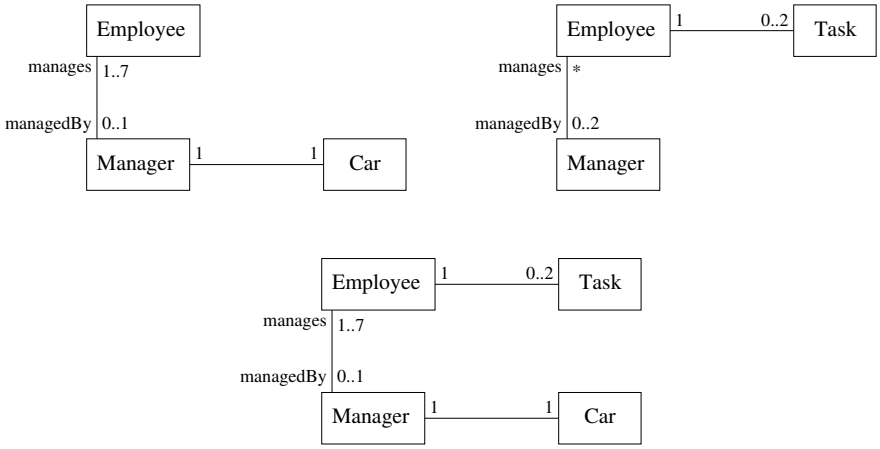


Fig. 2. Two class diagrams (above) and their merge (below), *cf.* [15].

As an example, consider the following scenario: An engineer wants to perform a three-way merge of two models which have evolved from a common ancestor. She will first use her development tools to attempt an entirely syntactic merge, which will detect some conflicts which are inessential because they only amount to syntactic differences while being semantically equivalent (given the developers' intuitive understanding of the semantics). These she can easily detect, and she can adjust her merge tool to take them into account (hence applying a partial semantics). However, due to semantic effects which propagate through the models, also the reverse may happen: there may be semantic conflicts which go undetected by the syntactic approach. These are more difficult to detect and require that even a merge which is syntactically correct be carefully checked by applying possibly several different semantics.

References

1. Kerstin Altmanninger. Models in conflict - towards a semantically enhanced version control system for models. In Holger Giese, editor, *MODELS Workshops*, volume 5002 of *LNCS*, pages 293–304. Springer, 2007.
2. Sebastian S. Bauer, Uli Fahrenberg, Axel Legay, and Claus R. Thrane. General quantitative specification theories with modalities. In Edward A. Hirsch, Juhani Karhumäki, Arto Lepistö, and Michail Prilutskii, editors, *CSR*, volume 7353 of *LNCS*, pages 18–30. Springer, 2012.
3. Sebastian S. Bauer, Line Juhl, Kim G. Larsen, Axel Legay, and Jiří Srba. Extending modal transition systems with structured labels. *Math. Struct. Comput. Sci.*, 22(4):581–617, 2012.
4. Sebastian S. Bauer, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wąsowski. A modal specification theory for components with data. *Sci. Comput. Program.*, 83:106–128, 2014.

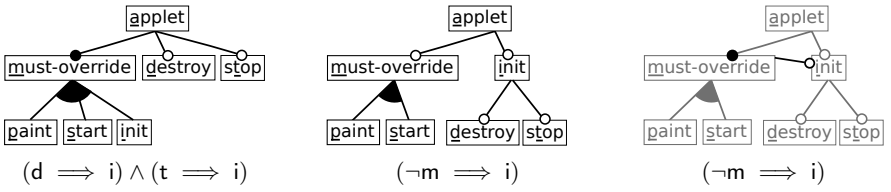


Fig. 3. Two feature diagrams and an over-approximation of their difference, *cf.* [18].

5. Nikola Beneš, Jan Křetínský, Kim G. Larsen, Mikael H. Møller, and Jiří Srba. Dual-priced modal transition systems with time durations. In Nikolaj Bjørner and Andrei Voronkov, editors, *LPAR*, volume 7180 of *LNCS*, pages 122–137. Springer, 2012.
6. Greg Brunet, Marsha Chechik, Steve Easterbrook, Shiva Nejati, Nan Niu, and Mehrdad Sabetzadeh. A manifesto for model merging. In *GAMMA*, pages 5–12. ACM, 2006.
7. Benoît Caillaud, Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wąsowski. Constraint Markov chains. *Theor. Comput. Sci.*, 412(34):4373–4404, 2011.
8. Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Freddy Y. C. Mang. Synchronous and bidirectional component interfaces. In Ed Brinksma and Kim Guldstrand Larsen, editors, *CAV*, volume 2404 of *LNCS*, pages 414–427. Springer, 2002.
9. Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In Rajeev Alur and Insup Lee, editors, *EMSOFT*, volume 2855 of *LNCS*, pages 117–133. Springer, 2003.
10. Barak Cohen and Shahar Maoz. Semantically configurable analysis of scenario-based specifications. In Gnesi and Rensink [19], pages 185–199.
11. Alexandre David, Kim G. Larsen, Axel Legay, Mikael H. Møller, Ulrik Nyman, Anders P. Ravn, Arne Skou, and Andrzej Wąsowski. Compositional verification of real-time systems using Ecdar. *J. Softw. Tools Techn. Transfer*, 14(6):703–720, 2012.
12. Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wąsowski. Timed I/O automata: a complete specification theory for real-time systems. In Karl Henrik Johansson and Wang Yi, editors, *HSCC*, pages 91–100. ACM, 2010.
13. Benoît Delahaye, Benoît Caillaud, and Axel Legay. Probabilistic contracts: a compositional reasoning methodology for the design of systems with stochastic and/or non-deterministic aspects. *Formal Meth. Syst. Design*, 38(1):1–32, 2011.
14. Benoît Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher, and Andrzej Wąsowski. Abstract probabilistic automata. In Ranjit Jhala and David A. Schmidt, editors, *VMCAI*, volume 6538 of *LNCS*, pages 324–339. Springer, 2011.
15. Uli Fahrenberg, Mathieu Acher, Axel Legay, and Andrzej Wąsowski. Sound merging and differencing for class diagrams. In Gnesi and Rensink [19], pages 63–78.
16. Uli Fahrenberg and Axel Legay. General quantitative specification theories with modal transition systems. *Acta Inf.*, 51(5):261–295, 2014.
17. Uli Fahrenberg and Axel Legay. The quantitative linear-time-branching-time spectrum. *Theor. Comput. Sci.*, 538:54–69, 2014.

18. Uli Fahrenberg, Axel Legay, and Andrzej Wąsowski. Make a difference! (Semantically). In Whittle et al. [31], pages 490–500.
19. Stefania Gnesi and Arend Rensink, editors. *Fundamental Approaches to Software Engineering - 17th Int. Conf., FASE 2014. Proceedings*, volume 8411 of *LNCS*. Springer, 2014.
20. Line Juhl, Kim G. Larsen, and Jiří Srba. Modal transition systems with weight intervals. *J. Log. Algebr. Program.*, 81(4):408–421, 2012.
21. Kyo Kang, Sholom Cohen, James Hess, William Nowak, and Spencer Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, CMU, 1990.
22. Kim G. Larsen. Modal specifications. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 232–246. Springer, 1989.
23. Kim G. Larsen, Axel Legay, Louis-Marie Traonouez, and Andrzej Wąsowski. Robust synthesis for real-time systems. *Theor. Comput. Sci.*, 515:96–122, 2014.
24. Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. A manifesto for semantic model differencing. In Jürgen Dingel and Arnor Solberg, editors, *MODELS Workshops*, volume 6627 of *LNCS*, pages 194–203. Springer, 2010.
25. Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. ADDiff: Semantic differencing for activity diagrams. In Tibor Gyimóthy and Andreas Zeller, editors, *SIGSOFT FSE*, pages 179–189. ACM, 2011.
26. Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. CDDiff: Semantic differencing for class diagrams. In Mira Mezini, editor, *ECOOP*, volume 6813 of *LNCS*, pages 230–254. Springer, 2011.
27. Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Semantically configurable consistency analysis for class and object diagrams. In Whittle et al. [31], pages 153–167.
28. MERgE ITEA2 project. http://merge-itea-project.irisa.fr/.
29. Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. Feature diagrams: A survey and a formal semantics. In *RE*, pages 136–145. IEEE Computer Society, 2006.
30. Ian Sommerville. *Software Engineering*. International computer science series. Addison-Wesley, 9th edition, 2010.
31. Jon Whittle, Tony Clark, and Thomas Kühne, editors. *Model Driven Engineering Languages and Systems, 14th Int. Conf., MODELS 2011. Proceedings*, volume 6981 of *LNCS*. Springer, 2011.