A Model Centered Perspective on Software-Intensive Systems

Heinrich C. Mayr[1], Judith Michael[2], Vladimir A. Shekhovtsov[1], Suneth Ranasinghe[1] and Claudia Steinberger[1]

Abstract: The aim of this paper contributing to resurrect research interest in conceptual modeling as a means for designing and producing software-intensive systems, as there is still no comprehensive and consistent use of conceptual modeling in practice. The idea is to see any software and information system as a construct consisting of model handlers (model consumers and/or producers). This leads to the paradigm of "Model Centered Architecture", which treats all processes, as well as the data they process, as instances of models. These models in turn are instances of metamodels, described using a particular domain specific modeling language (DSML), and represented using a corresponding domain specific representation language. Consequently, all system interfaces are defined through models (via an appropriate DSML) as well. The paper introduces the relevant MCA concepts and sketches open research questions in this field.

Keywords: Model Centered Architecture, Domain Specific Modeling, models@runtime, Language Hierarchies

1 Introduction

We start from the obvious fact, that any kind of data managed and/or processed within a software-intensive system, as well as the processes themselves, are instances of explicitly specified or implicitly underlying models and, clearly, are models again [ELP11]. We, therefore, see any software and information system as a construct consisting of model handlers (consumers and/or producers), which leads to the paradigm of "Model Centered Architecture (MCA)" [Ma17].

MCA can be seen as a generalization of Model Driven Architecture (MDA) [KWB03], Model Driven Software Development (MDSD) [Li11], and models@runtime [Be14]. MDA and MDSD deal with generating software through model transformation out of a (conceptual) requirements model. The Object Management Group (OMG) proposes to use the Unified Modeling Language (UML), the Meta Object Facility (MOF), the XML Metadata Interchange (XMI), and the Common Warehouse Meta-model (CWM) for this purpose. Models@runtime [Be14] uses models as artifacts at runtime with the goal to blur the boundaries between development time and runtime artifacts [BG10]. This allows for planning and executing maintenance and adaptation on model level [He15].

[1] Alpen-Adria-Universität Klagenfurt, <forename.surname@aau.at>
[2] Software Engineering, RWTH Aachen University, michael@se-rwth.de

Other initiatives focus on metaprogramming [SD12], domain specific modeling [KMM16], multilevel modeling [Fr14], and generative programming [CE00]. Metamodeling frameworks like ADOxx® [FK13] or language workbench and code generation frameworks like MontiCore [KRV10] support the definition of Domain Specific Modeling Languages (DSMLs) and the creation of related modeling tools. [Fr13] and [MM15] discuss the DSML creation as part of a comprehensive Domain Specific Modeling Method (DSMM).

Like multilevel modeling, MCA advocates, for any system aspect, the use of (possibly recursive) hierarchies of DSMLs, each embedded into a DSMM. I.e., MCA focuses on models (and their meta-models) in any development step up to the running system.

The aim of this paper is contributing to resurrect research interest in conceptual modeling as a powerful means for designing and producing software-intensive systems. For, despite of its power, there is still no comprehensive and consistent use of conceptual modeling in practice: for instance, in software development or business process management conceptual models mostly serve as prescriptive documents that, without synchronization with the developed artefact, stepwise diverge from reality.

The reasons for the restricted enthusiasm for modeling are manifold, so let's sketch only some of them: First, software developers generally seem to prefer working on the concrete and algorithms centered programming language level instead on the more abstract level of metamodels and models. This might be a consequence of insufficient and inadequate modeling education, in particular regarding domain specific modeling languages and their creation. Second, even when deploying generative approaches as cited above, mostly extensive manual interventions on the code level are necessary that not only jeopardize the accordance of models and code but also possible cost advantages. This carries even more, as these cost benefits are hard to anticipate due to the lack of appropriate assessment methods and therefore are considered as a risk.

With MCA, our long-term goal is to get rid of the need of such manual interventions so that system developers can concentrate on requirements analysis, metamodeling and modeling. This is why we want to attract attention to this field.

The paper's organization is as follows: Section 2 outlines the main MCA concepts. Section 3 discusses the variety of domain specific modeling and representation languages coming into play. Section 4 aims at drawing attention to some of the open issues.

Given the page limit, there will be no detailed discussion of related work beyond the already mentioned. However, when applicable, we will refer to work listed in section 5.

2 Model Centered Architecture

The basic idea of MCA is to treat all processes as well as the data they process (MOF level 0) as instances of models (MOF level 1). These models in turn are instances of meta-models (MOF level 2), described using a particular DSML, and represented using a corresponding domain specific representation language. Consequently, all system interfaces are defined through models (via an appropriate DSML) as well. This means, that the system components are seen as model handlers (producers and model consumers or both). Figure 1 shows the architectural patterns realizing such approach:

1. *Modeling tool pattern:* means for creating and managing models according to the given DSMLs; such tools are either custom built or generated using an existing meta-modeling framework like ADOxx (consumes meta-models and produces models).
2. *Model (and data) exchange interface pattern:* model transfer to runtime components.
3. *Model (and data) adapter pattern:* components transforming models/data into the format understood by the system, e.g., from an XML dialect to OWL.
4. *Model (and data) storage and storage manager patterns:* enable persistence.
5. *Model handler pattern:* use the adapted models to provide the functionality of the MCA-based system, which again leads to the production of models.

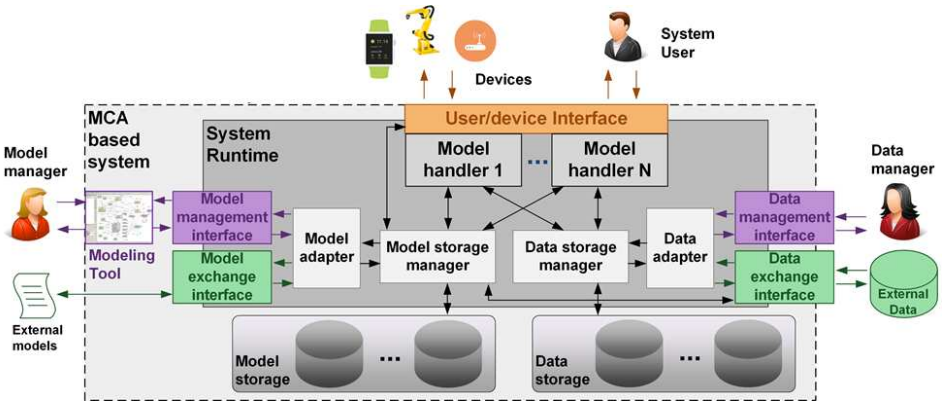

Fig. 1. Architecture of MCA-based systems

Figure 2 sketches the aspects conceptualized in the MOF hierarchy (for simplicity, we waive here the more complex view when using a multilevel modeling approach). (1) the application domain together with the application data exchange and user interfaces, each of them equipped with an appropriate DSML on level M2, and (2) the interfaces for model exchange (data on M0) and model management. The M2 interfaces allow for handling meta-models as MCA artifacts (meta-model management) and for integrating external meta-models (meta-model exchange). On the M1 (model) level the M2 meta-models are instantiated for a concrete application situation. On the M0 (instance) level, the application itself results from creating extensions of the M1 model elements.

If the application domain meta-model is comprehensive in the sense of providing concepts for structure, dynamics and functionality, the M0 extensions either form the models@runtime which might be handled by an interpreter (orchestrated by M2, visualized by the arrows from M2 to M0). Alternatively, when using a MDA tool, code may be generated from the M1 dynamic and function models. But even when following the conventional programming approach, effort may be saved by implementing the patterns as generic modules first and then reusing and adapting these for various projects.

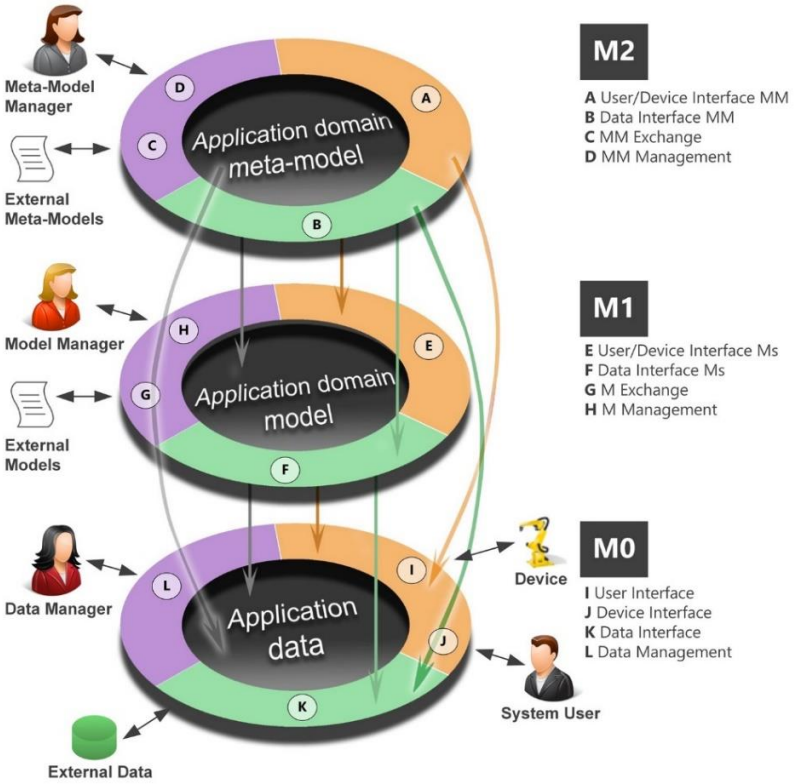

Fig. 2: MCA Model Hierarchy

Thus, MCA is a very flexible framework for creating complete software-intensive system by mere meta-modeling and modeling. For a proof of concept, we have successfully used and refined this approach in two large development projects: (1) QuASE [SM16], which provides flexible means for harmonizing stakeholder views and supports decisions in development and maintenance processes. (2) HBMS [Ma16], which provides ambient support services derived from integrated models of abilities, current context and episodic knowledge that an individual had or has, but has temporarily forgotten. The HBMS core, the Human Cognitive Model (HCM), preserves a person's episodic memory in the form

of comprehensive conceptual behavior models. The interfaces to activity recognition systems as well as multimodal user interfaces again are defined via DSMLs. In both projects we deployed ADOxx for DSML definition and modeling tool generation, and we reused model/data adapter and exchange module templates.

3 Language Hierarchies

The semantic concepts defined through the model hierarchy are to be represented by *syntactic constructs* of appropriate representation languages that again form a hierarchy as shown in Figure 3. We distinguish three levels in this language definition hierarchy:

The grammar definition level (top level of the hierarchy) contains the means of defining the language grammars. In our research, we use a specific version of EBNF, compatible with the ANTLR grammar definition language, for these means.

The language definition level defines grammars for the various representation languages (RL) related to the defined DSMLs: meta-meta-model RLs (not shown on Fig. 3 for brevity), meta-model RLs, model RLs and instance/data RLs.

The language usage level comprises the representations of the models of all levels.

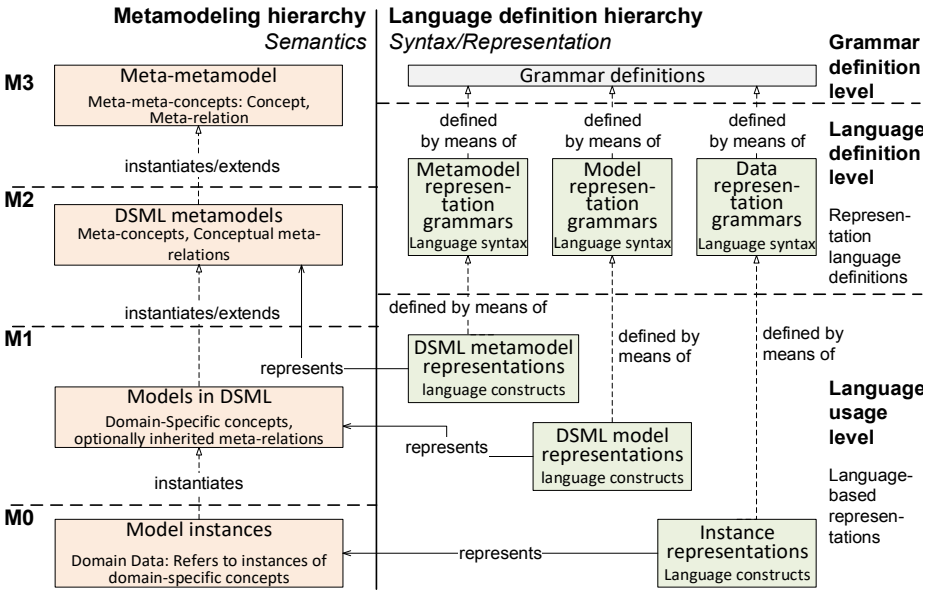

Fig. 3. Representation language definition hierarchy

This language definition hierarchy can also describe the ways of using existing languages for this purpose. For example, it is possible to use OWL 2 as a representation

language for interface or application concepts. In our projects, this proved to be very efficient, as it allowed us to reuse, among others, existing reasoning mechanisms for particular model handlers.

4 Research and Development Issues

While testing and refining the MCA approach in close linkage with the development of the aforementioned QuASE and HBMS systems, we identified a series of pending issues to be addressed by further research and development, among them:

1. Available metamodeling platforms and frameworks do not allow for handling several metamodels and DSMLs at the same time in the same production environment. This however, is essential for a smooth and integrated handling of various application domains, interfaces etc. to be supported by the same system under development.
2. For being comprehensive, such frameworks should be equipped by multi-model interpreting and/or code generating facilities.
3. For achieving acceptance in practice, a software process model as part of a MCA based development method has to be provided that, in particular, is aligned with Scrum to allow for agile modeling based [Ru17] development and flexible work division.
4. Similarly, an integrated method for cost assessment could help to alleviate the inhibitors in practice of changing to a model centered development paradigm.
5. Reusability of metamodel and language design might be fostered by extending MCA to the multi-level modeling paradigm [Fr14]. Within this context, a systematic analysis is needed which, and on which levels, concepts for covering quality, privacy and security issues should be provided by (meta-)models and languages. The same holds for concepts for interface modeling, as these are necessary for linking system parts in complex systems.
6. Ontological grounding on all levels might provide an even more stable semantic basis for development projects. The respective ontologies in turn, can be based on a common MCA foundational ontology. In addition, harmonization of views, notions and labels [LSM12] would enhance model understandability and quality.

We thank our reviewers for their valuable comments and advise, and we hope to initiate with this contribution a fruitful discussion on model centered system architecture.

5 References

[Be14] Bencomo, N., France, R.B., Cheng, B.H., Aßmann, U. (eds.): Models@runtime: Foundations, Applications, and Roadmaps, LNCS, Vol. 8378. Springer 2014

[BG10] Baresi, L., Ghezzi, C.: The Disappearing Boundary Between Development-Time and Run-Time. In Proc. FSE/SDP Workshop on Future of Software Engineering Research, pp. 17-22. ACM 2010

[CE 00] Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools,and Applications. Addison-Wesley Professional, Boston, Massachussets, USA 2000

[ELP11] Embley, D.W., Liddle, S.W., Pastor, O.: Conceptual-Model Programming: a Manifesto. In: Handbook of Conceptual Modeling, pp. 3-16. Springer 2011

[FK13] Fill, H.-G., Karagiannis, D.: On the conceptualisation of modelling methods using the ADOxx meta modelling platform. EMISA Journal 8, 4-25, 2013

[Fr13] Frank, U.: Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines. In: Domain Engineering, pp.133-157, Springer 2013

[Fr14] Frank, U.: Multilevel Modeling - Toward a New Paradigm of Conceptual Modeling and Information Systems Design. Bus Inf Syst Eng, 6(6), pp. 319-337, Springer Fachmedien, 2014

[He15] Heinrich, R. et al.: Runtime Architecture Models for Dynamic Adaptation and Evolution of Cloud Applications. Universität Kiel 2015

[KMM16] Karagiannis, D., Mayr, H.C., Mylopoulos, J. (eds.): Domain-Specific Conceptual Modeling: Concepts, Methods and Tools. Springer 2016

[KRV10] Krahn, H.; Rumpe, B.; Völkel, S.: MontiCore: a Framework for Compositional Development of Domain Specific Languages. In: Int. Journal on Software Tools for Technology Transfer (STTT), Volume 12, Issue 5, pp. 353-372, September 2010

[KWB03] Kleppe, A.G., Warmer, J.B., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publ. Co., Inc. 2003

[Li11] Liddle, S.W.: Model-Driven Software Development. In Handbook of Conceptual Modeling, pp. 17-54. Springer 2011

[LSM12] Leopold, H.; Smirnov, S.; Mendling, J.: On the refactoring of activity labels in business process models. Information Systems, 37(5), 443-459, 2012

[Ma16] Mayr, H. C. et al.: HCM-L: Domain-Specific Modeling for Active and Assisted Living. In: [KMM16], pp. 527-552, 2016

[Ma17] Mayr, H.C. et al: Model Centered Architecture. In Cabot, J. et al (eds.): Conceptual Modeling Perspectives, pp 85-104, Springer Nature 2017

[MM15] Michael, J., Mayr, H.C.: Creating a Domain Specific Modeling Method for Ambient Assistance. In: Proc. ICTer2015, pp. 119-124, IEEE 2015

[Ru17] Bernhard Rumpe: Agile Modeling with UML: Code Generation, Testing, Refactoring. Springer International, May 2017

[SM16] Shekhovtsov, V.A., Mayr, H.C.: View Harmonization in Software Processes: from the Idea to QuASE. In: Proc. INFORMATIK 2016, pp. 111-123, LNI, Vol. P-259, 2016

[SD12] Štuikys, V., Damaševičius, R.: Meta-Programming and Model-Driven Meta-Program Development: Principles, Processes and Techniques, Springer 2012