

Language Engineering for Multi-Level Modeling (LE4MM): A Long-Term Project to Promote the Integrated Development of Languages, Models and Code

Ulrich Frank^{1,*}, Tony Clark²

¹University of Duisburg-Essen, 45141 Essen, Germany

²Aston University, B4 7ET Birmingham, UK

Abstract

In this paper we present a long-term research project that started in 2011. *Language Engineering for Multi-Level Modeling* (LE4MM) combines research on fundamental challenges of conceptual modeling and software engineering with the design and implementation of a tool environment. This interplay between research and development has proved very fruitful. Notwithstanding, running a research project over a period of more than 10 years, especially when it involves software development, faces a number of specific challenges. While the main focus of the paper is on research questions and findings, we will also present lessons learned in the project.

Keywords

language architecture, model-driven development, DSML, reuse, design conflict

1. Introduction

It all started at the conference dinner of the ER conference 2010 in Vancouver. A German professor told the round table about a problem that had been bothering him for some time and for which he could not find a solution. The problem was due, among other things, to specific limitations of object-oriented programming languages. After he had to realize that this topic had not elicited the hoped-for response at the table, he refrained from going into further detail. But then his table neighbor, a professor from the UK, pointed out that he might have something that could be used to address the problem – and, indeed, he had. A short time later, the conception of a joint research project emerged. The project began in 2011, initially financed from existing funds of the participating research groups. During the first years, a PhD student and a PostDoc worked part-time in the project. From 2015 on, a full-time developer joined the team, which included an ever changing group of doctoral students and student assistants.

The project is based on two main pillars that originate from previous work of its two founders.

CAiSE 2023 : Research Project Exhibition, June 12–16, 2022, Zaragoza, Spain

*Corresponding author.


✉ ulrich.frank@uni-due.de (U. Frank); tony.clark@aston.ac.uk (T. Clark)

🌐 <https://www.umo.wiwi.uni-due.de/en/> (U. Frank); <https://research.aston.ac.uk/en/persons/tony-clark> (T. Clark)

📞 0000-0002-8057-1836 (U. Frank); 0000-0003-3167-0739 (T. Clark)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

Tony Clark's extensive work on reflexive language architectures, on language engineering and the language engineering workbench *XModeler* [1, 2, 3] proved to be an extremely viable basis for the further development of methods and tools. Ulrich Frank's previous work focused on the development of domain-specific modeling language (DSMLs) [4], methods and tools for conceptual modeling with specific emphasis on enterprise modeling. Among other things it led to a comprehensive method for enterprise modeling comprising of various integrated DSMLs [5], e.g., for goal modeling [6], for business process modeling [7], for modeling organization structures [8] and IT infrastructures [9], as well as corresponding modeling environments [10, 11]. Despite the results achieved, this work has also been a constant source of frustration because it has been constrained by fundamental limitations of existing language architectures, both modeling languages and programming languages.

From the beginning, the project was driven by the conviction that bringing these two research directions together would lead to significant synergies. To show that this hope was justified, we will first present the motivation on which the project is based. We then outline selected project goals before presenting and discussing key results. A comprehensive description of the project, including publications, videos, the language engineering environment and exemplary applications/models are available at the project's webpages <https://le4mm.org>.

Due to space limitations, we had to refrain from inserting graphics, even though they would be useful to support an appropriate understanding of the project. To compensate, we have included several references to graphics and videos available on the project's web pages.

2. Motivation

The project is motivated by a vision of application system architectures and the way they are developed, used and maintained. It evolved from our experience with developing modeling languages and accompanying tools, which led us to the conviction that the entire lifecycle of application systems benefits significantly from domain-specific (modeling as well as programming) languages and corresponding models. They should not only be accessible by developers, but also by users. Thus, they support the involvement of users during analysis and design. In addition, if they are provided as representation of an application system during its use and maintenance, they contribute to empowering users, since users get the chance to understand relevant aspects of the software they work with, which otherwise would widely remain a black box. A specific manifestation of this vision, referred to as "self-referential enterprise system", is outlined in [12]. It comprises an architecture of an enterprise software system that is integrated with a conceptual model of itself and of the relevant context it operates in, that is, models of the corporate goal system, of business processes and project, of the organization structure, etc. In other words: the enterprise software is integrated with an enterprise model during its entire life time.

Even though we regard this vision as extremely attractive, its realization is fraught with a number of question marks. They relate mainly to technical and economic challenges, but also to supposed reservations about modeling in practice. From a technical perspective, the integration of models and code during a long period of time is hampered by the notorious synchronization problem. Also, the economics of such an architecture may be seen as disadvantageous, since

the costs of developing and maintaining large conceptual models add to those required for the development and maintenance of code. Finally, the popularity of conceptual modeling in practice is limited, which raises doubts about whether the outlined version is suited to convince developers and users.

It has been known for long that DSMLs are suited to promote both the productivity of modelers and the quality of models. However, the design and implementation of DSMLs is a time-consuming process – especially if it starts from scratch with rudimentary general-purpose meta languages. Such an approach is in obvious contrast to the development of technical languages in professional domains: instead of starting from scratch, a more specific technical language, e.g., to describe a new production process, will usually be defined in terms of an existing, more general technical language, which does not only promote productivity by reusing existing domain-specific concepts, but also language quality, since the more general DSML will constrain the range of unreasonable specifications.

In order to promote reuse and adaptability, the design of DSMLs should aim at expressing domain knowledge that is regarded as invariant at the highest possible level of abstraction. Otherwise, it would have to be repeated at lower levels, resulting in redundancy. Unfortunately, the implementation of this principle is hindered by the insufficient expressive power of language architectures such as MOF. For example: when we design a process modeling language, we know that every process instance has a start and a stop time, but it is not possible to express this knowledge within the metamodel that is used to specify the process modeling language.

Reference models are a promising approach to significantly increase the economic efficiency of modeling: lower development costs go hand in hand with higher quality. However, despite the remarkable attention, reference models received at the beginning [13], they never took off. While various reasons may have prevented the success story that reference models were supposed to write, their wide-spread use certainly suffered from principal design conflicts that are characteristic for many modeling and software development projects. On the one hand, restricting an artefact to more general requirements promotes its range of reuse and, hence, economies of scale. On the other hand, making it more specific contributes to its utility in those cases, where it fits. This well-known “power/generalization trade-off” [14, p. 71 ff.] marks a conflict between integrity and adaptability, too, see [15].

3. Research Goals

The problems that hinder the realization of the vision outlined above led to ambitious research goals, which, however, seemed not exaggerated given the power of the previously developed reflexive language architecture, the XModeler is based on. Apart from the goals concerning the basic language architecture (1-4), the research objectives outlined below emerged during the course of the project.

Goal 1: Enable an arbitrary number of classification levels and allow for deferred instantiation. *Rationale:* The design of conceptual models and especially the design of DSMLs is often confronted with the problem that commonalities discovered within a range of classes cannot be expressed properly through generalization [15]. In these cases, additional levels of classification are suited to provide for the required abstraction. Achieving goals 1 and 2 is a prerequisite for

being able to specify DSMLs with less specific DSMLs (see goal 3).

Goal 2: Allow for deferred instantiation. *Rationale:* Sometimes knowledge about a range of objects or classes is available already above their direct (meta) classes. If this knowledge is expressed through properties such as attributes, operations or associations, it is required to make sure that these do not apply to direct instances, but only to instances of classes further down the instantiation chain. For example: a language for modeling documents may include the meta-class **Document**. At this level, we know already that every instance of a specific document class, e.g. **MasterThesis**, has a certain page count. But this could not be expressed without deferred instantiation, that is, without the possibility to define properties at level Mn that are to be instantiated only at a level below Mn-1.

Goal 3: Provide comprehensive support for the design and implementation of DSMLs – including concrete syntax. *Rationale:* The unchallenged advantages of DSMLs are offset by the considerable effort required to create them. Two more specific goals are suited to clearly reduce this effort: the development of a comprehensive language engineering environment and a language architecture that enables the definition of a DSML with a less specific DSML.

Goal 4: Enable a common representation of models and programs. *Rationale:* A common representation of models and programs is the best option to avoid effort and risk related to the synchronization of models and code. Among other things, it requires a programming language that allows classes at any level to serve simultaneously as objects.

Goal 5: Promote the design and use of reference models through additional abstraction. *Rationale:* Achieving this goal, which would include mitigating principle design conflicts, is suited to invigorate reference models and, hence, to clearly improve reuse.

Goal 6: Different ways of representing models/objects and interacting with them should be supported. *Rationale:* User preferences regarding diagrams, browsers, or other kinds of GUIs vary. Also, there are use cases where it may make sense to combine different modes of representation.

Goal 7: Allow for navigation at runtime. *Rationale:* Enabling users to navigate to models of the software they use – as well as to the corresponding meta-models – is an obvious contribution to their empowerment. This is especially the case, if the models they can navigate to are created with a DSML.

Goal 8: Enable changes at runtime. *Rationale:* Users who are sufficiently qualified and authorized could adapt an application system to changing requirements by changing models they are familiar with. This would clearly improve an organization's agility, since there would be no need for a lengthy detour through developers. Changes are especially challenging, if they concern deletion, also because of multiple dependencies between objects on different levels.

Goal 9: Provide a method to guide the design of DSMLs and models. *Rationale:* Traditional approaches to conceptual modeling are not sufficient to handle additional abstraction, as it is enabled by multiple levels of classification and deferred instantiation. It is especially important to aim at invariant relationships between levels.

Goal 10: Support the development of process models at different levels of classification. *Rationale:* Current process modeling languages suffer from a lack of abstraction. They offer only poor reuse. Enabling domain-specific process modeling languages are suited to provide more reuse and guidance.

Goal 11: Provide an approach to model-based application development that represents an

alternative to current low-code platforms. *Rationale:* Low-code platforms are based on the assumption that it suits users to focus on the design of a GUI with the underlying data/object models being created implicitly. While this assumption may be appropriate for many users, starting with GUI design has obvious weaknesses.

4. Results to Date

It was clear from the outset that achieving the project's goals would require some kind of *multi-level modeling* approach. Multi-level modeling was introduced by the pioneering work of Kühne and Atkinson at the beginning of the millennium [16], with ancestors that go back even further, cf. [17, 18, 19]. The various approaches that have evolved since then have in common that they allow for multiple classification levels and for classes having state. However, none of the existing approaches fulfilled our requirements sufficiently, since they focus on modeling only and did not include a programming language, which is essential for achieving goal 4. In addition to presenting an overview of the project's results, we will also briefly report on the experience gathered during the course of the project.

4.1. Languages, Models, and Tools

Since the scope of this paper is limited, we can only present an overview of the project results here. As this is hardly sufficient to allow a proper understanding of the main contributions of the project, the text includes references to additional material provided on the project webpages.

The main results that have been achieved so far comprise the FMML^x (Flexible and executable Multi-Level Language) [20], the XModeler^{ML} [21], a comprehensive language engineering, modeling and execution environment, and an accompanying design method (goal 7) [22]. The FMML^x is specified through an extension of XCore, the meta model of the previously developed XModeler [2, p. 40]. XCore represents the class-based type system of XOCL, an object-oriented “superlanguage” [1]. While XOCL already allowed the specification of classes at any level, the FMML^x enables assigning a specific level to a class and to define deferred instantiation. The metamodel shown at <https://le4mm.org/xmodelerml/#FMMLx> defines the concepts offered by the FMML^x. The example model at the same page illustrates the use of these concepts. It also illustrates the integration of executable languages, models and objects at M0 within one multi-level model. The demo “DSML for IT Management” (<https://le4mm.org/xmodelerml/#examples-1>) shows a multi-level model that integrates DSMLs at different levels, models and objects at M0. The classes at level 3 represent the specification of a more general DSML for modeling IT infrastructures. This DSML then serves the specification of a more specific DSML at level 2, which in turn is used to create the model at level 1. The objects instantiated from that model, e.g., one representing a particular desktop computer, are also part of the multi-level model. Since every class, no matter at what level, is an object, too, its operations can be executed and the returned values can be shown in the diagram. Whenever a class is added to the diagram, the diagram editor's palette is updated.

In addition to a diagram editor, the interaction with objects can be done through an object browser, an object editor, or through some other kind of GUI (goal 6). A further component

enables the convenient definition of graphical notations and their integration with the corresponding abstract syntax and semantics (see demo at <https://le4mm.org/xmodelerml/#Demo-1>). From this it follows that goals 1-4 are achieved, enabling the realisation of self-referential enterprise systems (see video and demo at <https://le4mm.org/xmodelerml/#Focus-on-Runtime>, <https://le4mm.org/xmodelerml/#Demo-2>). The language architecture featured by the XModeler^{ML} also allows for designing reference models that relax the power/generalizability trade-off. While higher level languages allow for a wide range of reuse, thus fostering economies of scale, more specific languages and models increase reuse productivity.

To support non-programmers with developing applications, we chose an approach that is different from prevalent low-code platforms (goal 11). Users start with developing a model of their domain using a DSML of choice. If the specification of classes involves the implementation of elaborate operations, the support of professional developers is required. Once the model is complete, a default GUI is generated and exported to an external GUI designer that allows the user to rearrange and polish the GUI elements. The finalized GUI is sent back to the XModeler^{ML} where it is transparently integrated with the model that constitutes the application. The current implementation [21] is in an early stage and subject of future research. For a demo, see screencast at <https://le4mm.org/xmodelerml/#advancedGUI>.

Multiple classification levels, deferred instantiation as well as the fact that the strict separation of language and model is removed mark an obvious and potentially confusing difference to traditional approaches to modeling. Therefore, multi-level modeling can be regarded as a new paradigm of modeling and software design in general. The sudden introduction of a comprehensive multi-level language engineering and execution environment is associated with the risk of overwhelming users. We have made corresponding experiences in our courses as well as in the exchange with professional software developers. An interactive introduction that gradually reveals the benefits will therefore often be wiser. To this end, one could start with using the XModeler^{ML} as a UML class diagram editor. It offers an immediate advantage over regular UML editors, since it allows the instantiation of models into objects and the execution of those within the diagram editor (see screencast at <https://le4mm.org/xmodelerml/#UML-pp>). Subsequently, multi-level concepts can be introduced step by step (see further screencast on same page).

To explore the potential of multi-level modeling and of the XModeler^{ML} in particular, it is recommended downloading it from <https://le4mm.org/xmodelerml/#download>. In addition to screencasts that guide the installation, the distribution includes several executable models.

4.2. Lessons Learned

It is demanding to run a research project at universities for more than ten years. This applies both to the persistence required and to securing funding. We were fortunate in both respects. We had access to various resources over the course of time that allowed us to fund the project throughout. Currently, our research is supported by a generous donation from a large American software company. Nevertheless, it was only possible to give the project a high priority over such a long period of time, because our enthusiasm for the project has not waned over the years.

In retrospect, the following factors have contributed to the project's persistence and productivity. The work in the project was largely oriented to the guiding principle of research through

development. Research questions were accompanied by challenging development tasks, which often led to new research questions. In addition to long-term goals such as the realization of a self-referential enterprise system, the development work regularly enabled tangible successes, which kept team motivation high. Furthermore, the small, but enthusiastic multi-level modeling community proved to be very helpful in this respect. The MULTI Workshop series provides a forum for exchanging and validating ideas. With its regular modeling challenges it fosters a healthy competition.

Finally, the inclusion of the topic in the teaching program proved to be very beneficial. On the one hand, teaching forced us to further specify the terminology used, and on the other hand, it allowed us to recruit a number of students to work on the project.

5. Conclusions and Future Work

So far, the project has been extremely profitable for all participants. It has enabled us to gain a number of important insights. The results achieved so far exceed our original expectations – and we are still excited about remaining challenges that will keep us busy for years to come.

Among the most prominent research goals that will occupy us in the near future are applying multi-level modeling to process modeling and the specification of contingent level classes, that is, classes that may change their level with the context they are used in, thus increasing the range of their possible reuse.

References

- [1] T. Clark, P. Sammut, J. Willans, *Superlanguages: Developing Languages and Applications with XMF*, Ceteva, 2008.
- [2] T. Clark, P. Sammut, J. Willans, *Applied Metamodeling: A Foundation for Language Driven Development*, 2 ed., Ceteva, 2008.
- [3] T. Clark, J. Willans, *Software Language Engineering with XMF and XModeler*, in: I. R. Management Association (Ed.), *Computational Linguistics*, IGI Global, Hershey, 2014, pp. 866–896.
- [4] U. Frank, *Domain-Specific Modeling Languages - Requirements Analysis and Design Guidelines*, in: Iris Reinhartz-Berger, Aron Sturm, et al. (Eds.), *Domain Engineering: Product Lines, Conceptual Models, and Languages*, Springer, 2013, pp. 133–157.
- [5] U. Frank, *Multi-Perspective Enterprise Modeling: Foundational Concepts, Prospects and Future Research Challenges*, *Software & Systems Modeling* 13 (2014) 941–962.
- [6] S. Overbeek, U. Frank, C. A. Köhling, *A Language for Multi-Perspective Goal Modelling: Challenges, Requirements and Solutions*, *Computer Standards & Interfaces* 38 (2015) 1–16.
- [7] U. Frank, *MEMO Organisation Modelling Language (2): Focus on Business Processes*, Technical Report 49, Institute for Computer Science and Business Informatics (ICB), University of Duisburg-Essen, 2011.
- [8] U. Frank, *MEMO Organisation Modelling Language (1): Focus on Organisational Structure*, Technical Report 48, Institute for Computer Science and Business Informatics (ICB), University of Duisburg-Essen, 2011.

- [9] U. Frank, M. Kaczmarek-Heß, S. D. Kinderen, IT Infrastructure Modeling Language (ITML): A DSML for Supporting IT Management, Technical Report, Institute for Computer Science and Business Informatics (ICB), University of Duisburg-Essen, 2021.
- [10] J. Gulden, U. Frank, MEMOCenterNG – A Full-Featured Modeling Environment for Organisation mModeling and Model-Driven Software Development, in: Proceedings of the 2nd International Workshop on Future Trends of Model-Driven Development (FTMDD 2010), 2010.
- [11] A. Bock, U. Frank, Multi-perspective Enterprise Modeling—Conceptual Foundation and Implementation with ADOxx, in: D. Karagiannis, H. C. Mayr, J. P. Mylopoulos (Eds.), Domain-Specific Conceptual Modeling, Springer, Cham, 2016, pp. 241–267.
- [12] U. Frank, S. Strecker, Beyond ERP Systems: An Outline of Self-Referential Enterprise Systems: Requirements, Conceptual Foundation and Design Options, Technical Report 31, Institute for Computer Science and Business Informatics (ICB), Universität Duisburg-Essen, 2009.
- [13] J. Becker, P. Delfmann (Eds.), Reference modeling: Efficient information systems design through reuse of information models, Physica Verlag, Heidelberg, 2007.
- [14] A. Newell, Heuristic Programming: Ill-Structured Problems, in: J. S. Aronofsky (Ed.), Progress in Operations Research. Relationship Between Operations Research and the Computer, Wiley, New York, 1969, pp. 361–414.
- [15] U. Frank, Multi-level Modeling: Cornerstones of a Rationale, Software and Systems Modeling 21 (2022) 451–480.
- [16] C. Atkinson, T. Kühne, The Essence of Multilevel Metamodeling, in: M. Gorgolla, C. Kobryn (Eds.), UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools, Lecture Notes in Computer Science, Springer, Berlin et al., 2001, pp. 19–33.
- [17] J. J. Odell, Power Types, Journal of Object-Oriented Programming 7 (1994) 8–12.
- [18] A. Pirotte, E. Zimányi, D. Massart, T. Yakusheva, Materialization: A Powerful and Ubiquitous Abstraction Pattern, in: J. B. Bocca, M. Jarke, C. Zaniolo (Eds.), Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 1994, pp. 630–641.
- [19] M. Jarke, S. Eherer, R. Gallersdörfer, M. Jeusfeld, M. Staudt, ConceptBase – A Deductive Object Base for Meta Data Management, Journal of Intelligent Information Systems 4 (1995) 167–192.
- [20] U. Frank, Multilevel Modeling: Toward a New Paradigm of Conceptual Modeling and Information Systems Design, Business and Information Systems Engineering 6 (2014) 319–337.
- [21] U. Frank, L. L. Mattei, T. Clark, D. Töpel, Beyond Low Code Platforms: The XModeler^{ML} - an Integrated Multi-Level Modeling and Execution Environment, in: J. Michael, J. Pfeiffer, A. Wortmann (Eds.), Proceedings of the Modellierung 2022 Satellite Events, GI, 2022, pp. 235–244.
- [22] U. Frank, Prolegomena of a Multi-Level Modeling Method Illustrated with the FMML^x, in: Proceedings of the 24th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, IEEE, 2021.