Specification and Management of Methods - A Case for Multi-Level Modelling

Ulrich Frank[0000−0002−8057−1836]

University of Duisburg-Essen
ulrich.frank@uni-due.de
https://www.umo.wiwi.uni-due.de/en

Abstract. The digital transformation creates an increasing demand for projects to prepare and realize change. The professional management of projects demands for methods. In particular, there is not alone need for method engineering, but also for managing the use of methods and for method maintenance. In this paper, it will be shown that traditional approaches to method engineering are not only limited with respect to reuse, they also do not support the integration of method engineering and method management. The approach presented in this paper addresses these limitations. It is based on a mult-level language architecture, which enables the common representation of models and code.

Keywords: Multi-level modelling · language engineering · method engineering.

1 Introduction: The Need for Methods

In times of the digital transformation, many organizations need to regularly adapt their products, their operations and possibly their entire business model to stay competitive. In most of these cases, the adaptation will involve the conjoint analysis and modification of an organization's action system and its information system. Corresponding projects do not only target a contingent and challenging subject, they also require remarkable skills and substantial resources. Most organizations are not capable of staffing and managing those projects on their own. As a consequence, a huge consultancy industry has evolved over the last decades. The major companies in this industry alone employ hundreds of thousands of consultants, many of which are still novices or not far above. Therefore, it is essential for these firms as well as for other organizations running projects under similar conditions to promote a professional management of projects. Among other things, that recommends the use of appropriate *methods*. A method is suited to foster the reuse of existing knowledge, to reduce risk and, hence, to promote the economics of projects. However, general-purpose methods that can be used in a wide range of projects will usually not be satisfactory. Instead, it seems more appropriate to focus on domain-specific methods. For this purpose, a consultancy firm would provide a repository of

2   U. Frank

methods, a project manager can choose from and, if required, add modifications. An example of such an approach is IBM's "Component Business Modeling (https://www.ibm.com/downloads/cas/6NMP1WEP). In any case, the management of method repositories faces serious challenges. On the one hand, it should allow for convenient adaptations of existing methods. On the other hand, it needs to ensure the integrity of a repository. While copy&paste might be regarded as a convenient way of creating new methods from existing ones, it is suited to create a maintenance nightmare. In academia, these challenges have been known for long. The field of method engineering is based on the assumption that the construction of particular methods should follow an engineering approach, which among other things recommends accounting for linguistic rigor, consistency and coherence as well as for the development of supportive tools. During the last 20 years, a plethora of approaches originating mostly in Requirements Engineering and Software Engineering have evolved (for an intermediate overview see [15]). The field has reached a stage of moderate maturity, which is also indicated by the existence of a respective ISO standard [1]. At the same time, it seems that research interest in method engineering has clearly declined during the last years. With respect to the current and further growing relevance of methods for mastering the digital transformation, this seems unfortunate. Against this background, the paper is intended to contribute to the revival of method engineering. It is structured as follows. First, an analysis of foundational terms will conclude with essential requirements related to the specification and management of modelling methods. Then, it will be shown that traditional approaches to meta-modelling have serious limitations that clearly compromise the specification and use of methods. Subsequently, a multi-level approach to the specification, modification and management of methods is presented. It improves reuse and adaptability. Furthermore, it enables the integration of method specification, method use, and project management.

2   Conceptual Foundation and Essential Requirements

Often, the term method is defined with respect to purpose: a method is aimed at solving a class of problems. However, such a functional definition is not sufficient for an approach to guide the specification and management of methods. To that end, a concept of method is required that reflects its constitutional elements. Only then, it is possible to specify a method referring to these elements and to represent a method in a repository. And only then, we can develop requirements to be satisfied by approaches that target the specification and management of methods.

2.1   Terminology

In systems development and method engineering various definitions of the term method can be found. "A method is based on models (systems of concepts) and consists of a number of steps which must/should be executed in a given

order." [20, p. 7] While this definition could be misinterpreted in the sense that, e.g., a particular data model could be constitutive for a method, its intention seems to correspond to that of the definition proposed by Karagiannis and Fill who regard a modelling method as being composed of a "modelling language and a modelling procedure" [7, p. 8] Instead of using the term "modelling language", Lyytinen speaks of "a multitude of conceptual structures to describe, interpret and prescribe a field of phenomena" [16, p. 5]. In line with these definitions, we shall regard a method in general as consisting of a linguistic structure and a process model. A linguistic structure such as a technical terminology defines concepts that allow for structuring the problem domain in a purposeful way. In the field of system development, the concepts should be suited to structure both, the system to be built and the domain it is supposed to represent. A process model provides guidelines for how to proceed with developing a solution. A *modelling* method is a refinement of the general term. It consists of one or more modelling languages and a corresponding process model (for a more comprehensive description see [9, p. 40]).

This conception of (modelling) method focusses the syntax and semantics of the specification that describes a method as an artefact. In addition to that, the pragmatics of a method needs to be accounted for. The pragmatics of a method results from the practices it is used in. These practices may be in line with the guidelines specified with the artefact or may deviate from them. People may only pay lipservice to those guidelines, misinterpret them, or develop their own workarounds. Understanding these pragmatic aspects of a method is of pivotal relevance for their success, but not of particular relevance for the focus of our investigation.

2.2   Requirements

Metamodels are a common approach to specify the abstract syntax and semantics of modelling languages. A process model could be created with a modelling language designed for that purpose. Hence, methods could be specified as (meta) models: on the one hand, metamodels would represent the modelling languages that are being used in a method. On the other hand, a metamodel would be used to represent the corresponding process. The modelling method would then result from an integration of these metamodels, which could be stored and managed in model repositories. While this conclusion is not inappropriate, it would be wrong to regard it as the solution to our problem. An approach to the specification, use and management of modelling methods should account for the following generic requirements.

R1 - Support for reuse: the approach should feature abstraction concepts that foster reuse. In an ideal case, it should be possible to reuse all knowledge available in a domain for the specification of a method. *Rationale*: reuse of mature knowledge does not only reduce the costs of developing methods substantially, it should also contribute to method quality. This requirements has been at the core of research on method engineering for long.

4   U. Frank

R2 - Relax conflict between range of reuse and productivity of reuse: with respect to economies of scale, a method should have a wide range of reuse. That recommends the construction of methods which are not designed for specific purposes, but that cover a wider range of possible project types. However, the more generic a method is, the lower is its contribution to productivity and integrity of a particular project. *Rationale*: relaxing this conflict promises clear economic advantages. One could benefit from economies of scale without giving up on the customized methods that are designed to specific needs. This conflict also reflects a practical problem that every language designer is confronted with. For every relevant concept in a domain, it has to be decided whether it should be part of the language or rather be specified with a language. While there are a few guidelines that support this decision, none of those is entirely convincing ([10]. For example: a concept like "Desktop Computer" could be part of a language for modelling IT infrastructures. Alternatively, it could be modelled as an instance of the more generic concept "Computer".

R3 - Support for integrity: an approach to specify methods should guide with the construction of methods that are consistent and coherent. *Rationale*: a method that lacks important aspects or that includes conflicting elements is likely to cause problems. Furthermore, method representations that lacks integrity compromise storing and retrieving of methods.

R4 - Integration of method and method use: the representation of a method's use, that is, a particular project should be integrated with a representation of the method. *Rationale*: method management is not restricted to the specification and dissemination of methods. It also includes the support of particular projects that use a method and their monitoring. Monitoring is required to assess a project's performance and to contribute to its improvement. If the representations of a method and its use are integrated, it is possible to use a method as a foundation for project management and to enable navigation between the two levels of abstraction.

As we shall see, satisfying these requirements is confronted with serious challenges.

3   Pitfalls of the Traditional Meta-Modelling Paradigm

Research on method engineering has resulted in various concepts to foster method configuration and reuse of existing knowledge. It also produced metamodelling tools that feature the convenient and fast realization of specific model editors.

3.1   Patterns of Reuse and Tools

Classification is probably the most prominent abstraction concept to foster reuse in method engineering. It is addressed by the use of metamodels. A metamodel provides foundational concepts that can be instantiated to specify particular methods. In addition, composition is often referred to as a measure to achieve

reuse. In that case, components of methods, referred to as "chunks" or "fragments", are stored in a repository. There seems to be no common definition of chunks and fragments. Henderson-Sellers et al. [13] suggest that a fragment can either represent a part of a process that constitutes a method or part of the product, that is, the documents (models, code ...) the creation of which a method is aimed at. According to this terminology, a chunk represents an aggregation of a fragment that represents part of a process with a corresponding fragment that represents part of a product. Both, classification and composition are well-known approaches to promote reuse in conceptual modelling. The use of metamodels corresponds to the construction of modelling languages. Reuse of knowledge is especially effective in the case of domain-specific modelling languages (DSML).

Both, instantiation and composition are supported by specific tools. Metamodelling environments such as Eclipse, MetaEdit [17], or ADOxx [7] enable the fast creation of modelling tools. For this purpose they take the metamodel that specifies the abstract syntax and semantics of a modelling language as an input. They also provide specific tools for the specification of the concrete syntax. Based on that, they generate a corresponding model editor. These tools usually focus on model editors and do not provide specific support for the specification of corresponding process models. Tools that focus on composition usually feature repositories. Chunks and fragments can be retrieved from a repository, e.g., through faceted classification [18], and somehow composed to a full method. However, the composition seems to be based on copy, paste & adapt [13].

Tools for method engineering are focussed on the specification of modelling languages and, in part, the configuration of methods. They usually do not allow for monitoring the use of a method, that is, they do not provide specific support for project management. However, this is not the case, because the use of a method is regarded as being out of scope. Instead, the application of methods is explicitly accounted for by various authors (e.g., [14], [4]) as well as in the ISO 24744 standard. The authors speak of "endeavour" in the sense of a particular project that is instantiated from a method. As we shall see, there is a principal reason, why the instantiation of projects (or actual uses of methods) from a method is not covered by metamodelling tools: the semantics of prevalent object-oriented programming languages.

3.2   Limitations

To illustrate limitations of traditional approaches to method engineering we look at the fictitious example of a large consultancy firm that wants to develop a method repository. The simplified metamodel depicted in fig. 1 serves as a language for defining specific methods. It can also be seen as a schema for storing methods. Note that this metamodel is not to be seen as a solution but as an illustration of a problem.
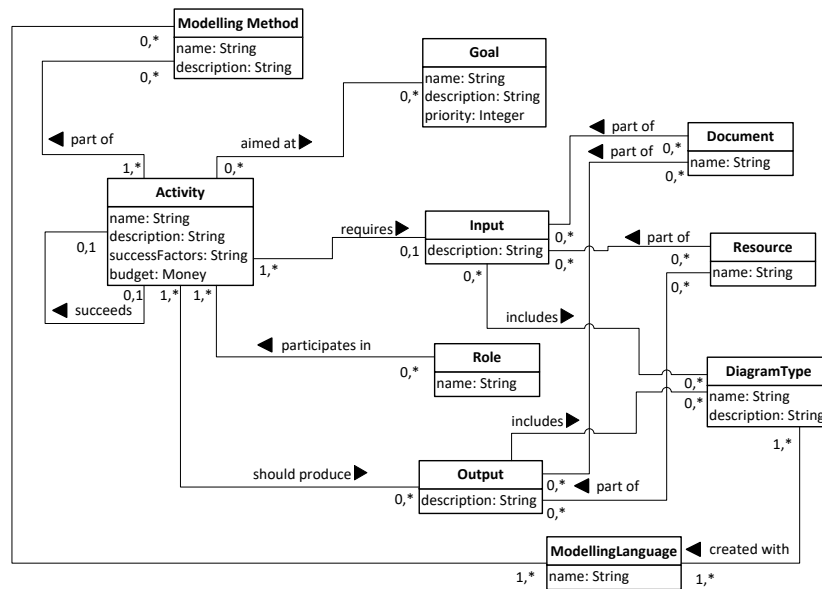
6   U. Frank

Fig. 1. Prototypical metamodel for the specification of modelling methods

Let us assume the metamodel is used to define a method to guide the selection of an ERP system. The process model in fig. 2 and the description of a selected activity in the process illustrate the method.
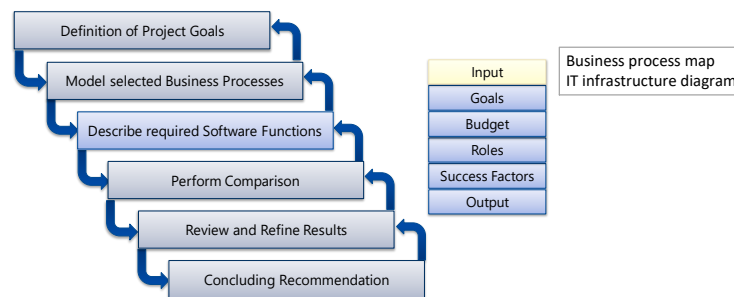

Fig. 2. Illustration of process model that is part of a modelling method

Now let us see, how the method could be instantiated from the metamodel. Apparently, the activities that form the waterfall are instantiated from the meta-class `Activity`. The object that serves to represent the required input is instantiated from `Input` and linked to instances of `DiagramType` each of which would refer to one or more representations of modelling languages, instantiated from

the metaclass `ModellingLanguage`. Similarly, one could instantiate objects to represent roles, goals, budget, resources, etc. But, maybe, this would not be sufficient. For example, a proper application of the method might require accounting for certain states. A business process map that is needed as input should, e.g., be in a state that includes the definition of certain performance indicators for each process type. Adding an attribute like "state" to `DiagramType` does not work, since we are not interested in the state of a diagram type, but rather in that of a particular diagram. While we could overload the object that represents a diagram type to also represent particular diagrams that would clearly compromise model integrity. It would also be impossible to define that two different instances of a diagram type were required in different states. Using two different meta types like "Diagram" and "DiagramType" would result in a loss of semanitics. This lack of expressiveness becomes even more obvious, if a particular method use should be represented. In that case, an instance of the metamodel would have to be further instantiated. While we know that each acitivity has a certain start and termination time, those cannot be expressed in the metamodel. Also, it is not possible to specify the actual budget available in a particular activity – or whether it was exceeded or not. Furthermore, it would be important to assign actors to roles. For this purpose, it would be required to instantiate a role type like "domain expert", which was instantiated from `Role`, into a particular role instance and link it to an instance of a class `Actor`. However, again that would not be possible, because in MOF like architectures the instantiation of a class cannot be further instantiated.

An approach to reuse method fragments would correspond to instantiate parts of a metamodel and reuse them with further instantiations. For example, a particular instantiation of `Activity` and associated instances of `Goal`, `Input`, `Role`, etc. could be stored in a repository and used for the creation of a new method. However, the challenge to avoid copy&paste semantics and the accompanying redundancy would remain: it is unlikely that the state of the components in the repository will be invariantly the same across all reuse cases. For example, within one method activity b may follow activity a, while another method requires activity b to follow activity x.

A further shortcoming of the traditional approach to method engineering becomes not immediately apparent. Imagine, a large company that frequently runs projects in areas such as software deployment, IT management, integration of IT infrastructures, etc. is using a metamodel like the one shown in fig. 1. Each new method would have to be created from scratch using the rather generic concepts defined in the metamodel – maybe supplemented by using existing method fragments through copy&paste. Imagine, the company would instead use more specific meta modelling languages to define particular methods. There could, for example, be a meta-metamodel that reflects common knowledge about software development methods. It may include a meta-activity `Analysis` that defines what is generally known about analysis of software systems. The knowledge represented by this meta-meta model (or, in other words: this meta DSML) would then be reused for the definition of a more specific method, e.g., a method for de-

8   U. Frank

veloping distributed systems. Again, such an approach would require sequences of multiple instantiation, which are not supported by MOF like architectures. This limitation has been known for some time. Various authors suggest the use of powertypes to cope with it ([12], [4], [14]). Powertypes are also supported by the ISO/IEC 24744 metamodel. Apart from powertype being a cumbersome concept that does not contribute to the readability of metamodels (even though that may be a subjective assessment), it is restricted by the fact that it is part of an extended MOF architecture, which means it can be used on M2 only, not on higher levels of classification. That would hinder the definition of multiple classification levels in order to define hierarchies of reusable DSML.

   Even more important is a limitation that is not implied by the concept itself: as long as no programming language provides generic support for powertypes, they cannot be used for the design and implementation of tools, in particular they could not be used for tools that integrate the representation of methods with representations of method use ("endeavour"). Table 3.2 presents an assessment of traditional approaches to method engineering with respect to the generic requirements proposed in 2.2.

R1	Classification allows for defining properties of direct instances. Composition allows for reusing particular instances. However, it is not possible to express knowledge related to instances of instances.
R2	Each metamodel reflects a specific trade-off between range of reuse and productivity of reuse. The concepts defined with a certain method cannot be reused for the definition of further more specific methods – except for the reuse of method fragments. Therefore, the conflict between the two objectives of reuse can hardly be relaxed.
R3	The integrity of a method depends on the extent of misleading interpretations its specification allows for. The more domain-specific a meta-model is, the better are the chances to constrain the specification of a model properly. However, it is not possible to define constraints on instances of instances, e.g. on specific states of diagrams.
R4	From a conceptual perspective, methods could be further instantiated into specific method uses, because methods are conceptually specified on M1. However, apart from the use of powertypes, it is not possible to define properties (attributes, associations, etc.) for the classes that represent a method. Therefore, they cannot be further instantiated. But even with powertypes, the integration of methods with particular method instances is not possible, as long as there is no programming language that allows for multiple levels of classification.

Table 1. Assessment according to generic requirements

4   Prospects of Multi-Level Language Architectures

The limitations of MOF like architectures as a foundation for method engineering call for language architectures that enable higher levels of abstraction. Multi-level

modelling has been around for some time ([3]. It provides concepts that allow for creating models on different levels of classification. The meta language and language engineering environment presented in this section follows this tradition.

4.1   Multi-Level Methods in a Nutshell

While multi-level modelling may appear unusual to some, it supports in fact a natural way of using language. In the traditional paradigm, a language is always defined from scratch, that is, using the generic concepts of a generic meta-modelling language (see example in fig. 1). However, that does not correspond to the creation and use of languages in advanced societies. If, for example, a company wants to create a method, it will very likely not start with generic concepts such as class or attribute to design a method that fits its needs. Instead, it will use domain-specific concepts that are known for a certain purpose, e.g., software development, and/or for a certain domain, such as a specific industry. These concepts in turn are also not defined from scratch, but by using some more general concepts known for describing methods and projects. The highest level of such a hierarchy could be regarded as the textbook level. It reflects generally applicable knowledge of a certain field. Fig. 3 illustrates this idea. The boxes represent (meta-) models that define (meta-) methods. A certain method is defined by concepts that are part of a more general method. At the top level, the range of reuse is the highest. With every refinement step, the range of reuse decreases, but the productivity of reuse in a particular case increases. At the same time, the distinction between modelling language and model gets blurred. What is a model at one level, is a language at a higher level. That corresponds to the natural use of language. Usually, we do not bother with asking whether a term is part of a language (which it usually is) or whether it was defined with some other concepts (which is usually the case). Apparently, such a language architecture clearly contributes to satisfying requirements R1 and R2, because it fosters reuse in general and promotes range of reuse at a higher level, while it contributes to productivity of reuse on more specific levels. It is also suited to foster the integrity of methods (R3), since every refinement step introduces more specific concepts that constrain the construction of methods on the level below. Finally, it enables the integration of representations of a method and its use (R4), because objects on M0 that represent a particular use of a method (or in other words: a particular project) are part of the language architecture. Note that the relationship called "specified with" represents a specific kind of intrinsic instantiation (see the description of the FMML^x), which in fact combines aspects of instantiation with aspects of inheritance.
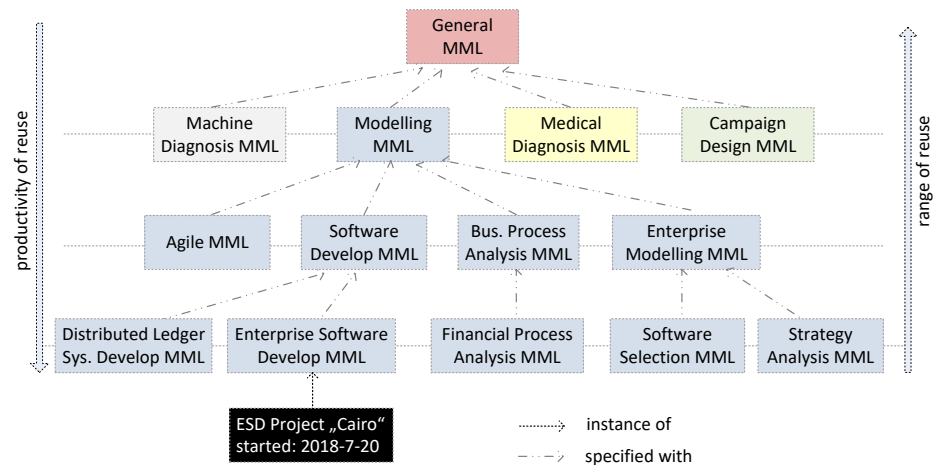
Fig. 3. Illustration of multi-level methods

In the remaining part of this section, it will be shown how a multi-level method architecture like the one illustrated in fig. 3 can be accomplished – both at design and at run-time.

4.2   An Executable, Multi-Level Meta-Modelling Language

The Flexible Meta Modelling Language (FMML^x) [11] is an executable multi-level modelling language. It allows for modelling classes on an arbitrary number of classification levels. To enable multiple instantiation steps, intrinsic properties, that is, attributes, operations and associations can be defined for each class. For each intrinsic property, the intended instantiation level has to be defined, which allows for deferred instantiation. The examples in fig. 4 illustrate the default notation of the FMML^x. The classification level of a class is indicated by the background color of the header field. The class `PeripheralDevice` is on level M3. Its intrinsic attribute `serialNo` is to be instantiated on M1 only. Since ever class is an object at the same time, it may have a state. For example, the class on M1 that represents a printer model includes data that represent technical properties and a price. Note that objects on M0 can be included in a model, too.
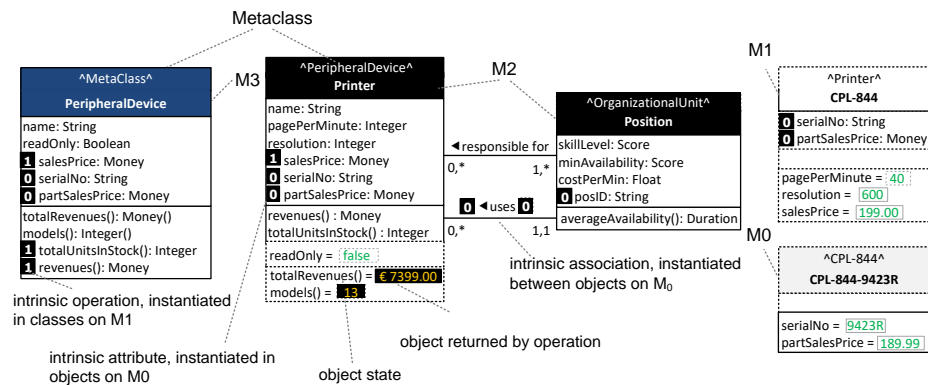
Fig. 4. Default notation of the FMML^x

The FMML^x is implemented with XMF (eXecutable Metamodeling Facility) ([5], [6]), which is a language execution engine that is based on a recursive metamodel, called Xcore. Modelling and programming languages that are specified as instances of Xcore can be executed within XMF. XMF allows accessing and modifying its own specification and its run-time system. Hence, there is no clear distinction between the language and a respective meta language. That is, XMF is reflective. Therefore, it facilitates navigation and introspection across all language levels represented in a particular system. XMF includes XOCL, an executable object constraint language. Xcore enables classes on an arbitrary number of classification levels. However, is is not possible to assign a particular classification level to a class. Instead, the classification level of the metaclass `Class` is contingent. The particular classification level of its instances is determined dynamically, depending on the actual number of possible instantiation steps. It does not provide direct support for deferred instantiation either. By default `Class` is on level M3 at first. A class that is instantiated from `Class` is on M2. If this class then inherits from `Class`, it is lifted to M3. The same procedure can be applied to its instances, which results in lifting the class higher to any intended level.

The specification and implementation of the FMML^x is based on an extension of Xcore and an intermediate layer that enables assigning a specific classification level to a class. Fig. 5 illustrates the recursive architecture of Xcore and the extension (properties marked with green background) added for the FMML^x. The interface layer includes the class `MetaAdaptor`, which is an instance of `Class` and inherits from it at the same time. It is instantiated into the class `MetaClass`, which serves to create classes on particular classification levels. The instantiation methods implemented in `MethodAdaptor` hide the process of lifting a class to an intended level described above.

12   U. Frank
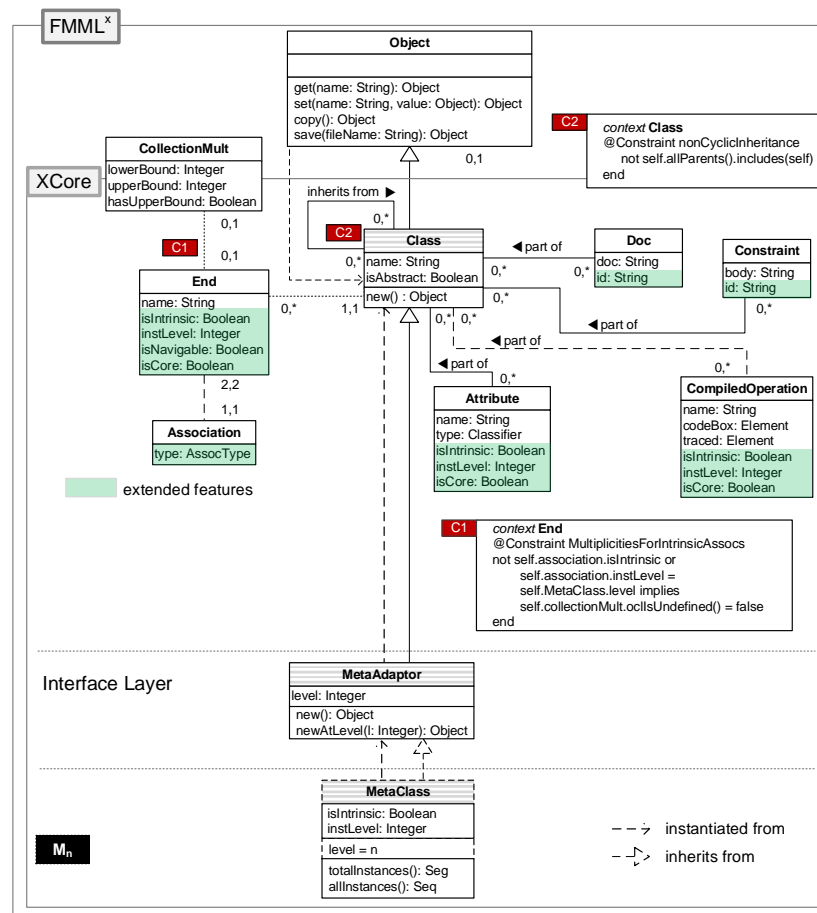
Fig. 5. Metamodel of the FMML[x] as an extension of Xcore

Apart from a slightly different terminology, the FMML[x] shares core features such as multiple classification levels and deferred instantiation with other approaches to multi-level modelling [2], [19]. However, it offers two distinct features that are of particular relevance for method engineering tools. Models defined with the FMML[x] may comprise objects on different levels of classification – down to M0. Furthermore, it features a common representation of models and code. Hence, there is no need to generate code from models and to cope with the challenges implied by the synchronisation of models and code.

4.3   Application to Method Engineering

To demonstrate the benefits of multi-level languages for method engineering, we will refer to the concepts in fig. 1 and the idea of a multi-level language hierarchy

shown in fig. 3. One essential principle of defining concepts of a language at any level is to express all knowledge available at that level. Only then, it is possible to avoid the redundant repetition of this knowledge at lower levels.
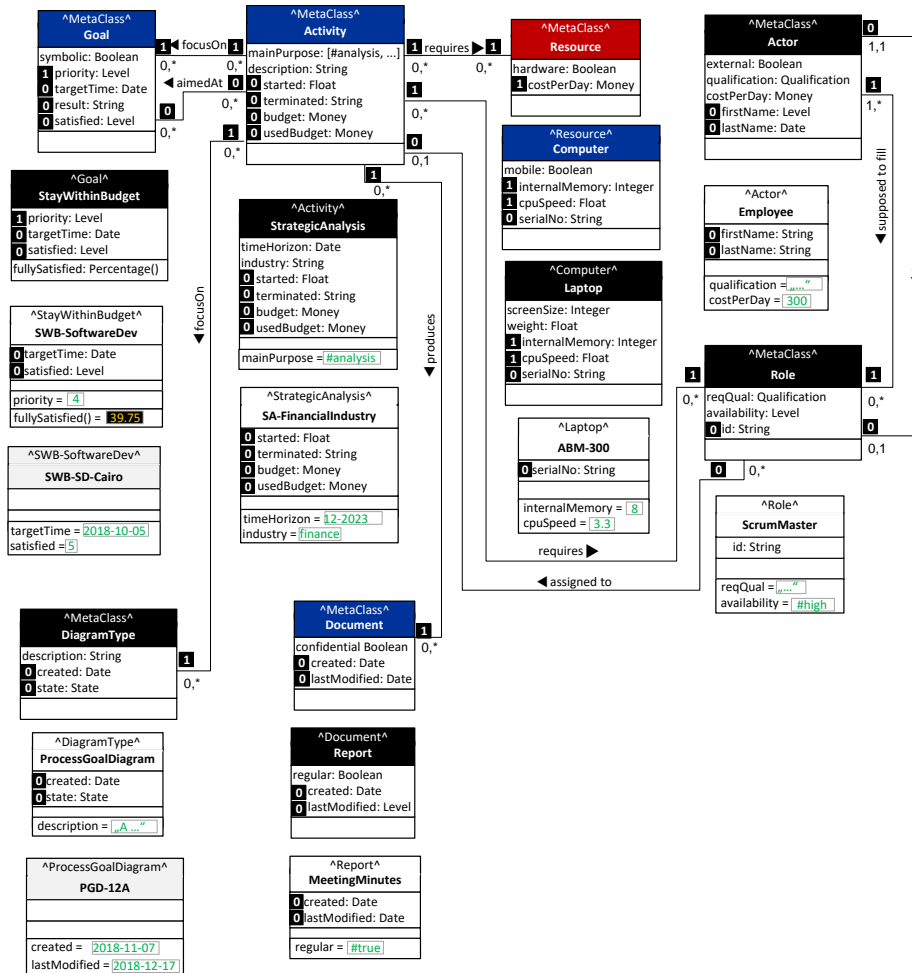

Fig. 6. Illustration of multiple levels of DSMLs integrated in one model[x]

The multi-level model shown in fig. 6 illustrates how DSML can be defined with more abstract DSML. The model shows only a small excerpt of classes that are part of corresponding DSML, such as a hierarchy of DSML for modelling resources, or documents. The representation of associations is restricted to a few examples only. On a high level of classification certain associations that are to be instantiated on lower levels, may be known already, e.g. the association "re-

14   U. Frank

quires" between the classes `Activity` and `Role`, which is to be instantiated on M1 only. The model illustrates a few important aspects of multi-level language architectures. There is no clear distinction between language and language application. Furthermore, a class may be associated with a class on any other level. Again, this corresponds to the use of concepts in natural language. Apparently, it is possible to integrate the representation of particular method uses on M0. The Xmodeler allows to execute the model hierarchy. In other words, the multi-level model becomes the conceptual foundation *and* the implementation of an integrated method specification and execution environment.

5   Conclusions and Outlook

The approach presented in this paper demonstrates the potential of multi-level language hierarchies for modelling language and models in the context of method engineering and method use. It is suited to relax the fundamental conflict between range of reuse and productivity of reuse. It also contributes to more consistent method definitions. The obvious prospects of multi-level modelling are constrasted with two remaining research challenges. Our work on multi-level models has shown that the classification level of a class may vary with the context it is used in. For example, `Document` may be on M2 in one context, on M3 in another context. While it is conceivable to use two different instantiations of the class, that would compromise integrity, because two classes that have common properties would have to be maintained separately. For this reason, the next version of the FMML^x [8] will support contingent level classes, which are already supported by the Xmodeler. Contingent level classes create the challenge to define a proper semantics. An approach based on modal logic is a promising option. In that case, a class would be on level n in one world, on level m in another world. A further challenge is related to multi-level models of dynamics. It would, e.g., be nice to define a (meta-) process for software development in general, which could then be refined step by step to more specific processes. Unfortunately, the specialization of process types is not possible without relaxing the substitutability constraint. The multi-level model in fig. 6 abstracts this problem away by specifying activities without accounting for the dynamic context they are supposed to be used in. Our future research is aimed at defining a specialization semantics for processes that is based on a relaxed notion of substitutability.

References

1. Iso/iec 42010:2007: Systems and software engineering – recommended practice for architectural description of software-intensive systems (2007)
2. Atkinson, C., Gutheil, M., Kennel, B.: A flexible infrastructure for multilevel language engineering. IEEE Trans. Software Eng. **35**(6), 742–755 (2009)
3. Atkinson, C., Kühne, T.: The essence of multilevel metamodeling. In: Gogolla, M., Kobryn, C. (eds.) Proceedings of the 4th International Conference on the Unified Modeling Language, Modeling Languages, Concepts, and Tools. pp. 19–33. Lecture Notes in Computer Science, Springer, Berlin and Heidelberg (2001)

4. Borges Ruy, F., Perini Barcellos, M., de Almeida Falbo, R., Guizzardi, G.: An ontological analysis of the iso/iec 24744 metamodel. Frontiers in Artificial Intelligence (2014)
5. Clark, T., Sammut, P., Willans, J.: Superlanguages: developing languages and applications with XMF. Ceteva (2008)
6. Clark, T., Willans, J.: Software language engineering with xmf and xmodeler. In: Mernik, M. (ed.) Formal and Practical Aspects of Domain-Specific Languages, pp. 311–340. Information Science Reference (2012)
7. Fill, H.G., Karagiannis, D.: On the conceptualisation of modelling methods using the adoxx meta modelling platform. Enterprise Modelling and Information Systems Architectures **8**(1), 4–25 (2013)
8. Frank, U.: The flexible multi-level modelling language (fmmlx): Version 2.0: Analysis of requirements and technical terminology
9. Frank, U.: Multi-perspective enterprise modelling: Background and terminological foundation. icb research report, no. 46, university of duisburg-essen (2011)
10. Frank, U.: Domain-specific modeling languages - requirements analysis and design guidelines. In: Iris Reinhartz-Berger, Aron Sturm, Tony Clark, Yair Wand, Sholom Cohen, Jorn Bettin (eds.) Domain Engineering: Product Lines, Conceptual Models, and Languages, pp. 133–157. Springer (2013)
11. Frank, U.: Multilevel modeling: Toward a new paradigm of conceptual modeling and information systems design. Business and Information Systems Engineering **6**(6), 319–337 (2014)
12. González-Pérez, C., Henderson-Sellers, B.: Modelling software development methodologies: A conceptual foundation. Journal of Systems and Software **80**(11), 1778–1796 (2007). https://doi.org/10.1016/j.jss.2007.02.048
13. Henderson-Sellers, B., González-Pérez, C., Ralyté, J.: Comparison of method chunks and method fragments for situational method engineering. In: Proceedings of the Australian Software Engineering Conference 2008
14. Henderson-Sellers, B., Ralyté, J.: Situational method engineering: State-of-the-art review. Journal of Universal Computer Science **16**(3), 424–478 (2010)
15. Henderson-Sellers, B., Ralyte, J., Ågerfalk, P., Rossi, M. (eds.): Situational Method Engineering (2014). https://doi.org/10.1007/978-3-642-41467-1
16. Kalle Lyytinen, Kari Smolander, Veli-Pekka Tahvanainen: Modelling case environments in systems development. In: Proceedings of CASE 1989. Stockholm (1989)
17. Kelly, S., Lyytinen, K., Rossi, M., Tolvanen, J.P.: Metaedit+ at the age of 20. In: Bubenko, J. (ed.) Seminal contributions to information systems engineering, pp. 131–137. Springer, Berlin, New York (2013)
18. Kornyshova, E., Deneckère, R.: Method chunks selection by multicriteria techniques: an extension of the assembly-based approach. In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Situational Method Engineering: Fundamentals and Experiences, pp. 65–78. IFIP The International Federation for Information Processing, International Federation for Information Processing, Boston, MA (2007)
19. Neumayr, B., Grün, K., Schrefl, M.: Multi-level domain modeling with m-objects and m-relationships. In: Link, S., Kirchberg, M. (eds.) Proceedings of the 6th Asia-Pacific Conference on Conceptual Modeling (APCCM), pp. 107–116. Australian Computer Society, Wellington (2009)
20. Rolland, C.: A primer for method engineering (1998)