# Using Multi Level-Modeling Techniques for Managing Mapping Information

Samir Al-Hilank[2], Martin Jung[1,2],
Detlef Kips[1,2], Dirk Husemann[2], and Michael Philippsen[1]

[1] Friedrich-Alexander University Erlangen-Nürnberg (FAU),
Programming Systems Group, Martensstr. 3, 91058 Erlangen, Germany
philippsen@cs.fau.de
[2] develop group Basys GmbH, Am Weichselgarten 4, 91058 Erlangen, Germany
alhilank|husemann|jung|kips@develop-group.de

**Abstract.** Traditional modeling approaches support a limited set of instantiation levels (typically one for classes and another adjacent one for objects). Multi-level modeling approaches on the other hand have no such limit to the number of levels. As a consequence, an arbitrary number of levels may be used to define models, and the distinction between class and instance is redefined.
The paper summarizes the experience gained from applying multi-level modeling techniques to a real application from the domain of development process improvement (DPI). The underlying case study has been conducted in cooperation with a large automotive supplier. We discuss the pros and cons of using multi-level modeling techniques and propose areas that we think would benefit from further research.

## 1 Introduction

Although problem domains often have a natural structure that spans more than two logical levels, traditional modeling languages like, for example, MOF [20] or UML [21] are limited to only two layers. As a consequence, using UML or MOF to capture such problem domains leads to squeezing several logical levels into two. This in turn causes accidental complexity [12] that does not originate from the problem domain itself. Consequently, concepts like powertypes [17], potency [12], dual classification [11], etc. have been developed to allow for the definition of an arbitrary set of classification levels.

This paper reports on experience with defining and using deep models in an industrial case study. It is organized as follows: Sec. 2 presents the case study's problem statement. Sec. 3 describes our experiences with using deep modeling techniques. Sec. 4 discusses related work. We conclude with a list of research areas that could improve the applicability of multi-level modeling.

## 2 Problem Statement

Most companies working in the automotive domain need to use development processes (DP) that comply with requirements defined by quality standards (QS) like CMMI [14] or ISO 26262 [18]. Company specific standards or laws enforced by governments may pose additional requirements. Moreover, those companies often have to provide evidence that their DP complies with all requirements, for example, due to economic reasons (precondition of contracts) or legal considerations (product liability). Hence, there is a strong need to collect this evidence information in a systematic way.

The basic approach is straightforward: Just collect and later analyze mappings between QS requirements and elements of the DPs. Here is a typical example mapping that we illustrate at the bottom of Fig. 1:

– `Integrate SW Component` is an item (e.g., a Task) defined by the company's DP.
– `SW Integration and Testing` is a QS requirement (e.g., a description of a phase) that has to be implemented by elements of the company's DP.
– `Mapping` connects both elements with the following semantics: The requirement `SW Integration and Testing` is fulfilled by the element `Integrate SW Components`.

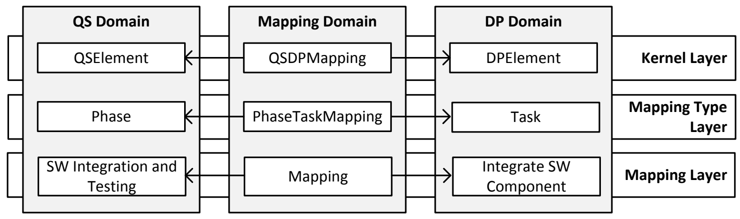

**Fig. 1.** Organization by responsibility (horizontal) and domain (vertical).

Note that these are items from three different domains. First, items from the **DP domain** need to be taken into account. There is no commonly used standard approach for describing DPs, but typical items found in DPs are Task, Step, Function or Template. In our case study, we use ARIS [6] to describe the company's DP. Second, we capture QS requirements in the **QS domain**. In our case, the company's process has to comply with three QSs simultaneously (CMMI, Automotive SPICE [13] and ISO 26262). Last, the **mapping domain** collects mappings between items of the DP and QS domain.

Fig. 1 also structures the problem into three orthogonal layers:

– The `Mapping Layer` collects mapping information.
– The `Mapping Type Layer` defines valid mappings.
– The `Kernel Layer` provides the foundation with respect to the terminology used within the DP, QS, and mapping domain.

The natural structure of the problem domain in Fig. 1 does not fit into two levels. Instead, Fig. 1 shows a scenario that is well suited for deep modeling techniques. Below we present such a deep model that we have implemented in an industrial application.

## 3 The Case Study

### 3.1 DeepML in a Nutshell

None of the research papers on concepts and features of deep modeling offers the flexibility and freedom needed for our case study (for details see Sec. 4). In general, existing approaches lack

- **freedom to combine concepts from several approaches.** There is no common set of deep modeling concepts, that would allow to cherry-pick suitable concepts from various approaches.
- **freedom to omit concepts.** Known approaches do not allow to omit concepts that are unnecessary for our solution.
- **freedom to experiment with new ideas.** None of the approaches we know of provides means to explore new ideas or to add new concepts.

As we needed this degree of freedom for our case study, we designed the DeepML language and infrastructure. DeepML supports the usual modeling concepts like inheritance, primitive data types, and enumerations. Furthermore, DeepML builds on the idea of dual classification and potency that is unified in the ontological classification architecture (OCA) [8]. OCA distinguishes between two kinds of instantiation relationships: The **linguistic instantiation** (lio) is a relation between elements of the model (that describes the core language capabilities, e.g., attributes, references, etc.) and elements of the problem domain. Linguistic instantiations therefore are not domain specific. Fig. 2 gives examples. On the other hand, **ontological instantiation** (oio) relations capture domain specific aspects. In Fig. 2, for example, `Book` is an ontological instance of `ProductType`.

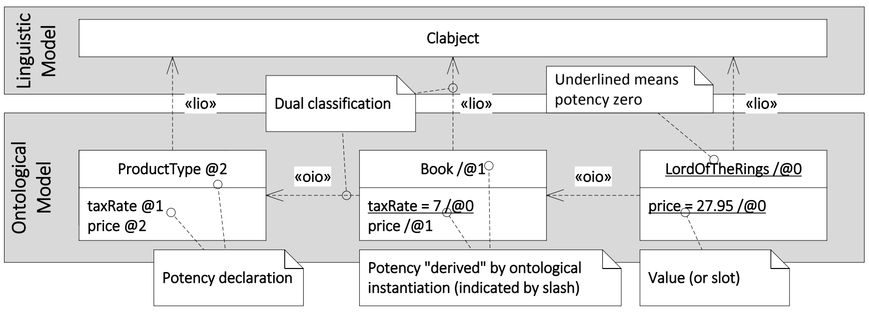

**Fig. 2.** DeepML in a Nutshell – Dual Classification.

Both elements (`Book`, `ProductType`) have a property called **potency** which is a positive integer value. Every ontological instantiation step decrements the potency by one. Model elements with a potency zero cannot be further instantiated. Consequently, the element `Book` has both an instance and a class facet. This is commonly called a **clabject** [10]. DeepML clabjects "derive" the potency in the same way as attributes. To express this inheritance, we use the "/" character as in `Book/@1`.

Due to space restrictions, instead of describing DeepML's language capabilities in full detail, the next sections highlight some of its key features. We also compare them to other deep modeling languages in Sec. 4.

### 3.2 Sample Mapping Scenario from the Case Study

As outlined in Sec. 2, collecting mappings is always done with a specific goal in mind. For example, in our case study, the DP needs to fulfil the requirements of a new and emerging QS (ISO 26262). However, the company's DP already fulfils a subset of the requirements as defined by the QS CMMI. One strategy to minimize the effort of gap analysis in such a situation is to map elements from both QSs onto each other. We consider those ISO 26262 requirements that we can map to CMMI requirements as already covered by the company's DP. ISO26262 requirements that cannot be mapped to CMMI necessitate further investigation.

The following subsections present parts of our Process Improvement and Quality Standard Harmonization Model (PIQSH-M), a deep model that underlies our mapping management application for the QS-DP-Mapping (Sec. 2) and QS-QS-Mapping (see below) scenario.

### 3.3 Ontological Containment

QSs are usually published as large text documents. These documents are typically organized with a specific ordering structure in mind. For example, each chapter at a specific level (e.g., "9.5.6 SW Integration and Testing") represents a phase. On its left side, Fig. 3 shows how to model the following clabjects for capturing QSs in DeepML:

- **QSDElement (Quality Standard Domain Element)**: Base clabject of all elements that belong to the QS domain.
- **QSTM (Quality Standard Type Model)**: Instances of this clabject are **top level** elements of models used for capturing the structure of one QS (e.g. entities, relations, etc.). An example is the clabject `ISO26262/@1`.
- **QSElement**: Instances of QSElements represent non top level elements of QSs (e.g., clabject `Phase/@1`).

To capture the relationships between elements of one QS, we introduce references. In DeepML, relationships among clabjects are expressed by references that are always unidirectional and owned by exactly one clabject. They are, thus,

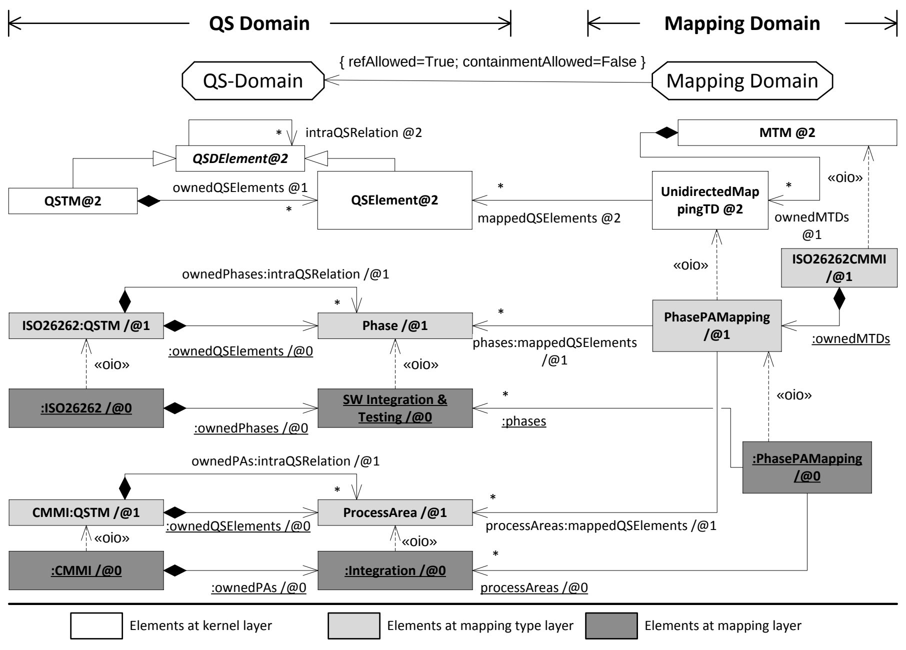

**Fig. 3.** An extract of the PIQSH-model.

more similar to references in MOF than to association classes in UML. References may participate in classification hierarchies in the same way as clabjects do. Consequently, the potency of a reference is either directly specified or derived from a classifying reference. For example, the reference `intraQSRelation@2` classifies the set of relationships (not values) between QS elements. Thus, instances of this reference, for example `ownedPhases:intraQSRelation/@1`, express that ISO 26262 is made up of a set of phases. A slot concept is used to store values and in DeepML underlined text is used as notation. For example, `:ownedPhases/@0` is a slot, and `SW Integration and Testing/@0` is owned by `:ISO26262/@0`.

A similar model structure captures mapping information: Instances of the clabject Mapping Type Model (`MTM`) are top level containers of models for specifying meaningful mappings. An example for such a mapping is `PhasePAMapping/@1` which links `Phases` (from ISO26262) and `ProcessAreas` (from CMMI). Concrete mappings can then be captured on the mapping layer (e.g., `:PhasePAMapping/@0`).

Although, references optionally may be containment references, most deep language specifications ignore this. Instead, DeepML adds an ontological containment concept because: Firstly, a containment is a common constraint that should be natively supported. Secondly, and more importantly, a containment hierarchy reflects the hierarchical structure that the domain expert had in mind. Finally, persistance layers typically use containments to find the one (or no) resource to store elements into. An example for a containment reference is ownedQSElements@1 (see Fig. 3). Given its definition, QSElements are part of exactly one QSTM.

### 3.4 Domain Stacks and Model Organization

One of the key requirements of the PIQSH-M is to support reuse. For example, other companies or organizational units may need to comply with a different set of QSs. The basic idea for addressing this problem is to develop a library of commonly used QSs that grows over time. Missing QSs can simply be transformed into deep models and added to that library for later reuse.

When managing such a library, it is important to define what kinds of models are part of it. Restricting and managing dependencies between models is equally important. We decided to use the matrix-like organization shown in Fig. 1 as a guideline because it separates model content with respect to the domain (QS, Mapping, DP) and to the role (kernel, mapping, and mapping type definition layer). For example, direct and indirect instances of QSTMs@2 (e.g., CMMI/@1 and :CMMI/@0, respectively) are models managed by the library.

As explained above, restrictions should apply to the set of permissible relations between models from different domains and layers. DeepML allows the definition of references without a classifying reference at any level. For example, we could define a direct reference between Phase and ProcessArea without defining the corresponding mapping type. However, such a direct reference would lead to a direct dependency between ISO26262/@1 and CMMI/@1 — thus breaking the intentional decoupling introduced by the mapping layer and leading to a pollution of model content.

To prevent pollution of this kind, the concept of **domain stacks** is added to DeepML, formalizing the separation into domains as illustrated in Fig. 1. Every clabject may be part of at most one domain stack. For example, all elements on the left side of Fig. 3 are members of the QS domain stack. References between elements within the same domain stack are allowed. References to elements of other domain stacks or to domainless elements are not allowed by default.

The situation is slightly different for members of the DP domain stack (right side of Fig. 3): They must refer to elements from other domain stacks, but must not contain them. To express these rules, a relationship between domain stacks is introduced that specifies which kind of references are allowed. An example is shown at the top of Fig. 3.

To summarize, by using top level containers, ontological containment and domain stacks, a clean separation of content is realized that spans multiple levels. However, a capability for managing mapping projects is still missing.

### 3.5 A Deep Model for DPI Project Management

With a deep model for organizing items of the QS and DP domains at hand, the practitioner also needs tool support to actually use this model for creating and managing mapping information. Let us now demonstrate how the DeepML model serves as a foundation of an application called PIQSH Support Center (PIQSH-SC) that covers the three main use cases of managing projects in the DPI domain:

(1) **Define Quality Standards**: Capture the structure and content of QSs.
(2) **Define Mapping Type Models**: Define the set of meaningful mappings between certain combinations of QSs (or QSs and DPs).
(3) **Collect Mapping Information**: Collect mapping information within a specific context (e.g. between QSs or between QSs and DPs).

Use case (1) is straightforward. As it is only concerned with creating models, we do not discuss it any further. Use case (2) involves at least three models, for example, two `QSTMs` (e.g., `CMMI/@1` and `ISO26262/@1`) and one `MTM` (e.g., `ISO26262CMMI/@1`) as shown in Fig. 3. The inter-model relationships can also be captured in DeepML as the PIQSH Project Management Model (PIQSH-PMM) and the clabject Model Type Definition Project (`MTDProject`). With this clabject, we string together all models in terms of an import relationship. The resulting model is shown on the left side of Fig. 4.

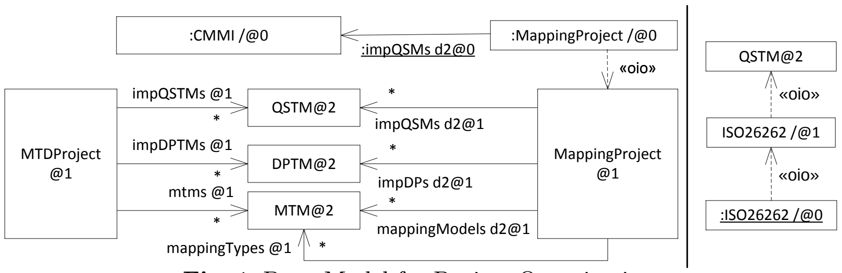

**Fig. 4.** Deep Model for Project Organization.

Use case (3) requires capturing mapping information. Within use case (2), `MTDProject` and the corresponding references had equal potency values, and both were reduced within an ontological instantiation step. In contrast, in use case (3) a project needs to store references to models on the mapping layer; that is, to instances of instances of `QSTMs` or `MTMSs`. Since references of this kind cannot be expressed with DeepML, we enhance references by the concept of distance.

The **distance** between two clabjects `A` and `B` is defined as the number of instantiation relationships that need to be traversed in order to reach A when starting from B. As an example, the distance between `QSTM@2` and `ISO26262/@0` is 2 (see right side of Fig. 4). The rule for assigning values to a reference depends on the distance: The distance between the reference's target type and the value to be assigned must be equal to the distance of the reference. Fig. 4 shows the clabject `MappingProject` and the notation (d<distance>@<potency >) that is used to specify both distance and potency. As a consequence, the semantics of the reference `impQSMs d2@1` in the context of a concrete mapping management project is to store instances of instances of `QSTMs`. There is an example for reference values (i.e., instances of `MappingProject`) shown at the top of Fig. 4.

### 3.6 DeepML's Framework Architecture

Fig. 5 gives an architectural overview of our framework. `DeepMLCore` is the core language implemented with the Eclipse Modeling Framework (EMF) [2]. On top of it there is a set of supporting libraries: The `DeepML Editor` is a generic component that provides views and editors for visualizing and modifying DeepML models from both perspectives, the ontological and the linguistic one. The `Epsilon Adapter` is a bridge for using the language family (M2M, M2T, etc.) provided by the Epsilon framework [3]. The `Epsilon Adapter` is also used to realize a bridge to BIRT (Business Intelligence and Reporting Tool) [1]. The PIQSH-SC in turn uses BIRT for generating reports (e.g., gap analysis).

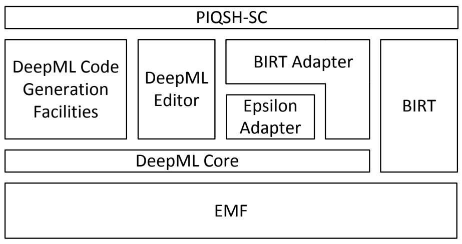

**Fig. 5.** DeepML's Architecture and Framework.

## 4 Related Work

The need to define efficient strategies for managing mappings in scenarios as outlined in Sec. 2 has been identified before, for example, in [16] and [22]. However, this paper focuses on experience made by using deep modeling techniques. So we will concentrate on related work in that area of research.

Melanee [4] provides a deep modeling framework with graphical syntax and editors. It is built on top of EMF and also incorporates ideas from the domain of ontologies. It therefore supports an additional so-called exploratory mode [9]. In exploratory mode, a reasoning engine establishes ontological instantiation relationships. However, our goal was to use a deep modeling language with a minimal set of language constructs that is optimized for building models in a constructive way, that is, by using explicit instantiation relationships. Also, Melanee does not define a concept similar to domain stacks for managing model partitioning. There is also no concept available in Melanee for realizing layer spanning references as defined in Sec. 3. However, some framework capabilities like for example emendation support [7] are neither provided by DeepML nor by any other deep modeling framework we are aware of.

Another deep modeling language is MetaDepth [5]. MetaDepth provides a convenient textual syntax for creating deep models and has some unique language features, for example, the *-potency. A clabject with a star potency has an unlimited potency. With each instantiation step, that potency may either remain unlimited or a concrete potency value may be assigned. Yet, MetaDepth lacks support for ontological containment and domain stacks that greatly simplified the architecture needed for our case study 3.

## 5 Conclusion and Future Work

In this paper we report on our experience gained by applying deep modeling techniques to a real life industrial use case. The corresponding application has been successfully used by our industrial partner for approximately one year, and we are currently discussing further extensions. The underlying deep model is very well suited for this kind of problem: The concept of layers and stacks provides a clean separation of concerns, and the model is very flexible. We continue to develop both, the application and the DeepML framework, mainly focussing on the following topics:

**First**, one of the main problems with traditional programming languages, such as Java, is, that they only support two levels of instantiation (class and object). We are currently investigating solutions to this problem either by using appropriate patterns to emulate multiple levels or by enhancing the programming language (like e.g. done in DeepJava [19]). **Second**, a lot of enhancements are planned with respect to the infrastructure. For example, diagram based editors, additional code generators for automatically generating code to ease the development of user interfaces, and so forth. **Third**, we are looking for other domains that might benefit from using deep modeling techniques. A promising candidate is the domain of variant management in software development and systems engineering. **Fourth**, we intend to explore the combination of PIQSH-M with executable process models such as eSPEM [15]. One possibility is to introduce a process execution layer below the mapping layer, but limited to the DP domain. This leads to a four-level model architecture.

To summarize, we think multi-level modeling techniques are very well suited for the kinds of problem outlined in Sec. 2. However, there remains a lot of research to be done. For example, we did not find any solutions for handling model migration in a multi-level aware way. Yet, in our view, the major stumbling block is the lack of an agreed and publicly available specification of the core concepts and terminology of multi-level modeling (a standard deep meta model).

Finally, we hope our experience report encourages other people to use deep modeling techniques — at least in areas similar to the one outlined in Sec. 2.

## References

1. BIRT. `http://www.eclipse.org/birt/` (July 2014)

2. Eclipse Modeling Framework. http://www.eclipse.org/modeling/emf/ (July 2014)
3. Epsilon. http://www.eclipse.org/epsilon/ (July 2014)
4. Melanee. `http://melanee.org/` (July 2014)
5. MetaDepth. `http://astreo.ii.uam.es/ jlara/metaDepth/` (July 2014)
6. software AG: ARIS. `http://www.softwareag.com/aris` (July 2014)
7. Atkinson, C., Gerbig, R., Kennel, B.: On-the-Fly Emendation of Multi-Level Models. In: Modelling Foundations and Applications. LNCS, vol. 7349, pp. 194–209. Springer (2012)
8. Atkinson, C., Kennel, B., Go, B.: The Level-Agnostic Modeling Language. In: Software Language Engineering, LNCS, vol. 6563, pp. 266–275. Springer (2011)
9. Atkinson, C., Kennel, B., Go, B.: Supporting Constructive and Exploratory Modes of Modeling in Multi-Level Ontologies. 7th Intl. Workshop on Semantic Web Enabled Softw. Eng. (2011)
10. Atkinson, C., Kühne, T.: Rearchitecting the UML Infrastructure. ACM Trans. Model. Comput. Simul. 12(4), 290–321 (Oct 2002)
11. Atkinson, C., Kühne, T.: Model-Driven Development: A Metamodeling Foundation. IEEE Software 20(5), 36–41 (2003)
12. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Software and System Modeling 7(3), 345–359 (2008)
13. Automotive SIG: Automotive SPICE Process Reference Model, Ver. 4.5 (May 2010)
14. CMMI Product Team: CMMI for Development, Ver. 1.3. Tech. Rep. CMU/SEI-2010-TR-033, Carnegie Mellon Univ. – Software Eng. Inst. (Nov 2010)
15. Ellner, R., Al-Hilank, S., Drexler, J., Jung, M., Kips, D., Philippsen, M.: A FUML-Based Distributed Execution Machine for Enacting software process models. In: ECMFA. LNCS, vol. 6698, pp. 19–34 (2011)
16. Ferreira, A.L., Machado, R.J., Paulk, M.C.: Supporting audits and assessments in multi-model environments. In: PROFES. Lecture Notes in Business Information Processing, vol. 6759, pp. 73–87 (2011)
17. Gonzalez-Perez, C., Henderson-Sellers, B.: A powertype-based metamodelling framework. Software and System Modeling 5(1), 72–90 (2006)
18. Intl. Org. for Standardization: ISO 26262: Road vehicles – Functional safety (Nov 2011)
19. Kühne, T., Schreiber, D.: Can Programming be Liberated from the Two-Level style? Multi-Level Programming with DeepJava. In: OOPSLA. pp. 229–244 (2007)
20. OMG: Meta Object Facilities. `http://www.omg.org/mof/` (July 2014)
21. OMG: Unified Modeling Language. `http://www.uml.org/` (July 2014)
22. Siviy, G. et al.: Maximizing your Process Improvement ROI through Harmonization. Tech. rep., Carnegie Mellon Univ. – Software Eng. Inst. (March 2008)