The Open Meta Modeling Environment

Bernhard Volz, Michael Zeising, Stefan Jablonski
University of Bayreuth
Bayreuth, Germany
{bernhard.volz,michael.zeising,stefan.jablonski}@uni-bayreuth.de

ABSTRACT
Drawing tools are typically used in an early phase of software development when outlining and communicating ideas is more important than representing problems and solutions formally. But in contrast to that during later development phases formal models are needed for different purposes like simulation or code generation. Within this contribution, we present our modeling environment OMME that is based on methods and paradigms known from meta modeling. With OMME models of different stages of the development process may be linked to each other and constraints are used to ensure the integrity of these links.

Categories and Subject Descriptors
D.2.10 [**Software Engineering**]: Design – Methodologies, Representation

General Terms
Algorithms, Languages

Keywords
Open Meta Modeling Environment, Linguistic Meta Model, Constraints

1. INTRODUCTION
Models are an important tool for communicating ideas and for reducing the complexity of problems. When it comes to creating or representing models there are commonly two basic approaches:

- Free form drawing tools which allow for drawing nearly any kind of diagram but lack formality and therefore produce diagrams that can hardly be reused. These tools include e. g. *Microsoft Visio* [1].
- Formal modeling environments based on a strict formal language usually designed for one special purpose. An example for this category of tools is the *IBM Rational Software Architect* [2] which is built on top of the *Unified Modeling Language* (UML).

In the early stages of a project people usually demand a tool from the first category. They want to have the freedom to sketch ideas without boundaries. When the project progresses and requirements concretize users move to formal modeling tools to produce reusable and unambiguous models.

FlexiTools'11, May 22, 2011, Waikiki, Hawaii, USA.

The challenge consists in bridging the gap between these two types of representations. One common approach is to start with an informal drawing, to impose some meaning afterwards and to finally generate the formal model from this "annotated" drawing. The catch is that the semantic link between the initial sketch and the resulting model gets lost on the way. There is no way to ensure that e. g. every box in the drawing has resulted in an entity.

We argue that this can be solved by representing whatever type of model using one formal language. This approach is referred to as *orthogonal classification* [3]. If every type of model is amenable to the same modeling platform, a semantic link between them may be easily established and maintained throughout the modeling process. As a platform implementing this principle we propose the *Open Meta Modeling Environment* (OMME).

The remainder of this contribution will first present a short summary of OMME including its design principles. After that we will focus on the capabilities of OMME to impose semantic restrictions on models which ensure that links between different models are not only valid in a syntactic style. Here we will discuss features of the OMME constraint language and give a simplified example of its usage. Details and implications not directly shown in the example are explained in detail in Section 3.

2. CORNERSTONES OF THE OPEN META MODELING ENVIRONMENT
Common approaches like the *Eclipse Modeling Framework* (EMF) [4, 5] and the *Meta Object Facility* (MOF) [6] rely on a fixed meta model that every model has to fit in. The idea behind orthogonal classification is to separate storage from semantics by expressing this hierarchy of (meta) models in terms of one storage representation – the so called *linguistic meta model* [7]. By building on this language one gains an environment for developing arbitrary models instead of a tool for one modeling language like the UML only. A key principle of our implementation of this language is neutral naming. Terms like "class", "object" or "clabject" are well understood by software engineers but unlikely to be used by others. Neutral terms like "concept" should help the framework to gain more acceptance by domain experts.

Formal modeling environments usually comply to a certain paradigm – rules defining how models should be structured [8]. Drawing an initial sketch certainly requires a very lose paradigm with only very few restrictions. That means it requires the tool to support non-strict modeling. On the other side, the formal model of a concrete application e. g. used for code-generation requires the tool to enforce rules and therefore to support strict modeling. Our framework supports those modeling paradigms to be

exchanged at any time and therefore supports both strict and non-strict modeling.

We have implemented a prototype called OMME [9] that is based upon a linguistic meta model. It features neutral naming and supports advanced modeling constructs like *Powertypes* [10, 11], *Clabjects* [12], *Deep Instantiation* [13] and *Materialization* [14]. A modeling constraint language has recently been incorporated to enable high-level model validation. Our implementation is available upon request.

3. WORKING PRINCIPLE OF THE OPEN META MODELING ENVIRONMENT

The foundation of the Open Meta Modeling Environment is the *Orthogonal Classification* approach of Atkinson and Kühne [3] that separates contents of models from their storage aspects. Following the principles known from meta modeling, a linguistic meta model can be built that is capable of representing arbitrary kinds of models which may consist of an unlimited number of meta levels. Each "model" within OMME is then an instance of this linguistic meta model. The benefit of this approach is that all infrastructure services of a modeling environment such as a model repository or a constraint evaluation engine operate on instances of a linguistic meta model. Thus, even though constraints refer to model content and are bound to the corresponding meta model ("only yellow boxes may correspond to an entity"), they can still be evaluated by a generalized engine since this engine is only based on the storage aspects of the model. Consequently the contents of a linguistic meta model describe how a model is being built.

In case of the OMME, a "model" is a container for an arbitrary number of "levels". "Packages" are used for structuring the contents of a level, i.e. "concepts" and "enumerations". A "concept" represents a modeled entity that consists of attributes and values assigned to attributes. Thus, a "concept" in OMME corresponds to a *clabject* [12]. Then the OMME constraint engine may validate a constraint like "only yellow boxes may correspond to an entity" as follows:

- Retrieve all "concepts" that are defined as being instances of a "concept" named "Box"
- Check the value assigned to the attribute "entity" of each concept; if it contains a reference to another "concept", the value of the attribute "color" has to be "yellow"

Thus, even the definition of the constraint is highly related to the content of a model, it can be evaluated taking only the storage aspect of a model into account.

Figure 1 then depicts the linguistic meta model of OMME together with an instance of it in the two boxes "L1" and "L0". L0 references L1 by means of an "instance of" relationship which is mapped to the creation of an object in Java within our prototype. Consequently, each element within L0 is again an instance of exactly one element of the linguistic meta model in L1.

Special invariants that are bound to elements of the linguistic meta model and that are evaluated against an instance of it ensure the validity of models within OMME. As mentioned in the previous section, these Linguistic Constraints can be changed dynamically.

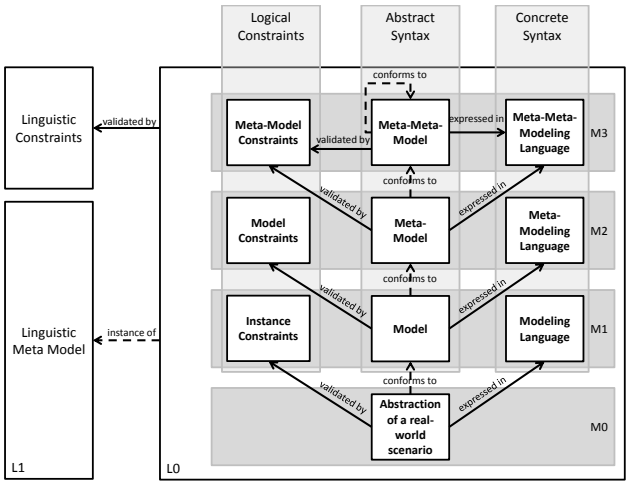

Figure 1. Conceptually, each level within a hierarchy of different meta levels in OMME is split into the definition of *Constraints*, an *Abstract* and a *Concrete Syntax*.

As each level can be seen as the definition of a modeling language for the lower level, an abstract and concrete syntax as well as a set of logical constraints can be defined within a level (Figure 1). Abstract and concrete syntax can both be seen as elementary building blocks of a modeling language. However, up to now, the OMME prototype does not allow for defining a concrete syntax freely. Therefore, Figures 2 and 3 show a similar syntax for each "level". The logical constraints on each level finally define rules such as "only yellow boxes may correspond to an entity" which are used for validating the content of lower levels.

4. STEPS OF THE MODELING PROCESS

The purpose of the OMME is to support a process that we imagine could be the modeling method of the future. This process starts at drawing without formal boundaries and leads to modeling on the basis of a strict meta model.

4.1 Starting with a drawing

During the inception phase of a project, tools must have the ability to transport ideas between humans. It is not necessary for the models to comply with a strict formal meta model or to exhibit any semantics at all. Tools that usually fall into that category are *Microsoft PowerPoint* or *Microsoft Visio*. Of course, humans have some semantics in mind when they are drawing these informal models but it's usually consuming too much time and effort to formalize it in this early stage of the project.

For being able to draw freely and to use the OMME like we would use *Visio*, we first have to define what may be drawn at all. This is expressed via a bundle of three models: the meta model, the so called *graphical definition* and the *editor definition* (see Figure 2). We could create them on our own but we decide to use one of the precast templates provided by our framework, e. g. "boxes and arcs". We load the template bundle and create a new diagram instance. Now we may draw boxes and connect them by arcs.

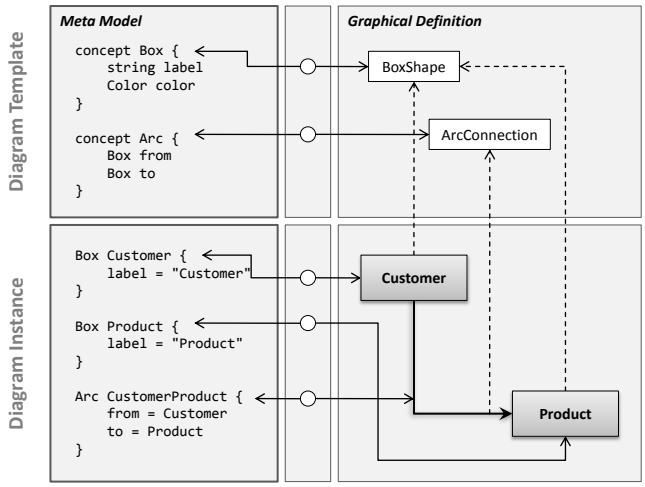

Figure 2. The *Meta Model* describes what may be drawn while the *Editor* and *Graphical Definition* describe how the concepts are graphically represented. Solid arrows denote references and dashed arrows denote "instance of" relationships.

We now have a drawing without any meaning concerning formal semantics. It must be noted that every model that constitutes this drawing is expressed in terms of the same language – our linguistic meta model – and is therefore amenable to the framework.

4.2 Linking to a formal model

In a next step we are deciding that the drawing reflects an entity-relationship-diagram. Each box maps to an entity while the arrows are relationships between them. The framework already features a complete meta model of an entity-relationship-diagram. There are now three possible ways of linking the drawing to the model:

- Each box references an entity
- Each entity references a box
- A mediating model establishes links each referencing a box and an entity

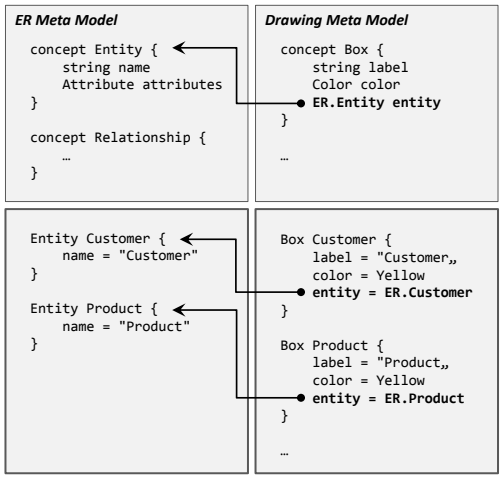

Figure 3. The drawing meta model has been connected to the meta model of an entity-relationship-diagram. Physical references may now be created between their instances.

We chose the first solution and add the necessary attribute to the Box concept within the drawing's meta model. Now we are able to establish a link between boxes and entities textually that means via the textual model editor.

To enable a more user-friendly experience one could add a graphical representation of the attribute to the graphical and editor definition. Links could then be actually drawn between the two models.

At this point, we would also like to mention that ideally the framework would now generate entities according to the boxes we've drawn. This ultimately corresponds to a model to model transformation which is a feature already planned but not implemented yet. So for now we create entities and link boxes to them.

There is now a physical reference between the free form drawing on the one side and the formal model on the other.

4.3 Imposing constraints

In a third step we want to ensure that the evolution from the drawing to the model was accurate. We want to guarantee that the mapping is correct and complete. So the third step renders our model more precisely by imposing constraints on it. We've decided to design a modeling constraint language for this that fully integrates with the OMME.

We believe that for achieving a certain level of expressiveness an expression language rather than a structural language is necessary. Graphical notations for such languages tend to become intricate so for now there is only a textual syntax. An important objective was that the language should be in the so called "problem domain". It must clearly set itself apart from a programming language on the implementation level. What does that mean for our constraint language?

- It must not reflect the machine model like an imperative language does by chaining memory operations in the form of statements and assignments.
- It must not expose an implementation-level type system like the *Object Constraint Language* (OCL) [15] does by e. g. forcing you to map a relationship `0..* employees` to a `Collection(Employee)`.
- There must be concise constructs that facilitate graph-based navigation on the model. This includes transitive closure and discovery of cyclic references.

We achieved the first objective by following the functional language paradigm. The second one is met by a type system that quantifies types with "multiplicities" like they are used in the UML. Compared to the OCL, graph-based navigation is substantially simplified by certain operators.

Concerning the link between our models again, we want to express that

- entities may only be linked to yellow boxes and
- every yellow box must be connected to an entity

In the words of our constraint language we say:

```
on Box
  rule ensures
    entity != undefined implies color = Yellow
    otherwise
      "Only yellow boxes may correspond to an
      entity"

  rule ensures
    color = Yellow implies entity != undefined
    otherwise
      "Every yellow box must correspond to an
      entity"
```

In case one of the rules is violated the according box is marked as defective and the message gets associated with the marker. The drawing and the entity-relationship-diagram which are fairly diverse models on the first view are now spanned by constraints ensuring their integrity. Furthermore, invariants regulating the drawing itself are imaginable, e. g. “boxes and arcs must not contain loops” and far more complex constraints could adjust the connection between the models.

Beyond this, OMME supports the modeling language itself to be refined by such constraints. They can forbid modeling constructs and enforce certain structures like single inheritance. This means we can switch the modeling paradigm by just referencing another collection of language-level constraints.

5. CONCLUSION AND FUTURE WORK

Of course, this doesn’t have to be the end of the process. Further models could be derived and linked to the others. Constraints could then validate whole chains of them.

This short example is a strongly simplified version of a use case that we are investigating. We want our framework to support the fluent evolution from a drawing to a process model and the connection to other models like entity-relationship-diagrams. Our focus is now on implementing features that further ease the described modeling method.

6. REFERENCES

[1] Microsoft. Microsoft Office 2010. 2010 [cited 2010 2010-08-12]; Available from: http://office.microsoft.com/en-us/visio/.

[2] IBM. IBM - Rational Software Architect for WebSphere Software. [visited 2010-08-12]; http://www-01.ibm.com/software/awdtools/swarchitect/websphere.

[3] Atkinson, C. and T. Kühne, Concepts for Comparing Modeling Tool Architectures. Lecture Notes in Computer Science, 2005(3713): p. 398-413.

[4] Eclipse Foundation. Eclipse Modeling Framework Project (EMF). 2010 [cited 2010 2010-07-08]; Available from: http://www.eclipse.org/modeling/emf/?project=emf.

[5] Steinberg, D., F. Budinsky, M. Paternostro, and E. Merks, Eclipse Modeling Framework. 2 ed. the eclipse series, ed. E. Gamma, L. Nackman, and J. Wiegand2008, Amsterdam: Addison-Wesley Longman. 744.

[6] Object Management Group. MOF 2.0 Specification. 2008 [cited 2008 2008-11-26]; Available from: http://www.omg.org/spec/MOF/2.0/.

[7] Volz, B., A Meta Model for Representing Arbitrary Meta Model Hierarchies, in Proceedings of the 2010 ACM Symposium on Applied Computing2010, ACM: Sierre, Switzerland. p. 2371-2372.

[8] Atkinson, C. Meta-Modeling for Distributed Object Environments. in 1st International Enterprise Distributed Object Computing Conference (EDOC '97). 1997. Gold Coast, Australia.

[9] Volz, B. and S. Jablonski, Towards an Open Meta Modeling Environment, in The 10th Workshop On Domain-Specific Modeling (DSM'10)2010: Reno, NV, USA.

[10] Odell, J., Advanced Object-Oriented Analysis and Design Using UML1998, New York, NY, USA: Cambridge University Press. 264.

[11] Henderson-Sellers, B. and C. Gonzalez-Perez, Connecting Powertypes and Stereotypes. Journal of Object Technology (JOT), 2005. 4(7, September - October 2005): p. 83-96.

[12] Atkinson, C. and T. Kühne, Meta-level Independent Modelling, in International Workshop Model Engineering 2000: Cannes, France.

[13] Atkinson, C. and T. Kühne. The Essence of Multilevel Metamodeling. in Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools. 2001. Toronto, Canada: Springer.

[14] Dahchour, M., A. Pirotte, and E. Zimányi, Materialization and its Metaclass Implementation. IEEE Transctions on Knowledge and Data Engineering, 2002. 14(5): p. 1078-1094.

[15] Object Management Group, Object Constraint Language Version 2.0, 2008, Object Management Group.