Electronic Communications of the EASST
Volume 18 (2009)

Proceedings of the Eighth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2009)

A Meta-Model-Based Approach for Specification of Graphical Representations

Merete Skjelten Tveit

15 pages

Guest Editors: Artur Boronat, Reiko Heckel
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

A Meta-Model-Based Approach for Specification of Graphical Representations

Merete Skjelten Tveit

merete.s.tveit@uia.no
Faculty of Engineering and Science, Department of ICT
University of Agder, Grimstad, Norway

Abstract: Meta-models are widely used for the specification of the internal structure of graphical modelling languages, and well-established standards (e.g. MOF) exist for this. For the graphical representation there is not the same agreement and no related standards. This paper presents a new meta-language for an *independent* specification of graphical representations. A diagram from the domain-specific language *Service* is used as a running example to show how this meta-model-based approach is appropriate for specifying the graphical representation in a precise way, but still on a high level of abstraction.

Keywords: language specification, graphical representations, meta-models, mapping descriptions

1 Introduction

Graphical modelling languages are an important part of information technology and in the software development process as they are very suitable for visualising structures, compositions and relationships between elements. Unfortunately, there is not the same broad understanding and agreement of how these graphical languages are specified in a formal way. A formal language specification for a graphical language normally consists of a structure definition (also known as abstract syntax), a graphical representation definition (concrete syntax) and a definition of the semantics. A formal specification of the language, including the graphical representation is important both for the users and for the tool developers that intend to build appropriate tool support for the language.

Meta-models are well-known as specification approach for the *structure* of a language, and there already exist standards like MOF [OMG03] for this purpose. MOF is a meta-language, also called a meta-meta-model, and defines all the concepts that are necessary to specify the structure of a language. A language structure that is specified based on the concepts in MOF is said to *conform to* MOF. The UML 2.0 [OMG04] meta-model is an example of a language structure definition that conforms to MOF. The structure meta-model is in turn instantiated when models in the language are made. This kind of instance hierarchy is one of the fundamental concepts in a meta-model-based approach.

For graphical representations, there still does not exist any standards similar to MOF. A meta-language for the graphical representation can however be based on the same principles as MOF, that is importing and reusing concepts from the *UML Infrastructure* [OMG07]. This paper

presents such a meta-language for specification of *graphical representation* based on reuse and extensions of concepts from UML Infrastructure, with some extensions for the graphics. The meta-language defines all the concepts that are required to specify the graphical representation of languages. The placement of the approach in the hierarchy is illustrated at the right-hand side in Figure 1. It is very important to note that the reuse of concepts from the UML Infrastructure does not restrict the application of meta-language to only the graphical representation of UML. This approach is suitable for most graphical languages, and is already applied to SDL [ITU99] in [PST07] and several domain-specific languages.

The main difference between the meta-language presented in this paper and already existing model driven approaches for specification of graphical representation is the combination of *independence*, *completeness* and *expressiveness*. The *independence* implies that the graphical representation is specified independent of the *structure meta-model*. This independence is a clear advantage when the languages are complex and when it is necessary to have more than one graphical representation for the structure. This independence is also identified as a crucial part of language specification approaches in [Tve08a]. Because of this independence, it is important to have precisely defined mapping relations between the structure and the graphical representation. In addition to this, the meta-language gives the possibility to describe the graphical representation of both languages and diagrams as *complete* constructions. This is opposed to the UML-DI [OMG06], the closest to a standard for graphical representation, whose purpose is to exchange diagrammatic information, not specify it. Features to describe all kinds of spatial relationships between elements in a graphical language also make the meta-language more *expressive* than many other approaches. The approach that is most similar to ours regarding independence is GMF [GMF], but this approach has weaknesses both regarding completeness and expressiveness. Section 5 presents these differences in more details.

The focus in this paper is mainly on the conceptual parts of the approach, nevertheless, a prototype that can be used to define graphical languages and generate graphical editors based on the description is implemented on the GMF platform, and is outlined in Section 4. The following sections will give a bottom-up description of the approach, starting at the diagram level. Section 2 presents the meta-model-based approach both for a specific diagram and for the language aspects (Section 2.3). Section 2.4 presents the most important concepts from the meta-language. Section 3 describes how the relationship between the structure and the representation is handled in this approach. The concluding remarks are found in Section 6.

2 An Overview of the Approach

The biggest difference between a string language and a graphical language is the dimensional space of the sentences in the language. While sentences in a string language are linear, the sentences (i.e. the diagrams) in a graphical language have a minimum of two dimensions. The differences are in the concrete representation. We consider a diagram expressed in a graphical language as a collection of graphical elements that are arranged in various ways. There are different methods for describing how the graphical elements are located and arranged to form valid diagrams. One way is to specify the placement of an element *physically* by using concrete coordinates. The coordinates will give the exact position of an element. Another method is

to specify the placement *logically*. A logical placement is normally described by using spatial relationships to describe where an element is placed relative to other elements. A deeper outline on how graphical elements are spatially related and arranged in diagrams is found in [BG04]. In the approach presented in this paper, the arrangements of the graphical elements are also specified in a logical way using *spatial relationships*.

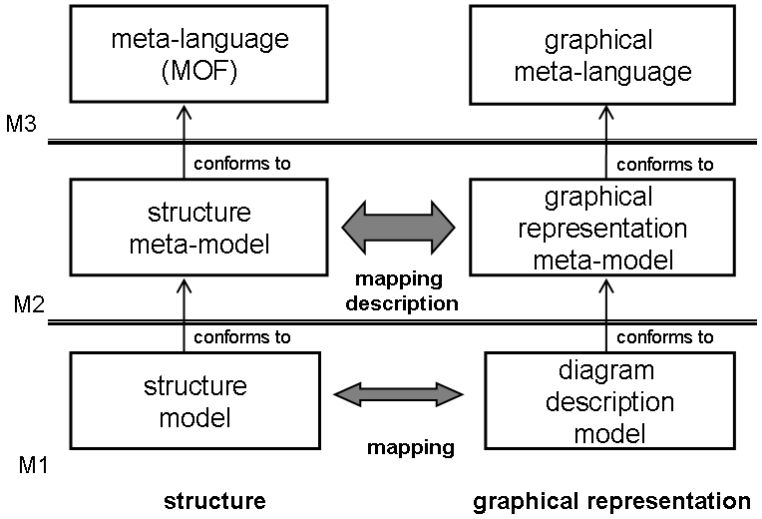

Figure 1: The meta-architecture of the approach

At the lowest level, M1, in the architecture presented in Figure 1, we have the *diagram description model*, which describes the graphical representation of one particular *diagram*. This includes instances of the graphical elements, their properties and how they are related. The model is an *abstraction of a diagram*, and is *complete* in the sense that the diagram can be constructed from it. The *diagram description model* conforms to the *graphical representation meta-model* for the language.

The *graphical representation meta-model* specifies all the graphical elements in the language, their *role* in the diagram and how they are spatially *related* to form well-formed diagrams. By role, we here refer to the *topological role* a graphical element plays, for instance *connection* or *container*. The approach provides a basis library which contains a set of pre-defined shapes. The library is language independent, that is, it can be used for all graphical representations. The graphical elements in the graphical representation meta-model specializes the pre-defined shapes in the library to get their geometrical properties. How the basis library is used in practise is described more detailed in Section 2.3.

At the upper-most level we have the *graphical meta-language* which defines all the *role concepts* that are necessary for specifying the graphical representation of a language. This meta-language re-uses concepts from UML Infrastructure. The three different levels are explained in more detail in Section 2.

2.1 Specification of Graphical Representation

As an example for the article, a diagram from the domain specific language *Services* will be used. The language is used to model service devices with plugs, and their connections. The example diagram in Figure 2 includes the following graphical elements: two *device symbols* with names "PC" and "Keyboard", one *female plug symbol* with name "USB_in" and one *male plug symbol* named "USB_out", one *connection point symbol* (the filled ellipse) and two *connector symbols*.

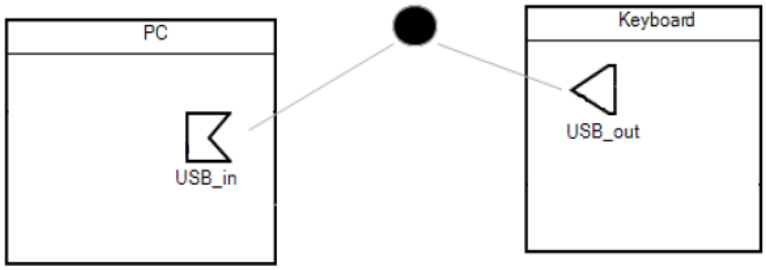

Figure 2: The service diagram used as running example

It is not only important to identify the graphical elements in a diagram, it is also necessary to describe how the elements could legally be related to each other. The relations between the elements are what actually create the diagrams in a language. For the service diagram we can recognise the following relations between the graphical elements: The *device symbols* have two *compartments* placed **inside**, one *name compartment* and one *plug compartment*. A horizontal line, also placed **inside** the device symbol, is separating the two compartments. The *device name* is placed **inside** the *name compartment*. The *plug symbols* are placed **inside** the *plug compartments* of the devices. The *plug names* are **associated with** the *plug symbols*. The *connector symbols* are **connected to** a *plug symbol* at their source end and to a *connection point* at their target end.

The next sections will give a bottom-up description of how the graphical elements and their spatial relationships can be specified in a meta-model-based way. We start with the description of the *diagram* in Section 2.2, the language graphical concepts in the service *language* are presented in Section 2.3 and finally the most important concepts in the graphical *meta-language* are presented in Section 2.4.

2.2 The Diagram Description Model (M1)

The graphical elements and how and where they are related form the most important aspects of the graphical representation of the service diagram, and are also what we would like to describe in the *diagram description model* in Figure 3. This diagram model is an object model which conforms to a meta-model defining the meta-classes like DeviceSymbol and ConnectorSymbol. The graphical elements are represented as objects and the relations as plain links with end names representing the spatial relationships between the elements involved in the relation. The meta-model is presented in Section 2.3.

Three kinds of relations were identified in the service diagram, and these three kinds of rela-

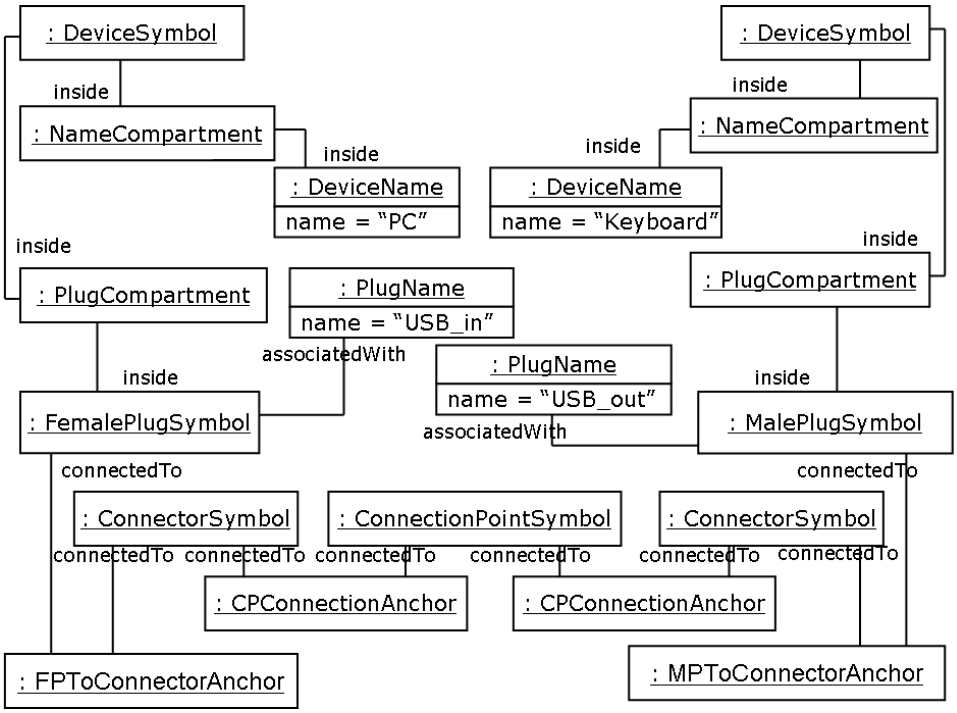

Figure 3: The diagram description model (M1) specifying a complete abstraction of the service diagram

tionships are also presented in the model: *inside*, *connectedTo* and *associatedWith*. The *inside* relation represents one graphical element placed inside another graphical element, and is given in the model as a link between the two objects that are involved in the relationship, e.g. the *device name* "PC" inside the *name compartment*.

The *associatedWith* relationship describes a special kind of relation since it is not directly visible in the diagram. An example of use is the instance of a *plug name* which is associated with an instance of a *male plug symbol* or *female plug symbol*. In the diagram it is not possible to see that these elements have a concrete relationship in between, but they are still related with an "invisible" association.

The *connectedTo* relationship is used to describe that two (or more) graphical elements are connected physically to each other. In the service diagram we have the *connector symbol* which at each target end is connected to the border of a *connection point symbol*. The connectedTo relationship (and also the associatedWith in some cases) also involves an *anchor* object (e.g. *cp connection anchor*) that specifies where the connecting appears. The anchor specifies a single point or a set of points that represent the concrete intersection point/area between the graphical elements that are involved. In the diagram description model (Figure 3), the *cp connection anchors* specify one single point at the border of the connection point where it intersects with the *target end* of a *connector symbol*.

The three spatial relationships, *inside*, *connectedTo* and *associatedWith*, are the only ones that are found necessary. Combining them with anchors gives possibilities to specify more complex

relationships, and also more specific connection areas, than for the connectors and connection points.

2.3 The Graphical Representation Meta-model (M2)

The model in Figure 3 is an abstraction of how the elements are arranged in the particular *diagram*. The objects in the model are instances of the graphical elements which are specified for the *language*. One particular diagram is just *one*, among many, legal representations in a language. At the M2 level in the meta-model hierarchy (see Figure 1) we have the *graphical representation meta-model* describing all the graphical elements in the language and their legal, spatial relations.

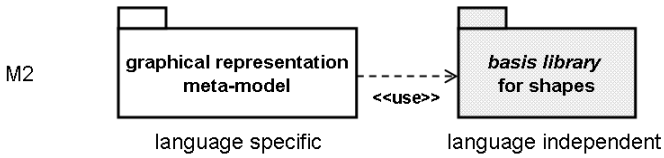

Figure 4: The graphical representation meta-model uses a language independent basis library

The *graphical representation meta-model* consists of an identification of all the graphical elements in the language, which roles (*connection*, *container shape* etc.) they play and how they are related to form well-formed diagrams. The graphical elements get their geometric properties from classes in a basis library (see Figure 4). The language independent *basis library* contains a number of pre-defined shapes that are considered as the most important shapes for graphical languages. The abstract descriptions of the geometrical shapes are all instances of UML::Class (cf. Section 2.4). The relationship between the graphical representation package and the basis library package is expressed by the UML dependency *use*, and the relationship between the graphical elements and the related pre-defined shapes is expressed by *inheritance* (also UML) in the meta-model for graphical representation. This implies that the graphical elements get the semantics of their roles from the meta-language at the M3 level, and their geometrical shape by inheritance.

A *device symbol* in the service language is shaped like a rectangle. Figure 5 illustrates how this is expressed in the meta-model: the *device symbol* inherits from *rectangle* in the basis library. In addition, the device symbol plays the role as a *container shape*, which means that device symbol can have other shapes inside. The role is set as a property *role*, where the type of the property indicates the actual role the element plays. While the role a graphical element is playing impact the spatial relationships the element can be involved in, the geometrical shape and properties the graphical element inherit from the basis library, impact the boundary and the attachment area(s). The boundary and the legal attachment areas are necessary to be able to specify where two graphical components can be connected together. The bounding is a *point set* representing the outer border (visible or invisible) of the geometrical shape. The attachment area is also a point set, representing the point or area where the shape could be legally related to other graphical elements through spatial relationships. In many cases, the bounding and attachment area are equal, like for the *connection point*. The classes Point and PointSet are part of the basis library.

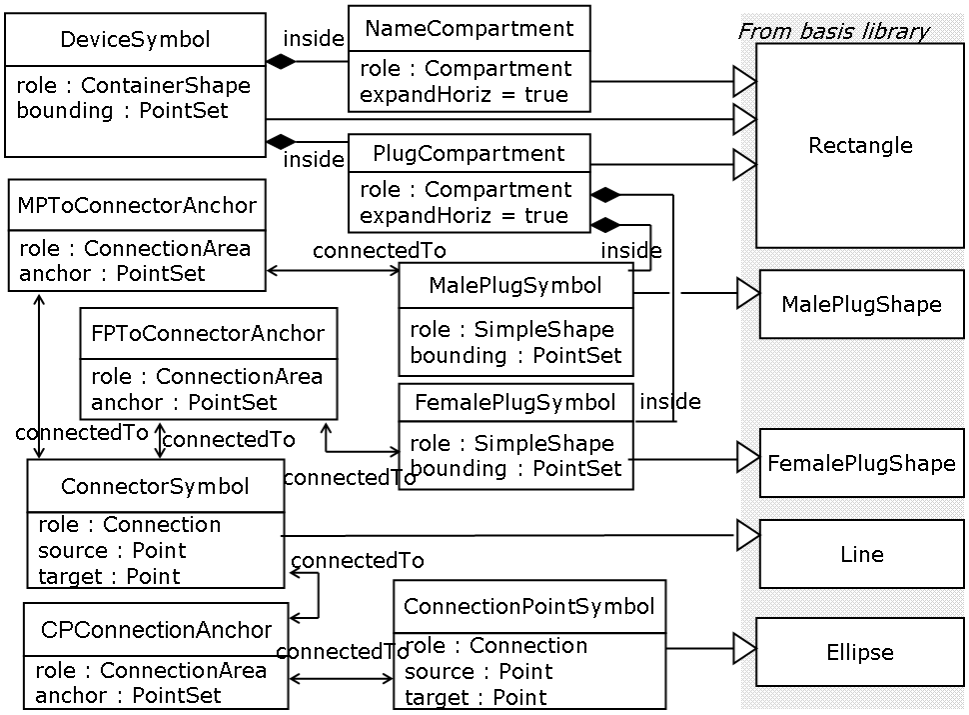

Figure 5: A subset of the *graphical representation meta-model* (M2) for the Service language

The *connector symbol* is shaped like a *line* and plays the role as a *connection*. A connection is special since it is always a part of a *connectedTo* relationship. For all the shapes in the basis library that are not characterised by their circumference, for instance different variations of lines, the attachment areas are set based on the endpoints of the shape. A *connector symbol* is shaped like a *line* and are in the basis library provided with two properties, *source* and *target*.

The *anchor* property in the *connection area* is derived from the *attachment* properties for the graphical elements involved in the relationship, and represents the possible intersection point/area. For the *cp connection anchor*, the anchor is specified (by constraint) to be all the possible points where the *target* of the *connection symbol* intersects the *attachment area* of the *connection point*. On this level, the anchor is relative to the involved elements. The *compartment* role is used for specification of compartments within a *container shape*. *Name compartment* and *plug compartment* are examples, and they both *expand* the container, the *device symbol*, horizontally.

Using a combination of spatial relationships (*inside*, *connected to* and *associated with*) and the connection areas for describing relationships between graphical elements makes this approach different from most other meta-model-based approaches. The advantage by using these features is the possibility to express complex spatial relationships on a high level of abstraction. The specification of the legal attachment areas for a graphical element is not a new idea itself, and especially in the traditional grammar-approaches (e.g. DiaGen [Min06]), attachment areas are used. In meta-model-based approaches on the other hand, these kinds of features are omitted in

many cases, which makes it difficult to specify attachment areas that are unequal to the shape boundings. The meta-model-based approaches that are most similar to the one presented in this paper, are described in more detail in Section 5.

2.4 The Graphical Meta-meta-model (M3)

The *graphical meta-meta-model* is the meta-language at the M3 level in the meta-model hierarchy (see Figure 1), and defines all the concepts that are necessary for specification of graphical representations. This section presents a slightly simplified version of the meta-meta-model, with focus on the concepts that are used in earlier sections.

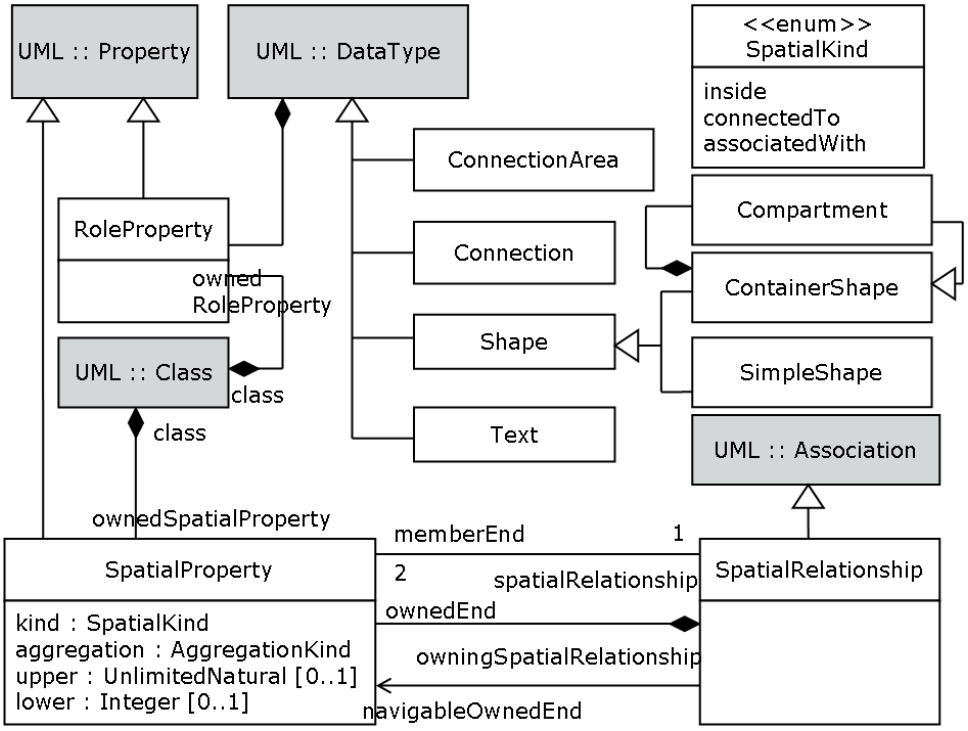

Figure 6: The meta-language (M3) for graphical representations

As a meta-language it is defined at the same level as MOF, and as for MOF it is defined by a meta-meta-model by re-using concepts from UML Infrastructure. The meta-classes representing *roles* are all defined as subclasses of UML::DataType which gives the type to the *role* property as illustrated in Figure 6. The role a graphical element plays, impacts which kind of spatial relationship the element can be a part of. There are four main kinds of types which all represent a special *role* a graphical element can have: *shape*, *connection*, *connection area* and *text*. A *container shape* is a special kind of shape that acts as container for other graphical elements. A container shape can contain *compartments*, which are used to arrange their contents.

To be able to describe how the graphical elements can be related to construct well-formed diagrams it is necessary to include features to express spatial relationships in the meta-language.

Also here, concepts from UML Infrastructure are used as basis. The most important concept is the *spatial property*, which is an extension of UML *property*. This concept gives the possibility to express *spatial relationships* (extension of UML association) with aggregation, multiplicity and navigable end. The new feature presented in spatial property is the *spatial kind*. This property is used to specify which kind of spatial relationship (*inside*, *connectedTo*, *associatedWith*) that exist between graphical elements.

The properties for the different roles on the language level are semantically generated from OCL constraints that are defined for the meta-language. For instance, only *container shapes* can act as containers for other graphical elements. This implies that whenever an *inside* relation is specified in the meta-model for graphical syntax, we need to assure that the element at the opposite end of the inside property, is of type *container shape*.

```
context SpatialProperty
   inv: self.kind = SpatialKind::inside implies
      spatialRelationship.memberEnd ->
      forAll( s | s <> self implies
      s.class.ownedRoleProperty.datatype ->
      oclIsKindOf(ContainerShape))
```

As we can see, the meta-language for specification of graphical representation is not very different from MOF, but it contains some additional semantics which are especially related to the different roles that a graphical element can have and the spatial arrangement of graphical elements.

3 Relating the Representation and the Structure

As we have seen in Section 2, the graphical representation is specified completely independent from the structure definition. This means there needs to be some kind of mapping defined between the two syntactic aspects to keep them synchronized. The complete separation is an important feature in this approach as it gives the possibility to have several representation definitions for the same structure and vice versa. The importance of keeping the structure and representation separated is discussed more widely in [Fon07]. The mapping meta-language that is defined within this approach is based on a study of the graphical languages SDL [ITU99] and UML [OMG04] and a categorisation of their relationships between the structure and the representation. From this study, three important mapping patterns were identified.

- **One-to-one** is the simplest kind of mapping, and also the most common.
- **Merge** is used to map graphical concepts which have several notation options. A well known example is signal declarations in SDL.
- **Partial description** is used to map graphical concepts which have a partial representation in addition to a complete. A typical example is partial descriptions of classes.

These patterns are also described with some more examples in [Tve08b]. The patterns are implemented as a mapping meta-meta-model, and a mapping meta-model that conforms to it is

transformed to the gmfmap model in the GMF framework. This is expressed in Figure 9 in Section 4. In the Service language, all mapping relationships are described using instances of the

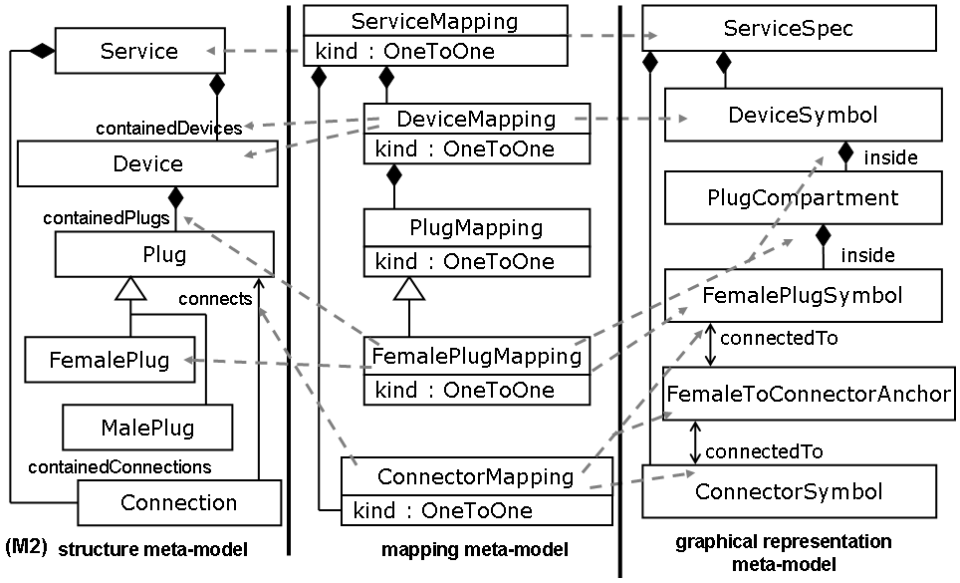

Figure 7: Illustration of mapping relationships in the Service Language

one-to-one pattern. This is illustrated in figure 7.

4 The Architecture of the Implementation

The prototype is implemented using the Eclipse framework, EMF, GMF and QVT for model-to-model transformation. The aim of the tool is to be able to use the approaches for specification of the graphical representation and the mapping description to generate a graphical editor. Figure 8 gives an overview of the workflow in the prototype.

To generate a graphical editor for a language using the prototype, the user needs to define:

1. The *structure* of a language.
2. The *graphical representation* of a language
3. The *mapping description* that specifies the relationships between the graphical representation and the structure.

   These three definitions, which together cover the complete syntactic aspects of a language, is automatically transformed using pre-defined QVT transformation, to GMF descriptions of the given language. The descriptions provide the necessary information to automatically generate:

4. A GMF graphical editor for the language

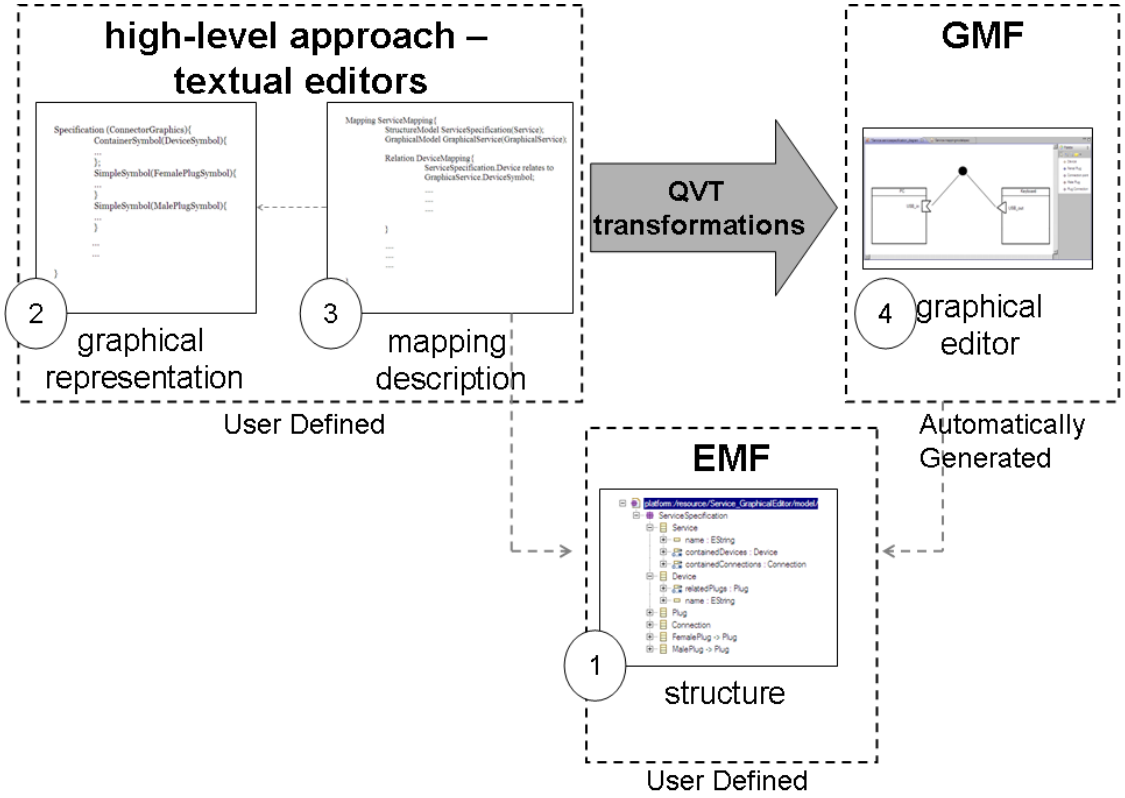

Figure 8: An overview of the implementation

The Eclipse Modeling Framework (EMF) [BSM+03] makes it possible to easily create tree-editors by generating code from the meta-languages for graphical representation and mapping that are defined as ecore-models. These tree-editors can be used directly to specify the graphical representation, and the mapping relationships to the structure. To make the prototype more user-friendly, the Textual Editing Framework (TEF) TEF [Sch08] have been used to specify textual representations and to generate textual editors for the two meta-languages. The textual editors makes it more straight-forward to specify a graphical language and its representation. This *high-level* description is then *transformed* to the lower level GMF framework [GMF] which is responsible for generating the graphical editor features that are necessary. The model-to-model transformations between the high-level specification and the GMF models are completely handled using QVT-R [OMG08], implemented using the tool *ikv++ medini QVT*. The transformations are illustrated in Figure 9 which shows the high-level approach on the left-hand side, the GMF framework on the right hand side (with the EMF in the middle since the structure meta-model is used by both approaches) and the transformation arrows in between. The three transformation descriptions are in the figure illustrated by numbered, stippled arrows:

1. from the graphical representation meta-model to graph.gmfgraph
2. from the graphical representation meta-model to tool.gmftool

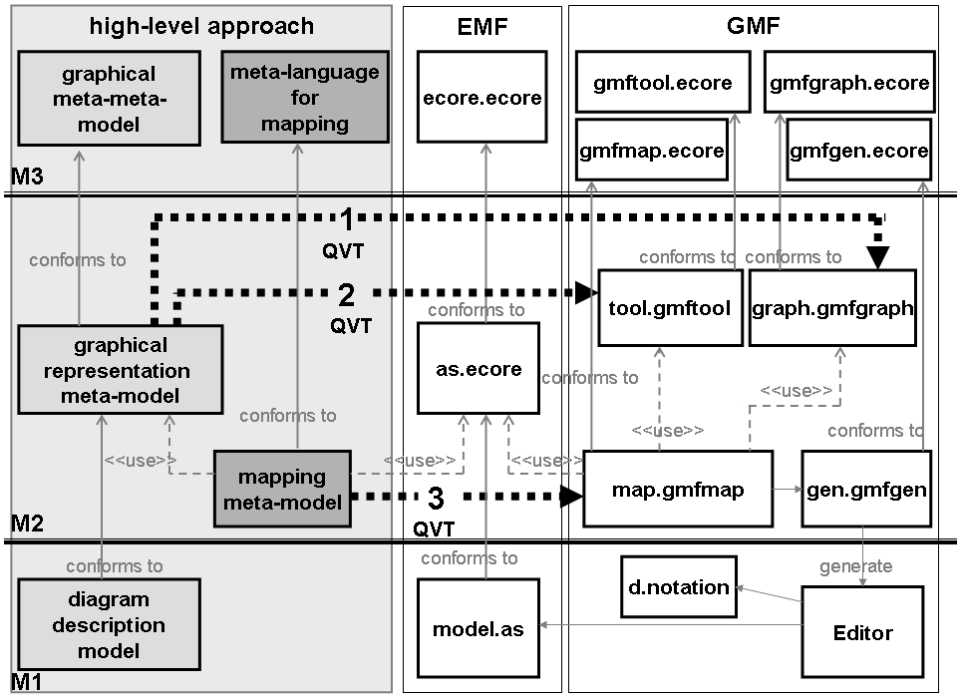

Figure 9: An overview of the high level approach (left-hand side) and its transformation to the GMF platform

3. from the mapping meta-model (high-level approach) to map.gmfmap

These three gmf-models are sufficient for generating a GMF editor.

There are some minor differences between the conceptual aspects presented in Section 2 and the practical aspects covered by the prototype. This is mostly related to the implementation of the meta-language, since the conceptual part of the approach are described re-using concepts from UML Infrastructure, while the implementation is based on EMF. These differences are solved as follows: The *UML::Class* is replaced by *EClass*. For the *spatial relationship*, which extends UML::Association, and *spatial property*, which extends UML::Property, the challenge is bigger. While UML uses associations, with properties for the association ends, EMF uses references to express relationships between classes. In the implementation, this is solved by merging the *spatial relationship* and *spatial property* into one unit, and let it extend *EReference*. The property *EOpposite* from *EReference* is used on the level below to express bidirectional spatial relations between graphical elements. By adding the *property kind* of type *spatial kind* to the new unit all the necessary semantics are kept.

5 Related Work

There exist a number of meta-tools that generate graphical editors from language specifications which involve meta-models to some degree: XMF-Mosaic [Lim05], The Graphical Modeling

Framework (GMF) [GMF], MetaEdit+ [Met05], The Generic Modeling Environment (GME) [LBM07] and Tiger [EEHT05] to mention some. All of these meta-tools present their own approach for handling and specifying the graphical representation. While most of these approaches (e.g. MetaEdit+ and GME) specifies the graphical representation on top of the structure meta-model and with this keep the graphical information as properties in the structure elements, GMF and XMF-Mosaic presents new, independent meta-languages for the graphical representation. The graphical representation description is then related to the structure meta-model by a syntactic mapping description. Of these two, GMF is the approach that shares most with the approach presented in this paper as it has a strong focus on separating the structure and the representation.

The biggest differences between the Eclipse-based GMF and the approach presented in this paper are related to two aspects: *expressiveness* and *completeness*. It is not possible to explicitly specify spatial relationships between graphical elements or specific attachment areas in GMF, except compartments, which make it possible to handle inside relationships. The relationships between nodes and connections are handled implicitly. This implicit specification makes the approach less expressive than the one presented in this paper. The other difference is related to the completeness of the graphical representation. The meta-language presented in this approach aims to specify the graphical representation as a complete construction (model), and not only single graphical elements collected in a container (canvas in GMF) without any relationships in between. In GMF, the relationships between a compartment and its content, and between a connection and its connected node, are specified in the mapping description, and not in the graphical representation itself.

This is also the case for another approach presented by F. Fondement in [Fon07] that is worth mentioning. This approach is different from both GMF and XMF as it does not provide a new meta-language for the graphical representation, but instead proposes to extend the already existing structure meta-model by an extra layer of visual objects. This approach is interesting as it presents features for specification of spatial relationship (contains, overlap, connects and nearby) between graphical elements. The weakness with this approach, as with GMF, is that the spatial information is specified as a part of the mapping description, and not in the graphical representation itself.

6 Conclusion

This paper presents an approach for meta-model-based graphical representation. The central part of the approach is a new meta-language based on re-use and extensions of concepts from UML Infrastructure. The meta-language provides features for specification of graphical elements and their spatial relationships, and this is done independently of the structure meta-model. This is an advantage since it allows languages to have more than one graphical representation. The introduction of *connection areas* together with three kinds of *spatial relationships* (*inside*, *connected to* and *associated with*) gives the possibility to specify both simple and more sophisticated relationships between and within diagram elements on a high level of abstraction. The expressiveness, the possibility to describe the graphical representation for both the language and its diagram as a complete construction, and the complete independence from the structure specification are the main advantages with the approach presented.

A prototype tool for the meta-languages for graphical representation and the mapping approach are implemented, based on EMF and transformed to the GMF platform. In the tool the meta-languages has textual representations and textual editors implemented using TEF [Sch08]. With this, the approach can be used to specify the graphical representation of a graphical language, which together with a structure specification and the mapping description makes it possible to generate a GMF editor from the specification. The future plan is to also specify a graphical representation for the meta-language and generate a graphical editor for it, using the approach and the meta-language itself.

Bibliography

[BG04] P. Bottoni, A. Grau. A Suite of Metamodels as a Basis for a Classification of Visual Languages. *2004 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2004), 26-29 September 2004, Rome, Italy*, pp. 83–90, 2004.

[BSM+03] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, T. J. Grose. *Eclipse Modeling Framework (The Eclipse Series)*. Addison-Wesley Professional, Aug. 2003.

[EEHT05] K. Ehrig, C. Ermel, S. Hänsgen, G. Taentzer. Generation of visual editors as eclipse plug-ins. In *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), November 7-11, 2005, Long Beach, CA, USA*. Pp. 134–143. 2005.

[Fon07] F. Fondement. *Concrete Syntax Definition For Modeling Languages. PhD Thesis*. Ecole Polytechnique Federale De Lausanne, 2007.
http://library.epfl.ch/en/theses/?nr=3927

[GMF] Graphical Modeling Framework.
http://www.eclipse.org/gmf

[ITU99] ITU-T. SDL - ITU-T Specification and Description Language (SDL-2000). ITU-T Recommendation Z.100, 1999.

[LBM07] A. Ledeczi, D. Balasubramanian, Z. Molnár. An Introduction to the Generic Modeling Environment. In *Proceedings of MDD-TIF07, Model-Driven Development Tool Implementers Forum*. 2007.
http://www.dsmforum.org/events/MDD-TIF07/GME.2.pdf

[Lim05] X. Limited. Language Driven Development and XMF-Mosaic. 2005.
http://www.xactium.com

[Met05] MetaCase. MetaEdit+. Version 4.0. Evaluation Tutorial. Technical report, MetaCase, 2005.
http://www.metacase.com/support/40/manuals/eval40sr2a4.pdf

[Min06] M. Minas. Syntax Definition with Graphs. *Electronical Notes in Theoretical Computer Science* 1:19–40, Feb. 2006.

[OMG03] OMG. *Meta Object Facility (MOF) 2.0 Core Specification*. Object Management Group, Oct. 2003. ptc/03-10-04.

[OMG04] OMG. *UML 2.0 Superstructure Specification*. Object Management Group, Oct. 2004. ptc/04-10-02.

[OMG06] OMG. *Diagram Interchange, version 1.0*. Object Management Group, Apr. 2006. ptc/06-04-04.

[OMG07] OMG. *OMG Unified Modeling Language (OMG UML) Infrastructure, V2.1.2*. Object Management Group, Nov. 2007. ptc/06-10-06.

[OMG08] OMG. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*. Object Management Group, Apr. 2008. ptc/07-07-08.

[PST07] A. Prinz, M. Scheidgen, M. S. Tveit. A Model-Based Standard for SDL. In *SDL Forum*. Pp. 1–18. 2007.

[Sch08] M. Scheidgen. Textual Modelling Embedded into Graphical Modelling. In *Model Driven Architecture - Foundations and Applications, 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings*. Lecture Notes in Computer Science 5095, pp. 153–168. Springer, 2008.

[Tve08a] M. S. Tveit. Specification of Graphical Representations - using hypergraphs or metamodels? In *Norsk informatikkonferanse NIK 2008*. Pp. 39–50. tapir akademiske forlag, 2008.

[Tve08b] M. S. Tveit. Towards Diagrammatic Patterns. In *Diagrammatic Representation and Inference, 5th International Conference, Diagrams 2008, Herrsching, Germany, September 19-21, 2008. Proceedings*. Lecture Notes in Artificial Intelligence 5223, pp. 427–429. Springer, 2008.