Specification of Graphical Representations - using hypergraphs or meta-models?

Merete Skjelten Tveit

Faculty of Engineering and Science, University of Agder
Grooseveien 36, N-4876 Grimstad, Norway
merete.s.tveit@uia.no

Abstract

This paper presents the essence of a larger study related to graphical languages and how their syntax is specified formally. The existing approaches for graphical language specification could be divided into two main categories: the traditional grammar-based approaches and the more modern meta-model-based approaches. The work presented in this paper is based on a comparison of two formalisms, one from each category. The two approaches are both applied to the same example language using one meta-tool for each approach: DiaGen for the hypergraph grammar and GMF for the meta-model-based approach. The comparison shows that apart from usage differences, the approaches have a different basic view on the syntactic aspects of a language and how the aspects are related. This again has an impact on the expressiveness and abstraction level of the language specifications.

1 Introduction

Graphical modelling languages are an important part of the software development process as they are good for visualising structures, compositions and relationships between objects. Unfortunately, there does not exist any common agreement in terms of standards on how to formally specify all the necessary aspects of these languages; and well-established, standardised languages like UML 2.0 [15] are still not specified completely formally. This paper focuses on the syntactic aspects of a graphical language specification, more precisely on the *structure* (also knowns as abstract syntax) in general and on the *graphical representation* (concrete graphical syntax) in particular. The terms structure and graphical representation will be used in this paper to cover the syntactic aspects. In addition, a complete language specification should include constraints and behaviour (also called semantics). For a deeper description of the different language aspects, see [12].

The biggest difference between textual languages and graphical languages is in the dimensional space of the concrete sentences in the language. While sentences in a textual language are one dimensional, the sentences (i.e. the diagrams) in a graphical

This paper was presented at the NIK-2008 conference; see http://www.nik.no/.

language have minimum two dimensions. This dimensional difference affects the possible relationships between the graphical objects in the representation of the language. We consider a diagram expressed in a graphical language as a collection of graphical objects that are related to each other by *spatial relationships* (e.g. *in*, *at*, *left of*) in a well-formed way. How to describe these spatial relationships and the arrangement of the graphical objects are one of the main challenges in the area of graphical language specification. A deeper discussion of spatial relationships in graphical languages is found in [6].

The absence of an agreement is not the same as an absence of approaches, and there exist a number of different formalism for specification of the syntactic aspects of a graphical language. The first formal approaches [8] were simple modifications of grammar formalisms used to specify textual languages and were very limited. The grammar-based approaches that are used today, like graph grammars, are still based on formalisms used for textual languages, but are way more expressive. In addition to these grammar-based approaches, more modern approaches based on meta-models have become popular in the last decades.

This paper is based on a larger study, where two different approaches, the *hypergraph grammar approach* and the *meta-model-based approach*, are compared. To be able to compare the two approaches, they are both applied to a common example by using automatic editor generators that are based on the given approaches. For the hypergraph grammar approach, a natural choice has been to use the well-known and established DiaGen [11]. The Graphical Modeling Framework (GMF) [3] is the meta-model-based tool that is used. Other editor generators that also could have been used are: VLHCC [6] and PROGRESS [16], for the traditional grammar-based approach and XMF-Mosaic [5], MetaEdit+ [10] and GME [9] for the meta-model-based approach, to mention some.

As a basis for the comparison and evaluation of the two approaches, a simplified version of State Machines from UML 2.0 [15] is used as example language. A specification of the graphical representation consists of two main parts: the graphical objects and their visual appearance, and how the objects can relate spatially to each other in a well-formed way. The *requirements* for the specification and the generated editors are as follows: A state diagram (see Figure 1) consists of states and transitions between states. We have four kinds of graphical objects: *initial state*, *final state*, *regular state* and *transition*. The initial state is represented as a small, solid filled circle, the final state is represented as a circle surrounding a small and solid filled circle, while the regular state is drawn as a rounded rectangle with the state name placed inside. *Transitions* between states are drawn as arrows, and they may be labelled with text. The transitions are *connected to the border of* one kind of state in each endpoint. A regular state may *contain* other states or state diagrams, and is then called a *composite state* ("CS" is an example in the diagram). The example language is simple, but this does not necessary means that the *description* of the language is simple. The requirements cover the diagrammatic aspects that are necessary to explore the nature of the specification approaches.

The main focus within this work and comparison is on how the approaches are used to specify the *language* (the syntactic aspects), but it is important to keep the tool (editor) specification in mind. It is important to be aware of the difference of specifying a language in isolation and specifying an editor for a given language. An editor requires information that goes beyond the language specification, which also could lead to solutions that differ from the basic principles in the specification approach the editor generator is based on.

In addition to get an overview of the differences between the two approaches, the motivation for doing this study was to use the comparison to high-light the most important

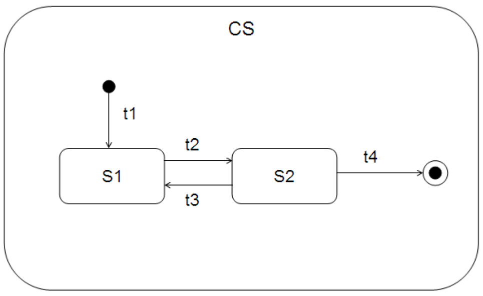

Figure 1: A simple state machine diagram

aspects in a description of the graphical representation, and to identify which parts of these descriptions that are most difficult to handle. The tools and approaches are studied from two different angles: On one hand, the expressiveness of the approaches is interesting: What is possible to express using the approach? How well does it satisfy the language requirements given? On the other hand, we have the conceptual aspects: What are the important aspects in the language specification? This paper will only give the essence of the comparison, and the presentation of the results will be based on four questions:

- How does the approach and the related tool satisfy the language requirements for state machines?
- Which techniques are used to specify the spatial relationships between the graphical objects in the language?
- What are the relations between the graphical representation and the language structure?
- What is the central language aspect in the specification? How does this impact the language specification?

Even if the main focus in this paper is on the graphical representation, it is difficult to consider this language aspect in total isolation from the internal *structure*, as the aspects need to be related to form a complete language specification. The theory of syntax specification and the different syntactic aspects are presented in section 2.

The rest of the article is structured as follows: Section 3 covers the hypergraph grammar approach and DiaGen, while section 4 presents the meta-model-based approach and the tool GMF. The evaluation of the two tools and approaches based on the experiment is discussed in section 5.

2 The syntactic aspect of a language

From the theory of textual language specification, we are familiar with the distinction between the syntax and the semantics of a language. The syntax specifies the structure of sentences in the language, while the semantics assigns a meaning to the sentences. According to Noam Chomsky [4], the syntax of a (sequential) textual language is specified using a generative string *grammar*. A string is an ordered sequence of symbols, where the symbols are taken from a predetermined set, also called an alphabet. A generative grammar G consists of a finite set of non-terminals, N, and a finite set of terminal symbols, T, that is disjoint from N. There also exist a finite set P of production rules and a start symbol that is in N. The *language* denoted as L(G) is defined as the *set of strings* that can be generated by starting with the start symbol and then applying the production rules until no more non-terminals are present. This way of specifying the language syntax is also forming the basis of most of the grammar-based methods that are used today, both for textual and visual languages. The traditional graph grammar is one approach that is based on an extension of the concepts of traditional string grammar [7]. A *graph* is simply a collection of nodes connected by edges. A *graph grammar* is similar to a generative grammar as it consists of terminals, non-terminals and production rules. The biggest difference is that the right-hand side and the left-hand side of the productions consist of graphs. A *graph language* is the *set of all graphs* that may be obtained by applying an arbitrary sequence of production rules to the initial graph until no more non-terminals exist.

Syntax division - the structure and the representation

We usually distinguish between two different kinds of syntax: the *structure* that describes the concepts in the internal structure of a language, and the *representation* that describes how the language structure is (re)presented to the user, both in text and by graphics.

The distinction between structure and representation has been made by several authors since the late 1950's. Noam Chomsky distinguished between the *deep structure* (structure) and the *surface structure* (representation) for natural textual languages [4]. The idea of separating the syntactic aspects was not transferred to the area of visual languages until the 1990's.

There are several reasons for putting so much focus in the distinction of syntactic aspects. One is that it gives the possibility to have several representation definitions for the same structure definition (and vice versa). Another important reason is that it gives a much cleaner language specification if the language is large and complex. An even more important reason is illustrated in Figure 2. It is not difficult to see that the state machine in Alt. 1 on the right-hand side correspond to the structure (ad-hoc notation) on the left-hand side. Alt. 2 shows another variation of the same state machine. These two state machines are both corresponding to the same structure definition. In contrast to the representation, the structure has always only one occurrence of each object. The structure definition is the *idea* of what we want to describe. This idea could then be represented in several different ways. In some cases, the structure and the representation would be very similar and the relationship will be one-to-one, but for most languages there will be situations like the one described here. It is important that the specification approach can handle this. The structure and the representation are two quite different views on the language, and to be able to specify the language in a correct way according to its language definition, these views need to be handled independently.

An interesting issue regarding the separation of the syntax definition, is which of the syntactic aspects is seen as the central syntactic aspect. Andries, Engels and Rekers presented in [16] their graph grammar approach with a *spatial relations graph* (SRG), an *abstract syntax graph* (ASG) and the relationship inbetween. This approach was one of the first that separated the structure and the representation, and the method is widely used in the area of graph grammars. A diagram expressed in the language is transformed to a SRG, a graph that describes the graphical *representation*. This graph is then *reduced* or *abstracted* into a new graph in a way that removes all information related to the graphics. The new graph, the ASG, represents the internal *structure*. The representation is the central aspect in this approach, and there is an abstraction-based relationship between the

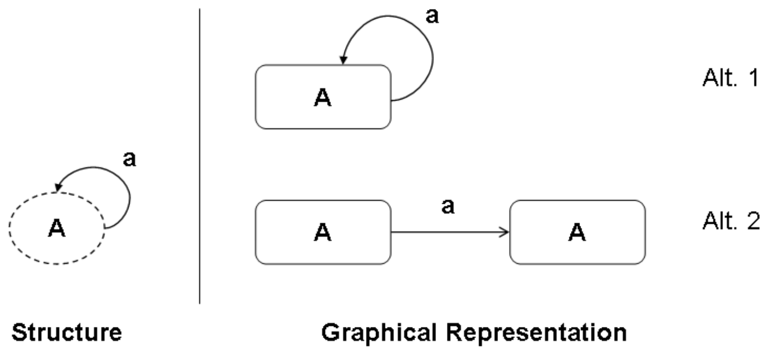

Figure 2: Two variations of graphical representation corresponding to the same structure definition

two syntactic definitions.

OMG's model-driven architecture (MDA) brings a new focus to the distinction of the syntactic aspects with the *separation of concerns* [14]. A meta-model-based approach adhering to MDA makes it possible to define the structure and the representation as two completely independent aspects and then provides a powerful *mapping* that binds together the related concepts. As a contrast to the grammar-based approaches, the structure is the central aspect in the meta-model based approaches, and is in turn *extended* with the graphical information.

Since the structure definition is on a higher level of abstraction than the graphical representation, which is more concrete, the specification approaches that treat the structure as the central aspect will produce language specifications that are on a higher level of abstraction. The approaches that see the representation as the central aspect will specify the language in a more concrete way, which, as we will see in section 3, often leads to a higher level of expressiveness.

3 Hypergraphs and hypergraph grammar

A labeled hypergraph, the internal model used in DiaGen, is a generalisation of a directed graph. In a hypergraph the edges are hyperedges, which means they can be connected to any fixed number of nodes. While graph edges visit pairs of nodes, hyperedges visit arbitrary sets of nodes. The formal definition of a hypergraph is: *A hypergraph H is an ordered pair (V, ϵ) where V is a set of vertices and ϵ is a set of edges such that $\epsilon \subseteq$ P(V)* [7].

A hypergraph grammar is used to specify the language *structure*. A context-free hypergraph grammar, which is the simplest kind of hypergraph grammar, consists of two sets of labeled hyperedges, one for *terminals* and one for *non-terminals*, and a starting hypergraph that contains non-terminal hyperedges with labels only. The syntax is described by a set of production rules of the form L::=R, where both L (left-hand side) and R (right-hand side) are a hypergraph. The hypergraph grammar defines the *set* of all hypergraphs, consisting of terminals only, that may be obtained by applying an arbitrary sequence of production rules to the starting hypergraph. Context-free hypergraph grammars is very simple, and are therefore not suitable to describe most graphical languages. DiaGen uses instead context-free hypergraph grammars with embeddings, which is way more expressive [11]. This kind of grammars allows extra embedding productions which have an additional hyperedge at the right-hand side. The rest of

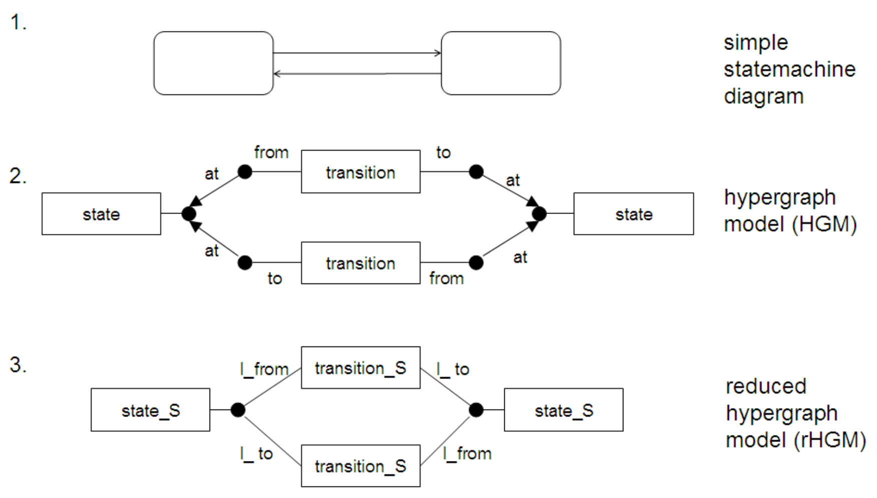

Figure 3: A simple state machine diagram, and its corresponding HGM and rHGM

this section will describe in more detail how DiaGen uses hypergraphs and hypergraph grammars to specify graphical languages and generate editors based on the specification.

Specifying state machines using DiaGen

DiaGen (The **Dia**gram Editor **Gen**erator)[11] is an editor generator where the language specification approach is based on labeled hypergraphs and hypergraph grammars. DiaGen consists of two main parts: the *designer* and the *editor framework*. The DiaGen designer is a GUI tool used to specify the language. The specification is internally represented as an XML document, which again is translated into Java code by the designer's code generator. This Java code will together with the DiaGen editor framework implement an editor for the specified language.

The language specification is described in the designer and consists of several parts. This section will describe the parts that are most relevant for the graphical representation and the comparison. For a more in-detail description, have a look at [11].

The DiaGen *components* are the graphical objects in the language, and they contain information about graphical appearance and attachment areas. The attachment areas specify where it is possible to relate the objects to other objects. The spatial relations that occur between the graphical objects (e.g. *inside*, *at*, etc.) are specified based on the attachment areas defined for each component. The spatial relationships need to be specified by java code for each specific relation type. This information covers the graphical representation aspect and gives a generated editor satisfying the requirements for the state machine language given in section 1 perfectly.

The diagrams modelled in an editor are translated to *hypergraph models* (HGM) based on the information described above. In this internal hypergraph model, each of the components is represented by a single hyperedge having *attachment areas* that in the model are represented as nodes. These nodes are visited by the component's hyperedges, and the nodes and edges define an unconnected hypergraph. If the components (actually the corresponding attachment areas) are related in a special way, these nodes will be connected by the specified relationship edges (based on the *spatial relationships* definitions). This internal hypergraph model can be compared to the SRG model for graph grammars mentioned in section 2.

Figure 3 illustrates the relationship between a simple state machine diagram and its corresponding internal HGM (1. and 2. in the figure). The diagram shows two *regular states* with a two *transition* in between, one in each direction. In the HGM this is illustrated by four *hyperedges* (rectangles in the figure), one for each graphical object. The two states have one *attachment area* each represented in the HGM as *nodes* (filled dots in the figure), and the transition has two attachment areas. The arrows between the attachment areas are special relationship edges that represents the *spatial relationship* between the graphical objects. Both the spatial relationships are at-relationships, indicating that the *transition* endpoint (its legal attachment point) is connected *at* the *state* border (its legal attachment area). The transitions also have labels (*to* and *from*) on their tentacles (line between the graphical object hyperedge and the node attachment area) to distinguish the two transition ends.

A hypergraph model is a rather complex model even for simple languages. To be able to syntactically analyse the internal model in an effective manner, it has to be *reduced* or *abstracted* to remove its complexity. This reduction is based on rules that are given for the specific diagram language. These *reduction rules* are like graph transformation rules that identify those sub-hypergraphs of the HGM that carry information about the diagram, and build a *reduced hypergraph model* (rHGM) according to that. The main idea behind this reduction step is to remove all the information irrelevant for the parsing process, and while the HGM represents each graphical object and each relationship between them directly, the rHGM is generally represented in terms of larger groups of *structure* components and their relationships. In the reduction process the spatial information is removed and the graphical objects are replaced with the equivalent *terminals*. This implies that the reduction rules specify how to transform an internal model that describes the graphical representation of a diagram (HGM) to a model that describes the structure (rHGM). These rules are representing the relationship between the structure aspect and the representation aspect of the language. This is also illustrated in figure 3 (2. and 3.) and we can see how the spatial relationship hyperedges are removed, and the graphical objects are replaced by their corresponding terminals. The attachment areas are replaced by links acting as nodes.

This way of specifying a graphical language, with the representation as the central aspect, makes the approach very expressive. It is clear that the approach is able to handle way more complex *language descriptions* than the one described in this paper. On the other hand, the concrete focus in the approach makes the specification more low-level than desired. The close relation between the representation and the structure also leads to difficulties of handling situations like the one described in section 2 and figure 2.

4 Meta-model-based approach

A meta-model is a model consisting of object-oriented classes with attributes which describe the language/domain concepts and their properties. Associations between the classes describe how the concepts are related to each other. Meta-models are a well-known specification approach for the *structure* of a language, and there already exist standards like MOF [13] 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 be *an instance* of MOF. The UML 2.0 [15] meta-model is an example of a language structure definition that is an instance of MOF. The structure meta-model is in turn instantiated for models of the language. For the graphical representation, there still does not exist any standard similar to MOF, but there exist approaches and tools providing meta-models also for the graphical representation. The structure definition is the central aspect in a

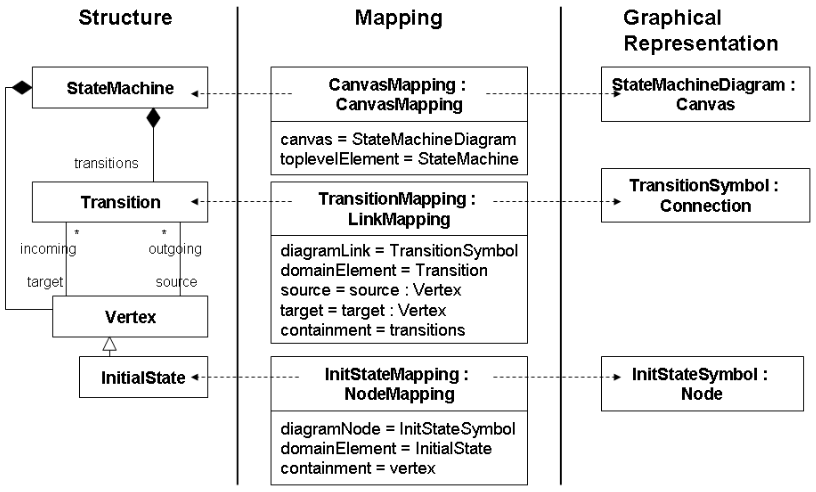
Structure
Mapping
Graphical Representation
StateMachine
transitions
Transition
incoming
outgoing
target
source
Vertex
InitialState
CanvasMapping : CanvasMapping
canvas = StateMachineDiagram
toplevelElement = StateMachine
TransitionMapping : LinkMapping
diagramLink = TransitionSymbol
domainElement = Transition
source = source : Vertex
target = target : Vertex
containment = transitions
InitStateMapping : NodeMapping
diagramNode = InitStateSymbol
domainElement = InitialState
containment = vertex
StateMachineDiagram : Canvas
TransitionSymbol : Connection
InitStateSymbol : Node

Figure 4: The mapping specifies the relationships between the structure and the representation

meta-model-based language specification, and the graphical representation is related to the structure by a mapping definition. The next section describes the meta-model-based approach provided by GMF.

Specifying state machines using GMF

The Graphical Modeling Framework [3] is a part of the Eclipse Modeling Project and acts as a bridge between the Eclipse plug-ins EMF [1] and GEF [2]. GMF defines and provides five specific meta-models that form the basis for the definition of the syntactic aspects of a graphical language and related editors: The **graphical definition model** is used to describe the graphical representation of the language and the **tooling definition model** is used to describe the diagram editor. The **mapping model** is responsible for the relationship between the information in the *graphical definition model*, the *tooling definition model* and the *structure (ecore) model.* In addition we have the **generator model** which is responsible for generating the editor code and the **notation model** which contains the graphical diagram information during runtime of the editor. Only the graphical definition model and the mapping model will be described in more detail in the next section.

In GMF, as for most meta-model-based tools, the *structure* forms the foundation of the language specification. The structure is an instance of the *Ecore*, the MOF-like meta-meta-language provided by the EMF framework. For the graphical representation GMF provides a separate language, the *graphical definition model*, which resides on the same meta-level as *Ecore*. The *graphical representation specification* is then an instance of the graphical definition model.

There are four kinds of graphical objects (called diagram elements in GMF) in the graphical definition model: *node*, *connection*, *compartment* and *label*. Each of these diagram elements has a reference to one *figure descriptor* which carries the graphical information. The same figure descriptor can be referenced by more than one diagram element. The graphical definition model does not provide the possibility of specifying the legal attachment areas explicitly. The nodes have implicit attachment areas at their borders, and the connections have one implicit attachment area at each endpoint. The spatial relationships between nodes and connections are also implicit; a connection has

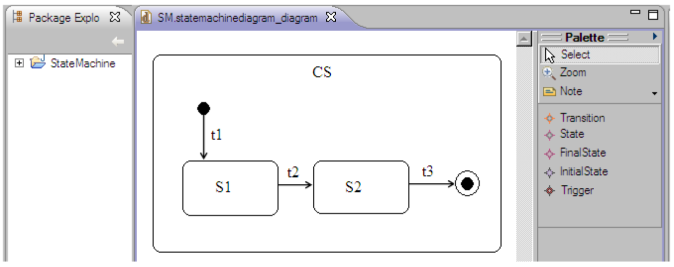

Figure 5: The generated GMF editor for state machines

to be attached to a node in each end. The only spatial relationship possible to specify explicitly is a containment relationship. The diagram element *compartment* is used for this purpose. A diagram element specified as a *compartment* is allowed to contain other diagram elements. These limitations in expressing spatial relationships more explicitly are a clear weakness in GMF and will lead to problems if the language description includes more complex spatial relationships.

The graphical information is defined independently from the structure meta-model (ecore-model), and to be able to generate an editor from this information, the graphics needs to be related to the structure. The *mapping model* defines all the mapping concepts that are necessary to specify the relationships between the two syntactic aspects. Figure 4 gives an illustration of how the structure definition and the graphical representation definition are related by the mapping definition. The left-hand side of the figure shows a simplified version of the structure definition for state machines containing four classes: *statemachine*, *transition*, *vertex* (superclass for the different kinds of states) and *initialstate*. This is based on the UML meta-model for state machines [15]. The right-hand side of the figure shows the specification of the *canvas* (i.e. the diagram), the *transition symbol* and the *initial state symbol* from the graphical representation definition. The mapping part of the figure shows how these two parts are related. The mapping model provides five different kinds of mappings: *canvas mapping*, *node mapping*, *link mapping*, *label mapping* and *compartment mapping*. The first three are used in the figure. The *canvas mapping* relates the top-level elements in the structure and the representation. Mappings of type *link mapping* are used to relate a graphical object of type connection with its corresponding structure element. The *transition mapping* is an example of a link mapping. In addition to specify which elements from the representation and structure to relate, it is also necessary to include which structure element the transition symbol is allowed to connect. The containment property *transitions* indicates how the structure element (*Transition*) is related to the top-level element.

Also GMF provides the necessary specification features to cover the requirements in a perfect way. Figure 5 shows the running state machine editor. This editor can also handle situations like the one described in section 2 and figure 2 in a good way.

5 Evaluation

This paper presents the essence of a larger study which has compared and contrasted the two main specification approaches for graphical languages. The two approaches, hypergraph grammar and meta-modelling have both been applied to a common example (state machines, language requirements given in section 1) by using automatic editor generators based on the given approaches: DiaGen for the hypergraph grammar and GMF for the meta-model-based approach. This evaluation will summarise the observations found in the previous sections, and by this answer the questions outlined in section 1.

How does the approach and the related tool satisfy the requirements for state machines?

- Both approaches and tools are able to satisfy the given requirements in a perfect way.

Which techniques are used to specify the spatial relationships between the graphical objects in the language?

- The hypergraph grammar approach gives the possibility to specify legal attachment areas for graphical objects and spatial relationships between them, explicitly. The attachment areas are represented as nodes for the hyperedges (graphical objects) in the hypergraph model and are as such perfect for the purpose. The spatial relationships are special links between the attachment areas. There do not exist any pre-defined set of attachment areas and spatial relationships in the tool, thus both the concepts need to be specified in detail by java code for each specific object and relationship. This process is both a bit cumbersome and also on a lower level than desired. An advantage is of course that it gives possibility to specify all kinds of spatial relationships in detail, and it makes the approach very expressive.
- The meta-model-based approach does not give any possibilities to specify special attachment areas on the graphical objects explicitly. The graphical objects of type node has implicit attachment areas at their border, the connections have one implicit attachment area in each end. The only spatial relationship that is possible to specify is a containment relationship. These limited possibilities for specification of attachment areas and spatial relationships are the main weakness with the meta-model-based approach, and makes it difficult, and even also impossible, to specify more complex language descriptions. It seems like the deficiency of features for specifying these graphical aspects on a high level of abstraction is not only a GMF weakness, but a problem for meta-model-based approaches in general.

What is the relationship between the language structure and the graphical representation?

- The hypergraph grammar approach uses two different kinds of internal models: the hypergraph model (HGM) and the reduced hypergraph model (rHGM). The HGM is a representation of the diagram and the graphical information. Because of its complexity it is reduced to a rHGM which describes the structure. In a hypergraph grammar approach, the graphical representation is the central aspect, and by reducing the HGM, i.e. removing the graphical information, we will have the structure model. The reduction rules, which act as the relationship between the two syntactic aspects, are artificial which implies that this relationship is a forced relationship. Referring to the reduction rules as artificial means that they are made with a view on the grammar approach, and not necessarily with a view on the language itself. For some languages, this reduction relationship will affect how the language is specified in the tool, and could lead to an incorrect syntactic specification. It is also worth mentioning that writing the reduction rules requires great knowledge on hypergraphs and also graph transformation, and this part is probably the most challenging part of the language specification.

- The meta-model-based approach deals best with the idea of complete separation between the language aspects as it makes it possible to define the graphical representation independent of the structure model. GMF provides a mapping model that is responsible for bringing together the concepts from the graphical representation with the related concepts from the structure. With this, GMF has a mapping approach that handles the separation issue in a very good way. Because of this separation, a GMF-based editor can handle situations with several occurrences of the same graphical object as presented in section 2.

What is the central language aspect in the specification approach? How does this impact the language specification?

- In the hypergraph grammar approach the representation is the central aspect in the specification. This gives a more concrete specification that is very expressive and able to handle very complex graphical languages, but it also means that the specification is more low-level than desired.

- In the meta-model-based approach the structure is the central aspect in the specification. This gives the possibility to specify the languages on a very high level of abstraction. As a disadvantage, the approach is still not very expressive and there are limitations regarding what kind of graphical features it is possible to specify.

Finally, which approach and related tool is best? When it comes to the quality aspect, there is no doubt that hypergraphs are most expressive and gives the best options related to what kind of diagram objects and spatial relations it is possible to specify. Even if the meta-model-based approach handled the requirements in a perfect way, this approach still has limitations according to languages that are not graph-like. When it comes to the conceptual aspects on the other hand, the meta-model-based approach in general gives a better opportunity to specify the language in a correct way by providing a clean separation between the syntactic aspects.

When we compare the amount of information that is necessary in the specification to get a running editor, we can see a clear difference between the two approaches. While the meta-model-based approach gives a rather simple and straightforward specification, the hypergraph grammar requires more information. The large amount of information necessary in the hypergraph approach is mainly related to the fact that all the spatial relationships needs to be explicitly specified for each particular relation. Also the reduction rules has an impact, and they are often more complex than the simple mapping relations provided in the meta-model-based approach. It is clear that the amount of information increases the expressiveness of the hypergraph approach, but in the same time it decreases the level of abstraction compared to the meta-model-based approach.

GMF is a promising tool, but unfortunately gives the impression of ad-hoc descriptions, and there is still some work left until it has the same expression possibilities as traditional grammar-based approaches as DiaGen. The ideal solution will be to have an approach that takes the best parts from both of the approaches and provide a way of specifying graphical languages which is both expressive and on a high level of abstraction.

References

[1] *EMF*. Available at
`http://www.eclipse.org/modeling/emf`.

[2] *GEF*. Available at
`http://www.eclipse.org/gef`.

[3] *GMF*. Available at
`http://www.eclipse.org/gmf`.

[4] Noam Chomsky. *Aspects of the Theory of Syntax*. Cambridge: The MIT Press, 1965.

[5] Tony Clark, Paul Sammut, and James Willans. *Applied Metamodeling – A Foundation for Language Driven Development*. Ceteva, 2008.

[6] Gennaro Costagliola, Andrea De Lucia, Sergio Orefice, and Giuseppe Polese. A classification framework to support the design of visual languages. *J. Vis. Lang. Comput.*, 13(6):573–600, 2002.

[7] F. Drewes, H.-J. Kreowski, and A. Habel. Hyperedge replacement graph grammars. pages 95–162, 1997.

[8] Bernd Meyer Kim Marriot and Kent B. Wittenburg. *A Survey of Visual Language Specification and Recognition. In Kim Marriot and Bernd Meyer, editors, Visual Language Theory, 1998*, pages 245–259. Springer-Verlag New York, Inc, 1998.

[9] Akos Ledeczi, Daniel Balasubramanian, and Zoltn Molnr. An introduction to the generic modeling environment. In *Proceedings of MDD-TIF07, Model-Driven Development Tool Implementers Forum*, 2007. Available at
`http://www.dsmforum.org/events/MDD-TIF07/GME.2.pdf`.

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

[11] Mark Minas. Syntax definition with graphs. *Electronical Notes in Theoretical Computer Science*, (1):19–40, February 2006.

[12] Jan Pettersen Nytun, Andreas Prinz, and Merete Skjelten Tveit. Automatic generation of modelling tools. In Arend Rensink and Jos Warmer, editors, *ECMDA-FA*, volume 4066 of *Lecture Notes in Computer Science*, pages 268–283. Springer, 2006.

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

[14] OMG. *Model Driven Architecture Guide, Version 1.0.1*. Object Management Group, June 2003. omg/03-06-01.

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

[16] Jan Rekers and Andy Schürr. Defining and parsing visual languages with layered graph grammars. *Journal of Visual Language and Computing*, 8(1):27–55, 1997.