A typed attributed Graph Grammar with Inheritance for the Abstract Syntax of UML Class and Sequence Diagrams

Frank Hermann [1,2] Hartmut Ehrig [3] Gabriele Taentzer [4]

Research Group TFS, Faculty IV
Technical University of Berlin
Berlin, Germany

Abstract

According to the UML Standard 2.0 class and sequence diagrams are defined in a descriptive way by a MOF meta-model and semi-formal constraints. This paper presents a formal and constructive definition of the abstract syntax of UML class and sequence diagrams based on the well-defined theory of typed attributed graph transformation with inheritance and application conditions. The generated language covers all important features of these parts of UML diagrams and is shown to satisfy all of the corresponding constraints by construction. An explicit model transformation demonstrates the close correspondence between the graph grammar and the MOF definition of UML class and sequence diagrams. The graph grammar is validated by well-established benchmarks showing that all important features of the MOF definition of UML are covered.

This formal constructive syntax definition of UML class and sequence diagrams is the basis for syntax directed editing, formal analysis, formal operational and denotational semantics and correctness of model transformations.

Key words: graph transformation, typed, attributed, inheritance, UML, sequence diagrams, class diagrams, abstract syntax

1 Introduction

Meta-modeling of visual languages, particularly the UML [10] defined by MOF [9], facilitates the definition of general structure elements and relations on

[1] Email: frank(at)@cs.tu-berlin.de
[2] Supported by the German Research Society (DFG)
[3] Email: ehrig@cs.tu-berlin.de
[4] Email: gabi@cs.tu-berlin.de

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

the one hand and the implementation of specific properties by constraints restricting the amount of valid instances on the other hand.

Due to the non-constructive nature of the MOF approach, i.e. there is no systematic method to generate all language elements, there exist well-known limitations, which are avoided by a constructive approach. Using typed attributed graph transformation with node type inheritance and application conditions as in [6] and [7] for defining a visual language allows the construction of elements of the language by applying rules of the corresponding graph grammar. The concept of inheritance allows creating an abstract rule, which defines an equivalent set of concrete rules, and therefore notably reduces the total amount of rules. The graph grammar GG_{CSD} for class and sequence diagrams, defined in [12], additionally uses a simple version of transformation units in the sense of [16] allowing to specify the construction of complex elements. This constructive definition shall not replace the original one, but build up a formal basis for certain applications.

Proving the correctness of GG_{CSD} relating the original specification of UML is not possible, because most of the constraints of the original definition of sequence diagrams are only informal. In contrast the formal definition eliminates some problems in the original definition (see 3.4). The explicit model transformation in Subsection 3.2 demonstrates the close correspondence to UML defined with MOF. Restrictions by multiplicities and constraints are already followed and argued at the corresponding rules.

A related approach for defining visual languages constructively is realized in [17] via an EBNF grammar. The application to UML is shortly sketched but not executed till now to our knowledge. In contrast to our visual specification this textual form includes many similarities to Java code as even the authors mention (p. 140). Previous applications of graph transformation describing the abstract syntax of UML diagrams used very simplified and restricted versions of the diagram types. The correspondence between the meta model for class diagrams and an implicit type graph is sketched in [15], but does not take advantage of a graph grammar to create the example diagrams needed for the described transformation. GG_{CSD} supports all important features of the current UML specification for class and sequence diagrams. Moreover an extension of the graph grammar to state machines was finalized in December 2005.

While UML class diagrams are widely known, the current version of UML sequence diagrams, which are special UML interactions and correspond to Life Sequence Charts as in [3], contain new and revolutionary features. Combined fragments as in Figure 1 offer the possibility to use control structures for managing the message flow in a sequence diagram. This leads to a compact notation for complex behaviors. The shown example specifies that a student is assigned to a class, if all previous costs were paid by him. Therefore the two scenarios of having a balanced account or having an unbalanced one are covered in one diagram by using the operator “opt” with its condition. Ad-

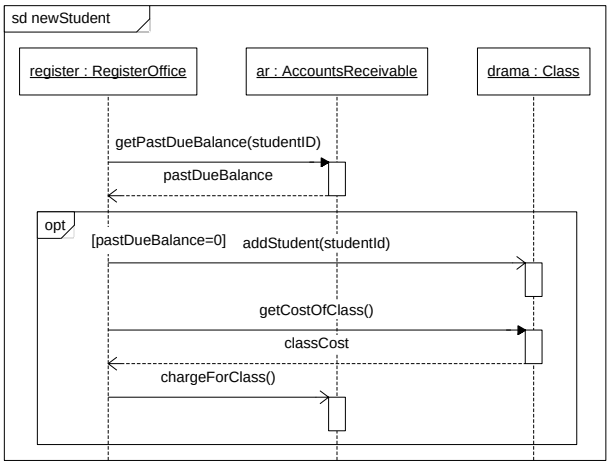

Fig. 1. Example of an UML Sequence Diagram (in [2] p. 9 fig. 9)

ditionally a variety of other operators together with multiple operands are available offering for example to specify parallel or alternative operands. A further new feature is the reuse of existing sequence diagrams in other sequence diagrams. Messages may cross the border of a used sequence diagram and lead into the using one. More details on sequence diagrams can be found in Chapter 14 of [10].

Implemented features of the grammar are validated by benchmarks as described in Subsection 3.3. Example diagrams in concrete syntax, originating from the IBM Rational Library [2] as shown in Figure 1, were recreated by applying the necessary rules leading to a graph representing the abstract syntax of the diagrams.

In a further step transformations into semantic domains shall be possible including operational and denotational semantics. These semantic representations may allow detecting internal and viewpoint conflicts as well as simulating the modeled system. Alternatively to sequence diagrams a specification by message sequence diagrams (MSCs) describes sequences of messages between objects. A formal semantics for MSCs was defined by Petri nets in [13] and allows simulation as well as analysis. Simulation and analysis of the graph grammar GG_{CSD} is possible using AGG (URL: http://tfs.cs.tu-berlin.de/agg/), a development environment for graph transformation systems, where transformation units can be simulated by using the command line input.

2 Graph Grammar for Class and Sequence Diagrams

The graph grammar GG_{CSD} for class and sequence diagrams generates instances of the corresponding parts of UML. It is defined by typed attributed graphs in the sense of [7], which integrate the graph structure and the attributes, which are elements of an algebra. Graph morphisms deliver the basis for typing and the definition of rules and transformations. All graphs of a language are typed over a given type graph via a type morphism. Rules

$(r : L \leftarrow K \rightarrow R)$ are specified using the double pushout approach, where L defines the pattern, that shall be found in a graph, K shows all remaining elements after deleting some elements of L, and finally R contains all preserved plus added elements. Application conditions in positive, negative, and general form restrict the application of a rule to graphs, which either have to contain a demanded pattern or are not allowed to. A rule is applicable, if the match from L to the graph G fulfills the gluing condition and all application conditions. The type graph includes an inheritance graph, which defines all generalization relations between the node types. This leads to a more compact definition of rules, as one abstract rule specifies a set of corresponding rules for all specialized node types. A language *Lang* is then defined by a type graph TG with inheritance, a start graph $S \in Lang$, and a set of abstract rules. Its elements are generated by applying rules to S and the relationship between graph grammar languages with abstract rules and inheritance on the one hand and with concrete rules on the other hand is used in the sense of [1]. Using transformation units [16] for creating complex language elements by a graph grammar is defined as controlled graph grammar in [12] and replaces the set of rules by a set of transformation units and the start graph by a set of start graphs.

2.1 Class Diagrams

The general structure of class and sequence diagrams is defined by the type graph TG_{CSD}. Figure 2 shows the important parts of it for class diagrams containing classes, their features, associations and inheritance relations. The gray marked node ConnectableElement connects this type graph component with the main part for sequence diagrams in Figure 5. A simple version of

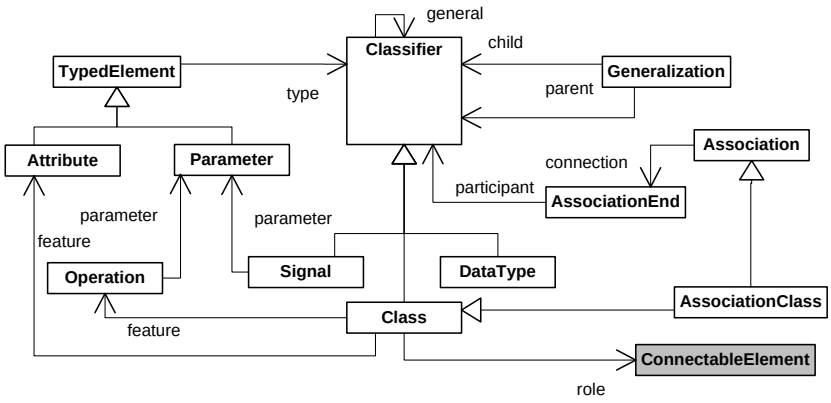

Fig. 2. Part of the type graph TG_{CSD}: main elements of class diagrams

transformation units of [16] combines different rules and imported units with the control structures ";" for sequential application and "!" to demand, that a rule or unit has to be applied as long as possible. For example the simple transformation unit "InsertGeneralization()" specifies, that a class transmits its features to another class and is shown in Figure 3. It imports the rules

"Generalization()" and "General()". After creating a new generalization the second rule is applied as long as possible to achieve again a transitive closed structure.

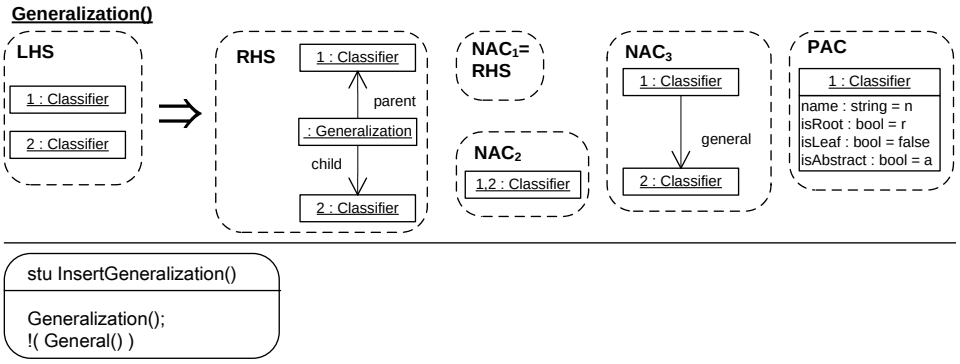

Fig. 3. Simple transformation unit for inserting a generalization

The node type Generalization connects a parent node with its child and is created via the rule "Generalization()" in Figure 3. As the inheritance relation shall be acyclic, a generalization relation in the opposite direction is strictly prohibited by the negative application condition NAC_3, where application conditions are used in the sense of [5]. The positive application condition PAC ensures, that the super class is not a leaf - a class, which is not allowed to transmit to further classes. Prevention of a double defined connection or a generalization link from one class to itself is handled by the other conditions NAC_1 and NAC_2. As Classifier is a generalization of Class, Datatype, and Signal this abstract rule implies nine concrete rules for each combination of the specializations.

Edges of type general supply the transitive generalization relation of all inheritance connections. These edges are created via the rule "General()" in Figure 4, where the positive application condition PAC_2 is used for inserting transitive links. Parallel edges are prevented by NAC and the condition PC allows to generate an edge because of a direct connection or a transitive one.

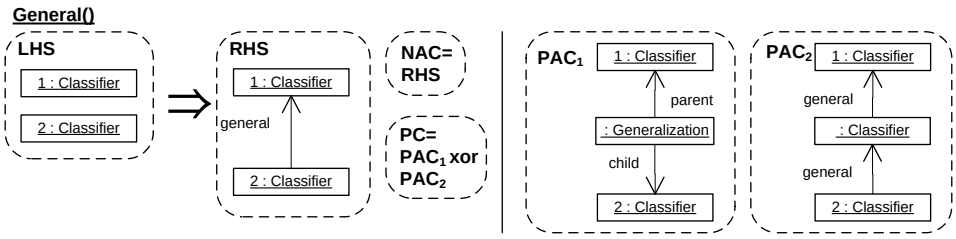

Fig. 4. Rule for creating transitive generalization relations

The simple transformation unit "InsertGeneralization()" in Figure 3 combines the two rules and allows multiple inheritance without cycles. The acyclic structure is demanded by the following constraint for Classifiers in the UML specification. It is mentioned exemplary to show how we argue that our graph grammar generates well-formed instances only.

```
[2] Generalization hierarchies must be directed and acyclic.
A classifier cannot be both a transitively general and transitively
specific classifier of the same classifier.
not self.allParents()->includes(self)
```

2.2 Sequence Diagrams

The main part of the type graph for sequence diagrams is shown in Figure 5, where arrows with closed arrow heads define inheritance realtions. Interacting objects are specified as ConnectableElements, which are already contained in the previous shown type graph component for class diagrams in Figure 2, and represents a role of a Classifier. A Lifeline is connected to anchor points of type OccurrenceSpecification on which elements like Messages can be attached. CombinedFragments are container structures to define control structures, like alternatives, loops, and parallel regions. Their content is structured in InteractionOperands, whose choice may be restricted by Constraints.

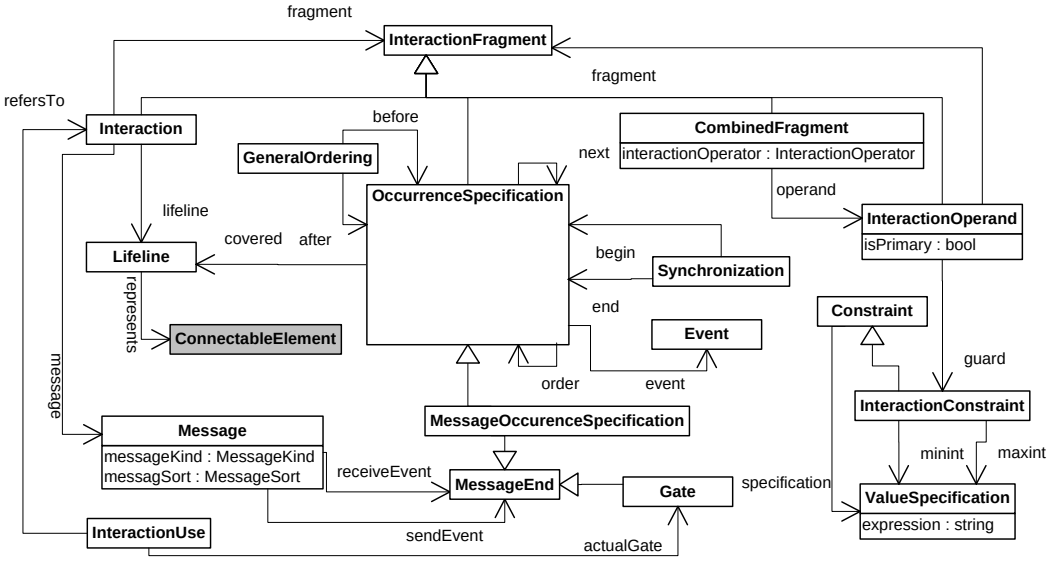

Fig. 5. Part of the type graph TG_{CSD}: main elements of sequence diagrams

Messages of sequence diagrams may be sent synchronously implying that the sender is not allowed to send other messages before receiving a reply. But the UML specification for interactions does not define a relation between these two message types. For this reason the language L_{CSD} additionally includes the node type Synchronization, which marks the beginning and the end of a synchronized interval. InteracitionUses allow to reuse existing sequence diagrams.

GG_{CSD} is fully presented by the two components GG_{CD} and GG_{SD} in [12] and contains more than 70 rules. Figure 6 shows a simple rule, which creates a Lifeline for an object and connects it to the ConnectableElement specifying the role this object executes in the interaction. Additionally it is linked to the enclosing interaction and a first anchor point is inserted. The negative

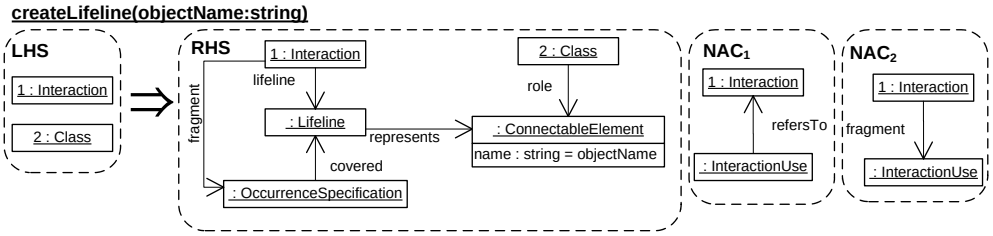

Fig. 6. Rule for creating a lifeline

application conditions NAC_1 and NAC_2 prevent an application of the rule, if the interaction is connected to an interaction use. They ensure, that hierarchical structured interactions remain consistent. This restriction in the order of the editing steps could be eliminated by a transformation unit including more complex rules.

3 Validation of the Graph Grammar

3.1 Testifying Multiplicity, OCL and General Constraints

Multiplicity constraints are respected by the rules of the graph grammar, which is argued at each relevant part of the UML meta-model in Chapter 3 of [12]. The implementation of the multiplicities into the rules is handled mainly by application conditions and well-formedness rules are also argued to be valid, independently of their formulation by natural language only or OCL.

3.2 Model Transformation: $L_{CSD} \rightarrow UML$

The abstract syntax of class and sequence diagrams is defined by GG_{CSD} and strongly corresponds to the definition of UML. As the rules of the graph grammar follow the UML well-formdeness rules, which was described before, the model transformation from each element of L_{CSD} to the corresponding diagram in UML syntax is simple and short. Some additional elements are deleted and bidirectional edges, which are redundant in their grade of information, are added and it is shown, that the transformation terminates and is confluent. The validation of the existing OCL constraints by a formal transformation and check will be available in the future.

3.3 Validation by Benchmarks

To show the coverage of UML features by GG_{CSD} common examples have been selected and its abstract syntax was generated by the grammar. The examples mainly belong to a paper of the IBM Rational Library [2] and are therefore independent benchmarks. They are concretely presented in Chapter 7 of [12] including the shown example in Figure 1 and the sequence of applied rules leading to the instance is given for every diagram. Covered features are for instance InteractionUses to reuse existing sequence diagrams an concurrent

ExecutionSpecifications for defining that an object calls a method which calls a subroutine. Scenarios with parallel or alternatively occuring fragments are other examples.

3.4 Eliminated Problems

The UML specification contains some inconsistencies and mistaken definitions. For example the following constraint occurring on page 476 in [10] is equivalent to true:

```
[2] The selector for a Lifeline must only be specified if the referenced
Part is multivalued.
(self.selector->isEmpty() implies not self.represents.isMultivalued()) or
(not self.selector->isEmpty() implies self.represents.isMultivalued())
```

Instead of the junction "or" it should contain "and". Furthermore the specification of arguments for messages and interaction uses in the meta model is inconsistent. On the one hand a "ValueSpecification" is possible, on the other hand an "Action". GG_{CSD} defines typed Elements as possible argument for both, including the specializations: "ValueSpecification", "Parameter", and "Attribute". A last example is the gap of information for the relation of a synchronous message and its reply mentioned in Subsection 2.2. All detected problems are solved in the graph grammar. Besides changing the text of OCL constraints also some connections and nodes in the meta-model had to be rearranged or inserted to cover the information of a diagram correctly.

4 Future Work and Conclusion

The abstract syntax of a visual model specifies all its semantic relevant properties in a very granular structured way leaving out all layout information. L_{CSD} with its non-descriptive but constructive definition GG_{CSD} offers possibilities to generate well defined specifications of UML in abstract syntax, which can be used directly in the following ways.

4.1 Model Transformation

The generated graphs by GG_{CSD} provide a formal basis to define transformations from L_{CSD} to some target language L_2 using graph transformations as described in [4]. As the source elements were created constructively no constraints have to be checked to ensure the syntactic correctness. Therefore the grammar can also be used for automatic generation of test cases used for model transformations from sequence diagrams.

4.2 Semantics, Simulation, and Animation

A formal semantics of L_{CSD} is planned to be applied, for example using Object-Oriented Transformation Systems (OOTS), where OOTS are an object-oriented variant of transformation systems of [11,14]. Simulation of a specification can

be realized by a transformation to an operational semantics, which also allows animation. All or a selection of possible sequences, defined by sequence diagrams, can be tested to show on the one hand the behavior of the modeled component and on the other hand liveliness, safety, and security properties.

4.3 Editor

In a next step the grammar shall be extended to deliver enough editing rules to automatically generate a syntax directed editor. The TIGER project [8] develops an Eclipse plug-in, which allows defining a graph grammar, connecting the abstract syntax with concrete layout information and generating a syntax directed editor for the language as new Eclipse plug-in. This editor can be used for modeling in the common concrete syntax but generating automatically the precise structured abstract syntax.

References

[1] R. Bardohl, H. Ehrig, J. de Lara, and G. Taentzer. Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation. In M. Wermelinger and T. Margaria-Steffens, editors, *Proc. Fundamental Aspects of Software Engineering 2004*, volume 2984. Springer LNCS, 2004.

[2] Donald Bell. "UML's Sequence Diagram". URL: http://www-128.ibm.com/developerworks/rational/library/3101.html, 4(th) August 2005.

[3] Werner Damm and David Harel. *"LSCs: Breathing Life into Message Sequence Charts"*. Number 19(1):45-80 in Formal Methods in System Design. 2001.

[4] H. Ehrig, K. Ehrig, J. de Lara, G. Taentzer, D. Varró, and S. Varró-Gyapay. Termination criteria for model transformation. In M. Wermelinger and T. Margaria-Steffen, editors, *Proc. Fundamental Approaches to Software Engineering (FASE)*, volume 2984 of *Lecture Notes in Computer Science*, pages 214–228. Springer Verlag, 2005.

[5] H. Ehrig, K. Ehrig, A. Habel, and K.-H. Pennemann. Constraints and application conditions: From graphs to high-level structures. In F. Parisi-Presicce, P. Bottoni, and G. Engels, editors, *Proc. 2nd Int. Conference on Graph Transformation (ICGT'04)*, LNCS 3256, pages 287–303, Rome, Italy, October 2004. Springer.

[6] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in TCS. Springer, 2006. to appear.

[7] H. Ehrig, U. Prange, and G. Taentzer. Fundamental theory for typed attributed graph transformation. In F. Parisi-Presicce, P. Bottoni, and G. Engels, editors, *Proc. 2nd Int. Conference on Graph Transformation (ICGT'04), Rome, Italy*. LNCS 3256, Springer, 2004.

[8] K. Ehrig, C. Ermel, S. Hänsgen, and G. Taentzer. Generation of visual editors as eclipse plug-ins. In *Proc. 20th IEEE/ACM International Conference on Automated Software Engineering*, IEEE Computer Society, Long Beach, California, USA, 2005.

[9] Object Managment Group et al. *Meta Object Facility (MOF) 2.0 Core Specification, available specification (ptc/04-10-15).* Object Managment Group, August 2005. URL: http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-15.pdf.

[10] Object Managment Group et al. *"Unified Modeling Language: Superstructure version 2.0, Specification (formal/05-07-04)".* Object Managment Group, August 2005. URL: http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf.

[11] M. Grosse-Rhode. Semantic Integration of Heterogeneous Software Specifications. In W. Brauer, G. Rozenberg, and A. Salomaa, editors, *Monographs in Theoretical Computer Science, An EATCS Series*, 2003.

[12] Frank Hermann. "Typed Attributed Graph Grammar for Syntax Directed Editing of UML Sequence Diagrams". diploma thesis, Technical University of Berlin, Department for Computer Science, 2005. URL: http://tfs.cs.tu-berlin.de/Diplomarbeiten/TFSdipl/05-F-Hermann.pdf.

[13] Olaf Kluge. *Compositional Semantics for Message Sequence Charts based on Petri Nets.* PhD dissertation, Technical University of Berlin, Department of Electrical Engineering and Computer Science, May 2002.

[14] Andreas Kniep. "Object-Oriented Transformation Systems". diploma thesis, Technical University of Berlin, Department for Computer Science, 2005.

[15] Oliver Köth and Mark Minas. Abstraction in Graph-Transformation Based Diagram Editors. In *Graph Transformation and Visual Modeling Techniques - GT-VMT 2001*, volume 50 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2001.

[16] Sabine Kuske. *Transformation Units - A Structuring Principle for Graph Transformation Systems.* PhD dissertation, University of Bremen, Department of Mathematics and Computer Science, February 2000.

[17] Yong Xia. *A Language Definition Method for Visual Specification Languages.* PhD dissertation, University of Zürich, Department of Economics, January 2005. URL: http://www.ifi.unizh.ch/ifiadmin/staff/rofrei/Dissertationen/Jahr_2005/thesis_xia.pdf.