A Combined Graph Schema and Graph Grammar Approach to Consistency in Distributed Modeling

Stefan Gruner
Institut für Kommunikations- und Softwaretechnik
Technische Universität Berlin
stefan@cs.tu-berlin.de

Abstract. In this overview-paper, a specification method based on coupled graph grammars is sketched that is able to uniformly describe legal domain configurations which distributed modeling as well as re-engineering tasks are based on. The specification method supports the development of proper (re)design tools, and the method is tool-supported itself.

1 Introduction

In this section, two well-known approaches to *distributed modeling* and to *re-engineering of legacy systems* are roughly sketched as a reminder for the reader. This is assumed as helpful for understanding some common characteristics of both which can be uniformly represented in an extended graph-grammatical framework — which is the topic of this paper. (Following a "deductive path", the argumentation of this paper proceeds from some general experiences to the specific contribution.)

1.1 Re-Engineering

In her contributions [1][2][3], Cremer describes how legacy COBOL systems can be transduced into an object-oriented architecture serving as a basis for re-implementation with a modern programming language, or even for distribution via the CORBA: In a first step of analysis, Cremer employs the meta language TXL for a selective parsing of the relevant structures of the legacy system. Then she exports the output of the TXL procedure into the specification environment PROGRES that is based on a set-oriented algorithmic graph transformation approach. In the PROGRES environment, a homomorphic representation of the legacy system is restructured according to the paradigm of graph grammar engineering described in [21].

In their contributions [13][14], Jahnke, Schäfer, and Zündorf describe the transformation of relational database systems into object-oriented ones. Both their approaches are based on graph grammar engineering, too: In [13], both the structures of the relational database system and the object-oriented database systems are modeled as graphs. Then, a non-deterministic graph transformation system specifies the domain transition according to the principles of [21]. As the

M. Nagl, A. Schürr, M. Münch (Eds.): Proc. Intl. Workshop AGTIVE 99
LNCS 1779, pp. 247 - 254.

tables of the relational database systems may be translated into structures of the object-oriented system in many different fashions, the re-engineering tool offers many alternative transition operations of which the re-engineer has to choose. In [14], the procedure has been further evolved.

1.2 Distributed Modeling

The purpose of distributed modeling is to reduce the design complexity by work-sharing among several designers. The VIEWPOINTS approach of Finkelstein *et al.* [6][7][8][9] is especially interesting because of its possibility of delaying consistency checks (and inconsistency repair) until explicit demand. Goedicke, Taentzer *et al.* have shown that VIEWPOINTS' original specification based on first-order logic can easily be substituted by a specification based on graph transformation [5][10].

1.3 Generalization

Both in re-engineering and in distributed modeling one is concerned with the mutual consistency of two or more *domains*. From this point of view, the main differences between those software engineering tasks are intentional differences reflected by the dimension of time t: In re-engineering, some system A (maybe a structure, a program, a document, etc.) is living in domain D_A from t_{-x} until t_0 while some corresponding system B is living in domain D_B (whereby not necess. $D_A \neq D_B$) from t_{-y} until t_z such that A and B are consistent with each other at t_0 (with $t_{-x} < t_{-y} < t_0 < t_z$). In distributed modeling, all involved systems $A_1, \ldots, A_n$ evolving in their domains $D_{A_1}, \ldots, D_{A_n}$ from $t_\perp$ to $t_\top$ need to be consistent (or, in weaker approaches: partially consistent) at certain times $t_\perp < \ldots < t_x < t_z < \ldots < t_\top$. Thus, by abstraction from those details, one may look at models or domains as graphs, and, consequently, one may look at the task of consistency management in re-engineering or in distributed modeling as graph transformation. Legal domain *configurations* can be described as sentences (or sentence forms) generated by special kinds of coupled graph grammars, as further explained in the following sections.

2 Graph Grammar Specifications

The following section sketches how different specification *intentions* may lead to different classes of specifications. Then, some related work is briefly discussed.

2.1 Tool oriented View — Domain oriented View

In many graph-grammatical approaches to the solution of practical software-engineering problems, a directly *tool-oriented* position is taken. In tool-oriented specification approaches, the employed graph grammars are viewed as *transformative* entities that describe the operations of a tool T on its objects *within* an

only implicitly given object-domain D_T. Assumed that an object O is a member of D_T, the tool-oriented specification ensures that a T-transformed object O' (thus: $O \rightarrow_T O'$) is a member of D_T as well. On the other hand, in a *domain-oriented* specification approach the transformative behavior of a Tool T is not regarded with first priority. Hence, the employed graph grammars are viewed as *generative* entities in such an approach that enumerates the relevant domains *D_T themselves.*[‡] As a consequence it is possible to ask *if* a given object O is a member of D_T.[§]

2.2 Related Work

The already mentioned contributions [1][2][13][14] follow the paradigm of graph grammar engineering presented in a paper by Schürr, Winter, and Zündorf [21]. According to this paradigm, *one* graph schema is used as a kind of hierarchic type declaration for *one* graph grammar (or, more general: one graph transformation system). Therefore, it seems quite difficult to meta-model the distributed modeling (dealing with *many* domains) within the paradigm of [21]. (Consequently, a generalization of this paradigm will be sketched below, wherein *many* graph schemas and *many* graph grammars can be involved.)

A VIEWPOINTS-oriented approach has been taken by the authors of [5] (also mentioned above already). Unlike the approach of [21], they do not employ a sophisticated graph schema for the purpose of typing, but they explicitly mention the different *views* of a distributed modeling environment. In order to represent those within a graph-grammatical formalism, the authors introduce the notion of partial grammars (or sub-grammars) being parts of a whole graph grammar. While the sub-grammars characterize the evolution of the particular domains in distributed modeling, the whole graph grammar represents the possible history of the total modeling environment. (Thus, the question may arise now if it is possible to have *both* the advantages of typing with graph schemas *and* the explicit recognition of more than one domain.)

As already suggested more than twenty years ago by Pratt [18][19], it is possible to glue two or more graph grammars together in a parallel fashion. Doing so, one can describe the construction of consistent configuration of sub-models in a distributed environment quite intuitively. Given n graph grammars with their graph languages $\mathcal{L}_1, \ldots \mathcal{L}_n$ representing the particular sub-model spaces, and given a proper coupling specification $\mathfrak{C}$, the language $\mathcal{L}_{\mathfrak{C}} :\subset \mathcal{L}_1 \times \ldots \times \mathcal{L}_n$ represents the space of all consistent model configurations. (This approach has been followed by several authors [15], and a generalization of it is sketched below.)

[‡] Both approaches can be combined for best specification results.

[§] The membership problem of grammars is undecidable in general, of course, but fortunately there is a large class of useful graph grammars whose membership problem is decidable, as proven by Rekers and Schürr in [20].

3 Coupled Graph Schemas — Coupled Graph Grammars

In [11] it is described how several *software development environments* (which are comprehensively explained in [15]) have been specified with *coupled graph grammars* according to the ideas of [18][19]. The aim of this section is to argue that the useful *design technique* of graph grammar coupling can be further improved by applying a corresponding design technique of *coupled graph schemas*. After the concepts are sketched in general, a small example is given below.

3.1 Concepts

In his dissertation [12] (embedded in the research context of [17]), the author describes how coupled graph grammars can be used to specify the *integration* of different yet mutually dependent sub-models in a CAD environment for chemical engineering. Thereby, the coupling of graph grammars is guided by the coupling of *graph schemas* representing some structural constraints on the visual design language generated by the coupled graph grammars. Formally defined in [12], the specification approach sketched in this section generalizes the graph grammar engineering approach of [21] in two directions: First, the concept of hierarchic typing and super-typing of graph nodes has been transferred to the edges as well. Second, the relation between one graph schema and one graph grammar has been extended to a relation between n graph schemas and n graph grammars for the sake of inter-domain consistency descriptions.

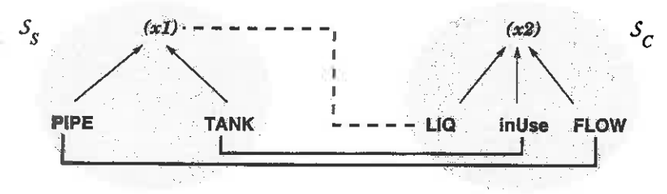

Fig. 1 *coupled graph schemas* ($n = 2$)

In Fig.1, a coupling of two graph schemas $\mathcal{S}_S$ and $\mathcal{S}_C$ is shown. Arrows "$\rightarrow$" denote that some ground types PIPE and TANK are super-typed by some type (x_1) in $\mathcal{S}_S$, whereas some ground-types LIQ, inUse, and FLOW are super-typed by some (x_2) in $\mathcal{S}_C$. The thick solid lines declare that the concepts PIPE and FLOW as well as TANK and inUse correspond to each other. However, any correspondences between $\mathcal{S}_S$ and the LIQ concept are declared as forbidden which is denoted by the broken line between LIQ and the super-type $x1$. (Like in the PROGRES system [21], the ground types can be instantiated while the super types cannot: they only serve as abstract property descriptors.)

In a similar way, the necessary correspondences between the rules of different graph grammars —generating the different domains of some distributed modeling task— can be declared. It is worth mentioning that the type-correctness of such graph grammar couplings can be checked via those axiomatic graph schema correspondences as they are sketched in the figure of above.

3.2 Example

Please imagine an engineering task wherein a system of tanks and pipelines shall be modelled in correspondence —but not immediately together— with some liquid materials flowing through the system. A graph grammar S shall describe the possible structure of such a system whereas a graph grammar C shall describe some properties of the liquid contents of the system.

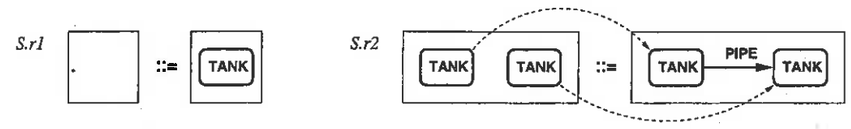

Fig. 2 *rules of graph grammar S*

Fig.2 shows two rules of S. With *S.r1*, new tanks can be added to the system. With *S.r2*, new pipelines can be added accordingly. (Please forget about the several possibilities of graph transformation semantics at this point of discourse.) On the other hand, let's assume that new liquid materials shall be introduced and their applications shall be handled. As depicted in Fig.3, this is done by the rules *C.r1*, *C.r2*, and *C.r3* of graph grammar C.

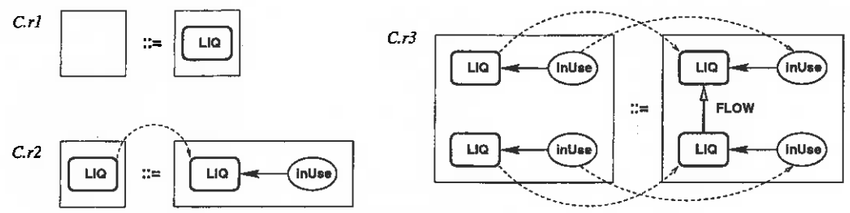

Fig. 3 *rules of graph grammar C*

Supposed now that a coupling $\mathfrak{S} \subset \mathcal{S}_S \times \mathcal{S}_C$ of graph schemas is given (Fig.1) such that {PIPE—FLOW} and {TANK—inUse} are declared as *the only* positive corresponding concepts of this example. Then it obvious that applications of rule *C.r1* must be completely independent from the other model domain described by graph grammar S. Moreover, it can be concluded automatically that no rule

correspondences {*S.r1—C.r3*} or {*S.r2—C.r2*} may be declared. Instead, a *tool* may *suggest* its user to establish the rule couplings {*S.r1—C.r2*} and {*S.r2—C.r3*} in order to characterize the integration of both involved model domains D_S and D_C.

When a rule coupling is declared *as a whole* (by rule names), the necessary *detailed* couplings between the nodes and edges of the involved rule bodies have to be declared accordingly by use of the information contained in $\mathfrak{S}$. Due to lack of space in this overview-paper, the reader must be referred to [11][12] at that point of discourse.

4 Results

The combined specification method of coupled graph schemas and coupled graph grammars is formally sound such that it can be implemented by a specification tool. The tool can support the domain specification tasks in the requirements engineering phase of a distributed project. Given such an integrated domain specification, further tools can be built for support the constructive design *within* the project domains defined by a coupled graph schema and graph grammar specification.

4.1 Tool Support for Integrated Domain Specifications

In [12] a prototype is reported which supports the proper declaration of coupled graph schemas and coupled graph grammars as sketched in the previous section. With that prototype, the user can construct schema correspondences and grammar correspondences in a syntax-directed fashion.* The tool is able to check the consistency of the grammar correspondences with respect to the given schema correspondences. It could be further enhanced by an interactive suggestion-component which provides the user with hints on possible couplings of rules. The coupling of graph schemas and graph grammars is not restricted to two dimensions. Instead, an n-ary approach ($\mathfrak{S} \subset \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$) is supported.

4.2 Tool Support for Distributed Modeling

In the research project reported in [17], the possibilities of tool support for distributed modeling in chemical engineering are studied. Important problems occurring in this field seem to be quite similar to the tasks mentioned in the introductory section above — and are, therefore, open to graph-grammatical solutions [4]. In the context of [17], another small prototype for experimental purposes is described in [12]. That prototype implements a partial graph-grammatical parse-and-generate approach to consistency of certain chemical-engineering domains. With that tool, the user can construct simple *structure views* of chemical

* The software system of the tool re-uses of some source code of the PROGRES environment [21] according to the framework-method reported in [15].

plants, which can be partially translated in corresponding *contents views* afterwards — thus: the functionality of that tool is inspired by certain aspects of the already mentioned VIEWPOINTS paradigm. The parse-and-generate approach results from the coupled specification method and has been transferred into the PROGRES environment [21] from which the prototype tool has been generated. The correspondence information residing in the underlying coupled domain specification is kept by an intermediate *parsing graph* which occurs as the tool proceeds with its integrative operations from one domain the other one.

4.3 Conclusion

- In this overview-paper it has been argued that *coupling* of graph grammars is able to serve as a sound and uniform method for describing and understanding important document-processing tasks like software re-engineering or distributed modeling.
- Provided with the notion of graph schemas, the coupling of graph grammars can be pre-specified by the coupling of graph schemas such that the relative *consistency* of the coupled grammar specification can be checked with respect to the coupled schema pre-specification.
- As the method is formally sound, *tool support* for the construction of coupled graph grammars —which can be applied in various domains and contexts of industrially relevant document-design tasks— is possible.

References

1. K. Cremer, *A Tool supporting the Re-design of Legacy Applications.* P. Nesi, F. Lehner (Eds.), Proc.2nd Euromicro Conf. on Softw. Maintenance & Re-Eng., pp.142-148. IEEE Comp. Soc. Press, 1998
2. K. Cremer, *Graph-based Reverse-Engineering and Re-Engineering Tools.* In [16]
3. K. Cremer, *Graph-basierte Werkzeuge zum Reverse Engineering und Re-Engineering.* Doct.Diss., RWTH Aachen 1999. To be published by Deutscher Universitätsverlag, Wiesbaden
4. K. Cremer, S. Gruner, M. Nagl, *Graph-Transformation-based Integration Tools: Application to Chemical Process Engineering.* H. Ehrig, G. Engels, H.-J. Kreowski, G. Rozenberg (Eds.), Handbook of Graph Grammars and Computing by Graph Transformation, vol.2, pt.IV, chpt.10, pp.369-394 World Scientific, Singapore 1999
5. H. Ehrig, G. Engels, R. Heckel, G. Taentzer, *A View-oriented Approach to System Modeling based on Graph Transformation.* M. Jazayeri, H. Schauer (Eds.), ESEC-FSE'97 Joint 6th Europ. Softw. Eng. Conf. & 5th ACM SIGSOFT Sympos. on the Foundations of Softw. Eng. LNCS 1301, pp.327-343, Springer-Verlag, Berlin 1998
6. S. Easterbrook, A. Finkelstein, J. Kramer, B. Nuseibeh, *Coordinating distributed View-Points: the Anatomy of a Consistency Check.* INTERNAT. JOURN. ON CONCURRENT ENG.: RESEARCH AND APPLIC. 2/3, pp.209-222, 1994
7. A. Finkelstein, B. Nuseibeh, J. Kramer, *A Framework expressing the Relations between multiple Views in Requirements Specifications.* IEEE TRANSACT. ON SOFTW. ENG. 20/10, pp.760-773, 1994

8. A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, B. Nuseibeh, *Inconsistency-handling in Multi-perspective Specifications.* IEEE TRANSACT. ON SOFTW. ENG. 20/8, pp.569-578, 1994
9. A. Finkelstein, G. Spanoudakis, D. Till, *Managing Interference.* Proc. ACM SIGSOFT'96 Workshop, pp.172-174, ACM Press 1996
10. M. Goedicke, B. Enders, T. Meyer, G. Taentzer, *Tool Support for ViewPoint-oriented Software Development: towards Integration of multiple Perspectives by distributed Graph Transformation.* In [16]
11. S. Gruner, M. Nagl, A. Schürr, *Integration Tools supporting Development Processes.* M. Broy, B. Rumpe (Eds.), *RTSE'97 Workshop on Requirements targeting Software and Systems Engineering.* LNCS 1526, pp.235-256, Springer-Verlag, Berlin 1998
12. S. Gruner, *Eine schematische und grammatische Korrespondenzmethode zur Spezifikation konsistent verteilter Datenmodelle.* Doct.-Diss., RWTH Aachen 1999. Published by Shaker-Verlag, Aachen 1999
13. J. Jahnke, W. Schäfer, A. Zündorf, *A Design Environment for migrating relational to Object-oriented Database Systems.* Proc. Internat. Conf. on Softw. Maintenance, IEEE Comp. Soc. Press, pp.163-170, 1996
14. J. Jahnke, A. Zündorf, *Using Graph Grammars for building the VARLET Database Reverse Engineering Environment.* G. Rozenberg, G. Engels (Eds.), TAGT'98 6th Internat. Workshop on Theory & Applic. of Graph Transformation (Paderborn 1998). To appear in the LNCS, Springer-Verlag, Berlin 2000
15. M. Nagl (Ed.), *Building tightly integrated Software Development Environments: the IPSEN Approach.* LNCS 1170, Springer-Verlag, Berlin 1996
16. M. Nagl, A. Schürr (Eds.), *AGTIVE'99: Applications of Graph Transformations with Industrial Relevance.* LNCS this volume, Springer-Verlag, Berlin 2000
17. M. Nagl, B. Westfechtel (Eds.), *Integration von Entwicklungssystemen in Ingenieuranwendungen: Substantielle Verbesserung der Entwicklungsprozesse.* Springer-Verlag, Berlin 1998/99
18. T. Pratt, *Pair grammars, graph languages and string-to-graph translations.* JOURN. OF COMP. & SYSTEM SC. 5, pp.560-595, 1971
19. T. Pratt, *Definition of programming language semantics using grammars for hierarchical graphs.* G. Rozenberg, H. Ehrig, V. Claus (Eds.), Proc. Internat. Workshop on Graph Grammars & their Applic. to Comp. Sc. and Biology. LNCS 73, pp.389-400, Springer-Verlag, Berlin 1979
20. J. Rekers, A. Schürr, *Defining and Parsing Visual Languages with layered Graph Grammars.* JOURN. OF VISUAL LANG. & COMPUTING 8/1, pp.27-55, Academic Press, London 1997
21. A. Schürr, A. Winter, A. Zündorf, *Graph Grammar Engineering with PROGRES.* W. Schäfer, P.Botella (Eds.), ESEC'95 Proc. 5th Europ. Softw. Eng. Conf. LNCS 989, pp.219-234, Springer-Verlag, Berlin 1995

Acknowledgments. Thanks to *M.Große-Rhode*, *T.Pratt*, and the AGTIVE99 referees for helpfull communication. Financial support was given by GETGRATS and the DFG.