# WHY SHOULD CHEMO-ENGINEERS BE INTERESTED IN GRAPH GRAMMARS?

## A SOLUTION TO THE INCONSISTENCY PROBLEM IN DISTRIBUTED MODELING

*Stefan Gruner*
Graduiertenkolleg für Informatik und Technik
Rheinisch-Westfälische Technische Hochschule
52056 Aachen/Aix-la-Chapelle/Aquisgrana, (D)
*gruner@kolleg.informatik.rwth-aachen.de*

ABSTRACT

In distributed modeling, a group of developers are elaborating some specification in a work-sharing manner. Sub specifications and partial models are constructed according to a divide-and-conquer principle. Maintaining the consistency of the evolving sub specifications, generally called documents, is the main meta problem of distributed modeling. The volume of literature on consistency maintenance is vast, especially in the fields of classical software engineering. In this contribution, it is shown how consistent document configurations can be represented by coupled graph schemas and graph grammars. Such models can serve as formal requirements definitions for various kinds of integration tools and consistency management systems. The approach presented here is not at all restricted to software engineering. As our team is busy in a long-term research project on software support for distributed modeling in chemical engineering, the running example of this contribution is taken from this engineering discipline.

KEY WORDS
*consistency maintenance, integration tool, specification, coupling, graph schema, graph grammar, modelling*

## 1 INTRODUCTION AND RELATED WORK

Graphs are not only of mathematical interest by themselves, but they also serve as intuitive modeling means in almost every practical application domain. For example, Petri nets [19][20] and Chen diagrams [4] have been such fundamental break-throughs in their times that many nowadays' graph based modeling approaches like dynamic task nets [13][30] or UML [9] are still related to them. Other examples of modeling graphs are algorithm flow diagrams, module interconnection diagrams [17], functional decomposition diagrams [6], and so on.

Speaking about modeling, it is important to remember that this can be a really industrial activity with very large documents to be constructed and many people involved. To overcome the complexity, the modeling task is usually shared among the developers who are elaborating different —yet not independent— sub specifications. Step by step, a complex document configuration evolves. Keeping this configuration in a consistent state is a crucial precondition of project success. At release time, all relevant sub parts of the specification must properly fit together.

On the other hand, model graphs can be viewed as 'sentences' of special kinds of grammars that have been called 'web grammars' in ancient times and that are now called graph grammars. The principle of poly-dimensionally producing and transforming graphs by graph grammars and graph transformation systems is similar to the production of sentences by mono-dimensional Chomsky grammars [5] and term rewriting systems [2]. Since the first known report on graph grammars [21], a large volume of literature has been published on that subject. Latest state-of-the-art compilations are [24][25] while tutorial introductions can be found, for example, in [1][7]. All of them provide sufficiently expressive lists of survey and specific references.

In distributed modeling, a configuration of mostly graphically denoted specification documents is evolving by the time. Graph grammars and graph transformation systems, on the other hand, describe the evolution of graphs in a formally sound manner. Therefore, it is no surprise that there can be found various approaches to the support of consistent distributed modeling by graph grammatical means and methods; see for example [3][12][18]. As a similar topic one can find graph grammatical schema transformation for consistent database management [14][27].

In [28] the authors describe how to split large graph grammar specification into adequate modules, and they introduce a new module concept to the graph programming language PROGRES which is still a monolithic language in the latest release [29]. This is interesting, because it is the module concept of a specification formalism that makes distributed modeling possible and applicable. Thus: the inconsistency problem —that I claim to be solvable by the means of graph grammars— is now in a strong position to 'strike back' onto this approach. As this problem is closely related to the view update problem of databases, the authors suggest database techniques like active constraints for compensation. My suggestion is not to apply distributed graph techniques for the purpose of solving inconsistency problems emerging from distributed modeling.

The advanced approach of [8] is conceptually similar to the method described in this contribution. There, separate graphic view of a system are maintained by a graph transformation system upon a common reference model. Moreover, a hierarchy of even more abstract meta views is possible. Particularly the graph typing facility of that approach is, however, not very advanced. It could be enforced by an inheritance hierarchy of vertex classes and arc classes as presented later in this paper.

As described in [10], there has been a decade of increasing experience in how to construct consistency maintenance tools employing Pratt's ideas of coupling grammars to each other [22] such that the coupled grammars characterize the space of all legal configurations constituting a whole problem description. In the early beginning, consistency maintaining tools have only been programmed ad hoc. Later one has found out that tool specification considerably improved the subsequent tool implementation. But then it was the specifications to be rather ad hoc and not yet as methodically sound as it could have been. The ongoing research of the author is to find a more concise way of pre-specifying consistency maintenance software tools.

Janning has published a specification approach based on static graph schemas [15] that, however, does not employ any graph transformational means. A dynamic specification approach with so called triple graph grammars is known from Lefering [16]. In fact, his 'triple' grammars are more or less pair grammars, while his transformation approach (I) strongly sticks to the language PROGRES —that possibly vanishes some day— and (II) is restricted to graph productions that may not destroy any vertex.

A sound combination of both the static and dynamic approaches in order to simplify the design of such integrative specifications, as explained by [11] in detail, is the topic of this paper. The new specification method shall be supported by a meta specification tool of which the implementation has already begun. In the following sections it is shown by a toy-size chemo-engineering example how a graph grammar specification of sub model inter-dependencies can be developed. The coherence of the grammar coupling definitions will be checked relative to a inter-submodel correspondence schema that serves as static pre-specification of the consistency maintenance problem.

As our group is busy in a long-term research project on software support for distributed modeling in chemical engineering [26], the running example of this paper is taken from that engineering discipline. (One can find several commercial design tools for particular chemo-modeling tasks, but their cooperation and distribution facilities are still quite insufficient.)

## 2 MOTIVATION BY A SIMPLE EXAMPLE

In Fig.1 one can see a sketch of a bubble tank. Some liquid is pouring into and out of the tank. Bubbles of vapor are pumped into the tank and leave the tank through another valve. Heat radiation is warming up the liquid from outside. The liquid and the vapor bubbles are connected to (and separated of) each other by a tiny film.

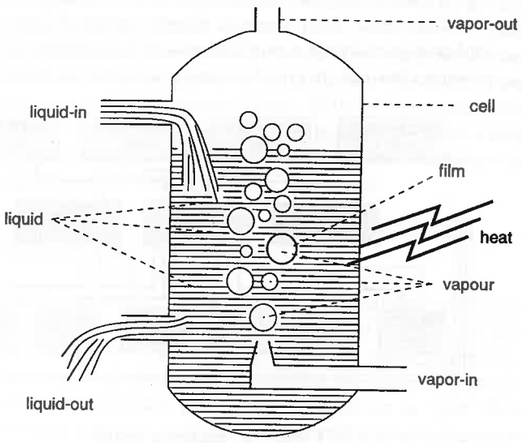


Figure 1: *bubble tank*

Let's further assume that this bubble tank shall be computer-simulated in order to explore its operational properties. For this final purpose (out of scope of this paper), a rough ER-like [4] specification of the bubble tank is constructed at first. This is done in a distributed manner as introduced above. In Fig.2 the structural essentials of the bubble tank are represented by an attributed ER-like graph. This structure graph[1] Has the following meaning: The devices liquid and vapor are the main entity nodes. The relation between them is the connection. All rectangular nodes with rounded corners represent the interfaces by which the devices communicate with their neighbourhood. The composition cell represents the bubble tank as a whole with all its components.

[1] *Fig. given by the Dept. of Proc. Eng., RWTH Aachen.*

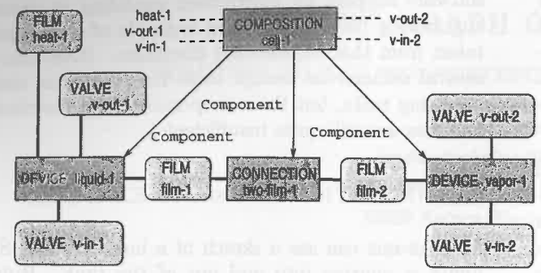


Figure 2: *structure graph*

Another view of the bubble tank is described by the sub model graph[2] in Fig.3, the material graph. This data structure is a cut-out of a (big) sub specification of what's going on with the chemical substances in the bubble tank. The chemo-engineers speak of four single phases containing a mixture of several chemical components that are involved in two homogeneous reactions:

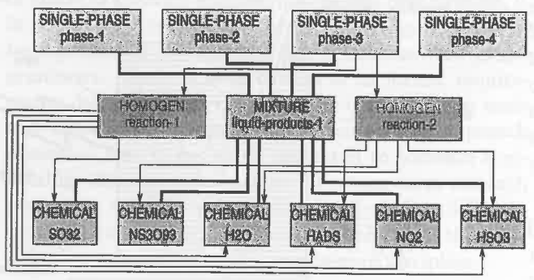


Figure 3: *material graph*

To support the structural consistency of modeling, it is important to know how these two sub model graphs are related to each other, but these relations are not represented in the sub models themselves and must, therefore, be made explicit.

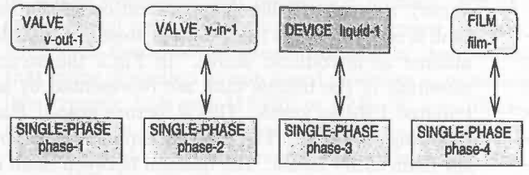


Figure 4: *consistency relations*

Fig.4 shows how certain nodes of the structure graph and the material graph are existentially related to each other. Thus, if we have a structure graph and a material graph for a bubble tank, and if not the liquid device as well as its liquid interfaces from the structure graph have got a partner concept in the material graph, then the whole configuration is inconsistent[3]. At release time, those kinds of inconsistency must not occur in the model.

*Definition: Consistency Relation*
A specification is consistent, if it has at least one solution, otherwise it is inconsistent. A *consistency relation* between different sub specifications is a constraint on the existence and the structure of certain sub specification parts such that the whole specification is inconsistent in case the consistency relation is violated. □

## 3 GRAPH SCHEMAS, GRAPH GRAMMARS

With graph schemas one can describe the gestalt of legal sub model graphs to be constructed. Thus, a graph schema can be interpreted as the type of a set of graphs. As graphs are constructed by graph grammars and as the rules of graph grammars are built of graphs themselves, a graph schema does also matter for the graph grammar constructing the graphs described by their common schema.

In the approach presented here, a graph schema is mainly a lattice over disjoint sets of vertex labels and arc labels. The lattice order $\preceq$ represents property inheritance, also called 'is-a relation' in a less rigorous way of (object-oriented) speaking. Lattice schemas offer the possibility of multiple inheritance modeling and they are, nevertheless, syntactically sound. Moreover, it is fair easy to discover redundancy of inheritance in lattices. The latticestructure is enriched with labeling functions defining the source vertex labels and the destination vertex labels of the arc labels:

*Definition: Graph Schema*
Let $\Sigma := \{\bot, \top\} \uplus S$ with $S \neq \emptyset$ a set of symbols, also called *classes*. A quadruple $\mathfrak{S} := (\Sigma, \preceq, \mathfrak{src}, \mathfrak{dst})$ is called *graph schema* over $\Sigma$, if:
- The partial order $\preceq \subseteq \Sigma \times \Sigma$ defines a lattice with $\bot$ and $\top$ as bottom and top elements.
- $\mathfrak{Fin}(\mathfrak{S}) := \{\sigma \in S | \nexists \tau \in S : \tau \preceq \sigma\}$ is the set of *final classes* of $\mathfrak{S}$, and $\mathfrak{Sup}(\mathfrak{S}) := \Sigma - \mathfrak{Fin}(\mathfrak{S}) - \{\bot, \top\}$ is the set of *super classes* of $\mathfrak{S}$.
- $\mathfrak{Fin}(\mathfrak{S}) = \mathfrak{A} \uplus \mathfrak{V}$, where $\mathfrak{V} \neq \emptyset$ is a set of vertex labels, $\mathfrak{A}$ a set of arc labels, and $\forall \mathfrak{v} \in \mathfrak{V}, \forall \mathfrak{a} \in \mathfrak{A}, \forall \sigma \in \Sigma : (\mathfrak{v} \preceq \sigma \wedge \mathfrak{a} \preceq \sigma) \Longrightarrow (\sigma = \top)$.
- $\mathfrak{src} : \mathfrak{A} \to \mathfrak{V} \cup \mathfrak{Sup}(\mathfrak{V})$ and $\mathfrak{dst} : \mathfrak{A} \to \mathfrak{V} \cup \mathfrak{Sup}(\mathfrak{V})$ are functions that map arc labels to source vertex labels and destination vertex labels, where $\mathfrak{Sup}(\mathfrak{V}) := \mathfrak{Sup}(\mathfrak{S}) - \{\sigma | \mathfrak{a} \preceq \sigma, \mathfrak{a} \in \mathfrak{A}\}$. □

[2] *Fig. given by the Dept. of Proc. Eng., RWTH Aachen.*

[3] *Stated by the Dept. of Proc. Eng., RWTH Aachen.*



Instance graphs may only be labeled with final class symbols, but the left hand sides of the graph grammar rules operating on the instance graphs may also contain super class labels serving as a more abstract description of the rules' application contexts. Please refer to [11] for more information (that is out of scope here).

Fig.5 shows a simple graph schema that describes graphs similar to the one shown in Fig.2 above. There is one edge label '—'. Source and target functions assign the node label Structure Concept to '—' such that also all children of Structure Concept (except of $\bot$) may appear as sources and targets of "—". The labels Structure Concept, Abstract Phase, Interface, Connection and nonterminals Nterm are node labels. They inherit the properties of Structure Concept. There is also a nonterminal label Nterm that is needed later for the graph grammar generating those simple structure graphs. The lines between the labels represent the lattice order $\preceq$.

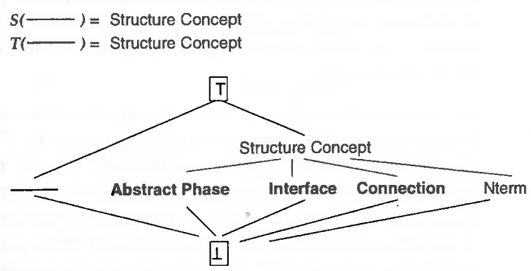


Figure 5: *graph schema for simple structure graphs*

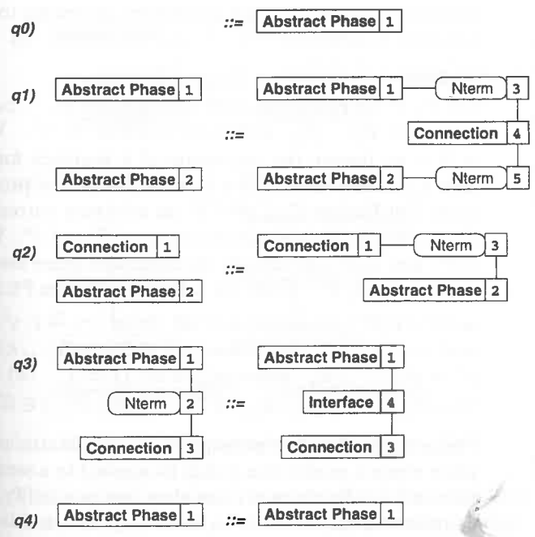


Figure 6: *Graph grammar obeying schema Fig.5*

A graph schema for a set of sub model graphs provided, it is possible to check if a graph grammar generates only those graphs intended by your schema. It only has to be shown that the right hand sides of each replacement rule does not violate the structural constraints required by the graph schema. This constraint is sufficiently fulfilled when the right hand side of the replacement rule belongs to the set of graphs typed by the same schema as the instance graph to be replaced.

In Fig.6 a small graph grammar is shown that generates graphs typed by the schema of above. In the vertices of the replacement rules are represented by numbers 1, 2, 3, 4, 5. If a vertex $i$ occurs both on the left hand side and on the right hand side of a rule $q$, then $i$ belongs to the unchanged context of $q$. All vertices are labeled according to the graph schema above, and it is easy to check that no graph can be generated which violates the given schema constraints. Rule $q_0$ allows to always create Abstract Phase vertices. Rule $q_1$ allows to create an interfaced Connection between two Abstract Phase vertices. Rule $q_2$ allows to plug further Abstract Phase vertices to a Connection vertex. Rule $q_3$ terminates the Nterm vertices to Interface vertices, and rule $q_4$ is a lazy dummy doing nothing. It will soon become obvious why especially the rules $q_3$ and $q_4$ are useful.

## 4 CORRESPONDENCES

Please remember that consistency relations have been stated between the sub model graphs of Fig.2 and Fig.3 in Fig.4. Now the question arises, if those dependencies can be represented employing the according graph schemas and graph grammars. The answer is: 'yes'. But for this purpose, a graph schema and a graph grammar for the other[4] involved sub model must be given, too. They are depicted in Fig.7 and Fig.8 below.

Again, it is possible to show that the graph grammar shown in Fig.8 is proper with respect to the graph schema given in Fig.7 describing chemical components in a very abstract fashion. The rules $p_0$ and $p_1$ allow the creation of Phase System nodes and Chemical nodes. Rule $p_2$ allows to plug both types of nodes together by the edge "—" as stated by the source and destination functions of the graph schema. Rule $p_3$ is, again, a (useful) lazy dummy.

So far, a graph schema and a graph grammar for each type of sub model graphs to be integrated have been constructed. In the following, a correspondence schema and a correspondence grammar describing the legal configurations of consistently related sub model graphs are constructed. Such correspondence specifications are

[4] *In general, the same method may be applied to configurations with more that two involved sub specifications as well.*

useful as a formal basis for the implementation of interactive integration tools that help to bridge the gap between the separated sub models in distributed modeling. Tool support of structural consistency is especially important when the sub specifications are subject to changes during some modeling activities [18].

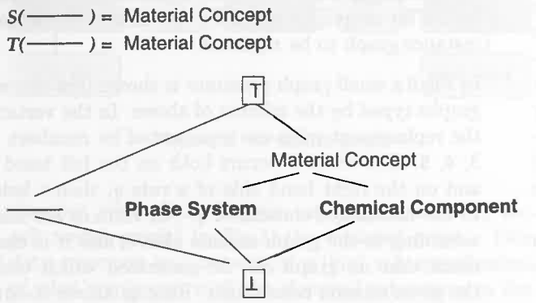


Figure 7: *graph schema for simple material graphs*

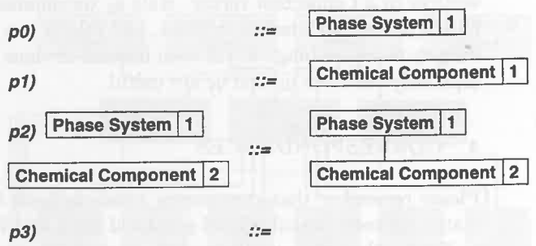


Figure 8: *graph grammar obeying schema Fig.7*

To construct a correspondence schema, you have to ask an expert of the application domain for the relevant consistency relations. In our case, a chemo-engineer will give the following answer: 'Each Abstract Phase and each Interface must find a Phase System in the corresponding sub model'[5]. Accordingly, the schemas of Fig.5 and Fig.7 are combined to the correspondence schema of Fig.9; (irrelevant labels omitted for clarity).

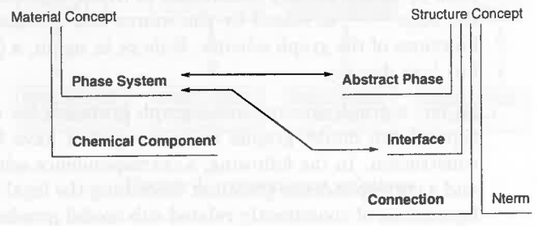


Figure 9: *correspondence schema of sub models*

[5] *Stated by the Dept. of Proc. Eng., RWTH Aachen*

Please do not mix up these schema correspondences with the consistency relations (shown in Fig.4) of their instance graphs.

*Definition: Correspondence Schema*
Let $\mathfrak{S}_1, \ldots \mathfrak{S}_n$ be graph schemas with alphabets $\Sigma_1, \ldots \Sigma_n$ and lattice orders $\preceq_1, \ldots \preceq_n$. A set $\mathfrak{C}$ is called *correspondence schema of size n* over $\mathfrak{S}_1, \ldots \mathfrak{S}_n$, if: $\mathfrak{C} := \pi \uplus \nu$, where $\pi \subseteq \Sigma_1 \times \ldots \times \Sigma_n$ and $\nu \subseteq \Sigma_1 \times \ldots \times \Sigma_n$ are sets of positive and negative correspondence axioms, and $\forall(\mathfrak{c}_1, \ldots \mathfrak{c}_n) \in \mathfrak{C} : \mathfrak{c}_j \neq \top_j$ $(j = 1 \ldots n)$, $\forall(\mathfrak{c}_1, \ldots \mathfrak{c}_n) \in \mathfrak{C} : \mathfrak{c}_j \neq \bot_j$ $(j = 1 \ldots n)$, $\nexists(\ldots \mathfrak{c}_i, \ldots \mathfrak{c}_j, \ldots) \in \mathfrak{C}$, $\nexists \mathfrak{v} \in \mathfrak{V}_i$, $\nexists \mathfrak{a} \in \mathfrak{A}_j : \mathfrak{v} \preceq_i \mathfrak{c}_i \wedge \mathfrak{a} \preceq_j \mathfrak{c}_j$, and $\nexists(\ldots \mathfrak{c}_i, \ldots \mathfrak{c}_j, \ldots) \in \mathfrak{C}$, $\nexists \mathfrak{a} \in \mathfrak{A}_i$, $\nexists \mathfrak{v} \in \mathfrak{V}_j : \mathfrak{a} \preceq_i \mathfrak{c}_i \wedge \mathfrak{v} \preceq_j \mathfrak{c}_j$. □

In a correspondence schema, only links between arc classes and arc classes respectively vertex classes and vertex classes are legal. The intuitive meaning of the correspondence axioms is the declaration of structural and existential dependencies between specification parts of different modeling domains in the universe of discourse — in other words: the declaration of consistency relationship types.

For reasons of simplicity, there is only one correspondence type in the example of this paper. In Fig.9, this correspondence type is represented by the thick line style of the arrows between Phase System, Abstract Phase and Interface. In [11] it is explained that the use of negative or even further correspondence types may be useful, too. Like in UML [9], it would also be possible to attach cardinality constraints to the schema correspondences, but this is not the topic here. Having combined graph schemas into a correspondence schema, it is also possible to combine graph grammars according to the following principle.

*Definition: Principle of Coupled Grammars*
Let $P$, $P'$ be grammars with productions $p_1, \ldots p_n$ respectively $p'_1, \ldots p'_m$, and start symbols $\gamma$, $\gamma'$. With $u \xrightarrow{x} v$ we denote the derivation of a sentence form $v$ from a sentence form $u$ by a finite sequence $x$ of productions. Let further $C :\subseteq P \times P'$ an arbitrary correspondence relation between productions of $P$ and $P'$. With $\mathcal{L}(P)$ and $\mathcal{L}(P')$ we denote the languages generated by $P$ respectively $P'$. Then, we call two sentences $l \in \mathcal{L}(P)$ and $l' \in \mathcal{L}(P')$ *consistent in* $C$ iff: $\exists x \exists x' : \gamma \xrightarrow{x} l, \gamma' \xrightarrow{x'} l'$ with $x := p_{(1)}; \ldots ; p_{(k)}$ where $p_{(i)} \in P$ $(i = 1, \ldots k)$ and $x' := p'_{(1)}; \ldots ; p'_{(k)}$ where $p'_{(j)} \in P'$ $(j = 1, \ldots k)$ with $(p_{(1)}, p'_{(1)}) \in C$, $(p_{(2)}, p'_{(2)}) \in C$, $\ldots$ $(p_{(k)}, p'_{(k)}) \in C$. □

Please note, that this principle does not determine the place where a production $p$ shall be applied to a sentence form $s$; (usually there is more than one possibility, and the grammars need not to be confluent). When this context indeterminism is dissolved by additional application rules, one obtains a particular correspondence definition from the general correspondence principle. For this paper, the graph parsing question [23] is out of the scope.



Coherent to this principle, the following couplings of the rules from Fig.6 and Fig.8 are defined: $c_0 = (q_0, p_0)$, $c_1 = (q_1, p_3)$, $c_2 = (q_2, p_3)$, $c_3 = (q_3, p_0)$, $c_4 = (q_4, p_1)$, $c_5 = (q_4, p_2)$. The according coupling decisions are based on 'insight'. They are made by an expert person who is familiar with the possible inconsistency problems of the examined application domain.

Based on these relations it is possible to define the coupled graph grammar of Fig.10 obeying to the correspondence schema shown in Fig.9 — Please note that the coupled graph grammar cannot be automatically derived from the correspondence schema, but the correspondence schema is important for reasoning about the adequacy of the coupled grammar.

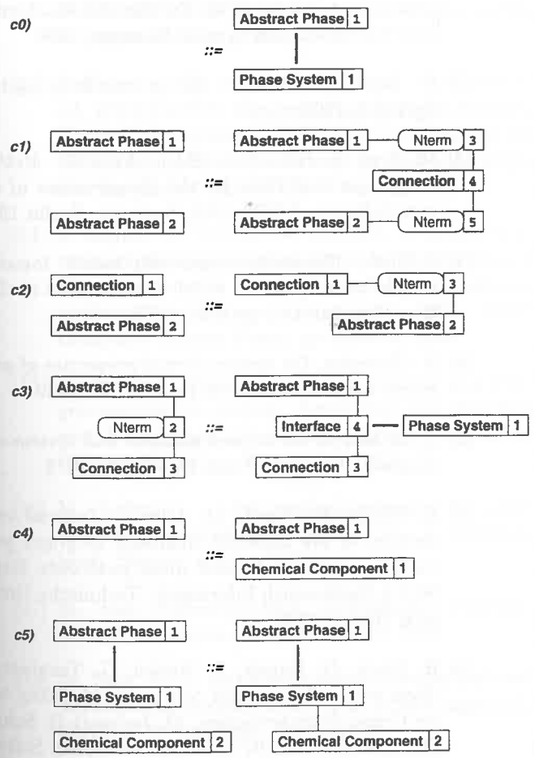


Figure 10: *correspondence grammar to Fig.9*

This coupled grammar constructively defines the legal configurations of all possible sub specification configurations in the here presented distributed modeling example. In general, a specification is regarded to be inconsistent if it cannot be produced by the according consistency defining coupled grammar.

The necessity of some trivial productions is due to the fact that every production $p_i$, $q_j$ has to occur in some coupling $c_k$ while not every concept in a graph schema $S_i$ has got a correspondent concept in a related graph schema $S_j$. Because the dummy $p_3$ is empty, the rules $c_1$ and $c_2$ are the same as $q_1$ and $q_2$. But the dummy $q_4$ is not empty: therefore, the application of $p_2$ in the context of $c_5$ is only allowed, if the Phase System node that the Chemical Component node shall be plugged onto is already integrated with an Abstract Phase node. This is an extra condition that cannot be found in the very sub model grammar of Fig.8. The extra edges that can be found in $c_0$, $c_3$ and $c_5$ do not violate the schema correspondence constraints defined in Fig.9. By the way, it is not difficult to generalize this grammatical model integration approach to problem domains that are divided in more than two sub model domains.

## 5 SUMMARY

Difficult engineering tasks, for example the construction of a chemical device model, require a careful specifying to avoid as many construction errors as possible. Such models are usually distributed into several sub models representing different views of the whole problem for clarity reasons. But with this complexity reduction by distributed modeling a meta problem arises, namely: how to keep the sub models coherent to each other in a consistent total configuration.

In this contribution I have shown an example of a new way of specifying sub model integration by a synergetic combination of graph schema and graph grammar coupling. A more elaborated theory of this approach can be found in [11]. The main idea of the method presented here is to test the coherence of a (sub model) graph grammar relative to a given (sub model) graph schema as well as the coherence of correspondence graph grammar relative to a correspondence schema representing the consistency relations between the sub models of a whole model configuration. The main steps, amongst others, of this method towards an integrative specification are the following:

• Exploit examples for a sufficient understanding of the problem domain. • Construct a graph schema for each sub view of the domain. • Discover integration relations between all involved sub models. • Construct a schema correspondence for these integration relations. • Construct schema-consistent graph grammars for each sub domain. • Construct a correspondence grammar according to the schema correspondence.

Graph models of practical relevance are considerably

bigger than the intuitive examples given in this contribution. Therefore, computer support seems necessary for the construction and maintenance of consistent integration specifications of realistic size. Those integration specifications serve as basis for the implementation of interactive consistency control tools that are to be used by the application experts, in our example: the chemo-engineer.

Different specifications and implementations of such consistency control tools are already reported in the literature on databases and software engineering. But the specification —if any!— of those integrative tools is still difficult and tedious. Therefore, a meta tool called 'correspondence editor' that shall support the specifier of consistency control tools is currently being implemented by our research group. The main tasks of this meta tool, amongst other analyses, shall be the following:

• Check, if a graph schema is syntactically correct and lattice-shaped. • Check, if a graph grammar is coherent relative to a graph schema. • Compute a list of legal couplings from given schema correspondence. • Check, if a correspondence grammar is coherent relative to the computed list.

In the title of my contribution I have asked the question why chemo-engineers should be interested in graph grammars. Now, at the end of the paper, it should be obvious that graph grammars can serve as a powerful means to describe the integration problems the engineer is confronted to, and thus pre-specify the integration tools to cope with those problems. As there already are approaches to automatically derive integration tools out of graph grammar specifications [14] we may hope to meet some more useful integration tools on that way some day. Of course, the running example from chemical engineering can be replaced by any other example of ER-like distributed modeling. However, software support for chemical engineering is a fair novel research field to be examined by our group.

To avoid misunderstandings I have to stress that the integration method presented in this contribution is sufficient only for the solution of s t r u c t u r a l inconsistency problems. Without incorporated knowledge of chemistry, the integration system cannot prevent the engineers from producing undesired chemicals by undesired reactions. But supposed that the engineers are experts of their subject, a structural consistency maintenance tool shall be able to increase the efficiency of their modeling part of work.

One of several interesting open questions is: if an appropriate correspondence grammar can automatically —or at least semi-automatically— be derived if the single graph grammars, the single graph schemas, and the correspondence schema are given.

ACKNOWLEDGMENTS

Drafts of this paper have been helpfully criticized by Ansgar Radermacher, Dr. Thomas Noll, and Prof. Dr. Andy Schürr from different institutes of computer science in Aachen and München.

Due to valuable remarks provided by the anonymous referees of the TAGT'98 workshop I have —hopefully— been able to improve as well the related work section as the general understandability of this contribution.

The German Research Community DFG is granting the author with a scholarship from a special fund for doctoral-students.

REFERENCES

[1] M. Andries, G. Engels, A. Habel, B. Hoffmann, H.-J. Kreowski, S. Kuske, A. Schürr, G. Taentzer, *Graph transformation for specification and programming.* Report 7/96, Fachbereich Mathematik und Informatik, Universität Bremen, 1996

[2] R. Book, F. Otto, *String rewriting systems.* Springer, 1993

[3] M. Broy, S. Jähnichen (Ed.), *KORSO: Methods, Languages and Tools for the Construction of Correct Software.* LNCS 1009, Springer, Berlin 1995

[4] P. Chen, *The entity relationship model: towards a unified view of data.* ACM Transactions on Data Base Systems 1/1, pp.9-36, 1976

[5] N. Chomsky, *On certain formal properties of grammars.* Inform. Control 2, pp.137-167, 1959

[6] T. de Marco, *Structured analysis and system specification.* Yourdon Press, New York, 1978

[7] H. Ehrig, M. Korff, M. Löwe, *Tutorial introduction to the algebraic approach of graph grammars based on double and single push-outs.* Report 90/21, Fachbereich Informatik, Technische Universität Berlin, 1990

[8] H. Ehrig, G. Engels, R. Heckel, G. Taentzer, *A View oriented Approach to System Modeling based on Graph Transformation.* M. Jazayeri, H. Schauer (Ed.), ESEC-FSE'97 Joint 6th European Software Engineering Conference and 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering. LNCS 1301, pp.327-343, Springer, Berlin 1998

[9] M. Fowler, K. Scott, *UML distilled.* Addison Wesley, New York, 1997

[10] S. Gruner, M. Nagl, A. Schürr, *Integration Tools Supporting Development Processes.* M. Broy, B. Rumpe (Ed.), RTSE'97 Workshop on Requirements Targeting Software and Systems Engineering. LNCS 1526, pp.235-256, Springer, Berlin 1998. Earlier version electronically available from ftp://ftp.informatik.rwth-aachen.de/pub/reports/index.html

[11] S. Gruner, *Einige Anmerkungen zur graphgrammatischen Spezifikation von Integrationswerkzeugen nach Westfechtel, Janning, Lefering und Schürr.* Report AIB-98-3, Fachgruppe Informatik, RWTH Aachen, 1998. Electronically available from ftp://ftp.informatik.rwth-aachen.de/pub/reports/index.html

[12] G. Heidenreich, *Ein generisches Organisationsschema der Konfigurationsverwaltung im Software-Engineering* (Doct. Dissertation). Arbeitsberichte des IMMD 29/13, Friedrich-Alexander-Universität Erlangen-Nürnberg, 1996

[13] P. Heimann, G. Joeris, C.-A. Krapp, B. Westfechtel, *DYNAMITE: Dynamic Task Nets for Software Process Management.* ICSE'96 Proceedings of the 18th International Conference on Software Engineering, pp.331-341, IEEE Computer, 1996

[14] J. Jahnke, W. Schäfer, A. Zündorf, *A design environment for migrating relational to object-oriented data base systems.* Proceedings of the international conference on software maintenance 1996, IEEE Computer Society Press, pp.163-170, 1996

[15] T. Janning, *Requirements Engineering und Programmieren im großen: Integration von Sprachen und Werkzeugen* (Doct. Dissertation). Deutscher Universitätsverlag, Wiesbaden 1992

[16] M. Lefering, *Integrationswerkzeuge in einer Software-Entwicklungsumgebung* (Doct. Dissertation). Shaker, Aachen 1995

[17] M. Nagl, *Softwaretechnik: methodisches Programmieren im großen.* Springer, Berlin 1990

[18] M. Nagl (Ed.), *Building tightly integrated software development environments: The IPSEN approach.* LNCS 1170, Springer, Berlin 1996

[19] C. Petri, *Kommunikation mit Automaten* (Doct. Dissertation). Schriften des Institutes für instrumentelle Mathematik, Bonn 1962

[20] C. Petri, *Grundsätzliches zur Beschreibung diskreter Prozesse.* Drittes Kolloquium über Automatentheorie, pp.121-140, Birkhäuser, Basel 1967

[21] John Pfaltz, Azriel Rosenfeld, *Web Grammars.* Proc. International Joint Conference on Artificial Intelligence, pp.609-619, Washington 1969

[22] T. Pratt, *Pair grammars, graph languages and string-to-graph translations.* Journal of Computer and System Sciences 5, pp.560-595, 1971

[23] J. Rekers, A. Schürr, *Defining and Parsing Visual Languages with Layered Graph Grammars.* Journal of Visual Languages and Computing 7/3, 1996

[24] G. Rozenberg (Ed.), *Handbook of graph grammars and computing by graph transformation*, Volume 1 (foundations). World Scientific, Singapore 1997

[25] G. Rozenberg et al. (Ed.), *Handbook of graph grammars and computing by graph transformation*, Volume 2 (specifications and programming). to appear 1998

[26] RWTH Forum of Informatics, *SFB 476 IMPROVE.* Informatik-Forschung und Entwicklung 13/2, pp.91-92, Springer, Berlin 1998

[27] G. Santucci, C. Batini, G. di Battista, *Multilevel schema integration.* R. Elmasri, V. Kouramajian, B. Thalheim (Ed.), Entity relationship approach ER'93, LNCS 823, pp.327-338, Springer, Berlin 1993/94

[28] A. Schürr, A. Winter, *Modules and updatable Graph Views for Programmed Graph Rewriting Systems.* Report AIB-97-3, Fachgruppe Informatik, RWTH Aachen, 1997. Electronically available from ftp://ftp.informatik.rwth-aachen.de/pub/reports/index.html

[29] A. Schürr, *The PROGRES Language Manual* Version 9.$_x$, 1998 http://www-i3.informatik.rwth-aachen.de/research/progres/index.html

[30] B. Westfechtel, *Graph-based models and tools for managing development processes* (Lecturer Habilitation). RWTH Aachen, to appear 1998