Electronic Notes in Theoretical Computer Science 50 No. 3 (2001) – Proc. GT-VMT 2001
URL: http://www.elsevier.nl/locate/entcs/volume50.html 8 pages

Comparing Notions of Hierarchical Graph Transformation ⋆

Giorgio Busatto and Berthold Hoffmann [1]

Fachbereich Mathematik/Informatik, Universität Bremen
Postfach 33 04 40, 28334 Bremen
{giorgio|hof}@informatik.uni-bremen.de

1 Introduction

The Unified Modeling Language is a prominent evidence for the steadily increasing importance of visual languages for modeling software. It is known that the syntax of a visual language can be represented by *graphs*, and its semantics can be specified by *graph transformation* [2].

Since software models may be large, their graph representation should provide a concept for dividing graphs into nested *packages*. Several notions of *hierarchical graphs* have been proposed for this purpose; they differ in aspects such as whether packages may be shared or not, and whether edges may cross package borders or not. *Transformation* has only been considered for some of them, and a commonly accepted notion of hierarchical graphs and their transformation is still missing. Aiming at filling this gap, we use a notion of hierarchical graph transformation [4] that is *generic*, i.e. not committed to a particular kind of graphs or graph transformation, and *decouples* the package structure from the underlying graph. The existing approaches of *H-graph grammars* [16], and *hierarchical hypergraph transformation* [6] are then compared by translating them to the generic notion. This shows up their similarities and differences clearly, and demonstrates that the generic notion may simulate many notions of hierarchical graphs. Space does only permit to outline definitions and results, which will be given in the full paper.

2 Generic Hierarchical Graph Transformation

Generic hierarchical graph transformation [4] aims at investigating structuring mechanisms for graphs. The definition *decouples* the flat underlying graph

⋆ This work has been partially supported by the ESPRIT Working Group *Applications of Graph Transformation* (APPLIGRAPH).

[1] The first author was supported by a grant of the European TMR Network *General Theory of Graph Transformation Systems.* (GETGRATS).

from its package structure, so that both aspects can be studied independently of each other. It is also *generic* so that it can be used to extend arbitrary notions of graph transformation with a package concept, or to simulate existing notions of hierarchical graph transformation.

Graphs. A set $\mathcal{G}$ defines a set of graphs if every $G \in \mathcal{G}$ has a *skeleton* $S_G = (N_G, E_G, I_G)$, where N_G and E_G are finite sets of *nodes* and *edges*, and $I_G \subseteq E_G \times N_G$ is an *incidence relation*. Having a skeleton is the minimal requirement for an entity to be considered a graph, and it serves as an interface to the hierarchical structure added to it.

A directed *graph* G consists of disjoint finite sets N_G of *nodes* and E_G of *edges*, with each edge attached to exactly one *source* and one *target* node, and each node (edge) labelled over a given set Σ (resp. Δ) of labels. Clearly, each directed graph G provides a skeleton; it is *rooted* if it has a distinguished *root* $r \in N_G$ so that there exists a path from r to n in G, for all $n \in N_G$. (Rooted graphs are used for package hierarchies.)

A bipartite graph C over (U, P)—i.e. a directed graph where all edges have one end in M and the other one in N—is a *coupling graph* if it induces an *association relation* $\leftarrow_C \subseteq P \times U$ that assigns every node of U to at least one node of P. A coupling graph C is *tight* if it also induces a *correspondence relation* $\approx_C \subseteq U \times P$ that anchors every node of P at a unique node of U. (Coupling graphs are used for connecting package graphs and underlying graphs.)

Generic Hierarchical Graphs. A *generic hierarchical graph* is a triple $H = (U, P, C)$, with an *underlying graph* U (of any kind, provided it has a skeleton), a rooted *package graph* P, and a bipartite *coupling graph* C over nodes $(N_U \cup E_U, N_P)$. If C is tight, H is called *tightly coupled*, and *loosely coupled* otherwise. Note that the union of H's components is *not* a graph, as C uses the edges of U as nodes.

Basic Transformation Approaches. The notion of a graph transformation approach has been formalized in [1] in order to specify the common features of as many kinds of graph transformation as possible. (See [17] for a survey of approaches.) Here we are only concerned with *basic* graph transformation approaches $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Rightarrow)$ where $\mathcal{G}$ is a class of *graphs*, $\mathcal{R}$ is a class of *rules*, and $\Rightarrow$ is a *rule application operator* that associates a binary relation $\Rightarrow_r \subseteq \mathcal{G} \times \mathcal{G}$ to every rule $r \in \mathcal{R}$. We ignore the control conditions and graph class expressions that are used for programming and specification in [1].

Generic Hierarchical Graph Transformation. A basic *hierarchical graph transformation approach* $\mathcal{A}_{\mathcal{H}} = (\mathcal{H}, \mathcal{R}_{\mathcal{H}}, \Rightarrow_{\mathcal{H}})$ is constructed by combining an underlying graph transformation approach $\mathcal{A}_u$ over graphs $\mathcal{G}_u$ with two graph transformation approaches $\mathcal{A}_p$ over rooted graphs, and $\mathcal{A}_c$ over coupling graphs, respectively, by componentwise composition. If its component approaches have the same application operator, we call $\mathcal{A}_{\mathcal{H}}$ *homogeneous*.

The classes of graphs and rules are defined as the cartesian products of the corresponding component classes, and their semantics is constructed componentwise, too. The application operator is defined as:

$$\Rightarrow_{\mathcal{H}} = \{((U, P, C), (U', P', C')) \in \mathcal{H} \times \mathcal{H} \mid U \Rightarrow_u U', P \Rightarrow_p P', C \Rightarrow_c C'\}$$

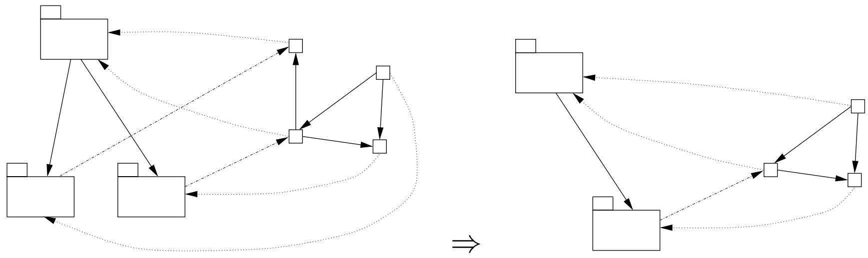

Fig. 1. A hierarchical graph transformation.

Figure 1 depicts a hierarchical graph transformation step where, both for the host and for the result graph, the hierarchy graph is depicted using big rectangular nodes (packages) with tabs, the underlying graph has small square nodes, and the coupling graph has dashed edges. (The associations of edges to packages are omitted.) The operation shown involves the deletion of a node and of the package anchored to it. The top right node in the underlying host graph, which was associated to the deleted package, is moved to the root package.

3 H-Graph Grammars à la Pratt

In [16], hierarchical graphs (so-called *H-graphs*) model runtime data structures for the definition of programming language semantics, and *H-graph grammars* model operations on them. An H-graph contains a global set of nodes N, where each node is either labeled over a given set of *atoms* A, or it contains a directed graph over N, thus inducing a hierarchical structure. Each *H-graph grammar* production specifies the substitution of an atomic node with a new H-graph. Edges attached to the replaced node in the original H-graph are redirected to two special nodes in the right-hand side of a production.

We use NLC graph rewriting (see e.g. [8]) for modeling H-graph grammars. In this approach, an induced subgraph of the host graph is matched by the left-hand-side graph of a rule, replaced with a copy of the right-hand side, and new connecting edges are created under the control of a global set of connection instructions.

An H-graph H is translated into a hierarchical graph $\mathbf{HG}(H)$ by splitting it into three graphs: the underlying graph $U(H)$ contains all the nodes of H and all the edges collected from all local graphs of the hierarchy; the hierarchy graph $P(H)$ contains one root package, one package for every non-atomic node, and a package q is nested in a package q' if the corresponding nodes n and n'

are nested in H; the coupling graph $C(H)$ contains all packages from $P(H)$, all nodes and edges from $U(H)$, and associates every node or edge to its owning package (where root nodes of H are assigned to the root package), and every package to its corresponding node. The node labels of the three component graphs encode the original label in H and additional information—used by connection instructions—determining whether a node is an input or an output node in a production, or a normal node.

An H-grammar is translated into a set of hierarchical graph rules—one triple of NLC rules $(\upsilon_p, \pi_p, \kappa_p)$ for each production p—together with a global set $\mathcal{C}$ of connection instructions. The production υ_p (π_p) specifies the substitution of a node (package) with a graph, whereas κ_p specifies the substitution of a node and its corresponding package with all nodes and packages from υ_p and π_p, and the insertion of the necessary coupling edges between them. Given such a hierarchical rule r, we consider special direct derivations from a hierarchical graph $HG = (U(H), P(H), C(H))$, where υ_p and κ_p are applied to the same node in $U(H)$ and $C(H)$, and π_p and κ_p are applied to the same package in $P(H)$ and $C(H)$. We denote such a derivation with $(U(H), P(H), C(H)) \Rrightarrow_r (U', P', C')$ and we call it an *amalgamated derivation step*.

The main result of this section—whose proof is given in the full paper [3]—says that we can simulate derivations of an H-graph grammar by means of amalgamated derivations in the corresponding grammar using triples of NLC rules. Therefore amalgamated derivation steps of translated H-graph grammar rules faithfully mimic the original H-graph grammar derivations as triples of NLC derivation steps.

Proposition 1 Let H and H' be two given H-graphs, Γ an H-grammar, p a rule of Γ, and $r = (\gamma_p, \pi_p, \kappa_p)$ the translation of p to a hierarchical graph rule. Then $H \Rightarrow_p H'$ iff $\mathbf{HG}(H) \Rrightarrow_r \mathbf{HG}(H')$.

4 Hierarchical Hypergraph Transformation

In [6], hierarchical hypergraph transformation is defined as a computational basis for programming with graphs [14].

A *hypergraph* is finite, and consists of nodes and labelled hyperedges that may be attached to any number of nodes. In a *hierarchical hypergraph*, some of the hyperedges (called *frames*) may contain hypergraphs that may be hierarchical again.

A hierarchical hypergraph H is translated to the *generic hierarchical hypergraph* $\mathbf{HG}(H) = (U, P, C)$ as follows: The underlying hypergraph U disjointly unites all top-level nodes and hyperedges with all nodes and hyperedges contained in all frames, recursively. The *package graph* P is a tree with a root package ρ, plus a package for every frame in H so that the edges in P represent the direct nesting of frames. Finally, the *coupling graph* C associates the top-level nodes and hyperedges to the package ρ, and all nodes and hyperedges directly contained in a frame to its package; furthermore, every nested package corresponds to a frame.

It is easy to see that a generic hierarchical graph is a translation of a hierarchical hypergraph iff it is *strict* in the following sense: *(i)* its underlying graph is a hypergraph; *(ii)* its package graph is a tree; *(iii)* every underlying node and hyperedge is associated to exactly one package; *(iv)* there are no *package-crossing hyperedges*: every hyperedge y is attached to nodes of the same package; *(v)* every nested package, except for the root package ρ, corresponds to an underlying edge.

A *hierarchical morphism* $m : H \to H'$ maps nodes and hyperedges of H and H' onto each other so that labels, attachments, and frames are preserved, and the contents of corresponding frames in H and H' is related by hierarchical morphisms, recursively. A *hierarchical hypergraph transformation rule* $t = P \stackrel{p}{\leftarrow} I \stackrel{r}{\rightarrow} R$ consists of two hierarchical morphism that embed a common *interface* I in a *pattern* P and a *replacement* R. (The morphism p must be injective.)

A transformation step $H \Rightarrow_t H'$ is constructed as follows: Match P as a subgraph in a package of the host graph H and construct an injective *matching* morphism m between P and that subgraph; remove $m(P)$ up to $m(p(I))$ from H to obtain the context graph C; add a copy of R to C and glue $m(p(I))$ with $r(R)$ to obtain H'. [2]

Amalgamated Generic Hypergraph Transformation. Every hierarchical morphism $m : H \to H'$ corresponds one-to-one to a triple of morphisms between the components of the generic hierarchical hypergraphs $\mathbf{HG}(H)$ and $\mathbf{HG}(H')$. A hierarchical hypergraph transformation rule $t = P \stackrel{p}{\leftarrow} I \stackrel{r}{\rightarrow} R$ can thus be translated into a triple of gluing rules on underlying graphs, package graphs and coupling graphs. Let $\mathbf{hg}(t)$ denote that generic hierarchical rule, and require that transformation steps $\mathbf{HG}(H) \Rrightarrow_{\mathbf{hg}(t)} \mathbf{HG}(H')$ are *amalgamated* so that the matching morphisms overlap in the nodes of their coupling component. Then we get the main result for this translation by the correspondence of morphisms:

Proposition 2 There is a hierarchical hypergraph transformation step $H \Rightarrow_t H'$ iff there is a generic hierarchical hypergraph transformation step $\mathbf{HG}(H) \Rrightarrow_{\mathbf{hg}(t)} \mathbf{HG}(H')$.

5 Conclusions

Generic hierarchical graph transformation turns out to be general enough to represent existing approaches to hierarchical graph transformation. Thus the

[2] This is a kind of gluing graph transformation [7] with injective matching [13].

decoupled representation makes it particularly easy to grasp differences of the approaches compared in this paper, which are summarized in Table 1.

	H-graph grammars [16]	**hierarchical hypergraph transformation [6]**
underlying graphs	simple graphs	hypergraphs
hierarchy	rooted graph	tree
coupling	tight	tight
package anchors	nodes	hyperedges
inter-packages edges	yes	no
transformation	NLC-like	injective gluing

Table 1
Comparison of Hierarchical Graph Notions

Although developed for a different application, our model of hierarchical graph transformation does remind of triple graph grammars [18], which also provide some kind of amalgamation, but are tied to a particular graph transformation approach. Related work has studied *encapsulation* concepts for hierarchical graphs [12] (yet without notion of transformation), and the construction of views [9] on (flat) graphs.

Further approaches to hierarchical graph transformation, namely *hierarchical graph transformation with variables* [6], *typed hierarchical graph transformation* [11], and *Higraphs* [15], shall be investigated in order to confirm our conjecture that practically every kind of hierarchical graph transformation can be simulated with the generic model.

Also, the generic model needs still to be refined with respect to the interrelation of elements in different components of the transformation rule triples. In our examples, this was no problem since the rule triples were *homogeneous* (using the same transformation approach) so that transformations could be amalgamated.

References

[1] M. Andries, G. Engels, A. Habel, B. Hoffmann, H.-J. Kreowski, S. Kuske, D. Plump, A. Schürr, and G. Taentzer. Graph transformation for specification and programming. *Science of Computer Programming*, 34:1–54, 1999.

[2] R. Bardohl, M. Minas, A. Schürr, and G. Taentzer. Application of graph transformation to visual languages. In Engels et al. [10], chapter 3, pages 105–180.

[3] G. Busatto and B. Hoffmann. Comparing notions of hierarchical graph transformation. Technical report, Fachbereich Mathematik-Informatik, Universität Bremen, 2001. In preparation.

[4] G. Busatto, H.-J. Kreowski, and S. Kuske. An abstract hierarchical graph data model. Technical report, Fachbereich Mathematik-Informatik, Universität Bremen, 2001. In print.

[5] V. Claus, H. Ehrig, and G. Rozenberg, editors. *Proc. Graph Grammars and Their Application to Computer Science and Biology*, number 73 in Lecture Notes in Computer Science. Springer, 1979.

[6] F. Drewes, B. Hoffmann, and D. Plump. Hierarchical graph transformation. *Journal on Computer and System Science*, 2001. Accepted for publication.

[7] H. Ehrig. Introduction to the algebraic theory of graph grammars. In Claus et al. [5], pages 1–69.

[8] J. Engelfriet and G. Rozenberg. Node replacement graph grammars. In Rozenberg [17], chapter 1, pages 1–94.

[9] G. Engels, H. Ehrig, R. Heckel, and G. Taentzer. A view-based approach to system modelling based on open graph transformation systems. In Engels et al. [10], chapter 16, pages 639–668.

[10] G. Engels, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. II: Specification and Programming.* World Scientific, Singapore, 1999.

[11] G. Engels and R. Heckel. Graph transformation as a conceptual and formal framework for system modelling and evolution. In U. Montanari, J. Rolim, and E. Welz, editors, *Automata, Languages, and Programming (ICALP 2000 Proc.)*, number 1853 in Lecture Notes in Computer Science, pages 127–150. Springer, 2000.

[12] G. Engels and A. Schürr. Encapsulated hierarchical graphs, graph types and meta types. In A. Corradini and U. Montanari, editors, *Proc. Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation*, number 2 in Electronic Notes in Theoretical Computer Science, `http://www.elsevier.nl/locate/entcs`, 1995. Elsevier.

[13] A. Habel, J. Müller, and D. Plump. Double pushout graph transformation revisited. In G. Engels and G. Rozenberg, editors, *Theory and Application of Graph Transformation (TAGT'98), Selected Papers*, number 1764 in Lecture Notes in Computer Science, pages 103–116. Springer, 2000.

[14] B. Hoffmann and M. Minas. A generic model for diagram syntax and semantics. In *ICALP Workshops 2000*, number 8 in Proceedings in Informatics, pages 443–450, Waterloo, Ontario, Canada, 2000. Carleton Scientific.

[15] A. Poulovassilis and M. Levène. A nested-graph model for the representation and manipulation of complex objects. *ACM Transactions on Information Systems*, 12(1):35–68, 1994.

[16] T. W. Pratt. Definition of programming language semantics using grammars for hierarchical graphs. In Claus et al. [5], pages 389–400.

[17] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. I: Foundations.* World Scientific, Singapore, 1997.

[18] A. Schürr. Specification of graph translators with triple graph grammars. In G. Tinhofer, editor, *Proc. WG94 int.workshop on graph theoretic concepts in comp. sci.*, number 903 in Lecture Notes in Computer Science, pages 151–163, Herrsching, 1994. Springer-Verlag.