Hierarchically Distributed Graph Transformation*

Gabriele Taentzer

Computer Science Department, Technical University of Berlin,
Franklinstr. 28/29, Sekr. FR 6-1, D-10587 Berlin,
e-mail: gabi@cs.tu-berlin.de

Abstract. Hierarchically distributed graph transformation offers means to model different aspects of open distributed systems very intuitively in a graphical way. The distribution topology as well as local object structures are represented graphically. Distributed actions such as local actions, network activities, communication and synchronization can be described homogeneously using the same method: graph transformation. This new approach to graph transformation follows the lines of algebraic and categorical graph grammars and fits into the framework of double-pushout high-level replacement systems.

Keywords: Graph transformation, distributed systems, communication, synchronization

1 Introduction

Graphical representations are an obvious means to describe different aspects of systems. Modeling distributed and concurrent systems graphs are often used to describe the *topological structure* of the system. The graphical structure shows then which parts are involved and what are the ways of communication. Graph transformations can be used conveniently to model dynamic changes of the system structure. For example, the distribution of some local parts is rearranged or communication channels are created or deleted. Local states are typically coded in some specification or programming text or not considered. This idea is followed, for example, in [3], by Δ-grammars in [9], in [13] and by actor graph grammars in [11].

Graphs can be used also to model complex *object relations* inside of local parts of a system as they arise, for example, in database systems (entity-relationship models described in [2]) or software process modeling (development graphs as used in [12] or project flow graphs in [10]). Graph transformations are useful then on these lower levels to specify changes of object relations.

The possibility to allow local actions to run concurrently can be modeled by distributed graph transformation following the algebraic approaches in [5], [8]

* This work has been partly supported by the ESPRIT Working Group 7183 "Computing by Graph Transformation (COMPUGRAPH II)"

and [15]. Here, some restricted types of network structures are allowed which can be changed by special operations, namely SPLIT and JOIN. SPLIT splits a graph into two or more local graphs with an interface graph between each two where the connection to the local graphs is described by graph morphisms. A local graph is not allowed to be split again, i.e. hierarchical network structures cannot be modeled. JOIN joins a distributed graph of such a kind again to one graph.

An approach which combines graph transformation on the network and on the local level is the work of categorical graph grammars in [14]. This kind of graph grammar allows a flexible change of network structures and a description of local actions for example by graph transformation. Communication is not expressed by means of graphs and graph morphisms, i.e. object identity has to be coded into names, for example.

With hierarchically distributed graph transformation the advantages of the categorical and algebraic graph grammars for modeling distributed systems are combined. It is possible to handle complex network structures, especially hierarchical ones. Communication and synchronization can be modeled by interface graphs which are connected to their local graphs by graph morphisms, i.e. object identities are described by graph morphisms between different local states.

This paper is organized as follows: In the next section the modeling of distributed systems based on graph transformations is discussed. A simplified example of distributed software development serves as illustration. All main features are introduced in that section. In section 3 the formal description of hierarchically distributed graph transformation is presented following the double-pushout approach to graph transformation ([4]). It is shown that this new approach fits into the framework of HLR-systems introduced in [6]. For this section the reader is supposed to be familiar with basic notions of category theory as they are presented, f. ex. in [1]. A reader not interested in the formal description of hierarchically distributed graph transformation can skip section 3 and gets a good impression of hierarchically distributed graph transformation anyway.

2 Distributed Systems Modeled by Graph Transformation

In this section the main features for modeling of distributed systems by graph transformations are presented. Distributed states are modeled by graphs and state transitions, i.e. local actions, network activities or communication are modeled by graph transformation.

Example 1 (Distributed software development). Several people developing a software system in parallel have to cooperate with each other. As an support *development graphs* (introduced in [12]) or *project flow graphs* (presented in [10]) can be used which describe different states of software development. These kinds of graphs have been introduced in order to assure software quality and possibly speed up the project. The developers are allowed to concurrently work on the software. Every *development step* is described by a graph transformation, in the following. All further examples are closely related to project flow graphs and their developments, although they are simplified in some minor points.

2.1 Distributed States

Usually a state of a system can be described by a graph where the nodes represent objects and the edges relations between them. If the state is a distributed one it can be reflected by several graphs where each of them shows a local state. In the following we call graphs describing a local state *local graphs*. The objects and relations in such a local state are called local, too.

Example 2 (Local software development graph). In so-called local development graphs dependencies between different units or, more concrete, documents, input and output relations of development tools and revisions of documents are modeled. Such a graph stores about the same information as a revision and configuration management system together with a tool like "make". The local graph in figure 1 contains three nodes of type "doc" modeling documents, two "tool"-nodes modeling two different development tools, i.e. editors, compilers, etc. and two "rev"-nodes standing for two revisions of documents. Furthermore, input and output relations are described by edges from "doc"-nodes to "tool"-nodes and vice versa. They are drawn as solid arrows. Edges drawn as dashed arrows model dependencies between different documents. A "doc"-node with a "rev"-node at its lower right part represents a document which has a revision. Notice that there is not an arrow which explicitly shows this relation. Internally the relation is modeled by an edge as all other relations, too.

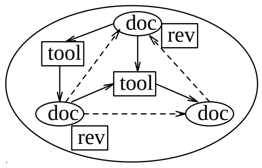

Fig. 1. A local development graph

Different local systems are usually connected by some kind of network. They interact with each other by some interface. The interfaces as well as the local systems are allowed to change their states. This means that we have not only local states but also interface states and, moreover, network states. In the network states the current distribution structure of the whole system is stored. It changes if, for example, new local systems are added or connections are changed.

Modeling the network structure of local systems by a *network graph* its nodes represent the local systems. These nodes are called *network nodes*. The edges of the network graph describe relations between local systems. These *network edges* can model some kind of links that have to be hold consistent.

The whole distributed state is described by a so-called *hierarchically distributed graph*, short HD-graph, which consists of a network graph where each network node is equipped with a local graph representing the current state of its local system. Each network edge is equipped with a total graph morphism describing the relation between two local states. These total graph morphisms are the essential basis for interaction of local systems. They are used to describe which local objects and relations correspond to each other in different local parts. Local graphs which are the target of such a graph morphism are called *target graphs*. Analogously, the sources of graph morphisms are called *source graphs*.

Example 3 (Development graph). Considering the development of a big software system it has to be determined first which groups and, more concrete, which persons have to do which parts. Such a distribution which is usually rather hierarchical can be modeled by a network graph. In our example a "big project" in the state described in example 2 is modeled by a so-called node which contains the local development graph in figure 1. The "big project" is distributed in those portions which are handled by one developer each. Thus, the network graph in figure 2 contains two "developer"-nodes which are connected with the "big project"-node by a network edge. It models how a developer part fits into the whole project shown by corresponding layout of local graphs.

Both developers should cooperate via some interface modeled by an "interface"-node and two network edges from this node to the "developer"-nodes. The local development graphs in the "developer"-nodes show the local states in each case which are parts of the "big project"-graph. The interface between the developers should contain those objects and relations which can be used or should be handled by both. These can be some kind of prereleases of produced documents which should be forwarded to other developers. All objects and relations which do not have a correspondence in the interface are considered as hidden for other developers, i.e. they do not have access.

Furthermore, the developers can use different views on their development parts, namely the "semantical view" and the "operational view" modeled by three so-called nodes and their connections to the corresponding developer parts.

The semantical view shows all documents as well as revisions and their interdependencies. The operational view is restricted to the connections between tools and documents.

In [5] and [8] distributed graphs with exactly two local graphs and one interface graph, i.e. a source graph which describes common parts are considered. This notion is extended in [15] to an arbitrary number of local graphs where an interface graph has to be established between each two. In both approaches it is not possible to define *distribution hierarchies*. This means, for example, that there can be an interface between interface graphs.

Usually a hierarchy means some kind of tree or, more generally, a directed acyclic graph. In our case of hierarchically distribution it is not an essential step to allow arbitrary graphs since loops or cycles model a distribution hierarchy, too, but in an abbreviated notation. For example, a network described by a

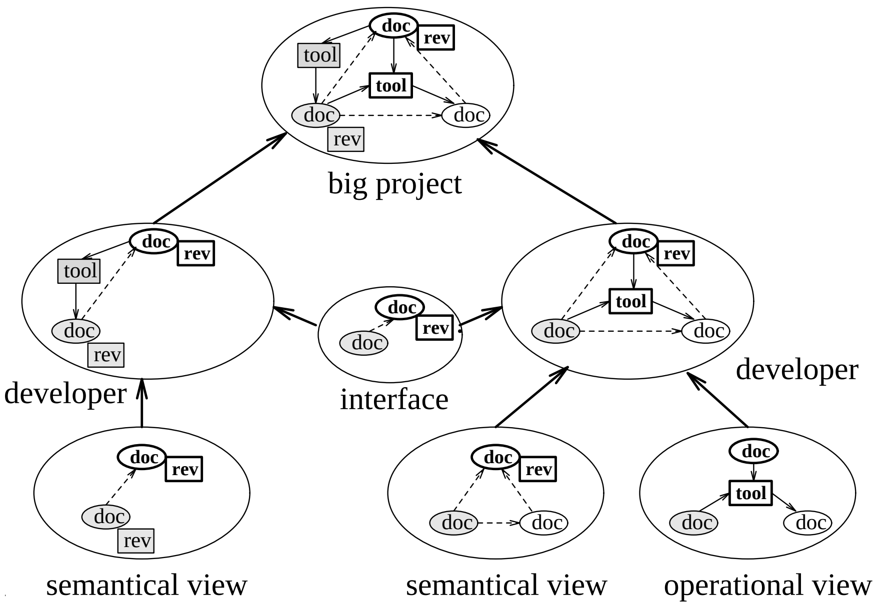

Fig. 2. An HD-graph for distributed software development

graph that consists of a node with an associated loop can also be modeled by one node and its copy with an edge in between.[2]

2.2 Local Actions

In the graph grammar field actions are usually described by *graph productions* and modeled by *graph transformation*. In this paper we use the double-pushout approach to graph transformation which characterizes some kind of cautious rewriting of graphs. This means that for a production and its matching part in some graph the following conditions have to be satisfied. Context edges are not allowed to dangle, i.e. a node which is connected to a context node has to be preserved, and two items (nodes or edges) are not allowed to be identified if at least one of them should be deleted. These conditions are combined in the well-known *gluing condition*. If the gluing condition is satisfied for a production and a match of its left hand side in the current graph a new graph is derived by deleting this occurrence and adding the right hand side of the production. (More details to this kind of graph transformation can be found in [4], etc.)

Conceptually local actions are described by *HD-graph productions* which consist of a *local production* describing the local action and a *network production* which is identical here, since the network graph is actually not changed. It is transformed by an identical production which preserves that network node where the local action took place.

[2] The property "hierarchical" does not belong to the levels of abstraction which have been invented to describe this kind of graph transformation. There are just two abstraction levels, the network level and the local level.

Such an HD-graph production can be applied to an HD-graph if the local action is somehow compatible with the context where it takes place. An action on a local graph which is a target graph is not permitted if it destroys the reference structure to other local graphs, i.e. a source graph cannot be mapped totally to its target graphs any more.

Applying an HD-graph production which describes a local action, first the network production is applied to the current network graph. This just means that the matching of the only network node is replaced identically. The local production is applied to that local graph which is equipped with the matched network node. The local production can be applied if the gluing condition is satisfied for this local graph transformation. After the application the matched network node is equipped with a new graph, the transformed one.

Example 4 (Local development steps). Typical development steps are the introduction of new documents, merging documents, storing new revisions, merging revisions, introducing new tools and therefore changing processing relations or changing dependency relations. These actions do not change the network structure. Development steps of that kind are considered in the next section. Here, we consider for example the introduction of a new tool as a local development step done by the "big boss" in the state of the "big project". The HD-graph production *new tool* in figure 3 describes this local development step where a tool requiring one input and one output document is introduced.

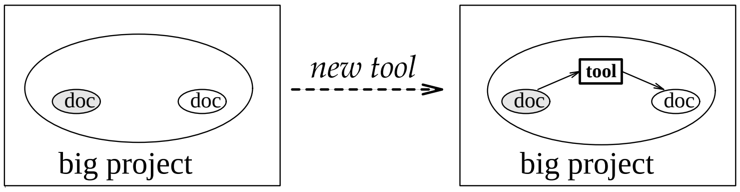

Fig. 3. Introduction of a new tool

Since there is nothing deleted in this production it can be applied to the development graph in figure 2. Otherwise a reference to an object or relation in the "big project" would be destroyed. Such destructions have to be arranged with the developers and, thus, are not local. A similar action as modeled in figure 3 cannot be performed by a developer since it has to be reported to the "big boss", i.e. all objects and relations belonging to the "developer"-node have to be totally mapped into the "big project".

2.3 Network Administration

Modeling distributed systems by graph transformations means usually the description of network activities. Changes of the network topology can be described by transformation of the network graph. The deletion and creation of network nodes and edges has to be done very cautious to avoid inconsistencies. Deletions may not destroy the reference structure. Creations have to fit into the reference structure, for example, a node in a new interface graph has to have a correspondent in all the local graphs it is interface of.

If such conditions are satisfied the network actions can be done concurrently, i.e. connections can be changed, new local parts can be inserted and other local parts (possibly with connections) can be deleted. Moreover, connections can be deleted and created, too.

Altogether a network activity is also described by an HD-graph production where its network production describes the changes of the network topology. For each network node which is preserved an identical production has to be applied to its local state graph. Each network node (edge) which is deleted or created is equipped with a local graph (morphism) on the left- or right-hand side of the HD-graph production, resp. The HD-graph production can be applied if the conditions for deletion and creation described above are satisfied. Moreover, the gluing condition has to be satisfied for the network part of the application.

Example 5 (Network activities). In a software development process network activities can be the introduction of a new developer or a new interface, the establishment or changing of connections between developers and the “big boss”, etc. In figure 4 the HD-graph production *change connection* is shown where the connection between a developer and a “big project” is relaxed which releases the developer from this “big brother is watching you”-situation modeled in example 3. This means that the local state of the developer is no longer a part of that of the “big project”.

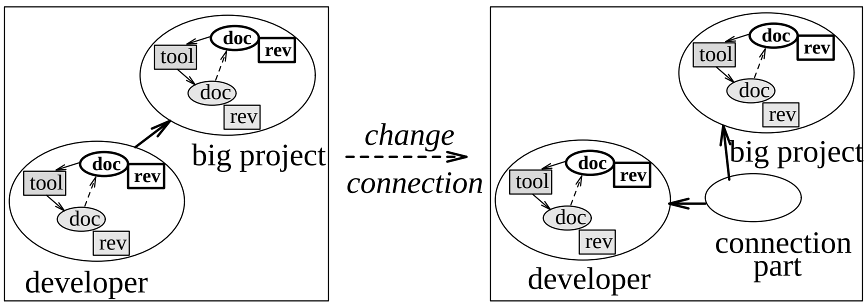

Fig. 4. Changing a connection

Applying this production to the HD-graph in figure 2 the left “developer”-graph is then connected to the “big project”-graph by an empty “interface”-graph. The whole relation between the corresponding local graphs is completely described in the HD-graph production and can be deleted.

2.4 Communication and Synchronization

Synchronization of different local actions is modeled by such actions on network nodes connected with each other. In this case an action on a source graph has to be a subaction in some sense of those on connected target graphs. This means more or less that the same actions on local items are allowed as in their adjoined local graphs. But in the source graph items may be deleted whereas some of their images in the corresponding target graph are preserved. Furthermore, a local item may be inserted in some source graph if its correspondents are preserved in all connected target graphs.

Synchronization is modeled by an HD-graph production where the network production is an identical production similarly to local actions. But here the whole network part which is more than one node is identically replaced. All nodes where a part of the synchronized action should be performed have to be described in the network production. Moreover, all network edges which should be used are identically replaced. For each local part a local production, which models that action part of the synchronized action that should be performed there, should be applied. As before, the application of an HD-graph production should not destroy the reference structure. For each local production application the gluing condition has to be satisfied.

Asynchronous communication consists of at least two concurrent actions. They can be performed by using some interface graphs (as in the examples) for the description of channels or common memory. First one local part puts something into the channel or memory and then the other gets it. Considering the interface as an independent local part the other local parts have to synchronize themselves with the interface. Thus, HD-graph productions modeling synchronization can be used here, too.

Example 6 (Communication between developers). Synchronous communication between two developers can be described by an "interface"-graph and its connections to the "developer"-graphs. In figure 5 HD-graph production *sync new revision* is shown where a new revision of a document created by one developer is established in the interface and also in the current state of the other developer.

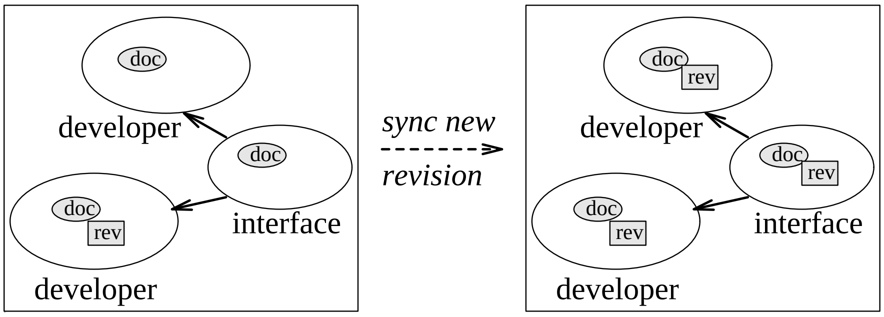

Fig. 5. Synchronous releasing of a new revision

The application of HD-graph production *sync new revision* to the HD-graph in figure 2 leads for example to the slightly modified HD-graph where new "rev"-nodes are established in the "interface"- and in the right "developer"-graphs. Additionally, the new "rev"-node in the "interface"-graph is mapped to its pendants in the "developer"-graph.

An asynchronous releasing of a new revision may be modeled by HD-graph production *async new revision* in figure 6. This action has not to be synchronized with actions of other developers. "New revision" just means the introduction of a new revision in the interface which has a link to the original revision where the developer worked on.

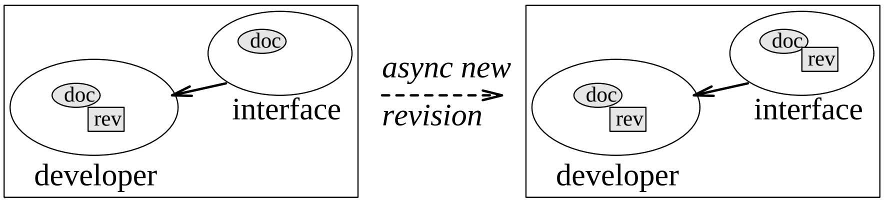

Fig. 6. Asynchronous releasing of a new revision

HD-graph production *async new revision* cannot be applied to the HD-graph in figure 2 since a new "rev"-node in the "interface"-graph cannot be mapped totally into the other "developer"-graph. But such a partial mapping of the "interface"-graph into the "developer"-graph should be possible for asynchronous communication. This can be achieved by a pair of total mappings of an additional "interface"-graph into the original "interface"- and the "developer"-graph. This additional interface graph would contain those objects and relations which should be mapped in any case, i.e. cannot be treated asynchronously. Applying an HD-graph production similar to *change connection* to the HD-graph in figure 2 an HD-graph as indicated in figure 7 can be achieved. This HD-graph can be transformed with HD-graph production *async new revision* yielding the HD-graph in figure 7 with the difference that a new "rev"-node is established in the left "interface"-graph. Additionally, this new node is mapped to the corresponding old "rev"-node in the left "developer"-graph.

2.5 Distributed Systems

Starting with some network topology and local initializations the initial state of a distributed system can be described by an HD-graph. Local actions, network activities, some kind of communicating actions and, moreover, also mixtures of these actions are allowed as state transitions. The whole distributed system is described by a so-called *HD-graph grammar* consisting of the starting HD-graph and all HD-graph productions modeling distributed actions.

Example 7 (Distributed software development system). A distributed software development system can be described by an HD-graph grammar consisting of

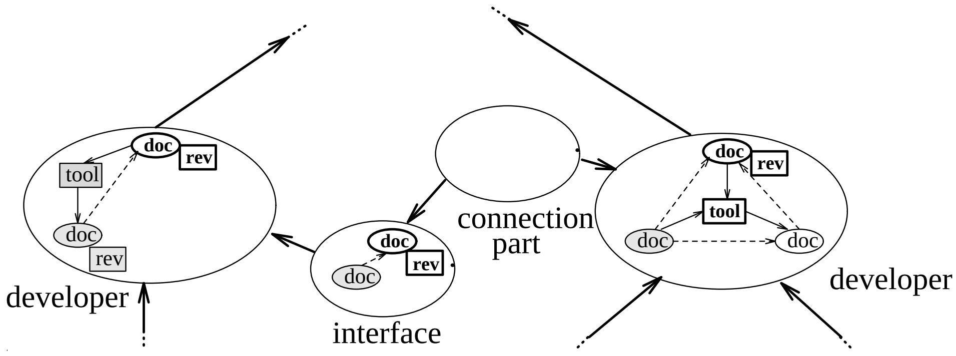

Fig. 7. Section of development graph with changed developer connection

a start graph which could be the HD-graph depicted in figure 2 and a set of HD-graph productions modeling distributed development steps also allowed to be performed concurrently. This set may contain the HD-graph productions *new tool*, *change connection*, *sync new revision*, *async new revision*, etc. As an example, a distributed development step may be the parallel application of *new tool* and *async new revision* to the state described in figure 2.

3 Formal Description of Distribution Concepts

The distribution concepts presented in the previous section are formally described in the framework of the double-pushout approach to graph transformation. This approach is comprehensively described in [4] for directed and labeled graphs. The double-pushout approach has been generalized to so-called high-level replacement systems in [6] where the main results of the graph grammar theory are abstracted to arbitrary objects and morphisms. In the following we show that HD-graph transformation fits into the framework of HLR-systems.

Definition 1. Let **GRAPH** be the category of labeled graphs and (total) graph morphisms which are label preserving. Given a graph G [3]of **GRAPH** which is called **network graph** a functor $\hat{G} : G \to$ **GRAPH** is called **hierarchically distributed graph (HD-graph)**[4].

A local graph is denoted $\hat{G}(i)$ for a network node $i \in G^N$. Given a network edge $e \in G^E$ the local graph $\hat{G}(s(e))$ is an interface graph or source graph, $\hat{G}(t(e))$ is called target graph and $\hat{G}(e)$ is a total graph morphism.

Example 8 (HD-graph). In figure 2 an HD-graph $\hat{G} : G \to$ **GRAPH** is shown. Graph G consists of all big ellipses with solidly drawn edges in between. Let d_1 be the left "developer"-node, its local graph which is drawn inside the ellipse is

[3] G^N describes the set of nodes and G^E the set of edges of G. s and t are the source and target mappings between G^E and G^N.

[4] Graph G and its induced small category are identified.

called $\hat{G}(d_1)$. Correspondingly the local graph of node b labeled by "big project" is called $\hat{G}(b)$. The network edge e with source node d_1 and target node b is equipped with a graph morphism $\hat{G}(e) : \hat{G}(d_1) \to \hat{G}(b)$ coded into the fill and frame styles of local nodes, i.e. nodes which are mapped to each other are filled and framed alike. Local edges are mapped in a structure compatible way.

Definition 2. Given two HD-graphs $\hat{G}$ and $\hat{H}$ an **HD-graph morphism** $\hat{f} = (\eta, f) : \hat{G} \to \hat{H}$ is a natural transformation $\eta : \hat{G} \to \hat{H} \circ f = (\hat{f}_i)_{i \in G^N}$ in **GRAPH** with $f : G \to H$ being a graph morphism of **GRAPH**, called **network morphism**.

If f is injective $\hat{f}$ is called **n-injective**. If moreover, all $\hat{f}_i$ with $i \in G^N$ are injective, $\hat{f}$ is called **injective**, too.

An HD-graph morphism $\hat{f}$ assigns to each node $i \in G^N$ a graph morphism $\hat{f}_i : \hat{G}(i) \to \hat{H} \circ f(i)$, called **local graph morphism** such that $\forall e \in G^E$ where $s(e) = i$ and $t(e) = j$ the diagram in figure 8 commutes. More concretely, given two local graphs $\hat{G}(i)$ and $\hat{G}(j)$ which are connected by $\hat{G}(e)$ and these graphs are mapped by local graph morphisms $\hat{f}_i$ and $\hat{f}_j$ to graphs $\hat{H} \circ f(i)$ and $\hat{H} \circ f(j)$ the connection $\hat{G}(e)$ has to be adapted leading to $\hat{H} \circ f(e)$.

$$
\begin{tikzcd}
i \arrow[d, "e"'] & \hat{G}(i) \arrow[r, "\hat{f}_i"] \arrow[d, "\hat{G}(e)"'] \arrow[dr, phantom, "="] & \hat{H} \circ f(i) \arrow[d, "\hat{H} \circ f(e)"] \\
j & \hat{G}(j) \arrow[r, "\hat{f}_j"] & \hat{H} \circ f(j)
\end{tikzcd}
$$

Fig. 8. HD-graph morphism

Example 9 (HD-graph morphism). HD-graph morphisms are shown in figures 3, 5 and 6. In all figures the network morphisms are not explicitly shown but given by the layout of the source and target graphs. Nodes and edges at corresponding places are mapped to each others.

Definition 3. The **composition** $\hat{f} = (\eta_f, f) : \hat{G} \to \hat{H}$ **of two HD-graph morphisms** $\hat{g} = (\eta_g, g) : \hat{G} \to \hat{K}$ and $\hat{h} = (\eta_h, h) : \hat{K} \to \hat{H}$ is defined by $f = h \circ g$ and $\eta_f = \eta_h \circ \eta_g : \hat{G} \to \hat{K} \circ g \to \hat{H} \circ h \circ g$.

Proposition 4. *All HD-graphs and HD-graph morphisms as defined above form a category* **DISTR(GRAPH)**.

In the following we do not consider the existence of pushouts in the category **DISTR(GRAPH)** in general, but concentrate on those which can be constructed component wise. Therefore, we need the following pushout conditions.

Definition 5. Two n-injective HD-graph morphisms $\hat{a} : \hat{A} \to \hat{C}$ and $\hat{b} : \hat{A} \to \hat{B}$ satisfy the **pushout conditions** (1) and (2) if (1) $\forall e \in C^E - a(A^E) : \exists y \in A^N$ with $a(y) = s(e)$ implies $\hat{b}_y$ is bijective and (2) $\forall e \in B^E - b(A^E) : \exists y \in A^N$ with $b(y) = s(e)$ implies $\hat{a}_y$ is bijective.

Proposition 6. *Given two n-injective HD-graph morphisms $\hat{a} : \hat{A} \to \hat{C}$ and $\hat{b} : \hat{A} \to \hat{B}$ which satisfy the pushout conditions (1) and (2) the pushout of $\hat{a}$ and $\hat{b}$ in* **DISTR(GRAPH)** *exists and can be constructed componentwise.*

Proof sketch: Let D with $c : C \to D$ and $d : B \to D$ be the pushout of a and b. (For pushouts in the category **GRAPH** compare [4], etc.) The pushout graph $\hat{D} : D \to$ **GRAPH** is constructed in the following way:[5]

$$\hat{D}(x) := \begin{cases} PO(\hat{a}_y, \hat{b}_y) = PO_x & \text{, if } \exists y \in A^N \text{ with } c \circ a(y) = x \\ \hat{C}(z) & \text{, if } \exists z \in C^N - a(A^N) \text{ with } c(z) = x \\ \hat{B}(v) & \text{, if } \exists v \in B^N - b(A^N) \text{ with } d(v) = x \\ IND(PO_{s(x)}, PO_{t(x)}) & \text{, if } \exists e \in A^E \text{ with } c \circ a(e) = x \\ \hat{c}_{t(x)} \circ \hat{C}(e) \circ \hat{c}^{-1}_{s(x)} & \text{, if } \exists e \in C^E - a(A^E) \text{ with } c(e) = x \\ \hat{d}_{t(x)} \circ \hat{B}(e) \circ \hat{d}^{-1}_{s(x)} & \text{, if } \exists e \in B^E - b(A^E) \text{ with } d(e) = x \end{cases}$$

where $IND(PO_{s(x)}, PO_{t(x)})$ is the induced morphism from pushout graph $\hat{D}(s(x))$ to pushout graph $\hat{D}(t(x))$ with $\hat{a}_{t(e)} \circ \hat{A}(e) = \hat{C}(a(e)) \circ \hat{a}_{s(e)}$ and $\hat{b}_{t(e)} \circ \hat{A}(e) = \hat{B}(b(e)) \circ \hat{b}_{s(e)}$. Pushout morphisms $\hat{c} : \hat{C} \to \hat{D}$ and $\hat{d} : \hat{B} \to \hat{D}$ are defined as follows:

$$\hat{c}_x := \begin{cases} id_{\hat{C}(x)} & \text{, if } x \in C^N - a(A^N) \\ \text{PO-morphism of } PO_{c(x)} & \text{, otherwise} \end{cases}$$

$$\hat{d}_x := \begin{cases} id_{\hat{B}(x)} & \text{, if } x \in B^N - b(A^N) \\ \text{PO-morphism of } PO_{d(x)} & \text{, otherwise} \end{cases}$$

It is straightforward to show that $\hat{D}$ is an HD-graph (using the pushout conditions) and $\hat{c}$ as well as $\hat{d}$ are HD-graph morphisms. Now, the pushout properties have to be shown. Commutativity follows directly from the construction. For the universal property the induced morphism $\hat{u}$ consists of the induced morphisms for the underlying pushouts in **GRAPH** where they exist. Otherwise it is defined suitable to the comparing HD-graph morphisms. Well-definedness of $\hat{u}$ follows from the universal property of local pushouts and the pushout conditions. The universal property can be obtained directly from the definition of $\hat{u}$. □

A pushout as described above can be constructed in the following steps. First the pushout on network morphisms is created. For all network nodes in A which have images in B and C the pushout on their local graph morphisms is constructed. All other local graphs of $\hat{B}$ and $\hat{C}$ are carried over to $\hat{D}$ unchanged. A network edge in D which has a preimage in A is equipped with the induced

[5] $PO(\hat{a}_y, \hat{b}_y)$ is the pushout graph of the pushout of $\hat{a}_y$ and $\hat{b}_y$

morphism between its source and target pushout graph. All other local graph morphisms of $\hat{B}$ and $\hat{C}$ are adapted to their new target graphs. The source graphs have to be mapped structure equivalent according to the pushout conditions.

Definition 7. An **HD-graph production** $\hat{p} = (\hat{L} \xleftarrow{\hat{l}} \hat{I} \xrightarrow{\hat{r}} \hat{R})$ consists of HD-graphs $\hat{L}$, $\hat{R}$ and $\hat{I}$, called left- and right-hand side and intermediate HD-graphs and two n-injective HD-graph morphisms $\hat{l}$ and $\hat{r}$.

If all $\hat{p}_x = (\hat{L}(l(x)) \xleftarrow{\hat{l}_x} \hat{I}(x) \xrightarrow{\hat{r}_x} \hat{R}(r(x)))$ for $x \in I^N$ are left-injective, i.e. all $\hat{l}_x$ are injective, $\hat{p}$ is called *left-injective*, too.

Example 10 (HD-graph production). The HD-graph morphisms in figures 3, 4, 5 and 6 are all left injective HD-graph productions if they are interpreted in the following way. Except figure 4 the figures mentioned show the right morphism $\hat{I} \xrightarrow{\hat{r}} \hat{R}$ of an HD-graph production. For each of these examples the left morphism $\hat{I} \xrightarrow{\hat{l}} \hat{L}$ is the identity $id_{\hat{I}}$. As usually in graph transformation this means that nothing is deleted. In figure 4 HD-graphs $\hat{L}$ and $\hat{R}$ are given explicitly. Graph I can be constructed by all nodes and edges drawn alike, i.e. the "developer"- and the "big project"-nodes. All local graph morphisms of this example are identities. All local and HD-graph morphisms shown or constructed are injective.

Definition 8. Given an HD-graph production $\hat{p} = (\hat{L} \xleftarrow{\hat{l}} \hat{I} \xrightarrow{\hat{r}} \hat{R})$ an n-injective HD-graph morphism $\hat{m} : \hat{L} \to \hat{G}$ is called **HD-match of** $\hat{p}$ if the following **distributed gluing condition** is satisfied:

1. m satisfies the *gluing condition* wrt. p
2. $\forall x \in I^N : \hat{m}_{l(x)}$ satisfies the *gluing condition* wrt. $\hat{p}_x = (\hat{L}(l(x)) \xleftarrow{\hat{l}_x} \hat{I}_x \xrightarrow{\hat{r}_x} \hat{R}(r(x)))$ (compare [4], etc.)
3. *connection condition*: (a) $\forall e \in G^E - m((L^E) - l(I^E))$ with $t(e) = m \circ l(y)$ for $y \in I^N$:

   $$\hat{G}(e)(\hat{G}(s(e)) - \hat{m}_{l(x)}(\hat{L}(l(x)) - \hat{l}_x(\hat{I}(x)))) \subseteq \hat{G}(t(e)) - \hat{m}_{l(y)}(\hat{L}(l(y)) - \hat{l}_y(\hat{I}(y)))$$

   where $s(e) = m \circ l(x)$, $x \in I^N$ or $\hat{L}(l(x))$, $\hat{I}(x)$, $\hat{m}_{l(x)}$ and $\hat{l}_x$ are empty
   (b) $\forall e \in G^E - m(L^E) : \exists y \in I^N$ with $m \circ l(y) = s(e)$ implies $\hat{l}_y$ and $\hat{r}_y$ are bijective.
4. *network condition*: (a) $\forall x \in L^N - l(I^N)$ and $\forall x \in l(I^N)$ where $\exists e \in L^E - l(I^E)$ with $s(e) = x : \hat{m}_x$ is bijective
   (b) $\forall e \in R^E - r(I^E) : \exists y \in I^N$ with $r(y) = s(e)$ implies $\hat{m}_{l(y)}$ is bijective.

The connection condition means the following: (a) An action on a target graph is not permitted to delete local items where some of their pendants in connected source graphs are not deleted. (b) A local action on a source graph is not allowed to extend this graph since new items would not have images in connected target graphs. Furthermore, such an action is not allowed to delete local items without deleting references to items in target graphs. Lastly, local objects in a source graph are not allowed to be glued together since this is not reflected in the connected target graphs.

The network condition is interpreted as follows: (a) The deletion of network nodes can be done if its local state graph is deleted as a whole in the same production, i.e. if the current local state corresponds with that in the production. If a network edge should be deleted its source graph has to correspond bijectively with that of the production. (b) The other way around, if a new connection from an existing source graph should be established this graph has to be structural equivalent with its correspondent given in the production.

The conditions above are shortly discussed within examples 4 and 6.

Definition 9. Given an HD-graph production $\hat{p} = (\hat{L} \xleftarrow{\hat{l}} \hat{I} \xrightarrow{\hat{r}} \hat{R})$ and an HD-match $\hat{m} : \hat{L} \to \hat{G}$ the following HD-graph $\hat{C} : C \to \mathbf{GRAPH}$ is called **HD-context graph** of $\hat{p}$ and $\hat{m}$. Let $C = G - m(L - l(I))$ be the context graph of p and m in **GRAPH**.

$$\hat{C}(x) := \begin{cases} \hat{G}(x) - \hat{m}_{l(y)}(\hat{L}(l(y)) - \hat{l}_y(\hat{I}(y))) & \text{, if } \exists y \in I^N \text{ with } m \circ l(y) = x \\ \hat{G}(x) & \text{, if } x \in G^N - m(L^N) \\ \hat{g}^{-1}_{t(x)} \circ \hat{G}(x) \circ \hat{g}_{s(x)} & \text{, if } x \in G^E - m(L^E - l(I^E)) \end{cases}$$

where HD-graph morphisms $\hat{g} : \hat{C} \to \hat{G}$ and $\hat{c} : \hat{I} \to \hat{C}$ are defined as follows. Let $c = m \circ l$ and $g = id_{G|_C}$ HD-graph morphism $\hat{c}$ is defined by $\hat{c}_y = \hat{m}_{l(y)} \circ \hat{l}_y$, $\forall y \in I^N$.

$$\hat{g}_x := \begin{cases} id_{\hat{G}(x)/\hat{C}(x)} & \text{, if } x \in m(l(I^N)) \\ id_{\hat{G}(x)} & \text{, otherwise }, \forall x \in C^N \end{cases}$$

Remark: $\forall y \in I^N$ the PO-complement $(\hat{C}(c(y)), \hat{g}_{c(y)}, \hat{c}_y)$ of $\hat{m}_{l(y)}$ and $\hat{l}_y$ in **GRAPH** is constructed.

Proposition 10 (Applicability of HD-graph productions). *Given an HD-graph production $\hat{p} = (\hat{L} \xleftarrow{\hat{l}} \hat{I} \xrightarrow{\hat{r}} \hat{R})$ and an HD-match $\hat{m} : \hat{L} \to \hat{G}$ there are an HD-context graph $\hat{C}$ as well as HD-graph morphisms $\hat{g}$ and $\hat{c}$ as defined above such that $(\hat{G}, \hat{m}, \hat{g})$ is the pushout of $\hat{c}$ and $\hat{l}$ in* **DISTR(GRAPH)***. Furthermore, the pushout of $\hat{c}$ and $\hat{r}$ exists.*

Proof sketch: $\hat{C}$ is an HD-graph since the gluing condition for all local transformations and the connection condition (a) are satisfied. Clearly, $\hat{g}$ and $\hat{c}$ are HD-graph morphisms. Next we construct the pushout of $\hat{l}$ and $\hat{c}$ (which is possible since the distributed gluing conditions contain the gluing condition for m and p and the pushout conditions which are part of network condition (a) and connection condition (b)) and have to show that the resulting pushout graph $\hat{X}$ is isomorphic to $\hat{G}$. According to pushout properties there is an HD-graph morphism $\hat{u} : \hat{X} \to \hat{G}$. Vice versa, a suitable HD-graph morphism $\hat{w} : \hat{G} \to \hat{X}$ can be defined using the local induced morphisms. $\hat{w}$ is well defined according to pushout complement properties in category **GRAPH** and network condition (a). □

Definition 11. Given an HD-graph production $\hat{p} = (\hat{L} \xleftarrow{\hat{l}} \hat{I} \xrightarrow{\hat{r}} \hat{R})$ and an HD-graph match $\hat{m} : \hat{L} \rightarrow \hat{G}$ an **HD-graph transformation** $\hat{G} \Longrightarrow_{hd} \hat{H}$ via $\hat{p}$ and $\hat{m}$, short $\hat{G} \overset{\hat{p},\hat{m}}{\Longrightarrow}_{hd} \hat{H}$, from an HD-graph $\hat{G}$ to an HD-graph $\hat{H}$ is given by the two pushout diagrams (1) and (2) in the category **DISTR(GRAPH)** shown in figure 9.

$$\begin{tikzcd}
\hat{L} \arrow[d, "\hat{m}"'] \arrow[dr, phantom, "(1)"] & \hat{I} \arrow[l, "\hat{l}"'] \arrow[r, "\hat{r}"] \arrow[d, "\hat{c}"'] \arrow[dr, phantom, "(2)"] & \hat{R} \arrow[d] \\
\hat{G} & \hat{C} \arrow[l, "\hat{g}"'] \arrow[r] & \hat{H}
\end{tikzcd}$$

Fig. 9. HD-graph transformation

An **HD-graph transformation sequence** $\hat{G} \overset{P}{\Longrightarrow}{}^{*}_{hd} \hat{H}$ is a sequence of $n \geq 0$ HD-graph transformations $\hat{G} = \hat{G}_0 \Longrightarrow_{hd} \hat{G}_1 \Longrightarrow_{hd} \ldots \Longrightarrow_{hd} \hat{G}_n = \hat{H}$ via HD-graph productions of a set P. $\hat{H}$ is also called **HD-derivable** from $\hat{G}$ by P.

Proposition 12 (Uniqueness of HD-graph transformation). *Given a left-injective HD-graph production $\hat{p} = (\hat{L} \xleftarrow{\hat{l}} \hat{I} \xrightarrow{\hat{r}} \hat{R})$ and an HD-match $\hat{m} : \hat{L} \rightarrow \hat{G}$ the HD-graph transformation $G \overset{\hat{p},\hat{m}}{\Longrightarrow}_{hd} H$ is unique up to isomorphism.*

Proof sketch: First, we have to show that $\hat{C}$ together with $\hat{c}$ and $\hat{g}$ as defined in 9 are unique. This is done similar to the proof of uniqueness of pushout complements in **GRAPH**. Together with the fact that pushouts are unique up to isomorphisms we can state that $G \overset{\hat{p},\hat{m}}{\Longrightarrow}_{hd} H$ is unique up to isomorphism. □

Definition 13. Given two HD-graph productions $\hat{p}_1 = (\hat{L}_1 \xleftarrow{\hat{l}_1} \hat{I}_1 \xrightarrow{\hat{r}_1} \hat{R}_1)$ and $\hat{p}_2 = (\hat{L}_2 \xleftarrow{\hat{l}_2} \hat{I}_2 \xrightarrow{\hat{r}_2} \hat{R}_2)$ the HD-graph production $\hat{p}_1 + \hat{p}_2 = (\hat{L}_1 + \hat{L}_2 \xrightarrow{\hat{l}_1+\hat{l}_2} \hat{I}_1 + \hat{I}_2 \xrightarrow{\hat{r}_1+\hat{r}_2} \hat{R}_1 + \hat{R}_2)$ defined by disjoint union of graphs and morphisms is called **parallel (HD-graph) production** of $\hat{p}_1$ and $\hat{p}_2$.

Definition 14. An **HD-graph grammar** $HDGG = (\hat{S}, P)$ is given by an HD-graph $\hat{S}$, called the **start graph**, and a set of HD-graph productions P.[6]

Let P^+ be the smallest extension of the set P including all parallel HD-graph productions $\hat{p}_1 + \hat{p}_2$ for $\hat{p}_1, \hat{p}_2 \in P^+$. The **observation set** $O(HDGG) = \{\hat{G} \mid \hat{S} \overset{P^+}{\Longrightarrow}{}^{*}_{hd} \hat{G}\}$ of $HDGG$ consists of all HD-graphs $\hat{G}$ HD-derivable from $\hat{S}$ by P^+.

[6] Since we are not interested in classical language aspects here, we do not distinguish terminal and nonterminal graphs.

Since all matches have to be n-injective parallel productions can be applied to different local graphs only, i.e. the matches of the original HD-graph productions are not allowed to overlap. Thus, it seems to be possible to show that given an HD-graph transformation $\hat{G} \overset{\hat{p}_1+\hat{p}_2}{\Longrightarrow}_{hd} \hat{H}$ where $\hat{p}_1$ and $\hat{p}_2$are applied in parallel there is a corresponding HD-graph transformation sequence $\hat{G} \overset{\hat{p}_1}{\Longrightarrow}_{hd} \hat{X} \overset{\hat{p}_2}{\Longrightarrow}_{hd} \hat{H}$ applying the original HD-graph productions $\hat{p}_1$ and $\hat{p}_2$ sequentially in some order.(See also the analysis construction of the parallelism theorem in [6].) This means that the observation set $O(HDGG)$ is "closed under parallelism", i.e. there is not a graph in $O(HDGG)$ which can only be derived by applying at least one parallel production.

4 Conclusion and Open Problems

In this paper hierarchically distributed graph transformation has been introduced to offer the possibility of modeling the main aspects of open distributed systems, such as distributed software development. This approach allows arbitrary distribution topologies, especially hierarchical ones, which can be handled dynamically. The internal structures of local parts can be modeled in a graphical way, too, and consistent copies are indicated by graph morphisms between local graphs. By means of graph transformation the topological structure as well as local object structures can be manipulated in an integrated way, i.e. graph transformation is performed on both levels of description.

It might be useful to increase the number of description levels to capture additional aspects of system modeling. For example in our running example of distributed software development, the documents, revisions and tools described by local nodes could be refined to graphs again showing their internal structures. Moreover, it might be useful to structure the local parts according to access rights. This would yield some kind of encapsulated components with well-defined interfaces. Such a concept for modular systems emphasizing encapsulation is described in [16]. It builds up on HD-graph transformation.

On the theoretical side HD-graph grammars are shown to fit into the framework of HLR-systems. Further results for HD-graph transformations such as independence or embedding results can be easily achieved if so-called HLR-conditions can be proven. These condition are mainly based on the existence of pushouts in the given category. In this paper we concentrated on that kind of pushouts in **DISTR(GRAPH)** being built component wise because it reflects best the distribution of local actions. The investigation of other kinds of pushouts and the cocompleteness of **DISTR(GRAPH)** altogether will be further work.

Allowing pushouts on HD-graph morphisms which need not to be n-injective the parallel execution of actions on one local graph can be modeled. This feature has to be used to simulate the special operations SPLIT and JOIN of distributed graph transformation in the algebraic approach by HD-graph transformation ([5]).

Acknowledgment: I thank Annika Wagner and the referees for their valuable

comments on this paper.

References

1. J. Adamek, H. Herrlich, and G. Strecker. *Abstract and Concerte Categories.* Series in Pure and Applied Mathematics. John Wiley and Sons, 1990.
2. I. Classen, M. Löwe, S. Wasserroth, and J. Wortmann. Static and dynamic semantics of entity-relationship models based on algebraic methods. to appear in proc. IFIP-Congress and GI-Fachgespräche, Hamburg, 1994.
3. P. Degano and U. Montanari. A model of distributed systems based on graph rewriting. *Journal of the ACM*, 34(2):411–449, 1987.
4. H. Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *1st Graph Grammar Workshop, Lecture Notes in Computer Science 73*, pages 1–69. Springer Verlag, 1979.
5. H. Ehrig, P. Boehm, U. Hummert, and M. Löwe. Distributed parallelism of graph transformation. In *13th Int. Workshop on Graph Theoretic Concepts in Computer Science, LNCS 314*, pages 1–19, Berlin, 1988. Springer Verlag.
6. H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. From graph grammars to High Level Replacement Systems. In Ehrig et al. [7], pages 269–291. Lecture Notes in Computer Science 532.
7. H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors. *4th International Workshop on Graph Grammars and Their Application to Computer Science.* Springer Verlag, 1991. Lecture Notes in Computer Science 532.
8. H. Ehrig and M. Löwe. Parallel and distributed derivations in the single pushout approach. *TCS*, 109:123 – 143, 1993.
9. S.M. Kaplan, J.P. Loyall, and S.K. Goering. Specifying concurrent languages and systems with Δ-grammars. In Ehrig et al. [7], pages 475–489. Lecture Notes in Computer Science 532.
10. D. Kips and G. Heidenreich. Project flow graphs – a meta-model to support quality assurance in software-engineering. to appear in proc. of IEPM'95, 1995.
11. M. Korff. Single pushout transformations of equationally defined graph structures with applications to actor systems. In *Proc. Graph Grammar Workshop Dagstuhl 93*, pages 234–247. Springer Verlag, 1994. Lecture Notes in Computer Science 776.
12. P. Pepper and M. Wirsing. KORSO: A methodology for the development of correct software. to be published in LNCS, 1995.
13. G. Schied. *Über Graphgrammatiken, eine Spezifikationsmethode für Programmiersprachen und verteilte Regelsysteme.* Arbeitsberichte des Institus für mathematische Maschinen und Datenverarbeitung (Informatik), University of Erlangen, 1992.
14. H.-J. Schneider. On categorical graph grammars integrating structural transformation and operations on labels. *TCS*, 109:257 – 274, 1993.
15. G. Taentzer. Towards synchronous and asynchronous graph transformations. accepted for special issue of Fundamenta Informaticae, 1995.
16. G. Taentzer and A. Schürr. DIEGO, another step towards a module concept for graph transformation systems. to appear in proc. of SEGRAGRA'95 " Graph Rewriting and Computation", published in Electronic Notes of TCS, 1995.