Towards Synchronous and Asynchronous Graph Transformations*

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

Synchronous and asynchronous graph transformations are discussed on the basis of distributed graph transformations. All types of graph transformations presented are formulated in the single-pushout approach to graph transformations and compared with each other. Some criteria are given to preserve or reestablish synchrony according to the definition proposed in this paper.

1 Introduction

Graphs and graph transformations have been used to model different kinds of parallel and distributed systems. (See, for example, the approaches of Degano and Montanari [3], Ehrig et.al. [5], Schneider [14], Goering [8], Kaplan et.al. [10], Janssens and Rozenberg [9], Ehrig and Löwe [7], Schied [13], Korff [11], etc.) In [5] and [7] the algebraic approach is used to model concepts of concurrency and distribution. In this framework graphs correspond to states of a system and graph transformations to operations or actions on these states. Distribution of graphs is modeled by splitting global state graphs into local ones together with common interface graphs. Local graphs are changed by local transformations running concurrently and usually asynchronously. Choosing two local transformations they can be synchronized via a transformation of their common interface graph. This means that interfaces may change during a transformation.

In this paper, the description of distributed transformations is based on the single-pushout approach to graph grammars and continues the work already started by Ehrig and Löwe in [7]. Basic notions of the single-pushout approach are reviewed in section 2 while a comprehensive presentation can be found in Löwe [12]. In [7] direct parallel transformations are considered where the application of two rules is either independent or asynchronous. Another kind of transformation considered there are amalgamated graph transformations describing some kind of synchronized parallel transformations. (For the amalgamation concept see Böhm, Fonio and Habel in [2].) Adding concepts of distribution, distributed graph transformations are formulated.

To get the possibility of formulating more complex distribution of graphs the kind distributed transformations is extended in this work. Global graphs are allowed to be

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

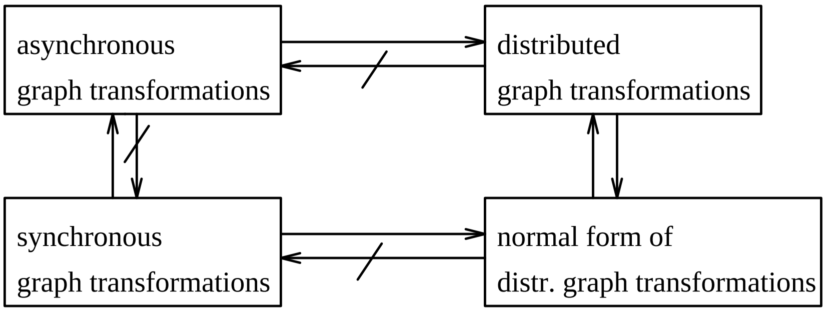

Figure 1: Overview over different kinds of graph transformations

distributed into arbitrary many local graphs where each two of them have a common interface.

Furthermore, distributed transformation sequences are considered where the restriction is dropped that the global graph has to be reestablished after each step. The kind of distributed transformations investigated in this paper extends the concepts of distribution as presented in [5]. Interface graphs may be changed during local transformations and some local transformations may synchronize while others are running asynchronously. This type of transformation seems to be convenient to model complex distributed actions.

The intention of this paper is to propose a definition for synchronous and asynchronous graph transformations which meets the intuition of these kinds of communication. Synchronous and asynchronous actions are usually distinguished along their kind of interaction. Synchronous actions are executed dependently of each other whereas asynchronous actions also allow independent behavior. Distributed graph transformations seem to be well suited to model communication and synchronization via shared objects which are equipped with a copy in each local part. Simultaneous updates of such objects would cover then the synchronous case. Asynchronous updates usually allow some delay of synchronizing actions. Modeling these ideas by distributed transformations, synchronous transformations would require direct local transformations of exactly one step while asynchronous transformations allow arbitrarily long local transformations which run completely independently.

On this basis, synchronous transformations are considered as a special case of asynchronous ones. Moreover, general distributed transformations contain asynchronous transformations because they allow arbitrary communication and may be incomplete, in general. Since communication is realized by copies of shared objects it is complete if all copies are updated.

Furthermore, criteria for rules and occurrences are given to check whether the transformation given is synchronous. For this aim a normal form for distributed transformations is introduced that allows direct distributed steps only.

An overview of all different kinds of graph transformations considered in this paper as well as their relations is shown in figure 1.

The paper is organized as follows: Section 2 reviews basic notions of the single-pushout approach to graph transformations as far as they are needed in this paper. In section 3 distributed as well as synchronous and asynchronous graph transformations are presented and demonstrated at several examples. In section 4, several properties of distributed and, especially, synchronous and asynchronous graph transformations are presented. A simplified graph transformation model of distributed software development is chosen as a running example explaining the concepts presented in this paper.

The reader is supposed to be familiar with the basic notions of category theory, especially with colimits as they are presented, for example, by Adamek, Herrlich and Strecker in [1]. All notions used are explained in the graph transformation framework.

2 Basics of the Single-Pushout Approach to Graph Transformations

This section provides the basic notions on the single-pushout approach to graph transformations as far as they are needed in this paper.

Definition 2.1 (Graphs and graph morphisms)

1. *Let C_E and C_N be fixed color alphabets for nodes and edges. A* graph $G = (G_E, G_N, s_G, t_G, m_{GE}, m_{GN})$ *consists of the sets of edges and nodes, G_E and G_N, mappings $s_G, t_G : G_E \to G_N$, called source resp. target map, and the coloring maps for arcs and nodes, $m_{GE} : G_E \to C_E$ and $m_{GN} : G_N \to C_N$.*

2. *Given two graphs G and G' a* partial (total) graph morphism $f : G \to G'$ *is a pair of partial (total) mappings $(f_E : G_E \to G'_E, f_N : G_N \to G'_N)$ such that f_E and f_N are compatible with source and target mappings and label preserving. Partial graph morphisms are often just called* graph morphisms *or* morphisms *for short. A partial graph morphism $f : G \to G'$ can also be described as a total graph morphism from some subgraph $G(f)$ of G to G'.*

Remarks: In the following figures partial morphisms are drawn by $\longrightarrow\!\!\!\!\!\!\!\circ\;\;\;$ if we want to stress that they are not total. The composition f of two partial morphisms g and h is defined component wise by $f_E = h_E \circ g_E$ and $f_N = h_N \circ g_N$. Graphs and partial (total) graph morphisms as defined above form a category $\mathbf{GRAPH^P}$($\mathbf{GRAPH}$). All following colimits are constructed in the category $\mathbf{GRAPH^P}$.

Definition 2.2 (Rules, occurrences and sequential graph transformations)

1. *A* rule $p : L \to R$ *is a partial graph morphism from the left hand side L to the right hand side R. The* empty rule *is the morphism $\emptyset : \emptyset \to \emptyset$ where $\emptyset$ denotes the empty graph.*

2. *An* occurrence *of a rule $p : L \to R$ in a graph G is a total graph morphism $m : L \to G$. Occurrences $\emptyset : \emptyset \to G$ of the empty rule are called* empty, *too.*

3. *Occurrence m is called* conflict-free *wrt. p if $m(x) = m(y)$ implies $x, y \in L(p)$ or $x, y \notin L(p)$.*

4. *Given rule p at occurrence m a* direct sequential graph transformation $G \overset{p,m}{\Longrightarrow}_s G'$ *consists of the pushout of p and m depicted in figure 2. Morphism $p^* : G \to G'$ is called transformation morphism.*

   The pair (p, m) is also called application pair. *A direct sequential transformation is called* empty transformation *if the empty rule is applied.*

$$\begin{tikzcd} L \arrow[r, "p"] \arrow[d, "m"'] & R \arrow[d, "m^*"] \\ G \arrow[r, "p^*"] & G' \end{tikzcd}$$

Figure 2: Sequential graph transformation

5. *A sequence of direct sequential graph transformations* $G \overset{(P,M)}{\Longrightarrow}_s G'$, *short a* sequential transformation, *consists of a sequence of* $n \geq 0$ *direct sequential graph transformations*

$$G = G_0 \overset{p_1,m_1}{\Longrightarrow}_s G_1 \overset{p_2,m_2}{\Longrightarrow}_s \ldots \overset{p_n,m_n}{\Longrightarrow}_s G_n = G'$$

where $P = \{p_i | 1 \leq i \leq n\}$ *and* $M = \{m_i | 1 \leq i \leq n\}$*. Transformation morphism* $t : G \to G'$ *is composed by transformation morphisms* $t_i : G_{i-1} \to G_i$ *for all* $1 \leq i \leq n$*. If we are not interested in* P *and* M *we just write* $G \overset{*}{\Longrightarrow}_s G'$*.*

Given p and m graph G' can be explicitly constructed by the following steps:

- Construction of the gluing graph: Let K be the largest subgraph of $L(p)$ such that for all nodes and edges $x \in K$ and $y \in L : p(x) = p(y)$ or $m(x) = m(y) \Longrightarrow y \in K$.
- Construction of definedness areas of m^* and p^*: Delete all nodes and edges from G that have preimages in L but not in K and all edges whose source or target nodes are deleted. Symmetrically, treat R. The remainders are $G(p^*)$ and $R(m^*)$.
- Gluing of graphs $G(p^*)$ and $R(m^*)$ along K: Now graph G' is constructed by the disjoint union of $G(p^*)$ and $R(m^*)$ where corresponding images of nodes and edges in K are identified with each other.

The above construction shows principal steps that have to be done for a direct sequential transformation. The exact construction is presented for example in [12] in the framework of graph structures. It is also achieved as a special case of star gluing of graphs with $n = 2$ in the following definition 3.1.

Proposition 2.3 (Conflict-free transformations) *Given a rule* $p : L \to R$ *and an occurrence* $m : L \to G$ *of a sequential transformation* $G \overset{p,m}{\Longrightarrow}_s G'$ *as defined in 2.2. Occurrence* m *is conflict-free wrt.* p *if and only if morphism* $m^* : R \to G'$ *is total.*

Proof: See [12].

□

Example 2.4 (Developing and testing software) In the following a small example is presented which describes developing and testing software by several programmers. The user gives a certain requirement specification for which the software is developed. It is implemented and tested by different persons. The implementation and the testing

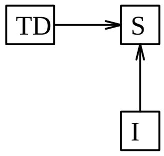

Figure 3: State of software development modeled by graph G

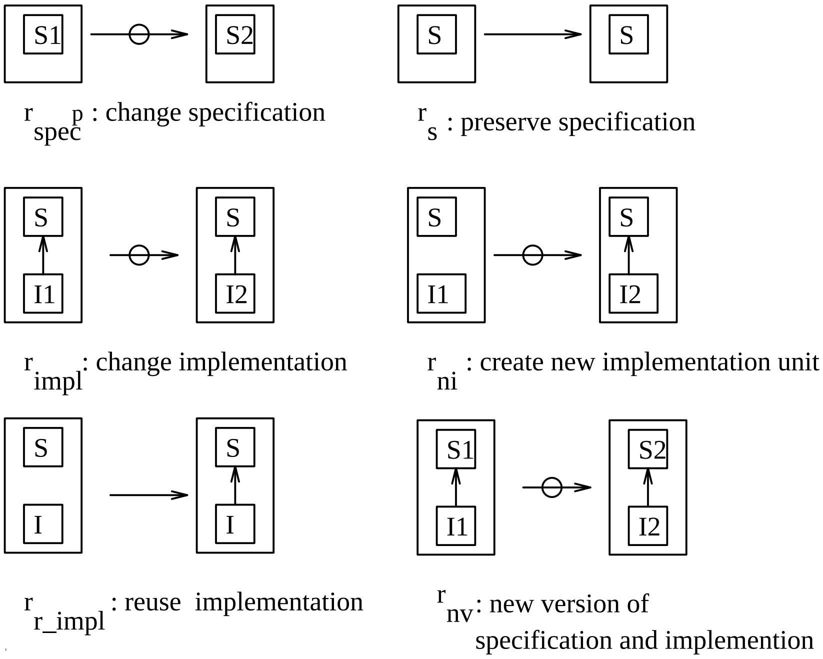

Figure 4: Rules for developing software

work is separated from each other for a better validation of the software. The implementing and testing persons know this specification and develop the corresponding program and testing data. The specification, the testing data, the implementation units and the testing results are represented by nodes labeled by “S”, “TD”, “I” and “TR”. Use relations between different parts are described by edges. One situation is represented by graph G given in figure 3 showing a snapshot of the development environment where testing data and an implementation have been created to a given specification.

In the environment of software development as described above the following actions can be performed:

- The user can change the specification.
- The implementing programmer can change or reuse a unit according to a given specification. Furthermore, she can create a new implementation unit.
- The testing programmer can change or reuse testing data according to the given specification. She can test a given implementation using testing data to the given specification and produces tests. Furthermore, she can install a new version consisting of the requirement specification and the implementation.

These actions are specified by the rules in figures 4 and 5. Morphisms map all nodes named alike to each other. Names consists of the label together with a unique number if there are several nodes with the same label in the context. Edges are mapped according to the mapping of their source and target nodes.

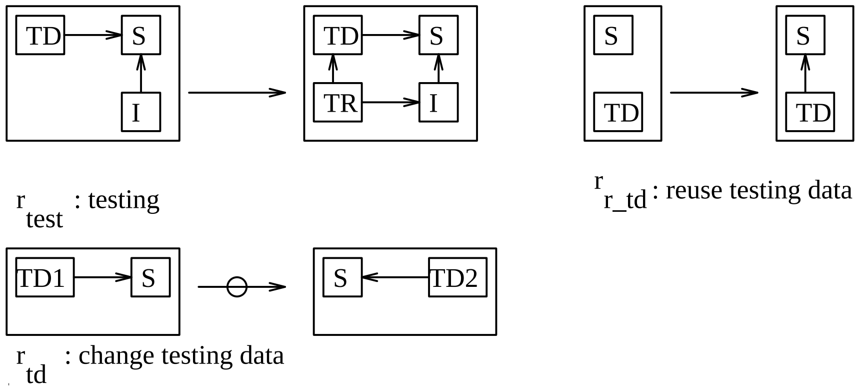

Figure 5: Rules for testing software

3 Distributed Graph Transformations

Building up on the basic concepts of single-pushout graph transformation reviewed in the previous section concepts of distribution are presented now. Distribution of graphs is modeled by splitting a global graph into local ones together with common interface graphs. A splitting of global graphs into exactly two local graphs with one common interface is described in [5] for the double-pushout approach and in [7] for the single-pushout approach to graph transformation. Joining the local graphs along their interface graph the global graph is achieved by gluing of graphs. This gluing can be characterized by a special colimit construction, i.e. a pushout. Gluing arbitrary many local graphs at common interfaces the star gluing construction described below can be used.

In the following, a global graph as well as a splitting into local graphs is called *distributed graph*. A global graph describes the special case that the state is undistributed.

Distributed graphs are transformed by distributed steps. Moreover, they can be split or joined to change between global and local graphs. In the distributed steps global as well as local graphs are transformed by sequential transformations. These transformations usually run concurrently and independently of each others. Synchronous as well as asynchronous communication is modeled by special distributed steps, so-called synchronous and asynchronous steps.

Distributed graph transformation sequences are regarded as a sequence of split, join and distributed steps. If all distributed steps are synchronous we talk about *synchronous graph transformations*. Similarly, if all distributed steps are asynchronous this kind of transformation is called *asynchronous*.

If there is an arbitrary number of graphs G_i, $i \geq 1$ where each two may have their own interface just partially embedded in the corresponding graphs the following construction shows the gluing of all the graphs G_i along their interfaces.

Definition 3.1 (Star gluing of graphs) *A* graph morphism star $MS = (G_i \overset{g_{ij}}{\leftarrow} G_{ij} \overset{g_{ji}}{\rightarrow} G_j)_{1 \leq i < j \leq n}$ *consists of graphs* G_i, G_j *and* G_{ij} *and (partial) graph morphisms* $g_{ij} : G_{ij} \rightarrow G_i$ *and* $g_{ji} : G_{ij} \rightarrow G_j$ *for all* $1 \leq i < j \leq n$.

Let $\bar{I}_G$ *be the equivalence relation generated by relation* $I_G = \{(\hat{g}_{ij}(x), \hat{g}_{ji}(x)) | x \in G_{ij}, 1 \leq i < j \leq n\}$ *where* $\hat{g}_{kl} : G_{kl} \rightarrow G_k \cup \{\perp\}$ *with*

$$\hat{g}_{kl}(x) = \begin{cases} g_{kl}(x), & \textit{if defined} \\ \perp, & \textit{otherwise} \end{cases}$$

The gluing graph G *of* MS *is defined by the largest subgraph* G *of* $(\biguplus_{1\leq i\leq n} G_i)_{/\bar{I}_G} - [\perp]$*, the quotient set of the disjoint union of all* G_i *except the class* $[\perp]$*. Partial functions* $g_i : G_i \to G$ *for all* $1 \leq i \leq n$ *send each element* x_i *of* G_i *to its equivalence class in* G*. If the class is* $[\perp]$ x_i *is not mapped.*

Remarks: If all morphisms of MS are total MS and the star gluing are called total. If $n = 1$ $MS = (G)$ consists of exactly one graph G. The gluing graph of MS is G again. Thus, we do not distinguish graphs and morphism stars of size $n = 1$ in the following.

Example 3.2 (Star gluing of graphs) Different examples of star gluing of graphs are given within the graph model of distributed software development. The graphs in the following examples are distinguished by small letters which can be translated into corresponding indices as used in definition 3.1.

In figure 9 at the end of this section the total gluing of two graphs G_i and G_t via interface graph I_{ti} is shown which yields graph G in figure 3 where node "S" corresponds to "S1" and "I" corresponds to "I1". This kind of gluing can be characterized as a pushout. Its construction is given after definition 2.2.

In figure 6 a morphism star of three graphs and corresponding interfaces is shown. The result graph of this total star gluing is graph G in figure 3. In this example nodes S1 in G_t and G_u are identified to S in G although their common interface graph I_{tu} is empty but they are transitively related via G_i and the remaining interfaces.

A star gluing where not all morphisms are total is shown in figure 11. There, morphism $I'_{ti} \to G'_i$ is not total such that node S1 of G'_t and I'_{ti} and associated edges are not in the gluing graph G'.

The star gluing of graphs can be characterized by a special colimit, i.e. the pushout-star. Its characteristics are given within the following proof.

Proposition 3.3 (Pushout-stars in $\mathbf{GRAPH^P}$) *Given a morphism star* $MS = (G_i \overset{g_{ij}}{\leftarrow} G_{ij} \overset{g_{ji}}{\rightarrow} G_j)_{1\leq i<j\leq n}$ *as in definition 3.1 a graph* G *and morphisms* $g_i : G_i \to G$ *for all* $1 \leq i \leq n$ *as they are constructed in 3.1 is a pushout-star in* $\mathbf{GRAPH^P}$*.*

Proof: For pushout-stars two properties have to be shown: (1) the commutativity $g_i \circ g_{ij} = g_j \circ g_{ji}$, $\forall 1 \leq i < j \leq n$, and (2) for each set of morphisms $g'_i : G_i \to G'$, $1 \leq i \leq n$, with $g'_i \circ g_{ij} = g'_j \circ g_{ji}$, there is a unique morphism $u : G \to G'$, called the *induced morphism*, such that $u \circ g_i = g'_i$, $\forall 1 \leq i \leq n$.

Due to definition 3.1, $g_i \circ g_{ij}(x) = g_j \circ g_{ji}(x)$, for all $x \in G_{ij}$ that are mapped via g_{ij} and g_{ji}. (See also [6] and [4].) For all other $x \in G_{ij}$ there is a pair $(g_{ij}(x), \perp)$ or $(\perp, g_{ji}(x))$ in I_G such that it is sent to the equivalence class of $\perp$. Hence, neither $g_{ij}(x)$ nor $g_{ji}(x)$ are mapped by g_i or g_j, resp. and (1) holds.

In order to prove (2) suppose that there are morphisms $g'_i : G_i \to G'$, $1 \leq i \leq n$, such that $g'_i \circ g_{ij} = g'_j \circ g_{ji}$. A morphism $u : G \to G'$ is defined as follows:

$$u(z) = \begin{cases} g'_i(y), & \text{if } z = g_i(y), y \in G_i, \\ \text{undefined}, & \text{otherwise} \end{cases}$$

It is straight forward to check that u is well-defined and $u \circ g_i = g'_i$, $\forall 1 \leq i \leq n$.

Assuming another morphism $v : G \to G'$ with $v \circ g_i = g'_i$, $\forall 1 \leq i \leq n$, it follows that v has to be defined in the same way as u. Hence, u is unique.

□

Based on the star gluing of graphs distributed graphs can be described by morphism stars of local graphs and the embedding of their interface graphs.

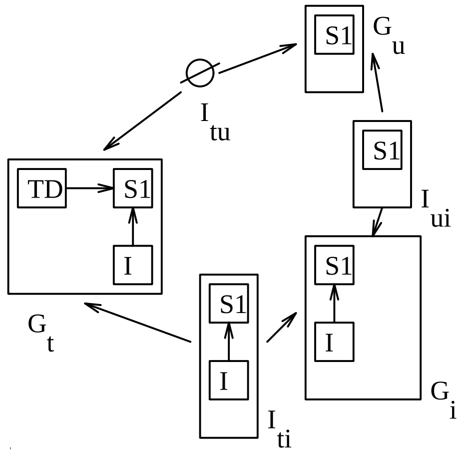

Figure 6: Distributed state of software development

Definition 3.4 (Distributed graph)

1. *A* splitting *of a graph G into graphs G_i, $1 \leq i \leq n$, with interfaces I_{ij}, $1 \leq i < j \leq n$, is a morphism star $MS = (G_i \leftarrow I_{ij} \rightarrow G_j)_{1 \leq i < j \leq n}$, written $SPLIT(G) = MS$, and its star gluing yields G. The splitting is called* total *if the partial star gluing of MS is total, in general, it is called* partial. *If a morphism star MS is given $JOIN(MS)$ is defined as star gluing yielding the gluing graph G.*

2. *A* distributed graph *is a morphism star MS as above where all G_i are called* local graphs *and all I_{ij} are called* interface(graph)s. *If MS is of size $n = 1$ it is called* global *graph.*

3. *If MS is total it is called* distributed graph with totally embedded interfaces, *in general, it has* partially embedded interfaces.

Example 3.5 (Distributed state) Each programmer has access to all units which have to be treated by her but is not interested in the other units. Thus, it would be natural to split the whole environment into local parts such that one programmer has access to all the units she works with. The interface graphs between them can be used to indicate which parts of the local graphs are copies of each other.

A possible distributed graph DG according to the local views of the three persons is depicted in figure 6. All interface graphs consist of those units which are shared by the corresponding persons. Distributed graph DG models a kind of communication channels where the user and the testing programmer communicate indirectly via the implementing one. A direct communication would be described by a non empty interface graph I_{tu}.

Gluing all local graphs of DG along their interfaces the result is graph G in figure 3 where node S corresponds to S1.

All persons have a local copy of the requirement specification. Moreover, the testing programmer has a copy of the implementation. These connections are indicated by the interface graphs I_{ti} and I_{ui}.

Definition 3.6 (Split, join and distributed steps) *Let D be called* distributed rule set *and N* distributed occurrence. *We consider the following* steps $DG \stackrel{D,N}{\Longrightarrow} DG'$ *on distributed graphs DG and DG':*

$$\begin{tikzcd}
& I_{ij} \arrow[dl, "g_{ij}"'] \arrow[dd, "g_{ji}"] \arrow[rr, Rightarrow, "*"] & & I'_{ij} \arrow[dl, "g'_{ij}"'] \arrow[dd, "g'_{ji}"] \\
G_i \arrow[rr, Rightarrow, "*"] & & G'_i & \\
& G_j \arrow[rr, Rightarrow, "*"] & & G'_j
\end{tikzcd}$$

Figure 7: Distributed step

1. Split step: *If* DG *is a global graph* $DG \overset{\emptyset,\emptyset}{\Longrightarrow} SPLIT(DG) = DG'$ *yields a morphism star* DG' *as defined in 3.4.*

2. Join step: *If* DG *is a morphism star of size* $n > 1$ $DG \overset{\emptyset,\emptyset}{\Longrightarrow} JOIN(DG) = DG'$ *yields a graph* DG' *as defined in 3.4.*

3. Distributed step: *If* $DG = (G_i \overset{g_{ij}}{\leftarrow} I_{ij} \overset{g_{ji}}{\rightarrow} G_j)_{1\leq i<j\leq n}$ *and* $DG' = (G'_i \overset{g'_{ij}}{\leftarrow} I'_{ij} \overset{g'_{ji}}{\rightarrow} G'_j)_{1\leq i<j\leq n}$ *a* distributed step $DG \overset{D,N}{\Longrightarrow} DG'$ *consists of sequential graph transformations* $G_i \overset{P_i,N_i}{\Longrightarrow}_s G'_i$ *and* $I_{ij} \overset{P_{ij},N_{ij}}{\Longrightarrow}_s I'_{ij}$, $1 \leq i < j \leq n$, *with transformation morphisms* $t_i : G_i \rightarrow G'_i$ *and* $i_{ij} : I_{ij} \rightarrow I'_{ij}$ *such that* $g'_{ij} \circ i_{ij} = t_i \circ g_{ij}$ *and* $g'_{ji} \circ i_{ij} = t_j \circ g_{ji}$, $1 \leq i < j \leq n$. $D = \{p_i \in P_i | 1 \leq i \leq n\} \cup \{p_{ij} \in P_{ij} | 1 \leq i < j \leq n\}$ *and* $N = \{n_i \in N_i | 1 \leq i \leq n\} \cup \{n_{ij} \in N_{ij} | 1 \leq i < j \leq n\}$. *In figure 7, a distributed step is depicted.*

4. *A distributed step* $DG \Longrightarrow DG'$ *as defined above is called* direct *if* $G_i \Longrightarrow G'_i$ *and* $I_{ij} \Longrightarrow I'_{ij}$, $\forall 1 \leq i < j \leq n$, *are direct sequential transformations.*

5. *A distributed step* $DG \Longrightarrow DG'$ *is called* asynchronous *if* DG *and* DG' *are distributed graphs with totally embedded interfaces.*

6. *A direct distributed step* $DG \Longrightarrow DG'$ *is called* synchronous *if* DG *and* DG' *are distributed graphs with totally embedded interfaces.*

Definition 3.7 (Distributed graph transformation)

1. *A* distributed graph transformation $DG \overset{D,N}{\Longrightarrow} DG'$ *of length* m *is a sequence*

$$DG = DG_0 \overset{D_1,N_1}{\Longrightarrow} DG_1 \overset{D_2,N_2}{\Longrightarrow} \ldots \overset{D_m,N_m}{\Longrightarrow} DG_m = DG'$$

of $m \geq 0$ *split, join and distributed steps* $d_k : DG_{k-1} \overset{D_k,N_k}{\Longrightarrow} DG_k$ *for* $1 \leq k \leq m$, *with* $D = \bigcup_{1\leq i\leq m} D_i$ *and* $N = \bigcup_{1\leq i\leq m} N_i$, *also written* $DG \overset{*}{\Longrightarrow} DG'$.

2. *A distributed graph transformation is called* synchronous *if all its distributed steps are synchronous.*

3. *A distributed graph transformation is called* asynchronous *if all its distributed steps are asynchronous.*

Remarks:

1. A morphism star $MS = (G)$ of size $n = 1$ formulates a graph not distributed. Due to this remark a direct sequential transformation $G \overset{p,m}{\Longrightarrow}_s G'$ is a special distributed transformation $G \overset{D,N}{\Longrightarrow} G'$ with $SPLIT(G) = (G)$, $SPLIT(G') = (G')$ and a synchronous step with $D = \{p\}$ and $N = \{m\}$.

2. Similarly to the local steps in [5] a distributed step can be called I-preserving if each two local transformations have an empty transformations $I_{ij} \Longrightarrow I'_{ij}$. Having this fact in mind distributed derivations as defined in [5] can be considered as special case of distributed graph transformations as defined here. Distributed derivations as formulated in the double-pushout approach[1], have to change to the global graph for synchronization. This restriction is removed for distributed graph transformations just defined.

3. In [5] also global derivation steps are defined as sequential derivations of global graphs. They are formulated here by special distributed steps where the morphism stars are of size $n = 1$. (Compare also the remark above.)

If the interfaces of distributed graphs show copies of shared objects distributed transformations describe the update of these objects in the following ways. Copies of shared objects in local graphs can be updated independently of other copies. This leads to inconsistent states, in general. Copies in interface graphs can be updated only if this is done in all local graphs that use it as interface. In the following examples different situations are shown. The first example shows a kind of simultaneous update of all copies of a shared object which is here the requirement specification. This update is modeled by a synchronous transformation. In the second example some copies of a shared object are updated in parallel while other copies first remain untouched. They are updated later on. This action is modeled by an asynchronous transformation. Lastly, two distributed transformations with several distributed steps are shown where two local parts synchronize each other while a third local transformation runs independently of the others.

Example 3.8 (Synchronous graph transformation) In the following a very delicate situation that the user changes the requirement specification is modeled by a synchronous graph transformation. We start at the distributed graph DG as depicted in figure 6. If all programmers get the new information the following state is derived by applying the rule r_{spec} shown in figure 4 at all local and all interface graphs of DG except of I_{tu} which is not changed, i.e. the empty rule is applied such that $I_{tu} = \bar{I}_{tu} = \emptyset$. This synchronous step $DG \overset{D,N}{\Longrightarrow} \hat{DG}$ is shown in figure 8. D consists of 5 copies of rule r_{spec} and N of the obvious occurrences in the local and interface graphs. All morphisms of the resulting morphism star $\hat{DG}$ are chosen as induced morphisms out of pushout graphs $\bar{I}_{ui}$, $\bar{I}_{ti}$ and $\bar{I}_{tu}$. The effect of these actions is the installation of a new specification in each local environment. The old testing data and implementation have to be adapted to that.

Example 3.9 (Asynchronous graph transformation) In this example, we just look at the testing and the implementing programmers. We start in the distributed state

[1]Using graphs and total graph morphisms as defined in 2.1 the double-pushout approach can be embedded in the single-pushout approach. See [12] for further reading.

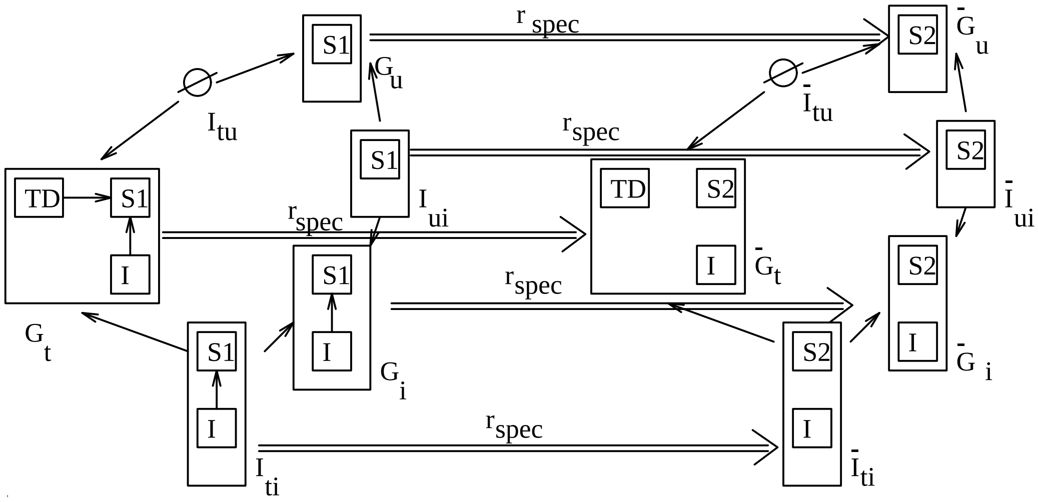

Figure 8: Change of requirement specification

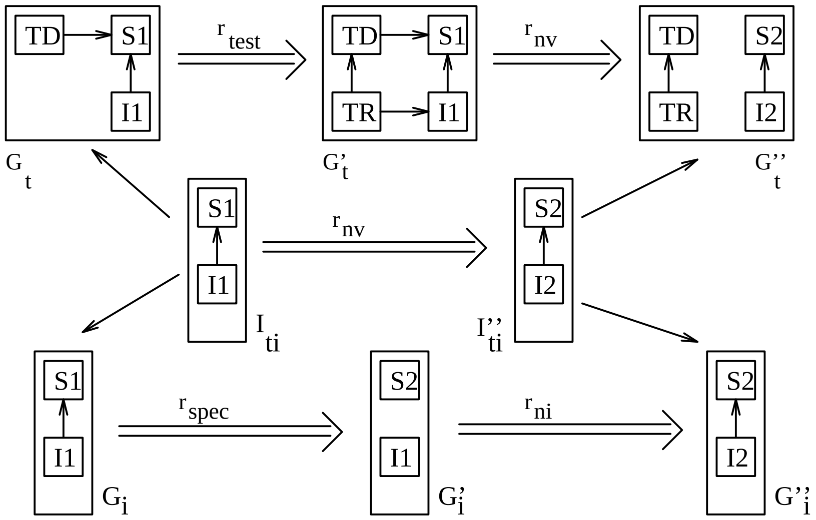

Figure 9: New version of specification and implementation

modeled by G_t, G_i, I_{ti} and morphisms $g_{ti} : I_{ti} \to G_t$ and $g_{it} : I_{ti} \to G_i$ as depicted in figure 6 and figure 9 where node "I" corresponds to "I1". Both programmers change the specification and the implementation asynchronously. The implementing one gets first a new requirement specification and develops then a new implementation for it. The testing person is first testing the old version and gets then a new version of the specification as well as the implementation. The interface graph which shows the objects shared by both programmers is adapted to the new version by applying the rule r_{nv}, too.

Example 3.10 (Distributed graph transformations)

1. If the user only tells the implementing programmer but the testing one is not informed that she changed the specification the resulting state becomes somewhat inconsistent. Modelling this situation the resulting distributed graph is achieved by applying rules r_{spec} at G_u, G_i and their interface graph I_{ui}. The testing programmer is testing the implementation of the old specification in between by applying rule r_{test} at G_t. Interface graphs I_{tu} and I_{ti} are not changed which means

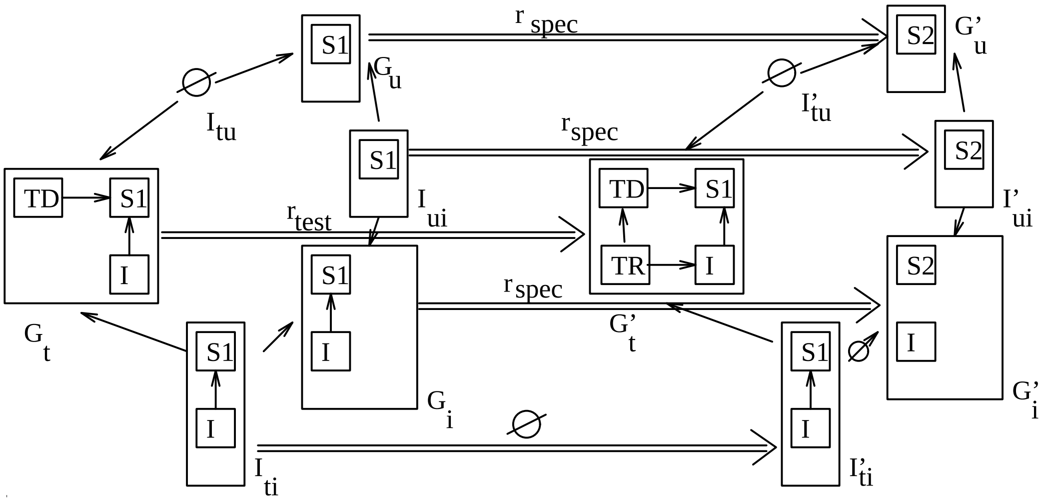

Figure 10: Incomplete update of requirement specification

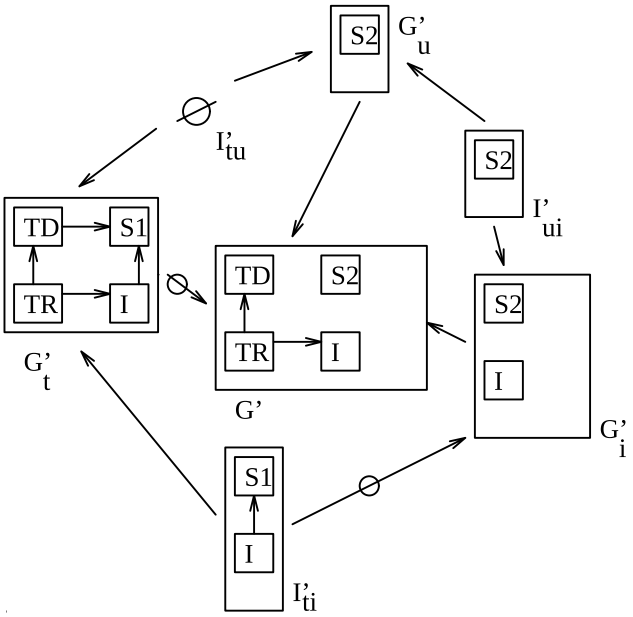

Figure 11: Star gluing of distributed graph DG'

that the empty rule is applied. Distributed graph transformation $DG \Longrightarrow DG'$ is depicted in figure 10. All morphisms of the resulting distributed graph DG' are chosen as induced morphisms out of pushout graphs I'_{ui}, I'_{tu} and I'_{ti}. Since the empty rule is applied at I_{tu} and I_{ti} the old specification is not deleted while it is in G_u and G_i.

Looking at the resulting distributed graph DG', morphism $I'_{ti} \to G'_i$ becomes partial indicating a missing update. Joining all local graphs according to their interfaces this would have a synchronization effect. In the global graph G' the old specification is deleted although it has been used by the testing programmer. The resulting splitting of G' is shown in figure 11. Thus in general, this would mean that morphism stars where some morphisms are not total represent a distributed state which is not consistent. Gluing of such a star leads to a global state where all conflicts are solved.

Another way of synchronization is to do the missing updates in the local envi-

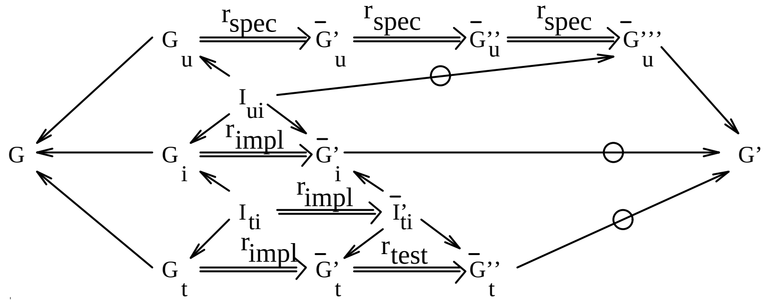

Figure 12: Example of a distributed graph transformation

ronment of the testing programmer later on. This possibility of synchronization describes some kind of asynchronous update of shared objects since this synchronization can be performed after arbitrary many other local transformations.

2. A distributed graph transformation sequence can have a shape as in figure 12. We start at the situation in figure 6 where the empty interface graph I_{tu} is left out. The global graph G is depicted in figure 3. Now, the user changes the requirement specification locally without any communication with the others. Implementing and testing goes hand in hand meaning that a new implementation is immediately transfered to the testing programmer. After some steps all three persons have a common meeting for an update of their information.

   This distributed proceeding is modeled by a distributed graph transformation as depicted in figure 12 on an abstract level. Graph G' is shown in figure 11 but the corresponding splittings are different.

4 Properties of Synchronous and Asynchronous Graph Transformations

In this section, a criterion for distributed rule sets and distributed occurrences is given which is sufficient to characterize synchronous graph transformations. This criterion restricts distributed occurrences to those without deletion conflicts wrt. the corresponding distributed rule set.

In section 3, synchronous and asynchronous graph transformations are defined as special cases of distributed transformations. The question arises if there is a proper implication between asynchronous and distributed transformations and, moreover, between synchronous and asynchronous graph transformations. Such a hierarchy can be shown at the examples about distributed software development presented in the previous sections. On the other hand, it can be shown that for each arbitrary distributed step there is a sequence of steps with direct transformations, only, as this is required for synchronous graph transformations. This result can be achieved if the application of the empty rule is allowed modeling some kind of no-operation. But in general, intermediate constructed distributed graphs have partial embeddings of interfaces. Transformations with direct distributed steps and, in general, partial interface embeddings are regarded as normal forms of distributed transformations, in the following. This normal form is useful because synchrony can be checked easier.

Synchronous distributed steps which are direct steps with additional conditions can be characterized by distributed steps where the local actions and corresponding

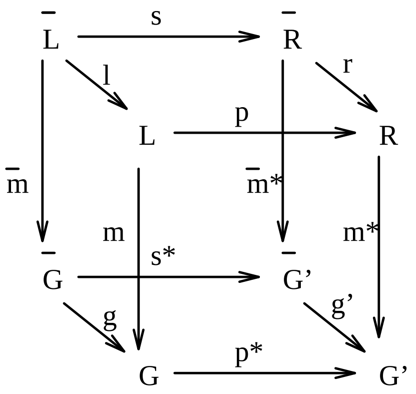

Figure 13: D-complete subrule

interactions have harmonize in the deletion of objects. A local action is only then allowed to delete its copy of a shared object if this is also done in all its interfaces. To describe this idea in the framework of graph transformations the concept of subrules presented in [2] and [12] is used. Moreover, a notion of suboccurrence is invented to describe so-called deletion completeness between subrule and superrule applications.

Definition 4.1 (Subrules and suboccurrences)

1. *A rule $s : \bar{L} \to \bar{R}$ is a* subrule *of a rule $p : L \to R$ if there are total morphisms $l : \bar{L} \to L$ and $r : \bar{R} \to R$ such that $r \circ s = p \circ l$. The pair (l, r) is called* subrule embedding. *(l, r) is called* weak *subrule embedding if l and r are partial.*

2. *Given a subrule $s : \bar{L} \to \bar{R}$ of $p : L \to R$ with embedding (l, r), two occurrences $\bar{m} : \bar{L} \to \bar{G}$ and $m : L \to G$ and a total graph morphism $g : \bar{G} \to G$. $\bar{m}$ is called* suboccurrence *of m wrt. g if $g \circ \bar{m} = m \circ l$. If (l, r) is weak and g is partial $\bar{m}$ is called* weak *suboccurrence of m wrt. g.*

3. *Let $s : \bar{L} \to \bar{R}$ be a subrule of $p : L \to R$ and $\bar{m} : \bar{L} \to \bar{G}$ a suboccurrence of $m : L \to G$ wrt. $g : \bar{G} \to G$ such that $\bar{m}$ (m) is conflict-free wrt. s (p). Then application pair $(s, \bar{m})$ is called deletion complete, short* d-complete, *wrt. application pair (p, m) if for all $x \in L - L(p)$, $y \in \bar{G}$ we have: if $g(y) = m(x)$ then there exists some $z \in \bar{L} - \bar{L}(s)$ with $\bar{m}(z) = y$. (See figure 13.)*

Example 4.2 (Deletion completeness) As we can see in example 3.8 in the synchronous transformation modeling the simultaneous update of all copies of a shared object all rule applications are d-complete. In contrast to that updating of just a part of the copies is modeled by distributed transformations where not all rule applications are d-complete. This is shown in example 3.10. The first transformation leads to a resulting distributed graph with totally embedded interfaces, the second one to a graph with partially embedded interfaces.

Thus, the following proposition can be stated which gives us a characterization of synchronous graph transformations.

Proposition 4.3 (Deletion-completeness) *Given two direct sequential transformations $G \overset{p,m}{\Longrightarrow}_s G'$ and $\bar{G} \overset{s,\bar{m}}{\Longrightarrow}_s \bar{G}'$ where s is a subrule of p with embedding (l, r). Moreover, let $g : \bar{G} \to G$ be a total graph morphism such that $\bar{m}$ is a suboccurrence of m wrt. g. Application pair $(s, \bar{m})$ is d-complete wrt. (p, m) if and only if the resulting morphism $g' : \bar{G}' \to G'$ is total.*

Proof: Let p^* and m^* be the pushout morphisms of p and m and s^* and $\bar{m}^*$ that of s and $\bar{m}$. Since g' is the induced morphism out of PO-graph $\bar{G}'$ wrt. $p^* \circ g$ and $m^* \circ r$ we have $g' \circ s^* = p^* \circ g$ and $g' \circ \bar{m}^* = m^* \circ r$.

1. Assume that there is a node or edge $x \in \bar{G}'$ not mapped to G'. Since $\bar{G}'$ is a PO-graph x comes from $\bar{R}$ or $\bar{G}$.

   (a) If there is a $y \in \bar{R}$ with $\bar{m}^*(y) = x$ it has to be mapped also by $m^* \circ r$ such that $m^* \circ r(y) = g' \circ \bar{m}^*(y) = g'(x)$. Since m^* is total by conflict-freeness as stated in proposition 2.3 and r is total because it is part of the subrule embedding $g'(x)$ is defined.

   (b) If there is a $\bar{z} \in \bar{G}$ with $s^*(\bar{z}) = x$ there is also a $z \in G$ with $g(\bar{z}) = z$.

      i. If there is $\bar{v} \in \bar{L}(s)$ with $\bar{m}(\bar{v}) = \bar{z}$ there is also $v \in L$ with $l(\bar{v}) = v$ since l is part of the subrule embedding. Since $g \circ \bar{m} = m \circ l$ $m(v) = z$. If $v \in L - L(p)$ there has to be some $\bar{v}' \in \bar{L} - \bar{L}(s)$ with $\bar{m}(\bar{v}') = \bar{z}$ because of d-completeness but then $\bar{m}$ is not conflict-free wrt. s. If $v \in L(p)$ there has to be some $v' \in L - L(p)$ with $m(v') = z$ to delete z but then m is not conflict-free wrt. p.

      ii. If there is $\bar{l} \in \bar{L} - \bar{L}(s)$ with $\bar{m}(\bar{l}) = \bar{z}$ $\bar{l}$ cannot be mapped to x via $\bar{R}$.

      iii. If there is not an $\bar{v} \in \bar{L}$ with $\bar{m}(\bar{v}) = \bar{z}$ z has to be mapped to an object of G' according to d-completeness.

2. If $(s, \bar{m})$ is not d-complete wrt. (p, m) we have to investigate the following cases:

   (a) Assume there is a $z \in \bar{L}(s)$ with $\bar{m}(z) = y$ and $g(y) = m(x)$ for some $x \in L - L(p)$. If $l(z) = x$ there is not a subrule embedding (l, r) because x is deleted and r has to be total. Since l has to be total, too, there has to be a $x' \in L$ with $l(z) = x'$. x' has to be preserved because (l, r) is a subrule embedding. Since $\bar{m}$ is a suboccurrence of m wrt. g $m(x') = m(x)$. This leads to a conflict. Hence, m is not conflict-free wrt. p.

   (b) If there is not a $z \in \bar{L}$ with $\bar{m}(z) = y$ and $g(y) = m(x)$ for some $x \in L - L(p)$ there is a $s^*(y) \in \bar{G}'$ but not $p^*(m(x)) \in G'$ since $x \in L - L(p)$. Hence, g' becomes partial.

If x is a dangling edge the same arguments hold for the source or target node which has to be deleted.

□

Corollary 4.4 (Characterizing synchronous graph transformations) *Let DG be a distributed graph with totally embedded interfaces, D a distributed rule set and N a distributed occurrence of a synchronous step $DG \overset{D,N}{\Longrightarrow} DG'$. The pair (D, N) is d-complete if and only if the resulting distributed graph DG' has totally embedded interfaces.*

Proof: Direct consequence of proposition 4.3.

□

Normally, distributed steps contain arbitrarily long local transformations. The following theorem states that there is a corresponding sequence of direct distributed steps between the distributed graphs given. Such a sequence is regarded as normal form of distributed transformations. Although it is not unique for a given distributed transformation, in general, it shows how to synchronize the actions if all distributed graphs in between have totally embedded interfaces.

Definition 4.5 (Normal form of distributed graph transformations) *A distributed graph transformation is called* to be in normal form *if all its distributed steps are direct.*

Theorem 4.6 (Distributed steps) *For each distributed step $DG \stackrel{D,N}{\Longrightarrow} DG'$ as defined in 3.6 there is a sequence of direct distributed steps $DG \stackrel{D',N'}{\Longrightarrow} DG'$ with $D' \supseteq D$, $N' \supseteq N$ and $\forall p \in D' - D : p = \emptyset$.*

Proof: Let $DG = (G_i \stackrel{g_{ij}}{\leftarrow} I_{ij} \stackrel{g_{ji}}{\rightarrow} G_j)_{1 \leq i < j \leq n}$, first all local transformations $G_i \stackrel{*}{\Longrightarrow}_s H_i$, $\forall 1 \leq i \leq m$, are performed step by step while the interface graphs I_{ij} are left unchanged, i.e. the empty rule is applied. We start to construct a sequence of local transformation steps $DG = \hat{DG}^0 \Longrightarrow \hat{DG}^1 \Longrightarrow \ldots \Longrightarrow \hat{DG}^m = \hat{DG}$ where m is the length of the longest local transformation $G_i \stackrel{*}{\Longrightarrow}_s H_i$ and $\hat{DG} = (H_i \stackrel{\hat{g}_{ij}}{\leftarrow} I_{ij} \stackrel{\hat{g}_{ji}}{\rightarrow} H_j)_{1 \leq i < j \leq n}$. If all local transformations have length 0 $\hat{DG} = \hat{DG}^0 = DG$. Otherwise assume that $\hat{DG}^k = (G_i^k \stackrel{g_{ij}^k}{\leftarrow} I_{ij} \stackrel{g_{ji}^k}{\rightarrow} G_j^k)_{1 \leq i < j \leq n}$ with $k < m$ is already constructed. To get $\hat{DG}^{k+1}$ we have to distinguish two cases: For all $G_i \Longrightarrow H_i$ with length less or equal to k $G_i^{k+1} = G_i^k = H_i$, $g_{ij}^{k+1} = g_{ij}^k$, $1 \leq i \neq j \leq n$. For all $G_i \Longrightarrow H_i$ with length greater than k the $(k+1)$. steps of all $G_i = G_i^0 \Longrightarrow_s G_i^1 \ldots \Longrightarrow_s G_i^k \Longrightarrow_s G_i^{k+1} \ldots H_i$ are performed. This leads to $g_{ij}^{k+1} = t_i^k \circ g_{ij}^k$ with $t_i^k : G_i^k \to G_i^{k+1}$ being the corresponding transformation morphism. Notice that in general, morphism g_{ij}^{k+1} becomes partial.

In a second step, the interface graphs are transformed via $I_{ij} \stackrel{*}{\Longrightarrow}_s J_{ij}$, $\forall 1 \leq i < j \leq m$, while the local graphs H_i are retained by applying the empty rule. Now, we construct a sequence of local transformation steps $\hat{DG} = DG^0 \Longrightarrow DG^1 \Longrightarrow \ldots \Longrightarrow DG^m = DG'$ where m is the length of the longest local transformation $I_{ij} \stackrel{*}{\Longrightarrow}_s J_{ij}$ and $DG' = (H_i \stackrel{h_{ij}}{\leftarrow} J_{ij} \stackrel{h_{ji}}{\rightarrow} H_j)_{1 \leq i < j \leq n}$. If all transformations $I_{ij} \stackrel{*}{\Longrightarrow}_s J_{ij}$ are of length 0 $DG' = DG^0 = \hat{DG}$. If $m > 0$ assume that $DG^k = (H_i \stackrel{h_{ij}^k}{\leftarrow} I_{ij}^k \stackrel{h_{ji}^k}{\rightarrow} H_j)_{1 \leq i < j \leq n}$ with $k < m$ is already constructed and $h_{ij} \circ j_{ij}^k = h_{ij}^k$ with j_{ij}^k being the transformation morphisms from I_{ij}^k to J_{ij}. For DG^{k+1} we distinguish two cases: For all $I_{ij} \Longrightarrow_s J_{ij}$ with length less or equal to k $I_{ij}^{k+1} = I_{ij}^k = J_{ij}$, $h_{ij}^{k+1} = h_{ij}^k$ and $h_{ji}^{k+1} = h_{ji}^k$, $1 \leq i < j \leq n$. For all $I_{ij} \Longrightarrow J_{ij}$ with length greater than k the $(k+1)$. steps of all $I_{ij} = I_{ij}^0 \Longrightarrow_s I_{ij}^1 \ldots \Longrightarrow_s I_{ij}^k \Longrightarrow_s I_{ij}^{k+1} \ldots J_{ij}$ are performed. $\forall 1 \leq i < j \leq n$, morphism h_{ij}^{k+1} is the induced morphism out of gluing graph I_{ij}^{k+1} with $h_{ij}^{k+1} \circ i_{ij}^k = h_{ij}^k$ where i_{ij}^k is the transformation morphism of $I_{ij}^k \Longrightarrow_s I_{ij}^{k+1}$. This is due to the fact that $h_{ij} \circ j_{ij}^{k+1} \circ i_{ij}^k = h_{ij}^k$ using the induction assumptions. Analogously, h_{ji}^{k+1} is the unique morphism with $h_{ji}^{k+1} \circ i_{ij}^k = h_{ji}^k$. (See figure 14.)

□

Corollary 4.7 (Normal form of distributed graph transformations) *For each distributed graph transformation $DG \stackrel{D,N}{\Longrightarrow} DG'$ there is a distributed graph transformation $DG \stackrel{D',N'}{\Longrightarrow} DG'$ in normal form with $D' \supseteq D$, $N' \supseteq N$ and $\forall p \in D' - D : p = \emptyset$.*

Proof: Direct consequence of theorem 4.6.

□

Remark: Clearly, each normal form of distributed graph transformation is a distributed graph transformation per definition.

Normal forms of distributed transformations can be optimized in the sense that local and interface transformations can be executed simultanously if they are independent of each other or related via weak subrule embeddings.

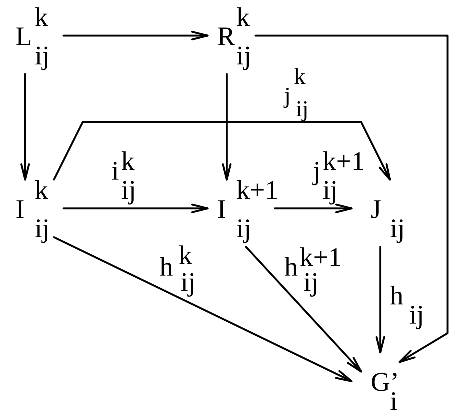

Figure 14: Distributed steps

Lemma 4.8 (Related sequential transformations) *Given two direct sequential transformations $G \overset{p,m}{\Longrightarrow}_s G'$ and $\bar{G} \overset{s,\bar{m}}{\Longrightarrow}_s \bar{G}'$ and a graph morphism $g : \bar{G} \to G$ there is a graph morphism $g' : \bar{G}' \to G'$ if there is a weak subrule embedding (l, r) of s into p with $\bar{m}$ being a weak suboccurrence of m wrt. g. (See figure 13.)*

Proof: Given s, $\bar{m}$, p and m as defined in 4.1. According to the assumptions we have $p^* \circ g \circ \bar{m} = p^* \circ m \circ l = m^* \circ p \circ l = m^* \circ r \circ s$. Thus, there is a unique morphism $g' : \bar{G}' \to G'$ with $g' \circ s^* = p^* \circ g$ and $g' \circ \bar{m}^* = m^* \circ r$ due to the universal property of the pushout over s and $\bar{m}$.

□

Corollary 4.9 (Optimization of normal forms) *Given a distributed transformation $DG^1 \Longrightarrow DG^2 \Longrightarrow DG^3$ consisting of two direct distributed steps there is a sequence of two direct distributed steps $DG^1 \Longrightarrow \hat{DG}^2 \Longrightarrow DG^3$ with $DG^1 = (G^1_1 \overset{g^1_{12}}{\leftarrow} I^1_{12} \overset{g^1_{21}}{\rightarrow} G^1_2), DG^2 = (G^2_1 \overset{g^2_{12}}{\leftarrow} I^2_{12} \overset{g^2_{21}}{\rightarrow} G^2_2)$, $DG^3 = (G^3_1 \overset{g^3_{12}}{\leftarrow} I^3_{12} \overset{g^3_{21}}{\rightarrow} G^3_2)$, $G^1_i \overset{p^1_i,m^1_i}{\Longrightarrow}_s G^2_i \overset{p^2_i,m^2_i}{\Longrightarrow}_s G^3_i$ for $i = 1, 2$ and $I^1_{12} \overset{p^1_{12},m^1_{12}}{\Longrightarrow}_s I^2_{12} \overset{p^2_{12},m^2_{12}}{\Longrightarrow}_s I^3_{12}$ if they satisfy one of the following conditions:*

1. *$G^2_i \Longrightarrow G^3_i$ for $i = 1, 2$ and $I^1_{12} \Longrightarrow I^2_{12}$ are empty transformations. Then $\hat{DG}^2 = (G^3_1 \overset{g^3_{12}}{\leftarrow} I^3_{12} \overset{g^3_{21}}{\rightarrow} G^3_2)$ with $DG^1 \Longrightarrow \hat{DG}^2$ consisting of $G^1_i \Longrightarrow G^2_i$ for $i = 1, 2$ and $I^1_{12} \overset{p^2_{12},m^2_{12}}{\Longrightarrow} I^3_{12}$. $\hat{DG}^2 \Longrightarrow DG^3$ consists of empty transformations only.*

2. *$I^1_{12} \Longrightarrow I^2_{12}$ is an empty transformation and p^2_{12} is a weak subrule of p^1_1 and p^1_2, moreover, m^2_{12} is a weak suboccurrence of m^1_1 wrt. g^1_{12} and of m^1_2 wrt. g^1_{21}. Then $\hat{DG}^2 = (G^2_1 \overset{h^2_{12}}{\leftarrow} I^3_{12} \overset{h^2_{21}}{\rightarrow} G^2_2)$ where h^2_{12} and h^2_{21} are the induced morphisms out of PO-graph I^3_{12}. $DG^1 \Longrightarrow \hat{DG}^2$ contains $I^1_{12} \overset{p^2_{12},m^2_{12}}{\Longrightarrow} I^3_{12}$ whereas $\hat{DG}^2 \Longrightarrow DG^3$ has the empty transformation $I^3_{12} \Longrightarrow I^3_{12}$ then.*

Proof:

1. According to commutativity of diagrams.

2. Direct consequence of lemma 4.8.

□

According to corollary 4.9 we are able to condense distributed steps as far as possible in order to get more information about synchrony.

That synchronous and asynchronous graph transformations have actually a synchronization effect which is differently restrictive is shown by the following hierarchy theorem.

Theorem 4.10 (Hierarchy for distributed graph transformations) *Let D and D' be distributed rule sets, N and N' distributed occurrences and $D' \supseteq D$, $N' \supseteq N$ and $\forall p \in D' - D : p = \emptyset$. Then, for $k = 1, 2$, statement k implies statement $k+1$ and none of the implications is an equivalence:*

1. *There is a synchronous graph transformation $DG \stackrel{D,N}{\Longrightarrow} DG'$.*
2. *There is an asynchronous graph transformation $DG \stackrel{D',N'}{\Longrightarrow} DG'$.*
3. *There is a distributed graph transformation $DG \stackrel{D',N'}{\Longrightarrow} DG'$.*

Proof: Statement *1* implies statement *2* per definition setting $D = D'$ and $N = N'$ since it has exclusively direct distributed steps between distributed graphs with total interface embeddings which are special distributed steps.

Statement *2* implies statement *3* also per definition because distributed graphs with totally embedded interfaces are special distributed graphs with partially embedded interfaces.

For an example satisfying statement *2* but not statement *1* see the asynchronous graph transformation in figure 9. The application of the empty rule to interface graph I_{ti} before or after $I_{ti} \stackrel{r_{nv}}{\Longrightarrow}_s I''_{ti}$ would offer the possibility to construct direct steps. Applying the empty rule to I_{ti} would not change it and would lead to a partial embedding into G'_i because the specification has changed there. Applying first r_{nv} to I_{ti} while r_{test} is applied to G_t and r_{spec} to G_i would not yield a distributed step. That is the case because the node "S1" of I_{ti} is mapped to "S1" of G'_t via G_t but not via I'_{ti}. The same is true if we take node "I1" and the edge in between. This phenomena can also be observed using G_i instead of G_t.

Considering the distributed transformation $DG \Longrightarrow DG'$ in figure 10 the resulting distributed graph DG' has partially embedded interfaces which is forbidden for asynchronous graph transformations. Thus, statement *3* is satisfied but not statement *2*.

□

Remark: In [7] an hierarchy theorem for interface preserving derivations is stated. Among several types of parallel and distributed derivatons I-distributed derivations with total and partial splittings are compared. This kind of derivation does not allow dynamic interfaces (see the remark about I-preserving derivations after definition 3.7). I-distributed derivations with total splittings are contained in the synchronous case. Using partial splittings a special case of the general distributed transformations is described. The hierarchy theorem in [7] shows that I-distributed derivations with total splittings are a special case of those with partial splittings. This is covered by the fact that statement *1* of theorem 4.10 implies statement *3*. Since in the hierarchy theorem of [7] direct distributed derivations are considered only asynchronous graph transformations do not have a proper reflection there.

5 Conclusion

In this paper we presented distributed graph transformations in the single-pushout approach. It turned out that all distributed transformation sequences can be transformed into distributed graph transformations in normal form by sequentializing local transformations and interface transformations in some way (corollary 4.7). Future work should investigate criteria under which equivalence of arbitrary distributed sequences is achieved using a suitable notion of equivalence. Moreover, the whole theory should be extended to distributed transformations where the distribution of graphs can be iterated and rules can be distributed, too.

We discussed synchronous and asynchronous graph transformations on the basis of distributed graph transformations. Distributed transformations are proposed to be synchronous if all distributed graphs involved have totally embedded interfaces and direct distributed steps. This means that all copies of shared objects have to be updated simultaneously. Otherwise some intermediate inconsistent state is achieved indicated by partial splittings. If totally embedded interfaces are achieved at last asynchronous communication is modeled.

Synchronous graph transformations which preserve total splittings are characterized by distributed graph transformations without deletion conflicts in each direct distributed step. It may be expected that they are comparable with amalgamated graph transformations where amalgamated occurrences are conflict-free wrt. corresponding amalgamated rules. Amalgamated graph transformations describe some kind of parallel replacement on graphs which is synchronized by the additional application of subrules. The relation of amalgamated and distributed graph transformations is investigated in [7] for two parallel rules with a common subrule in the single-pushout approach. Amalgamated transformations for arbitrary many rules in parallel is described in [15] for the double-pushout approach. This type of transformation can be adapted to the single-pushout approch very easily and the relation to distributed transformations can be generalized to more than two rules in a direct way.

The example of a distributed software development gives some intuition how synchronous and asynchronous graph transformations model corresponding types of communication via copies of shared objects. Investigating further examples of distributed systems it hopefully turns out that this proposal of synchronous and asynchronous transformations is successful from the distributed system point of view.

Acknowledgements

I thank Ralph Betschko, Hartmut Ehrig, Reiko Heckel, Annika Wagner and the referees for their valuable comments on this paper.

References

[1] J. Adamek, H. Herrlich, and G. Strecker. *Abstract and Concrete Categories.* Series in Pure and Applied Mathematics. John Wiley and Sons, 1990.

[2] P. Böhm, H.-R. Fonio, and A. Habel. Amalgamation of graph transformations: a synchronization mechanism. *Journal of Computer and System Science*, 34:377–408, 1987.

[3] P. Degano and U. Montanari. A model of distributed systems based on graph rewriting. ACM 34(2):411–449, 1987.

[4] H. Ehrig. Introduction to the algebraic theory of graph grammars. In *1st Graph Grammar Workshop, Lecture Notes in Computer Science 73*, pages 1–69, 1979.

[5] H. Ehrig, P. Boehm, U. Hummert, and M. Löwe. Distributed parallelism of graph transformation. LNCS 314, pages 1–19, Berlin, 1988.

[6] H. Ehrig and H.-J. Kreowski. Parallel graph grammars. In A. Lindenmayer and G. Rozenberg, editors, *Automata,Languages, Development*, pages 425–447. Amsterdam: North Holland, 1976.

[7] H. Ehrig and M. Löwe. Parallel and distributed derivations in the single pushout approach. TCS 109, pages 123 – 143, 1993.

[8] S. K. Goering. *A graph grammar approach to concurrent programming.* PhD thesis, University of Illinois at Urbana-Champaign, 1990.

[9] D. Janssens and G. Rozenberg. Structured transformations and computation graphs for actor grammars. in *Graph Grammars and Their Application to Computer Science*, H. Ehrig, H.-J. Kreowski, G. Rozenberg (eds.), LNCS 532, pages 446–460. Springer, 1991.

[10] S.M. Kaplan, J.P. Loyall, and S.K. Goering. Specifying concurrent languages and systems with Δ-grammars. in *Graph Grammars and Their Application to Computer Science*, H. Ehrig, H.-J. Kreowski, G. Rozenberg (eds.), LNCS 532, pages 475–489, Springer,1991.

[11] M. Korff. Single pushout transformations of equationally defined graph structures with applications to actor systems. in *Graph Transformations in Computer Science*, H.-J. Schneider, H. Ehrig (eds.), LNCS 776, Springer, 1994.

[12] M. Löwe. Algebraic approach to single-pushout graph transformation. *TCS*, 109:181–224, 1993.

[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. Describing distributed systems by categorial graph grammars. In *15th International Workshop on Graph-theoretic Concepts in Computer Science.* Springer Lecture Notes on Computer Science, 1989.

[15] G. Taentzer. Parallel high-level replacement systems. Technical Report, TU Berlin, No. 92-10, to appear in TCS, 1994.