Local Views on Distributed Systems and their Communication

Ingrid Fischer*[1], Manuel Koch[2], Gabriele Taentzer[2]

[1] International Computer Science Institute, Berkeley, USA, and IMMD2, University of Erlangen-Nuremberg, Germany, idfische@informatik.uni-erlangen.de
[2] Technical University of Berlin, Germany, {mlkoch,gabi}@cs.tu-Berlin.de

Abstract. Distributed graph transformation has been used to specify static as well as dynamic aspects of distributed systems. To support distributed designs by different developers, local views are introduced. A local view on a distributed system consists of one local system, its import and export interfaces, and connected remote interfaces. The behavior of a local system is specified by a set of graph rules that are applicable only to the local view of the local system. Local systems communicate either synchronously or asynchronously via their import and export interfaces. Asynchronous communication is modeled by sequential application of graph rules, synchronous communication by the amalgamation of graph rules. We compose a distributed system grammar from the rule sets for local systems. The operational semantics of the distributed system is given by distributed transformation sequences.

1 Introduction

Distributed graph transformation has been presented as a visual means for specifying static as well as dynamic aspects of distributed systems [8, 3, 6]. Whereas [8] introduces the basis concepts of distributed graph transformation based on the double-pushout approach to graph transformation, several extensions towards a specification technique customized for distributed systems have been proposed. In [3], distributed graph rules were equipped with application conditions for additional control of the rule application. Furthermore, attribution concepts have been integrated [4] to specify occurring data. In [6], both extensions have been combined to yield a specification technique suitable for describing the main aspects of distributed systems.

However, distributed graph transformation as introduced up to now assumes a global view on the whole distributed system, which is desirable in the early stage of system development to get an overview. In this first phase, network and interface structure graphs are developed and the principle network activities and interface services are described by distributed graph rules. After the network structure and its reconfiguration possibilities as well as the interfaces are fixed,

* I. Fischer was partly supported by a postdoc stipend of "Gemeinsamen Hochschulsonderprograms III von Bund und Ländern" from the DAAD.

the system developers are supposed to take a local view on network nodes and local system parts running on them.

To support this second phase of the development process as well, local views on distributed graphs are introduced in this paper. A local view contains a local system itself, its import and export interfaces and remote import and export interfaces to which the local system has connections. The behavior of a local system is specified by a set of local view rules, i.e. graph rules applicable only to the local view of the local system. Two local systems can communicate synchronously or asynchronously via their import and export interfaces. Asynchronous communication is modeled by sequential application of local view rules, whereas synchronous communication is formalized by rule amalgamation as presented in [5] for arbitrary cocomplete categories.

The synchronization of local view rules describes a kind of service request which is not already available, but can be computed. The result is put into an export interface.

A distributed system grammar contains the set of local view rules for each local system participating in the distributed system. The operational semantics of the distributed system is given by the set of all distributed transformation sequences starting at the start graph, which represents the initial state of the distributed system.

To simplify the presentation in this paper, we present concepts for labeled graphs instead of attributed graphs. All main concepts of distributed graph transformation and their local views are illustrated in a case study concerning distributed configuration management [9].

2 Distributed Configuration Management

First, we introduce a case study dealing with *Distributed Configuration Management*, as described in [9], for verifying the usability of the ideas developed later on.[1]

The size of software projects and requirements for high quality make it difficult to plan and coordinate projects. Two third of time and money spent on them go into maintenance and development after the project was finished. Furthermore, the knowledge needed to complete a greater larger project is hardly available in a single company. External project partners must contribute. Often it is also cheaper to allocate tasks to external experts. One possible scenario is to have just the core developers located in the original company. Additional personnel is situated around the world wherever the knowledge needed is available for a low price. With an ever growing demand on distributed software development, communication, coordination and quality management are needed, as all project sites must have access to a consistent actual set of project documents. When a project site changes a document leading to a new revision, it must become known in all other project sites, too. This is especially a problem when no

[1] The authors would like to thank V. Volle for numerous discussions.

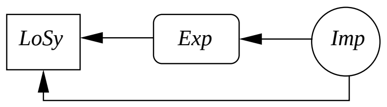

Fig. 1. The type graph *DiSy*.

central online archive can be used by all project partners. In this situation each project site has its own *revision archive*, a local repository where all documents and their different revisions are stored. Documents are *replicated* among revision archives to ensure that each project site has an up-to-date document set. This can be done by different means via the Internet daily, hourly, weekly or even by sending a floppy via postal mail. *Workspaces* are used when a revision archive document is to be changed or a new document has to be inserted. A workspace is connected to one archive and each archive may have an arbitrary number of workspaces. When the owner of a workspace wants to change a document he/she *checks it out* from the archive into the workspace. Then the actual change can take place or something new can be created. When this work is finished, the documents are *checked back into* the revision archive. In the following sections parts such a system will be described with distributed graph transformations.

3 Distributed Typed Graphs

We consider a distributed system consisting of local systems communicating via export and import interfaces. In export interfaces, local systems present objects accessible for remote systems whereas import interfaces contain local copies of objects from remote export interfaces. This idea of a distributed system's topology is modeled in the graph $DiSy$ shown in Figure 1. We provide a node $LoSy$ for local systems, a node Exp for export interfaces and a node Imp for import interfaces. Connections are possible between local systems and any type of interface and between import and export interfaces.

The nodes of the graph $DiSy$ are abstract in the sense that they show only the main components of a distributed system. In order to instantiate the abstract types for a concrete application, an application specific graph NTG is chosen. This graph has to possess the structure of our distributed system, which is ensured by the existence of a graph morphism $t_{NTG} : NTG \to DiSy$, called a *network type graph* in the following.

Definition 1 (network graph). *A* network type graph *is a graph morphism $t_{NTG} : NTG \to DiSy$. Each graph morphism $t_G : G \to NTG$ in NTG is a* network graph *w.r.t. NTG. We often write simply NTG for the network type graph t_{NTG} and G for a network graph w.r.t. NTG if no confusion is possible.*

Example 1. The network type graph for the distributed system of the case study as introduced in Section 2 is shown in Figure 2 on the right-hand side. It contains two kinds of local systems, namely *Workspace* and *Revisionarchive*, two

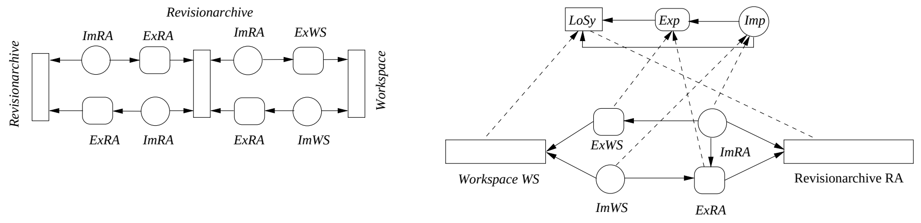

Fig. 2. A network graph modeling two revision archives with a workspace connected to one of the revision archives (left) and its network type graph (right).

different kinds of export interfaces, namely *ExWS* for workspaces and *ExRA* for revision archives, and two different types of import interfaces *ImWS* and *ImRA* for workspaces and revision archives, respectively.[2] Via the interfaces the replication between different archives and the checking in and out between archive and workspace can be realized. In Figure 2 on the left-hand side a network graph w.r.t. the network type graph on the right is shown. It represents a snapshot of a small project with two revision archives and one workspace, together with their interfaces.

The internal state of local systems and interfaces is described by a labeled graph. To simplify the presentation of the paper, we assume a common label set $\mathcal{L}$ for all local systems and interfaces.

Example 2. The internal states of workspaces and revision archives contain the revisions of documents (text files, code files, etc.) the project partners work on. Documents belonging together are packed into configurations that can contain other configurations. Each time a document or a whole configuration is changed the old version is kept and the new one is stored as a new revision of the old one. Documents or configurations cannot be changed in the revision archive. To change a document, it has to be checked out into a workspace. However, only one document can be checked out into the workspace and configurations have to be checked out completely. An example of a revision archive's local graph and its label alphabet is given in Figure 3. It consists of two configurations, where one configuration is a revision of the other. Each configuration contains two documents with one existing in both configurations.

Given a network graph G, each node in G_V is refined to an $\mathcal{L}$-labeled graph. If the node is a local system, this labeled graph represents the local state of the local system. If the node is an interface, it is the local state of the interface. An edge in G_E indicates a connection between local systems and their interfaces or between interfaces. This connection is refined to a label-preserving graph morphism that shows the relation between the local state of the source node of the network edge and its target node.

[2] It may be also possible to use just one export interface of a revision archive for workspaces and other revision archives.

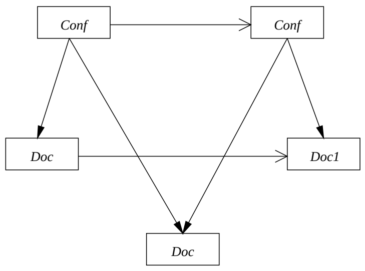

Fig. 3. A local graph consisting of one configuration with two documents and its revisions.

The refinement of a network graph $t_G : G \to NTG$ w.r.t. a network type graph NTG is formally defined by a functor from the small category $\mathbf{G}$ induced by the graph G into the category $\mathbf{Graph}(\mathcal{L})$ of $\mathcal{L}$-labeled graphs and label preserving graph morphisms. A *distributed graph* integrates topological and local state aspects.

Definition 2 (distributed graph). *Given a network type graph $t_{NTG} : NTG \to DiSy$ and a set $\mathcal{L}$ of labels. A* distributed graph *over G is a pair $\hat{G} = \langle t_G, \mathcal{G} \rangle$ where $t_G : G \to NTG$ is a network graph w.r.t. NTG and $\mathcal{G} : G \to G(\mathbf{Graph}(\mathcal{L}))$ is a graph morphism from G to the underlying graph $G(\mathbf{Graph}(\mathcal{L}))$ of the category $\mathbf{Graph}(\mathcal{L})$.*

For a distributed graph $\hat{G}$, the $\mathcal{L}$-graph $\mathcal{G}(v)$ for each v in G_V is called the *local state* of v. For each edge e in G_E, the graph morphism $\mathcal{G}(e)$ is called the *local graph morphism* for e. A morphism between distributed graphs $\hat{G}$ and $\hat{H}$ over G resp. H relates the network graphs by means of a graph morphism between G and H, and relates the local state of each node in G to a local state in H by means of a label-preserving graph morphism. It can be seen as a natural transformation from G to $H \circ f$.

Definition 3 (distributed morphism). *A* distributed morphism *between distributed graphs $\hat{G}$ over G and $\hat{H}$ over H is a pair $\hat{f} = \langle f, \tau \rangle$ where $f : G \to H$ is a graph morphism and $\tau : \mathcal{G} \to \mathcal{H}$ is a family of arrows $\{\tau(a) | a \in G_V\}$ such that*

- *for each node a in G, $\tau(a) : \mathcal{G}(a) \to \mathcal{H}(f(a))$ is a label preserving graph morphism and*
- *for each edge $e : i \to j$ in G, $\tau(j) \circ \mathcal{G}(e) = \mathcal{H}(f(e)) \circ \tau(i)$.*

Example 3. An example of a distributed graph is shown in Figure 4. The level refinement is given in the left-hand side of the figure. Additionally, the notation used in the following is shown. It shows the local states of network nodes and the local graph morphisms for network edges, but omits the explicit representation

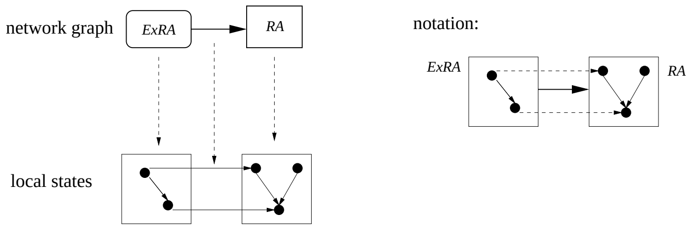

Fig. 4. A distributed graph and its notation.

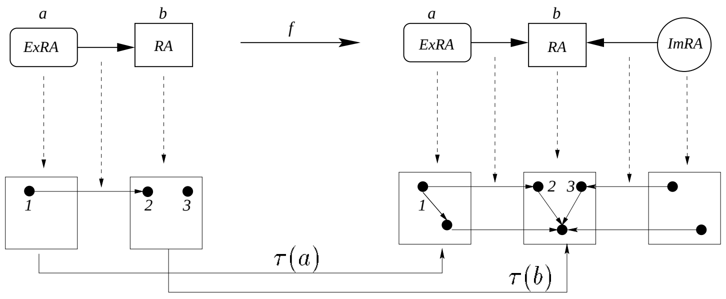

Fig. 5. A distributed morphism.

of the network graph itself and its refinement. However, a name at the local state indicates the network node is also found in a local state. The local system of type RA and the interface of type $ExRA$ are refined to labeled graphs that represent their local state (for the sake of readability we omit labels in the figures). The network edge is refined to a graph morphism. In the example of a distributed morphism $\hat{f} : \hat{G} \to \hat{H}$ in Figure 5, the graph morphism f and the local morphisms $\tau(a)$ and $\tau(b)$ map elements to elements with the same name.

For a given network type graph NTG and a label set $\mathcal{L}$, distributed graphs and distributed morphisms form a category $\mathbf{Distr}(NTG, \mathcal{L})$. Composition of distributed morphisms is defined componentwise for graph morphisms between network graphs and local graph morphisms. The identity for each distributed graph $\hat{G}$ is given by $\hat{id}_{\hat{G}} = \langle id_G, \tau \rangle$ where $\tau(a) = id_{\mathcal{G}(a)}$ for each a in G_V.

The pushout of two distributed morphisms $\hat{f} = \langle f, \tau^f \rangle : \hat{A} \to \hat{B}$ and $\hat{g} = \langle g, \tau^g \rangle : \hat{A} \to \hat{C}$ is constructed by constructing first the pushout of f and g in **Graph**. Then the pushout of $\tau^f(a)$ and $\tau^g(a)$ for each node a in A_V in category $\mathbf{Graph}(\mathcal{L})$ is constructed. The pushout object and the pushout morphisms for $\hat{f}$ and $\hat{g}$ in $\mathbf{Distr}(NTG, \mathcal{L})$ are then made up of these pushout components and the resulting unique pushout morphisms. However, this construction does not yield a pushout for all distributed morphisms as the counter examples in Figure 6 show.

In the example on the left, a node is added to a local source graph. In order to make the local graph morphism total, a node is inserted in the target graph

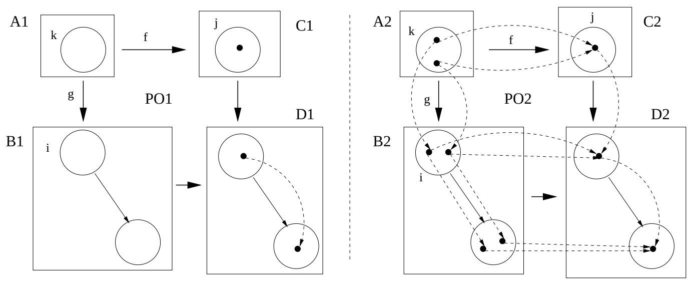

Fig. 6. Examples of a non componentwise pushout construction.

as well. The second example shows that the gluing of two nodes in a local source graph is propagated to the target graph in order to maintain the well-definedness of the local graph morphism.
We provide *locality conditions* ensuring that the pushout over two distributed morphisms can be constructed componentwise for the network and all pairs of local morphisms. (For the proof consider the corresponding proof in [2] for category $\mathbf{Mod}_C$ with C being category $\mathbf{Graph}(\mathcal{L})$.)

Definition 4 (locality conditions). *Two distributed morphisms*

$$\hat{f} = \langle f, \tau^f \rangle : \hat{A} \to \hat{B} \text{ and } \hat{g} = \langle g, \tau^g \rangle : \hat{A} \to \hat{C}$$

satisfy the locality conditions *if and only if*

- $f : A \to B$ *and* $g : A \to C$ *are injective,*
- *for each edge* $e \in B_E - f(A_E)$ *and for each node* $y \in A_V$, $f(y) = s(e)$ *implies* $\tau^g(y)$ *is bijective and*
- *for each edge* $e \in C_E - g(A_E)$ *and for each node* $y \in A_V$, $g(y) = s(e)$ *implies* $\tau^f(y)$ *is bijective.*

In the next section it is described how the pushout construction on distributed graphs can be used to model transformation rules along the lines of [8].

4 Distributed Graph Transformation

Distributed rules are given by a span of injective distributed morphisms. A transformation includes the transformation of the network graph as well as the local states. A local transformation is performed in each network node, which is preserved by the network transformation. The network as well as the local transformations are formulated as double-pushouts on graph morphisms. The local graph morphisms between transformed local graphs are induced as universal pushout morphisms. Furthermore, each network node and edge deleted, sees its local graphs and graph morphisms deleted as well. Creating a network node or edge is combined with the creation of a corresponding local graph or graph morphism. The result of a distributed graph transformation is again a distributed graph, since a distributed graph transformation can be characterized by a double-pushout ([8]).

Definition 5 (distributed rule). *A* distributed rule p *consists of a span*

$$(\hat{L} \overset{\hat{l}}{\leftarrow} \hat{I} \overset{\hat{r}}{\rightarrow} \hat{R})$$

of injective distributed morphisms.

A production p can be applied to a distributed graph $\hat{G}$, if there is an occurrence $\hat{m} = \langle m, \tau^m \rangle : \hat{L} \rightarrow \hat{G}$ of the left-hand side of the production in the distributed graph. The derivation of a distributed graph via a distributed rule is given by two pushouts in category $\mathbf{Distr}(NTG, \mathcal{L})$. In order to guarantee the existence and uniqueness of the pushout complement as well as the componentwise construction of the pushouts, the morphism $\hat{m}$ has to satisfy the so-called *distributed gluing condition.* This condition is satisfied by $\hat{m}$ if m is injective and satisfies the gluing condition for $(L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R)$ (defined e.g. in [1]), $\tau^m(l(x))$ satisfies the gluing condition for $(\mathcal{L}(l(x)) \overset{\tau^l(x)}{\leftarrow} \mathcal{I}(x) \overset{\tau^r(x)}{\rightarrow} \mathcal{R}(x))$ for all nodes $x \in I_V$ and $\hat{m}$ satisfies the *connection* and *network condition.* The connection condition is satisfied if whenever p deletes objects in some source local graph $\mathcal{G}(s(e))$ resp. in some target graph $\mathcal{G}(t(e))$, the local mapping $\mathcal{G}(e)$ must be changed correspondingly. If p adds new objects to a local graph $\mathcal{G}(s(e))$, the local mapping $\mathcal{G}(e)$ must be extended, too. The network condition is satisfied if a network node is only deleted together with its entire local graph and a network edge e can be only deleted if the local graph $\mathcal{G}(s(e))$ is completely mentioned in the rule, so that the local mapping can also be deleted. A new edge e is only inserted at an existing node v becoming the source node of e if $\mathcal{G}(v)$ is completely mentioned in the rule, i.e. the local morphism $\mathcal{G}(e)$ is completely specified.

Definition 6 (direct derivation). *Given a distributed rule p and a distributed morphism $\hat{m} : \hat{L} \rightarrow \hat{G}$ that satisfies the distributed gluing condition, then a* direct derivation $\hat{G} \overset{p,\hat{m}}{\Longrightarrow} \hat{H}$ *via p is given by two pushouts (1) and (2) in category* $\mathbf{Distr}(NTG, \mathcal{L})$.

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

Example 4. In Figure 7, a rule is given modeling the check in of a document from the workspace into the revision archive as a new revision. This document was exported from the revision archive and imported from the workspace where it was changed. Then it is exported by the workspace and imported by the revision archive where the changed document becomes a successive revision of the version originally used for export.

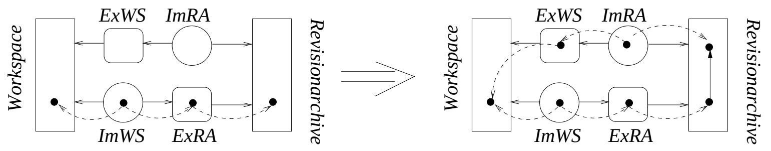

Fig. 7. Inserting a revision into the archive.

We omit the intermediate graph in the notation of rules. Only the left-hand and the right-hand side of a rule are shown. The span can be achieved from this notation due to the position of nodes and edges in the graphs.

5 Local Views on a Distributed System

Distributed rules as defined in Def. 5 allow one to specify actions within a distributed system affecting several local systems in one rule application. An example is the rule in Figure 7, where an action affects the workspace and the revision archive. Thus, distributed actions are specified in a global view. In the early stage of system development, a global view on the entire distributed system is desirable to get an overview. Once the network structure and its reconfiguration possibilities as well as the interfaces are fixed, the system developers are supposed to take a local view on network nodes and local system parts running on them. Therefore, we are going to restrict distributed rules to so-called *local view rules*. A *local view graph* for a local system in the distributed system specifies the visible parts of the local system. It contains the local states of the local system itself and of export and import interfaces to which the local system has connections. While it is natural that a local system knows the exports from where it imports, it is not as clear that it knows the local states of imports that are connected to its exports. This information can be advantageously used to inform the imports immediately when the export changes which is not possible otherwise.

Definition 7 (local view). *Let $t_{NTG} : NTG \to DiSy$ be a network type graph as defined in Def. 1. Let $\hat{G} = \langle t_G, \mathcal{G} \rangle$ over G be a distributed graph and v a node in G_V such that $t_{NTG}(t_G(v)) = LoSy$. An interface node $w \in G_V$, i.e. $t_{NTG}(t_G(w)) = Imp$ or $t_{NTG}(t_G(w)) = Exp$, is an* interface for *v if w is directly connected to v, i.e. there is an edge $e \in G_E$ such that $s^G(e) = w$ and $t^G(w) = v$. Otherwise, the interface w is a* remote interface w.r.t. *v.*

The distributed graph $\hat{G}$ is called local view graph *w.r.t. v if G is connected and there does not exist a v' in G_V such that $v' \neq v$ and $t_{NTG}(t_G(v')) = LoSy$.*

To include remote import interfaces into the view of a local system is also formally motivated. If remote import interfaces would not be visible for a local system, the local system cannot delete any object of its own export interfaces as long as other systems import these objects, what is formally forced by the distributed gluing condition.

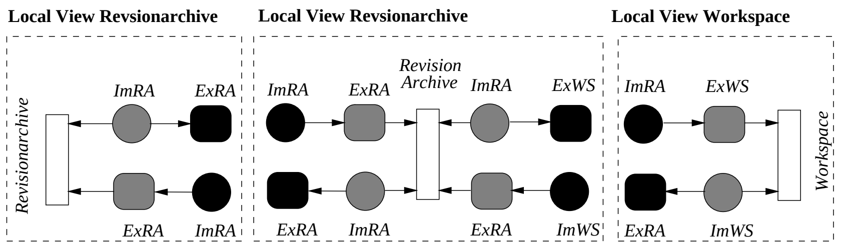

Fig. 8. Three local view graphs for Figure 2.

Example 5. Taking Figure 2, three local systems are shown which yields three local view graphs (Figure 8). In these graphs the local systems are white, interfaces are grey and remote interfaces are filled black.

Given a local system, we now introduce distributed rules for the local system. The knowledge of the distributed system's state restricted to the local view of the local system is sufficient to apply those rules. We distinguish rules for transforming the local states of a local system, rules for creating new interfaces or new local systems and rules for the deletion of the local system or parts of it.

Definition 8 (local transformation rule). *A* local transformation rule *is a distributed rule* $p = (\hat{L} \xleftarrow{\hat{l}} \hat{I} \xrightarrow{\hat{r}} \hat{R})$ *where*

- *the network graph morphisms l and r are the identity on I, i.e. $l = r = id_I$ and*
- *$\hat{L}$, $\hat{I}$ and $\hat{R}$ are local view graphs w.r.t. $v \in I_V$.*

The definition of a local transformation rule ensures that the network graph remains unchanged and that only local states are transformed. The transformation of local states includes deletion, preservation and creation of objects in the state. With local transformation rules different kinds of actions can be described. If remote interfaces are not involved, local actions are described. Moreover, asynchronous or synchronous actions can be modeled. In these cases the remote interfaces are just read or, in the synchronous case, are allowed to be changed. Here, the same actions have to be performed by those local systems which own these interfaces.

In order to additionally transform the network graph, *local deletion* and *local creation rules* are introduced. A local system is allowed to delete only itself and its interfaces, but not remote interfaces. By means of the *local creation rule* new local systems with their interfaces can be created.

Definition 9 (local creation rule and local deletion rule). *Rule p is a* local creation rule *if*

- *$\hat{L}$ does not contain any local system node,*
- *$\hat{R}$ is a local view graph w.r.t. v in R_V,*

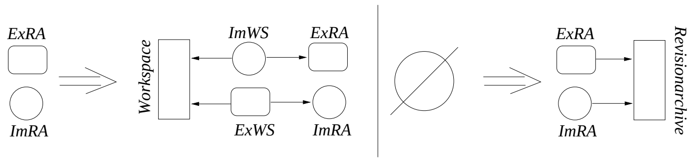

Fig. 9. Local creation rules for a workspace (left) and for an archive (right).

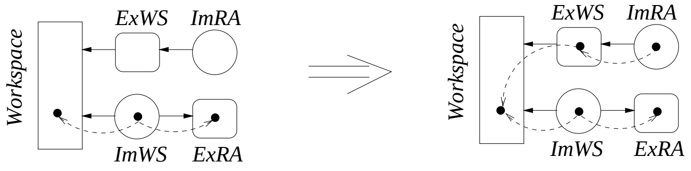

Fig. 10. The workspace exports a previously imported document again.

– *$\hat{l}$ is the identity on $\hat{I}$ and $\hat{I} = \hat{L}$ are subgraphs of $\hat{R}$,*
– *all remote interface nodes for v in L are preserved, i.e. if $w \in L_V$ is an remote interface for v then $w \in l_V(I)$, and*
– *no new remote interface node is created, i.e. there is not any interface node v which is not in the codomain of r.*

Rule p is a local deletion rule *if it the inverse rule of a local creation rule.*

Definition 10 (local view rule and local view grammar). *Given a network type graph $t_{NTG} : NTG \to DiSy$ and a local system node x in NTG, i.e. $t_{NTG}(x) = LoSy$, a* local view rule *for x is each local transformation, local deletion and local creation rule such that each local system node v occuring in the rule is of type x, i.e. $t_L(v) = x$ or $t_R(v) = x$. A* local view grammar *$LGG(x) = P(x)$ for x is given by a set of local view rules for x.*

Example 6. Two different kinds of local view grammars are necessary for the running example: one handles rules concerning workspaces, the second one is for revision archives. Local view rules for the workspace must include rules for checking in from the workspace into a revision archive as shown by the local transformation rule in Fig. 10. A document previously imported by the workspace, is exported again.[3] In Figure 9, a local creation rule for a workspace is given on the left. On the right the local creation rule for revision archives is shown.

In the local view grammar of the revision archive a rule corresponding to Fig. 10 can be found. It is shown in Figure 11 and used to import a revision exported by the workspace and adding it as a successor in the revision archive.

The application of the local rules in Fig. 11 and Fig. 10 on the same document should lead to the same result as the application of the global rule in Fig. 7, i.e. it must be ensured somehow that a changed document in the workspace is inserted

[3] The case where a new document is created is omitted here.

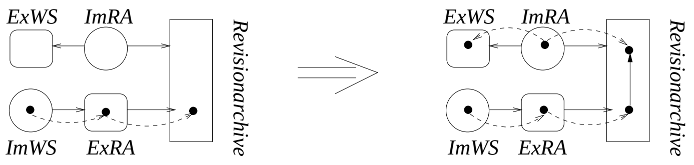

Fig. 11. The revision archive requests a document from the workspace to import it.

into the revision archive as a direct successor after the revision from which it was originally checked out. To achieve this, the local rules have to be synchronized.

6 Synchronizing Local Rules for Communication between Local Systems

This section is concerned with communication between local views. Whereas asynchronous communication is modeled by sequential application of local view rules, synchronous communication is expressed by amalgamating local view rules over a common subrule in category $\mathbf{Distr}(NTG, \mathcal{L})$ [5]. While the asynchronous communication describes the usage of a service already available in some interface, the synchronization models a kind of service request for objects not already available in the interface (cf. the rules in Figures 10 and 11).

For amalgamation so called *interface subrules* are necessary. An interface subrule p_s of a local view rule p for a local system v is a distributed rule, where all graphs of p_s are subgraphs of the corresponding graphs in p such that they contain only interfaces of v, but not v itself. Since we intend to describe synchronization via interface subrules and communication takes place over export and import interfaces, we require *handles* of export and import interfaces in the network graphs of the subrule. More exactly, a handle consists of an export interface and an import interface connected by an edge. This requirement prohibits export interfaces without an import interface connected, and vice versa. Considering local deletion and creation rules which have to be synchronized, handles can be found only on the left or on the right-hand side of an interface rule.

Definition 11 (interface subrule). *Let $p_s = (\hat{L}_s \xleftarrow{\hat{l}_s} \hat{I}_s \xrightarrow{\hat{r}_s} \hat{R}_s)$ be a distributed rule such that*

- *L_s, I_s and R_s contain interface nodes only,*
- *there is an $X \in \{L, I, R\}$such that for each export node $v \in X_s$, i.e. $t_{NTG}(t_X(v)) = Exp$, there is an import node $v' \in X_s$, i.e. $t_{NTG}(t_X(v')) = Imp$, and an edge $e : v' \to v$ and*
- *there is an $X \in \{L, I, R\}$ such that for each import node $v' \in X_s$, i.e. $t_{NTG}(t_X(v')) = Imp$, there is an export node $v \in X_s$, i.e. $t_{NTG}(t_X(v)) = Exp$, and an edge $e : v' \to v$.*

Then, p_s is an interface subrule *of* $p = (\hat{L} \xleftarrow{\hat{l}} \hat{I} \xrightarrow{\hat{r}} \hat{R})$ *if there are injective distributed morphisms* $\hat{in}_L : \hat{L}_s \to \hat{L}$, $\hat{in}_I : \hat{I}_s \to \hat{I}$ *and* $\hat{in}_R : \hat{R}_s \to \hat{R}$, *called* subrule embeddings *such that* $\hat{in}_L \circ \hat{l}_s = \hat{l} \circ \hat{in}_I$ *and* $\hat{in}_R \circ \hat{r}_s = \hat{r} \circ \hat{in}_I$.

A distributed rule has to be synchronized with others, if it contains one or more remote interfaces changed by the rule. This part of the rule just reflects what has to be done by remote systems within their interfaces (cf. the rules in Figures 10 and 11). To synchronize two rules an interface subrule is needed which contains at most the intersection of the two rules, but is also allowed to be smaller. If the distributed rule resulting from a synchronization step still contains remote interfaces, it can and has to be further synchronized. A distributed rule not fully synchronized is not applicable because of the distributed gluing condition (namely the connection condition).

Definition 12 (synchronization of rules). *Two distributed rules p_1 and p_2 are* synchronized *w.r.t. s if s is an interface subrule embedding of p_1 and p_2 consisting of interface subrule p_s and the subrule embeddings $\hat{in}_{X_1}$, $\hat{in}_{X_2}$ satisfy the locality conditions in Def. 4 for $X \in \{L, I, R\}$. Moreover, for each interface w in $\hat{X}_s$, either w is a remote interface* for all *local systems in $\hat{X}_1$ or w is a remote interface* for all *local systems in $\hat{X}_2$.*

The synchronized rule *of p_1 and p_2 via s is given by the amalgamated rule* $p_1 \oplus_s p_2 = (\hat{L} \xleftarrow{\hat{l}} \hat{I} \xrightarrow{\hat{r}} \hat{R})$. *The distributed graphs $\hat{L}$, $\hat{I}$ and $\hat{R}$ are the pushout objects given by the pair $\hat{in}_{L_1}$ and $\hat{in}_{L_2}$, the pair $\hat{in}_{I_1}$ and $\hat{in}_{I_2}$ and the pair $\hat{in}_{R_1}$ and $\hat{in}_{R_2}$ in* $\mathbf{Distr}(NTG, \mathcal{L})$. *The rule morphisms $\hat{l}$ and $\hat{r}$ are obtained as the universal pushout morphisms.*

Example 7. Regarding the check-in of a revision from the workspace into the revision archive, its intended semantics is described as a revision which becomes a successor of that revision in the archive from which it was checked out. In order to achieve this, we have to synchronize the local view rule for a workspace in Figure 10 and the local view rule for a revision archive in Figure 11. These rules are not applicable separately because of the distributed gluing condition. E.g. the revision archive rule in Figure 11 is not applicable without synchronization, because of the insertion of the revision in the export interface of the workspace. Because of the local creation rule for a workspace there is always an edge from an export of a workspace to the workspace itself. Therefore, the revision inserted has to be assigned to a revision in the workspace as well, which, however, is not specified by the rule.

The synchronized rule of the rules in Figure 11 and Figure 10 using the interface subrule in Figure 12 looks like that in Fig. 7.

Next, we construct all possible rules for a given set of local view rules used to model the operational semantics of the distributed system. Here, the rules are synchronized as much as possible, i.e. the largest interface subrule embeddings are chosen for synchronization. An interface subrule embedding is one of the largest if there no other interface subrule which contains the rule of the first

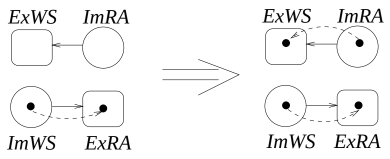

Fig. 12. Interface subrule for Fig. 10 and 11 to construct the rule in Fig. 7

embedding as subrule. Since an interface subrule may be the empty rule, the largest interface subrule always exists.

Definition 13 (set of synchronized rules). *Let $LVRules$ be a set of local view rules, then, $Syn(LVRules)$ is the smallest set of rules such that*

- *$p \in Syn(LVRules)$ for all $p \in LVRules$, and*
- *$p_1 \oplus_s p_2 \in Syn(LVRules)$ for all $p_1, p_2 \in Syn(LVRules)$ where s is one of the largest interface subrule embedding of p_1 and p_2, i.e. there is no interface subrule embedding s' of p_1 and p_2 such that p_s is a subrule of $p_{s'}$.*

A *distributed system grammar* w.r.t. a network type graph $t_{NTG} : NTG \rightarrow DiSy$ consists of a global start graph and a local view grammar for each local system type in NTG, i.e. for each v in NTG_V such that $t_{NTG}(t_G((v)) = LoSy$. For the operational semantics the set of all rules that can be constructed by the local view rules according to Definition 13 are considered.

Definition 14 (distributed system grammar). *Let $t_{NTG} : NTG \rightarrow DiSy$ be a network type graph and $LS = \{v \in NTG_V | t_{NTG}(v) = LoSy\}$ be the set of all local system types in NTG. A* distributed system grammar *w.r.t. NTG is a pair $DSG(NTG) = \langle \hat{G}_0, (LGG(v))_{v \in LS} \rangle$ where $\hat{G}_0$ is a distributed graph, $LGG(v) = P(v)$ is a local view grammar for each $v \in LS$.*

The operational semantics *for a distributed system grammar $DSG(NTG)$ is given by the set of all distributed derivations starting at $\hat{G}_0$ using rules of $Syn(\bigcup_{v \in LS} P(v))$.*

Example 8. The set of synchronized rules $SynRules$ of our case study contains for example the synchronized rule in Figure 7 created by the synchronization of the local view rules in Figures 11 and 10. It may also contain the corresponding local view rules, but these are never applicable due to the distributed gluing condition. Also the rules in Figure 9 should be in $SynRules$.

7 Conclusion

In this paper, we introduced local views on distributed graph transformation. A local view is concerned with one local system, its import and export interfaces, and remote import and export interfaces to which the local system may have connections. Local systems can communicate asynchronously by simply applying local view rules sequentially, or synchronously by constructing amalgamated distributed rules from local view rules. The concepts of local views are presented on graphs without attributes. We expect that they can be directly lifted to attributed graphs. Moreover, application conditions for rules should be integrated to support a convenient specification of distributed systems by distributed graph transformation. The integration – formally as well as informally – of graph transformation with attributes and application conditions has been done in [7]. We use the amalgamated rule construction to describe the synchronization between local view activities. In general, the resulting rules are not local view rules anymore, but distributed rules in a global setting. For the operational semantics definition of a distributed system all possible synchronizations of rules are computed.

References

1. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. *Handbook of Graph Grammars and Computing by Graph Transformations. Vol. I: Foundations*, chapter Algebraic Approaches to Graph Transformation Part I: Basic Concepts and Double Pushout Approach. World Scientific, 1997.
2. M. Koch. *Integration of Graph Transformation and Temporal Logic for the Specification of Distributed Systems*. PhD thesis, Technische Universität Berlin, FB 13, 1999. to defend.
3. Manuel Koch. Bedingte verteilte Graphtransformation und ihre Anwendung auf verteilte Transaktionen. Technical Report 97-11, TU Berlin, 1997.
4. Manuel Koch and Gabriele Taentzer. Distributing Attributed Graph Transformations. In *Proc. Workshop on "General Theory of Graph Transformation Systems", Bordeaux*, 1997.
5. G. Taentzer. Parallel high-level replacement systems. *Theoretical Computer Science*, (186), 1997.
6. G. Taentzer, I. Fischer, M. Koch, and V. Volle. *Handbook of Graph Grammars and Computing by Graph Transformations*, volume III, chapter Distributed Graph Transformation with Application to Visual Design of Distributed Systems. World Scientific, 1998. to appear.
7. G. Taentzer, I. Fischer, M Koch, and V. Volle. Visual design of distributed systems by graph transformation. In G. Rozenberg, U. Montanari, H. Ehrig, and H.-J. Kreowski, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency and Distribution*. World Scientific, 1999. to appear.
8. Gabriele Taentzer. *Parallel and Distributed Graph Transformation: Formal Description and Application to Communication-Based Systems*. PhD thesis, TU Berlin, 1996. Shaker Verlag.
9. Karsten Victor Volle. Verteilte Konfigurationsverwaltung: *COMAND*. Technical report, Basys GmbH, Am Weichselgarten 4, 91058 Erlangen, 1997.