Categorical Foundations of Distributed Graph Transformation

Hartmut Ehrig[1], Fernando Orejas[2], and Ulrike Prange[1]

[1] Technical University of Berlin, Germany
{ehrig, uprange}@cs.tu-berlin.de
[2] Technical University of Catalonia, Spain
orejas@lsi.upc.edu

Abstract. A distributed graph (N, D) consists of a network graph N and a commutative diagram D over the scheme N which associates local graphs $D(n_i)$ and graph morphisms $D(e) : D(n_1) \to D(n_2)$ to nodes n_1, n_2 and edges $e : n_1 \to n_2$ in N.

Although there are several interesting applications of distributed graphs and transformations, even the basic pushout constructions for the double pushout approach of distributed graph transformation could be shown up to now only in very special cases.

In this paper we show that the category of distributed graphs can be considered as a Grothendieck category over a specific indexed category, which assigns to each network N the category of all diagrams D of shape N. In this framework it is possible to give a free construction which allows to construct for each diagram D_1 over N_1 and network morphism $h : N_1 \to N_2$ a free extension $F_h(D_1)$ over N_2 and to show that the Grothendieck category is complete and cocomplete if the underlying category of local graphs has these properties.

Moreover, an explicit construction for general pushouts of distributed graphs is given. This pushout construction is based on the free construction. The non-trivial proofs for free constructions and pushouts are the main contributions of this paper and they are compared with the special cases known up to now.

1 Introduction

When modelling computation by means of (standard) graph transformation, a graph is supposed to denote the (centralized) state of a given system, and computation steps are modelled as transformations of this graph by means of some productions. To model distributed computation, where the state of the given system is not monolithic, G. Taentzer [1] introduced an extension of graph transformation called distributed graph transformation. The idea is to consider that, on one hand, a graph N (the *network graph*) describes the topology of the given system and, on the other, that the global state is, in some sense, partitioned along that graph. In particular, this is done associating to every node n in N a graph G_n that denotes the local state at this node, and to every edge $e : n \to n'$ in N a graph morphism $h_e : G_n \to G_{n'}$. These graph morphisms allow one

A. Corradini et al. (Eds.): ICGT 2006, LNCS 4178, pp. 215–229, 2006.

to describe the shared parts of the local states. Formally, then, a distributed graph is just a functor from the network graph into the category of graphs. In this context, (distributed) graph transformation is defined adapting the double-pushout approach to the (functor) category of distributed graphs.

The practical relevance of distributed graph transformation has been demonstrated in [2, 3], where this approach is used to keep coherence between models of different views. This allows an integrated management of modifications in the code and in the global UML model underlying a software artifact. Using distributed graph transformation we can define in a uniform way different kinds of computation steps. For instance, we can describe not only computations that occur in a single location (i.e. in the graph associated to a given node), but computations that occur simultaneously in several locations that are synchronized through the shared parts of the states involved. Moreover, we can also define transformations on the network, for instance allowing us some forms of refactoring. In some sense, this approach is related to Community (see, e.g. [4]), where the local states are tuples rather than graphs, and Goguens General Systems Theory [5].

Unfortunately, the basic constructions for defining distributed graph transformation as presented in [1] depend on some ad-hoc conditions that, on one hand, limit the power of the approach and, on the other hand, make it difficult to generalize the approach to cases where the states are not modelled as basic graphs, but as attributed graphs or some other kind of arbitrary structures [6, 7]. In particular, even the basic pushout constructions for the double pushout approach of distributed graph transformation could be shown up to now only in very special cases.

In this paper we provide categorical foundations for distributed graph transformation that allow us to provide the basic constructions with full generality. In particular, we generalize distributed graphs to distributed objects, where the local diagrams are not necessarily graphs, but consist of objects and morphisms in a certain category $\mathbf{C}$. Then we show that the category of distributed objects can be considered as a Grothendieck category over a specific indexed category, which assigns to each network N the category of all diagrams D of shape N in $\mathbf{C}$. In this framework it is possible to give a free construction which allows to construct for each diagram D_1 over N_1 and network morphism $h : N_1 \to N_2$ a free extension $F_h(D_1)$ over N_2 and to show that the Grothendieck category is complete and cocomplete if the underlying category of local objects has these properties. Moreover, an explicit construction for general pushouts of distributed objects is given. This pushout construction is based on the free construction. The non-trivial proofs for free constructions and pushouts are the main contributions of this paper and they are compared with the special cases known up to now.

The paper is organized as follows. In section 2 we study the category of distributed objects and present the free diagram extensions. Section 3 is dedicated to the category of distributed objects as a Grothendieck category. In section 4 we show the explicit construction of pushouts of distributed objects and, through an example, how these pushouts are used in distributed graph transformation. In

section 5 we introduce persistent morphisms and discuss their role with respect to strongly componentwise pushouts as considered in [1]. Finally, in section 6 we draw some conclusions.

We assume the reader to be familiar with the basic notions of category theory, as presented in, e.g., [4, 8, 9].

2 The Category DisC and Free Diagram Extensions

A distributed graph representing the distributed state of a system can be described, on one hand, by a graph N (the *network graph*) defining the topology of the object and, on the other, associating to every node n in N a graph $D(n)$ that denotes the local state at this node, and to every edge $e : n \to n'$ in N a graph morphism $D(e) : D(n) \to D(n')$. In particular, it is assumed that these graph morphisms describe the shared parts of the local states (see example 1).

Formally, in categorical terms, this means that a distributed graph (N, D) consists of the network graph N and a diagram $D : N \to \mathbf{Graph}$ which associates local graphs $D(n_i)$ and graph morphisms $D(e) : D(n_1) \to D(n_2)$ to nodes n_1, n_2 and edges $e : n_1 \to n_2$ in N. However, if we consider that states are not specifically modelled by basic graphs, but by some other kind of structure (as, e.g., typed attributed graphs) then we can easily generalize this definition. In particular, we can consider that a distributed object is a diagram $D : N \to \mathbf{C}$, where $\mathbf{C}$ is an arbitrary category. Obviously, we may require $\mathbf{C}$ to satisfy some specific properties.

In addition, we require that a diagram $D : N \to \mathbf{C}$ is commutative. We believe that this should be a consequence of assuming that the morphisms associated to the edges (or to the paths) in N denote the shared parts of the distributed states. In particular, suppose that we have two paths p_1 and p_2 from a node n into n'. According to our intuition, this means that we can consider that for the state at node n, $D(n)$, there is a (not necessarily injective) image $D(p_1)(D(n))$ of $D(n')$, and similarly for $D(p_2)$. Now, if $D(p_1)$ would denote a different morphism from $D(p_2)$, then, it would mean that we could also identify $D(p_1)(D(n))$ and $D(p_2)(D(n))$, which are different parts of the state at n'.

A graph $G = (V, E, s, t)$ consists of a set of nodes (or vertices) V and a set of edges E, with functions $s, t : E \to V$ assigning a source and target node to each edge, respectively. This concept has been extended to many different kinds of graphs, like hypergraphs, labelled graphs, typed and/or attributed graphs, which we do not define explicitly. Instead, we assume to have some category $\mathbf{C}$ and present the theory of distributed objects on the categorical level, which can be instantiated by various graphs and graph-like structures.

Given a graph $G = (V, E, s, t)$, it can be interpreted as the scheme of a category. This means, the reflexive and transitive closure of G is a category with objects V. Vice versa, a category $\mathbf{C}$ can be seen as a (possibly infinite) graph. In the following, we switch between both concepts as needed in the particular context. As a consequence, we use the terms functor and diagram as synonyms in this context.

Definition 1 (path morphism and commutative functor). *Given a graph N, a functor $D : N \to \mathbf{C}$ (interpreting N as a category) and a path $p : n \xrightarrow{e_1} ... \xrightarrow{e_k} n'$ in N, we define the path morphism $D(p) : D(n) \to D(n')$ of D along p as $D(p) = D(e_k) \circ ... \circ D(e_1)$. For the empty path $\epsilon_n : n \xrightarrow{0} n$, $D(\epsilon_n) = id_{D(n)}$.*

A functor $D : N \to \mathbf{C}$ is commutative, if for any two paths $p_1, p_2 : n \xrightarrow{*} n'$ in N we have $D(p_1) = D(p_2)$.

Remark 1. If D is commutative, we obviously have $D(c) = id_{D(n)}$ for each circle $c : n \xrightarrow{*} n$ in $\mathbf{C}$. For paths $p : n \xrightarrow{*} n'$, $p' : n \xrightarrow{*} n' \xrightarrow{f} n''$ and $p'' : n'' \xrightarrow{f'} n \xrightarrow{*} n'$ it follows that $D(p') = D(f) \circ D(p)$ and $D(p'') = D(p) \circ D(f')$.

We can now define the category of distributed objects:

Definition 2 (distributed object and distributed morphism). *Given a category $\mathbf{C}$, a distributed object (N, D) over $\mathbf{C}$ (or just a distributed object, if $\mathbf{C}$ is implicit in the given context) consists of a graph N, called network graph, and a commutative functor $D : N \to \mathbf{C}$, called diagram functor.*

A distributed morphism over $\mathbf{C}$ (or just a distributed morphism, if $\mathbf{C}$ is implicit in the given context), $f = (f_N, f_D) : (N_1, D_1) \to (N_2, D_2)$, consists of a graph morphism $f_N : N_1 \to N_2$ and a natural transformation $f_D : D_1 \to D_2 \circ f_N$.

Distributed objects and distributed morphisms over $\mathbf{C}$ form the category **DisC**.

In particular, we may notice that our previous definition implicitly associates to every network graph N a category consisting of all the commutative functors from N to $\mathbf{C}$. This construction can be extended to a functor.

Definition 3 (functor Diag). *The functor Diag :* $\mathbf{Graphs^{op}} \to \mathbf{Cat}$ *is defined by*

- *for a graph N, $Diag(N) = comFunct[N, \mathbf{C}]$, the category of commutative functors (diagrams) $D : N \to \mathbf{C}$,*
- *for a graph morphism $f : N \to N'$ in* **Graphs**, *$Diag(f)(D' : N' \to \mathbf{C}) = D' \circ f : N \to \mathbf{C}$ and $Diag(f)(t : D'_1 \to D'_2) = t \circ f$.*

In order to construct pushouts and colimits in **DisC** in section 3 and 4 we need to show that each commutative diagram $D_1 : N_1 \to \mathbf{C}$ has a free extension $D_2 : N_2 \to \mathbf{C}$ for each network morphism $h : N_1 \to N_2$. In fact, if $\mathbf{C}$ is cocomplete then each network morphism has an associated free construction (extension), leading to a free functor left adjoint to $Diag(h)$. Note, that we only need finite cocompleteness of $\mathbf{C}$ if all network graphs N are finite.

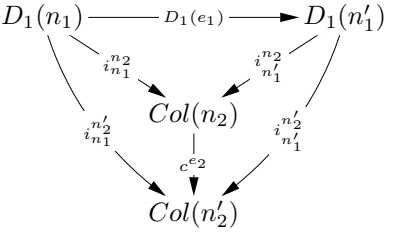
Theorem 1. *If $\mathbf{C}$ is cocomplete then for all network morphisms $h : N_1 \to N_2$ there is a functor $F_h : comFunct[N_1, \mathbf{C}] \to comFunct[N_2, \mathbf{C}]$, that is free with respect to $Diag(h)$.*

Construction. We have to show that there is a free construction $(D_2, u_h^{D_1})$ with $D_2 : N_2 \to \mathbf{C}$ and $u_h^{D_1} : D_1 \to Diag(h)(D_2)$ for each diagram $D_1 : N_1 \to \mathbf{C}$.

For $n_2 \in N_2$ define $N_1(n_2)$ as the full subgraph of N_1 induced by the node set $V(N_1(n_2)) = \{n_1 \in N_1 \mid \exists\, path\ p : h(n_1) \xrightarrow{*} n_2 \in N_2\}$.

The restriction $D_1|_{N_1(n_2)} : N_1(n_2) \to \mathbf{C}$ is a functor. Let $(Col(n_2), (i^{n_2}_{n_1})_{n_1 \in N_1(n_2)})$ be the colimit of $D_1|_{N_1(n_2)}$, with $i^{n_2}_{n_1} : D_1(n_1) \to Col(n_2)$ and $i^{n_2}_{n'_1} \circ D_1(e_1) = i^{n_2}_{n_1}$ for all $e_1 : n_1 \to n'_1 \in N_1(n_2)$.

For an edge $e_2 : n_2 \to n'_2$ we have $N_1(n_2) \subseteq N_1(n'_2)$ and therefore $(Col(n'_2), (i^{n'_2}_{n_1})_{n_1 \in N_1(n_2)})$ is a cocone of $D_1|_{N_1(n_2)}$. This means that there exists a unique morphism $c^{e_2} : Col(n_2) \to Col(n'_2)$ with $c^{e_2} \circ i^{n_2}_{n_1} = i^{n'_2}_{n_1}$ for all $n_1 \in N_1(n_2)$.



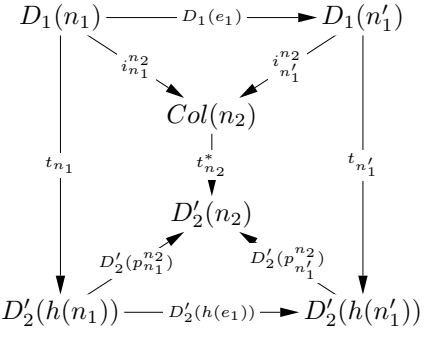
Define $F_h(D_1) = D_2 : N_2 \to \mathbf{C}$ by $D_2(n_2) = Col(n_2)$ and $D_2(e_2) = c^{e_2}$, and $u^{D_1}_h = (i^{h(n_1)}_{n_1})_{n_1 \in N_1}$. □

Proof idea. From the construction it follows that D_2 is a commutative functor and $u^{D_1}_h$ is a well-defined natural transformation.

For a distributed object (N_2, D'_2) and a natural transformation $t : D_1 \to D'_2 \circ h$ we have to show that there is a unique natural transformation $t^* : D_2 \to D'_2$ with $(t^* \circ h) \circ u^{D_1}_h = t$.

For a node $n_2 \in N_2$ and $n_1 \in N_1(n_2)$, by construction there exists a path $p^{n_2}_{n_1} : h(n_1) \xrightarrow{*} n_2$ in N_2. Since D'_2 is commutative, $D'_2(p^{n_2}_{n_1})$ is independent from the chosen path (if there is more than one). Then $(D'_2(n_2), (D'_2(p^{n_2}_{n_1}) \circ t_{n_1})_{n_1 \in N_1(n_2)})$ is a cocone of $D_1|_{N_1(n_2)}$ and there exists a unique morphism $t^*_{n_2} : D_2(n_2) \to D'_2(n_2)$ with $t^*_{n_2} \circ i^{n_2}_{n_1} = D'_2(p^{n_2}_{n_1}) \circ t_{n_1}$ for all $n_1 \in N_1(n_2)$.

$t^* = (t^*_{n_2})_{n_2 \in N_2}$ is a natural transformation, and the uniqueness follows from the uniqueness of its components.



For $n_1 \in N_1$ we have $t^*_{h(n_1)} \circ i^{h(n_1)}_{n_1} = D'_2(p^{h(n_1)}_{n_1}) \circ t_{n_1} = t_{n_1}$, therefore $(t^* \circ h) \circ u^{D_1}_h = t$, because $p^{h(n_1)}_{n_1} : h(n_1) \xrightarrow{*} h(n_1)$ implies $D'_2(p^{h(n_1)}_{n_1}) = id$ (see [10] for more detail). □

Example 1. Consider the network graphs N_1 and N_2 shown in Fig. 1 on the left hand side, the inclusion $h : N_1 \to N_2$ and the diagram $D_1 : N_1 \to$ **Graphs** shown in Fig. 1 on the right hand side. In the figure on the right, the thick lines represent the network structure, and the diagram morphism is indicated by the small numbers.

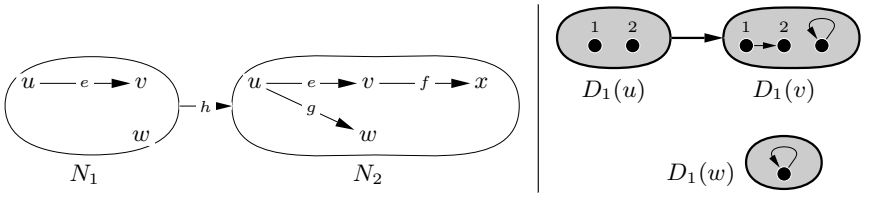


Fig. 1. Two network graphs and a diagram

From the construction we get for each node $n_2 \in N_2$ the corresponding subgraphs $N_1(n_2)$ of N_1, where $N_1(u)$ contains only the node u, $N_1(v)$ contains the nodes u and v and the edge e, $N_1(x) = N_1(v)$ and $N_1(w)$ contains the nodes u and w. The corresponding colimit constructions lead to the following free construction diagram $D_2 : N_2 \to$ **Graphs** over $D_1 : N_1 \to$ **Graphs** with $D_2(u) = Colim(D_1|_{N_1(u)}) = D_1(u)$, and similarly $D_2(v) = D_1(v)$, $D_2(x) = D_1(v)$ and $D_2(w) = D_1(u) \dot{\cup} D_1(w)$ as shown in Fig. 2.

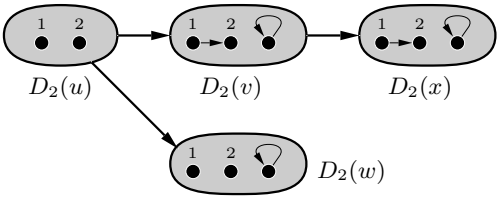
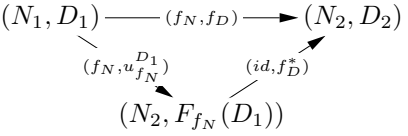


Fig. 2. The corresponding free construction

In section 4 we will use the following decomposition.

Proposition 1. *A distributed morphism $f = (f_N, f_D) : (N_1, D_1) \to (N_2, D_2)$ can be decomposed into the following diagram, where $f^*_D : F_{f_N}(D_1) \to D_2$ is the adjunction morphism associated to the morphism $f_D : D_1 \to Diag(f_N)(D_2)$.*



Proof. This follows directly from the free construction (Theorem 1). □

3 DisC as a Grothendieck Category

In this section we show that the category **DisC** can be considered as a Grothendieck category, because there are general categorical results how to construct limits and colimits in Grothedieck categories [4, 11, 12]. We start by defining indexed categories.

Definition 4 (indexed category). *Given a category* $\mathbf{I}$*, called index category, an indexed category is a functor* $F : \mathbf{I^{op}} \to \mathbf{CAT}$*, where* **CAT** *denotes the category of all categories.*

Definition 5 (Grothendieck category). *The Grothendieck category* $\mathbf{Gr}(\mathbf{F})$ *of an indexed category* F *has as objects pairs* (i, A) *with* $i \in \mathbf{I}$ *and* $A \in F(i)$*. A morphism* $(i, A) \to (i', A')$ *is a pair* (f, g) *with* $f : i \to i' \in \mathbf{I}$ *and* $g : A \to F(f)(A') \in F(i)$*.*

Given morphisms $(f, g) : (i, A) \to (i', A')$ *and* $(f', g') : (i', A') \to (i'', A'')$*, the composition is defined by* $(f' \circ f, F(f)(g') \circ g)$*. For an object* (i, A)*, the identity* $id_{(i,A)}$ *is given by* (id_i, id_A)*.*

According to [11] we have:

Fact 1. *Let* $F : \mathbf{I^{op}} \to \mathbf{CAT}$ *be an indexed category with Grothendieck category* $\mathbf{Gr}(\mathbf{F})$*. If* $\mathbf{I}$ *and* $F(i)$ *are complete for all* $i \in \mathbf{I}$*, and* $F(f)$ *is continuous for all* $f : i \to j \in \mathbf{I}$ *then also* $\mathbf{Gr}(\mathbf{F})$ *is complete. If* $\mathbf{I}$ *and* $F(i)$ *are cocomplete for all* $i \in \mathbf{I}$*, and* $F(f)$ *has a left adjoint for all* $f : i \to j \in \mathbf{I}$ *then also* $\mathbf{Gr}(\mathbf{F})$ *is cocomplete.*

Remark 2. As shown in the proof in [11], limits are constructed componentwise on the index and the functor level. However, this componentwise construction does not work for colimits, where the free construction has to be taken into account.

As a consequence, we can also form the category of distributed objects as the Grothendieck category associated to the indexed category *Diag* as defined in Def. 3.

Theorem 2. *The category* **DisC** *is a Grothendieck category over the indexed category* $Diag : \mathbf{Graphs^{op}} \to \mathbf{Cat}$*.*

Proof. This is a direct consequence of the definitions of distributed objects and morphisms, the given functor *Diag* and the construction of a Grothendieck category. □

Theorem 3. *If* **C** *is (co)complete, then also* **DisC** *is (co)complete.*

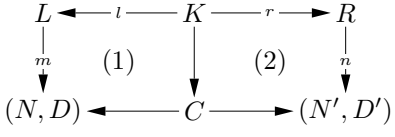
Proof idea. According to Fact 1, if **C** is (co)complete, **DisC** being (co)complete follows from the facts that **Graphs** is (co)complete, $comFunct[G, \mathbf{C}]$ is (co)complete for all $G \in$ **Graphs** and $Diag(h)$ is continuous (has a left adjoint) for all $h : G \to G' \in$ **Graphs** by Theorem 1 (see [10] for more detail). Note that Theorem 1 shows that $Diag(h)$ has a left adjoint F_h. This means that $Diag(h)$ is a right adjoint and hence continuous. □

4 Graph Transformation in DisC

In this section, we define graph transformations on distributed objects in the double pushout (DPO) approach based on [8]. In particular, we present explicit pushout and pullback constructions in **DisC** and discuss the gluing condition.

Definition 6 (distributed transformation system). *A distributed transformation system* $TS = (\mathbf{DisC}, S, P)$ *consists of a category* **DisC** *over some category* **C***, a start object* S *and a set of distributed productions* P*, where*

1. *a distributed production* $p = L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ *consists of distributed objects* L*,* K *and* R *and distributed morphisms* $l : K \to L$ *and* $r : K \to R$*,*
2. *a direct distributed transformation* $(N, D) \stackrel{p,m}{\Longrightarrow} (N', D')$ *of a distributed object* (N, D) *via the production* p *and a match* $m : L \to (N, D)$ *is given by the following diagram, where* (1) *and* (2) *are pushouts in* **DisC***,*



3. *a distributed transformation is a sequence* $(N_0, D_0) \Rightarrow (N_1, D_1) \Rightarrow \ldots \Rightarrow (N_n, D_n)$ *of direct distributed transformations, written* $(N_0, D_0) \stackrel{*}{\Rightarrow} (N_n, D_n)$*,*
4. *the language* $L(TS)$ *consists of all distributed objects* (N, D) *in* **DisC** *derivable from the start object* S *by a transformation, i.e.* $L(TS) = \{(N, D) \mid S \stackrel{*}{\Rightarrow} (N, D)\}$*.*

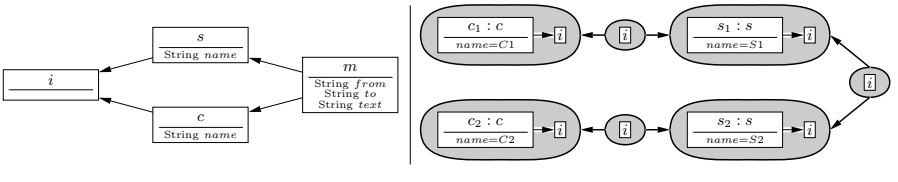


Fig. 3. The type graph and an example of a distributed network

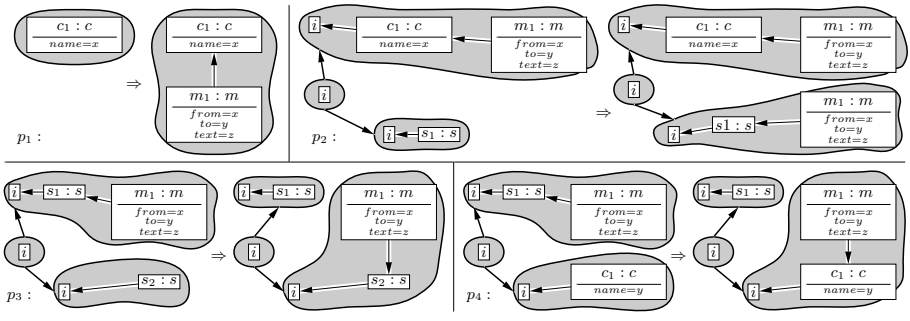


Fig. 4. Example communication productions

Example 2. In the following, we model a small client-server system with asynchronous communication using typed attributed graph transformation. In this case **C** is the category $\mathbf{AGraphs_{ATG}}$ of typed attributed graphs (see [8] for more detail) leading to distributed graphs in **DisC** over typed attributed graphs. The type graph of the local graphs is shown in Fig. 3 on the left hand side. Each client (c) and server (s) has a name and can be connected to an interface connector (i). Messages (m) can be assigned to clients and servers, and they contain the sender ($from$), the receiver (to) and the message itself ($text$).

On the network level, clients and servers can be connected to other servers via the interface connectors. An example of a distributed graph is given in Fig. 3 on the right hand side, where two clients $c1$ and $c2$ are connected to different servers $s1$ and $s2$, which themselves are connected.

The communication between the clients is modeled by graph transformation using communication productions p_1 - p_4 in Fig. 4, that do not change the structure of the underlying network. As usual, only the left- and the right-hand side of the productions are shown - the gluing object is their intersection. First, a client may create a message using the production p_1. Then the message is sent to the server with production p_2. Between different servers, the message can be transmitted using production p_3. If the receiver of the message is connected to the current server where the message is stored, this client can receive the message using production p_4.

Fig. 5 shows some productions for network administration. With production q_1, a new server is added, and q_2 adds a new client.

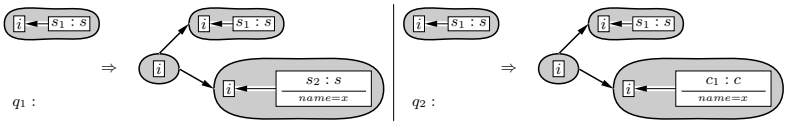


Fig. 5. Example network productions

The result of an application of the production q_2 with c_1 replaced by $x = c_3$ and $s_1 = s_2$ to the distributed graph shown in Fig. 3 is depicted in Fig. 6, where a new client c_3 is added, changing the network graph.

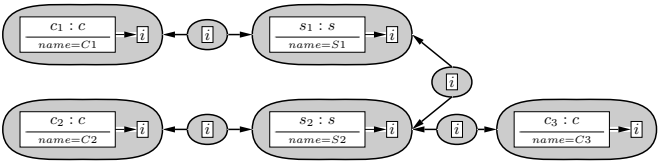


Fig. 6. Application of the distributed production q_2

From Theorem 3 it follows that if **C** is cocomplete the pushout over arbitrary distributed morphisms f and g exists. Since pushouts are the underlying structure of transformations, we want to characterize them more explicitly. The following construction has been introduced as generalized amalgamation in [13, 14].

Theorem 4. *Given distributed morphisms $f = (f_N, f_D) : (N_0, D_0) \to (N_1, D_1)$ and $g = (g_N, g_D) : (N_0, D_0) \to (N_2, D_2)$ in* **DisC**. *According to Proposition 1 these morphisms can be decomposed. Then the diagram in the upper part of Fig. 7 is a pushout over f and g in* **DisC**, *where $(1')$ is a pushout in* **Graphs** *with $g'_N \circ f_N = h_N = f'_N \circ g_N$ and $(4')$ is a pushout in $comFunct[N_3, \mathbf{C}]$.*

Proof idea. It can be shown that the squares (1), (2), (3) and (4) are pushouts in **DisC**. Then by pushout composition also the complete diagram is a pushout in **DisC** (see [10] for more detail). □

Remark 3. It may be noted that Prop. 1 and the above Theorem are formulated for the category **DisC**, but they hold for any Grothendieck category with free constructions, as shown for a similar general framework in [13, 14].

Example 3. Fig. 8 shows an example pushout construction, as defined above in Fig. 7. The network morphisms can be obtained from the relative positions of the nodes. Square (1) shows the pushout on the network level, and the free extensions of the diagram D_0. Squares (2) and (3) show the corresponding extensions for diagrams (D_2) and (D_1), respectively. Square (4) gives the componentwise pushout on the diagram level.

We also have an explicit construction of pullbacks in **DisC** with complete **C**.

Theorem 5. *Given distributed morphisms $f = (f_N, f_D) : (N_1, D_1) \to (N_3, D_3)$ and $g = (g_N, g_D) : (N_2, D_2) \to (N_3, D_3)$ in* **DisC**. *Then the diagram (1) is a pullback over f and g in* **DisC**, *where (2) is a pullback in* **Graphs** *with $g_N \circ f'_N = h_N = f_N \circ g'_N$ and (3) is a pullback in $comFunct[N_0, \mathbf{C}]$.*

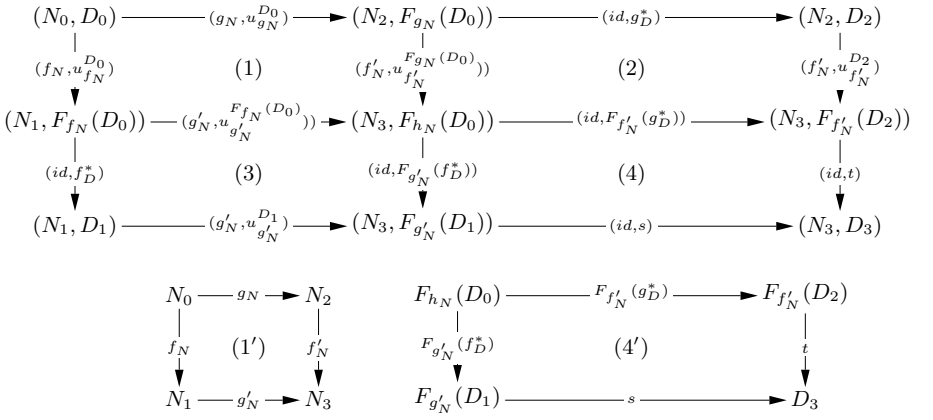
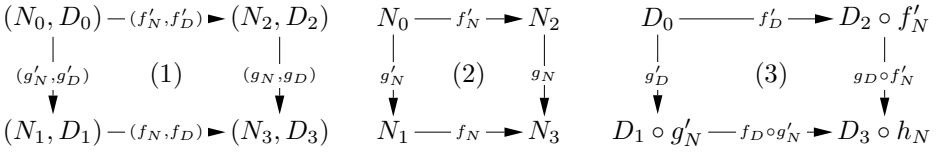


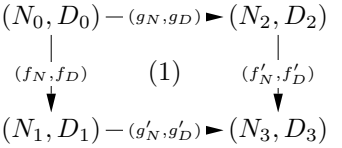
Fig. 7. Explicit pushout construction in **DisC**



Proof. Follows from the proof of Fact 1 in [11] and Remark 2.

5 Persistent Morphisms and Componentwise Pushouts

In [1], the author does not study the construction of general pushouts of distributed graphs. Instead, the paper concentrates on studying when it is possible to build *strongly componentwise pushouts*. Intuitively, if the network morphisms involved are injective, a componentwise pushout of distributed graphs (1) can be seen as the gluing of two distributed graphs (with respect to the common subgraph (N_0, D_0)), where for each node n_3 in N_3, if $n_3 = g'_N(f_N(n_0))$ then the graph at this node, $D_3(n_3)$, is the gluing of the graphs $D_0(n_0)$, $D_1(f_N(n_0))$, and $D_2(g_N(n_0))$ with respect to the corresponding morphisms defined by f_D and g_D. In the following, we call this "componentwise pushouts". But in [1], *strongly* componentwise pushouts are considered with the following additional property: if n_3 is not the image of any node in N_0, but just of a node n_1 in N_1 (respectively n_2 in N_2) then $D_3(n_3)$ is equal to $D_1(n_1)$ (respectively $D_2(n_2)$).



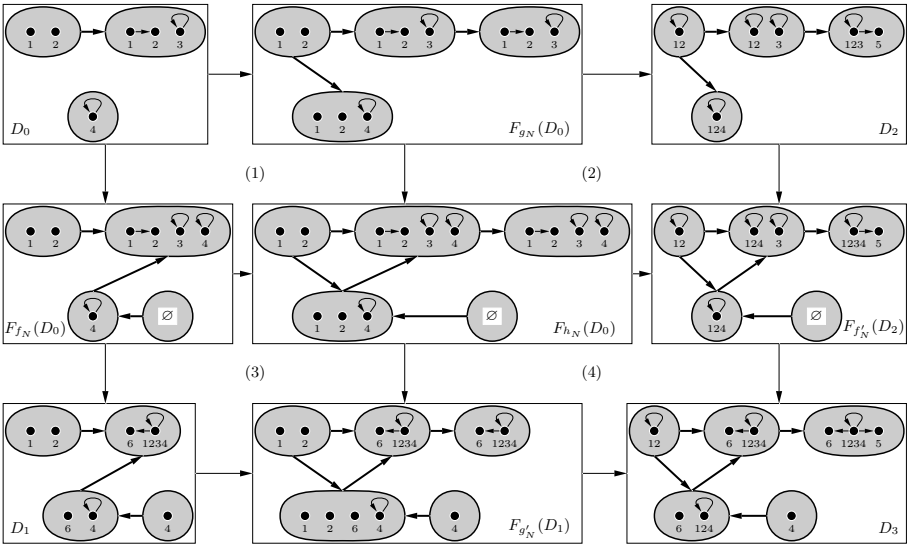


Fig. 8. Example of an explicit pushout construction

This means, in [1], Taentzer provides properties for an if-and-only-if characterization of the existence of strongly componentwise pushouts. Unfortunately, these properties are quite ad-hoc and depend not only on the span of network morphisms, but also on the diagrams, and thus are difficult to generalize to categories of distributed objects over a category **C** different than **Graphs**.

We think that it is important for several applications to have componentwise pushouts, but not necessarily stronlgy componentwise as in [1]. Fig. 8 is an example of a componentwise pushout, which is not strongly componentwise. The upper right node in D_2 has no preimage in D_0, but the local graph is different from the corresponding local graph in D_3. However, in general, arbitrary pushouts of distributed graphs will not be componentwise. The key property to ensure in Proposition 4 componentwise pushouts is that the given network morphisms are *persistent* in the following sense:

Definition 7 (persistent network morphism). *If* **C** *is cocomplete, a morphism* $h : N_1 \to N_2$ *is persistent if for every* D *in* $comFunct[N_1, \mathbf{C}]$ *the unit of the adjunction,* $u_h^D : D \to Diag(h) \circ F_h(D)$*, is an isomorphism.*

For a characterization of persistent morphisms, we need the following property of colimits.

Proposition 2. *Given a commutative functor* $D : N \to \mathbf{C}$ *with colimit object* $Col(D)$ *of* D*, then we have for any* $n \in N$*:*
If for all $n' \in N$ *there is a path* $p_{n'} : n' \xrightarrow{*} n$ *in* N *then* $D(n) \cong Col(D)$*.*

Proof idea. Since path morphisms are unique, $(D(n), (D(p_{n'}))_{n' \in N})$ is a cocone of D leading to a unique morphism $x : Col(D) \to D(n)$. Using the properties of

colimit $Col(D)$ and commutative D it can be shown that x is an isomorphism (see [10] for more detail). □

Taking into account the construction of free functors in Theorem 1, we are now able to characterize persistent network morphisms for cocomplete categories **C**. Intuitively, a morphism $h : N_1 \to N_2$ is persistent if for a path from $h(n_1)$ to $h(n_2)$ in N_2 there is already a path from n_1 to n_2 in N_1.

Proposition 3. *A morphism $h : N_1 \to N_2$ is persistent if we have for all nodes $n_1, n_1' \in N_1$ the following property:*
If there exists a path $h(n_1) \overset{*}{\to} h(n_1') \in N_2$ then there exists a path $n_1 \overset{*}{\to} n_1'$ in N_1.

Proof. Given $D : N_1 \to \mathbf{C}$. For $n_1 \in N_1$ we have $Diag(h) \circ F_h(D)(n_1) = F_h(D)(h(n_1)) = Col(D|_{N_1(h(n_1))})$ as defined in the construction of Theorem 1. If $n_1' \in N_1(h(n_1))$ then there is a path $h(n_1') \overset{*}{\to} h(n_1)$ in N_2. The above condition makes sure that there is also a path $n_1' \overset{*}{\to} n_1$ in N_1. Applying Proposition 2 with $F = D|_{N_1(h(n_1))}$, this means that $Col(D|_{N_1(h(n_1))}) \cong D|_{N_1(h(n_1))}(n_1) = D(n_1)$ and hence $Diag(h) \circ F_h(D)(n_1) \cong D(n_1)$. □

Remark 4. This property is also necessary for persistency for all categories **C**, where the colimits of an arbitrary $F : \bullet\!\!\rightarrow\!\!\bullet \to \mathbf{C}$ and of its restriction $F|_{\bullet\ \bullet} : \bullet\ \bullet \to \mathbf{C}$ are in general not isomorphic.

Then, using the construction of pushouts in Theorem 4 we can show that if the associated network morphisms are persistent then the pushouts of the interface graphs are componentwise.

Proposition 4. *If f_N and g_N are persistent, then the pushout in* **DisC** *is a componentwise pushout on the interface network, i.e. $D_3(h_N(n_0))$ is the pushout of $D_1(f_N(n_0)) \overset{f_{D,n_0}}{\leftarrow} D_0(n_0) \overset{g_{D,n_0}}{\rightarrow} D_2(g_N(n_0))$ for all $n_0 \in N_0$.*

Proof idea. In **Graphs**, pushouts can be shown to be closed under persistent morphisms. This means that also f_N', g_N' are persistent and we have $Diag(h_N) \circ F_{h_N}(D_0) \cong D_0$, $Diag(g_N') \circ F_{g_N'}(D_1) \cong D_1$ and $Diag(f_N') \circ F_{f_N'}(D_2) \cong D_2$. Since pushouts in functor categories are constructed componentwise, this means that $D_3(h_N(n_0))$ is the pushout of $D_1(f_N(n_0)) \overset{f_{D,n_0}}{\leftarrow} D_0(n_0) \overset{g_{D,n_0}}{\rightarrow} D_2(g_N(n_0))$ for all $n_0 \in N_0$ according to the pushout (4) in Fig. 7. □

6 Conclusion

In this paper we have presented categorical foundations for distributed graph transformation that, in our opinion, considerably improve [1]. In particular, we have seen that the category of distributed objects has free constructions and is complete and cocomplete (provided that the underlying category is so). Moreover, we have shown how to explicitly build pushouts using the concept of generalized amalgamation introduced in [13, 14] and we have characterized the class of

morphisms (persistent morphisms) that ensure that in a pushout the interfaces will be glued componentwisely and discussed the relationship with [1].

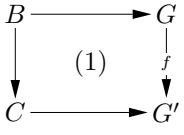
6.1 Towards a Theory of Distributed Graph Transformation

We have provided the basic constructions for defining transformations. However, this is just a first step for fully studying distributed graph transformation in a general setting.

According to Theorem 3, **DisC** is complete and cocomplete provided that **C** is complete and cocomplete. Since the categories **Graphs** of graphs, $\mathbf{Graphs_{TG}}$ of typed graphs and $\mathbf{AGraphs_{ATG}}$ of typed attributed graphs satisfy both properties [8, Thm. 11.3], **DisGraphs**, $\mathbf{DisGraphs_{TG}}$ and $\mathbf{DisAGraphs_{ATG}}$ are complete and cocomplete. This means especially that we are able to construct pushouts and pullbacks, which are needed in the DPO approach. The key question is, whether there is a suitable class $\mathcal{M}$ for **DisC** such that $(\mathbf{DisC}, \mathcal{M})$ becomes a (weak) adhesive HLR category. This would allow to instantiate the corresponding theory in [8] to distributed graph transformation over **C**. Unfortunately, for the most obvious choices $\mathcal{M}_1 = Monos \times Monos$, $\mathcal{M}_2 = Persistent\ Monos \times Monos$ and $\mathcal{M}_3 = Persistent\ Monos \times Mor_{\mathbf{C}}$, $(\mathbf{DisC}, \mathcal{M}_i)$, $i = 1, 2, 3$ is in general not (weak) adhesive HLR, where persistent morphisms are defined in section 5. In order to obtain a (weak) adhesive HLR category, $\mathcal{M}$-morphisms have to be monomorphisms (this rules out choice $\mathcal{M}_3$), pushouts along $\mathcal{M}$-morphisms have to be pullbacks (this rules out choice $\mathcal{M}_1$) and $\mathcal{M}$-morphisms have to be closed under pullbacks (this rules out choice $\mathcal{M}_2$).

But we can show that (injective) persistent network morphisms are closed under pushouts, which implies that at least $\mathcal{M}_3$ is closed under pushouts. This means that we can obtain the Local Church-Rosser Theorem [8, Thm. 5.12] for $(\mathbf{C}, \mathcal{M}_3)$, provided that we require a stronger notion of independence including the $\mathcal{M}_3$ PO-PB decomposition property for the given pair of direct transformations in the corresponding proof.

Moreover we obtain a weaker version of the Embedding Theorem [8, Thm. 6.14], which usually requires an initial pushout (1) over a morphism f.



It is an interesting open question under which conditions initial pushouts over distributed morphisms exist and how they can be constructed for suitable **C**. This would immediately lead to a necessary and sufficient gluing condition [8, Thm. 6.4], which is important for the construction of direct transformations. If we do not have initial pushouts, we can also take some other pushout (1) over f, including the trivial case $B \to C = G \to G'$, and replace the notion of consistency based on pullback-constructions and the boundary object B in (1) by B-consistency depending on the chosen B. In fact, the proof in [8] only uses

the pushout property of (1) and not the initiality, which, however, is used for the Extension Theorem [8, Thm. 6.16].

Acknowledgements. This work has been partially supported by the Research Network SEGRAVIS (HPRN-CT-2002-00275) and by the Spanish project GRAMMARS (ref. TIN2004-07925-C03-01).

References

[1] Taentzer, G.: Distributed Graphs and Graph Transformation. Applied Categorical Structures **7**(4) (1999) 431–462
[2] Bottoni, P., Parisi-Presicce, F., Taentzer, G.: Specifying Integrated Refactoring with Distributed Graph Transformations. In Pfaltz, J., Nagl, M., Böhlen, B., eds.: Proc. of AGTIVE 2003. Volume 3062 of LNCS., Springer (2004) 220–235
[3] Bottoni, P., Parisi-Presicce, F., Taentzer, G., Pulcini, S.: Maintaining Coherence between Models with Distributed Rules: From Theory to Eclipse. In Bruni, R., Varró, D., eds.: Proc. of GT-VMT 2006. ENTCS, Elsevier (2006) 81–91
[4] Fiadeiro, J.: Categories for Software Engineering. Springer (2006)
[5] Goguen, J.: Sheaf Semantics for Concurrent Interacting Objects. Mathematical Structures in Computer Science **2**(2) (1992) 159–191
[6] Ehrig, H., Habel, A., Padberg, J., Prange, U.: Adhesive High-Level Replacement Categories and Systems. In Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G., eds.: Proc. of ICGT 2004. Volume 3256 of LNCS., Springer (2004) 144–160
[7] Ehrig, H., Prange, U., Taentzer, G.: Fundamental Theory for Typed Attributed Graph Transformation. In Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G., eds.: Proc. of ICGT 2004. Volume 3256 of LNCS., Springer (2004) 161–177
[8] Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer (2006)
[9] Mac Lane, S.: Categories for the Working Mathematician. Volume 5 of Graduate Texts in Mathematics. Springer, New York (1971)
[10] Ehrig, H., Orejas, F., Prange, U.: Categorical Foundations of Distributed Graph Transformation: Long Version. Technical report, TU Berlin (2006)
[11] Tarlecki, A., Burstall, R., Goguen, J.: Some Fundamental Algebraic Tools for the Semantics of Computation: Part 3: Indexed Categories. Theoretical Computer Science **91**(2) (1991) 239–264
[12] Goguen, J.: Information Integration in Institutions. In Moss, L., ed.: Jon Barwise Memorial Volume, Indiana University Press (2006) to appear.
[13] Ehrig, H., Baldamus, M., Orejas, F.: New Concepts for Amalgamation and Extension in the Framework of Specification Logics. Technical Report 91/05, TU Berlin (1991)
[14] Ehrig, H., Baldamus, M., Cornelius, F., Orejas, F.: Theory of Algebraic Module Specification including Behavioural Semantics, Constraints an Aspects of Generalized Morphisms. In Nivat, M., Rattray, C., Rus, T., Scollo, G., eds.: Invited Lecture Proc. of AMAST'91, Springer (1991) 145–172