A Framework of Hierarchical Graphs and its Application to the Semantics of $SRML$

Nikos Mylonakis   Fernando Orejas

January 13, 2012

Abstract

Hierarchical graphs or, in general, hierarchical graphical structures are needed when describing visual models at different levels of abstraction. This is the case of the semantic model of $SRML$, the service modeling language of the project SENSORIA. In particular, the state model of this language is considered at two level of abstraction. Roughly, at the lowest level, a *state configuration* is a graph consisting of interconnected components and, at the highest level, *business configurations* are graphs consisting of interconnected activities, where each activity is a graph of components. Then, a state configuration is the flattening of the corresponding business configuration.

Following these ideas, in this paper, we present a new framework of hierarchical graphs, showing that it is m-adhesive. Then we show how this framework can be used to define (part of) a graph transformation semantics of $SRML$.

1 Introduction

$SRML$ ([8, 9, 10]) is a service modeling language designed within the project SENSORIA. Its state model is considered at two level of abstraction. Roughly, at the lowest level, a *state configuration* is a graph consisting of interconnected components and, at the highest level, *business configurations* are graphs consisting of interconnected activities, where each activity is a graph of components. This definition at two levels of abstraction is needed to allow for dynamic service binding. Unfortunately, the operational semantics of $SRML$ is defined in a relative ad-hoc way, which means that, to animate its models, one would have to build a specific implementation.

The goal of this work is to provide a graph transformation semantics for $SRML$, so that its models could be animated using some graph transformation tool, such as the Maude implementation of graph transformation [2]. Following these ideas, in this paper, we present a new framework of hierarchical graphs, based on the notion of symbolic graph and symbolic graph transformation [11], showing that it is m-adhesive. Then we show how this framework can be used to define (part of) a graph transformation semantics of $SRML$.

The paper is organized as follows. In Sections 2 and 3, we present an overview of $SRML$ and symbolic graphs, respectively. Then, in Section 4, we present our framework of hierarchical graphs. Section 5 is dedicated to show how can we define part of the semantics of $SRML$ using hierarchical

1

graph transformation. Finally, in Section 6, we discuss some related work and conclude the paper.

The paper includes a (long and) detailed appendix of the proof that our category of hierarchical graphs is m-adhesive. This appendix is not intended for publication. It is included so that the reviewers can check the correctness of our results. However, the paper without the appendix should be readable on its own.

2 Introduction to SRML

The essential concept of the Sensoria Reference Modeling Language ($SRML$) is the notion of module which is inspired by the constructions presented in Service Component Architecture (SCA). See [8, 9, 10] for a detailed description of the language. Roughly speaking, a module can be seen as a graph of components which are connected by wires. Moreover a module also includes some provides and requires interfaces, which are also connected by wires to the components. As an example of a module we present a booking agent. This module, which is graphically depicted in Fig. 1, is supposed to offer a service for booking trips (flight and hotel). It includes a single component (BookAgent) which is supposed to take care of the booking and three interfaces. A provides interface (Customer) for customer requests and two requires interfaces (FlightAgent and HotelAgent). The BookAgent is supposed to receive trip reservation requests from customers that are connected to the Customer interface. Then, BookAgent is supposed to request a flight and a hotel to services connected to the FlightAgent and HotelAgent, respectively, which are supposed to provide the corresponding reservation confirmations through a hotel and a flight code. These codes will then be returned to the customer. However, due to lack of space, in our example, we only see how the Bookagent requests a flight to the FlightAgent.
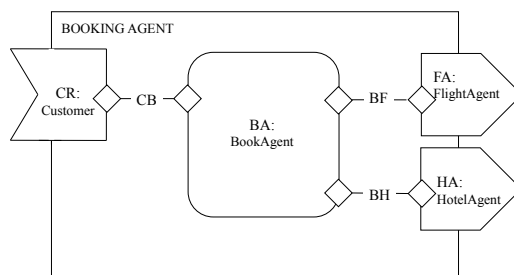

Figure 1: BOOKING AGENT service module

Components are specified by a business role consisting of a signature and an *orchestration* part. The signature declares the events in which that component may take part and the orchestration describes the behavior

of the component. For instance, below we can see a small part of the specification of the main component of the booking agent module.

In this specification we declare that BookAgent has an interaction called booktrip in which the component participates receiving and then sending information (r&s) and another interaction called bookflight in which the component participates sending and then receiving information (s&r). For example, booktrip has four input parameters (from, to, out and in) and one output parameter (tconf). Then, in the orchestration part, first we declare the local variables of the component and possibly their initialization, and then we specify the effects of the interactions in which the component may take part. For instance, in the example, the local variables from, to, in, and out are supposed to store the basic data of the trip being booked (source, des-

BUSINESS ROLE BookAgent **is**
r&s booktrip
   ♯from,to:string; out,in:integer;
   ✉tconf: (fcode,hcode);
s&r bookflight
   ♯from,to:string; out,in:integer;
   ✉fconf: fcode;
. . .
ORCHESTRATION
  local
    from,to:string; out,in:integer;
    fconf: fcode; hconf: hcode;
  transition Torder
    triggered by booktrip ♠
    effects
      from' = booktrip.from ∧
      to' = booktrip.to ∧
      out' = booktrip.out ∧
      in' = booktrip.in
    sends bookflight ♠
      bookflight.from = from' ∧
      bookflight.to = to' ∧
      bookflight.out = out' ∧
      bookflight.in = in' ∧
. . .

tination, departure and return dates, respectively), and fconf and hconf are supposed to store the flight and hotel reservation codes that have been booked. In the example, we also declare the local effects of an interaction, called Torder, in which the component takes part. This interaction is triggered by the event book trip that the component receives. The contents of the local variables from, to, in, and out after the interaction are the contents of the corresponding parameters of booktrip. Moreover, the interaction triggers an event bookflight, which is sent by the component with the corresponding input parameters.

External interfaces are specified through business protocols. They also include a signature and they specify the conversations that the module expects relative to each party. It is the responsibility of the coparty to adhere to these protocols. Finally, wires bind the names of the interactions and specify the protocols that coordinate the interactions between two parties. For instance, this module includes the wire CB that connects the business protocol of the customer of the BOOKING AGENT module and the business role BookingAgent of the same module. We do not include here an example of an interface or of a wire specification, since they are not relevant for this paper.

3 Symbolic graphs and symbolic graph transformation

Symbolic (hyper)graphs [11] can be seen as a specification of a class of attributed graphs (i.e. of graphs including values from a given data algebra in their nodes or edges). In particular, in a symbolic graph, values are replaced by variables and, moreover, a set of formulas, Φ, specifies the values that the variables may take. Then, we may consider that a symbolic graph SG denotes the class of all graphs obtained replacing the variables in the graph by values that satisfy Φ. For instance, the symbo-

lic graph in Figure 2 specifies a class of attributed graphs, including distances in the edges, that satisfy the well-known triangle inequality.
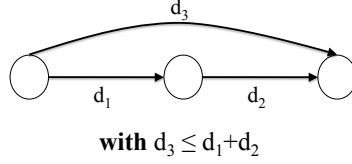

with $d_3 \leq d_1 + d_2$

Figure 2: A symbolic graph

The notion of symbolic graph is based on the notion of E-graph (for details, see [6, 7]). The only difference of the notion of E-graph that we use with respect to the notion in [6] is that we deal with hypergraphs. This means that instead of having source and target functions that map edges to nodes, we have an attachment function that maps each (hyper)edge to a sequence of nodes, i.e. the nodes connected by the edge.

Definition 3.1 (Symbolic graphs and morphisms) *A symbolic graph over the data algebra D is a pair $\langle G, \Phi_G \rangle$, where G is an E-graph over a set of variables X, and Φ_G is a set of first-order formulas over the operations and predicates in D including variables in X and elements in D.*

Given symbolic graphs $\langle G_1, \Phi_{G_1} \rangle$ and $\langle G_2, \Phi_{G_2} \rangle$ over D, a symbolic graph morphism $h : \langle G_1, \Phi_{G_1} \rangle \to \langle G_2, \Phi_{G_2} \rangle$ is an E-graph morphism $h : G_1 \to G_2$ such that $D \models \Phi_{G_2} \Rightarrow h(\Phi_{G_1})$, where $h(\Phi_{G_1})$ is the set of formulas obtained when replacing in Φ_{G_1} every variable x_1 in the set of labels of G_1 by $h_X(x_1)$.

Symbolic graphs over D together with their morphisms form the category **SymbGraphs$_D$**.

In [11] it is shown that **SymbGraphs$_D$** is an adhesive HLR category.

In symbolic graph transformation we consider that the left and right-hand sides of a rule are symbolic graphs, where the conditions on the left hand side on the rule are included in the conditions in the right hand side of the rule. This means that applying a transformation to a symbolic graph $\langle G, \Phi_G \rangle$ reduces or narrows the instances of the result. For instance, G may include an integer variable x such that Φ_G does not constrain its possible values. However, after applying a given transformation, in the result graph $\langle H, \Phi_H \rangle$ we may have that Φ_H includes the formula $x = 0$, expressing that 0 is the only possible value of x.

Definition 3.2 (Symbolic graph transformation rules) *A symbolic graph transformation rule is a triple $\langle \Phi_L, L \leftarrow K \to R, \Phi_R \rangle$, where L, K are E-graphs over the same set of labels X_L, R is an E-graph over a a set of labels X_R, with $X_L = X_K \subseteq X_R$, $L \leftarrow K \to R$ is a standard graph transformation rule, and Φ_L and Φ_R are sets of formulas over X_L and X_R, respectively, and over the values in the given data algebra D, with $\Phi_L \subseteq \Phi_R$.*

As an example, in Figure 3 we show a rule with two events and a bookagent component. The rule states that when arriving a booktrip event, the bookagent registers them and sends a new bookflight event. The formula below expresses that the origin, destination, and departure and return dates are the same in the incoming and in the outgoing events. For simplicity, we do not depict the intermediate graph K, nor do we state explicitly which are the sets X_L and X_R of the given rule. Instead, we assume that X_L consists of all the variables that are explicitly depicted in the left-hand side graph, and X_R consists of all the variables that are depicted in the rule. Similarly, we just depict a single set of formulas for a given rule, assuming that Φ_R is the set consisting of all these formulas and Φ_L is the subset of Φ_R consisting of the formulas that only include variables in X_L.
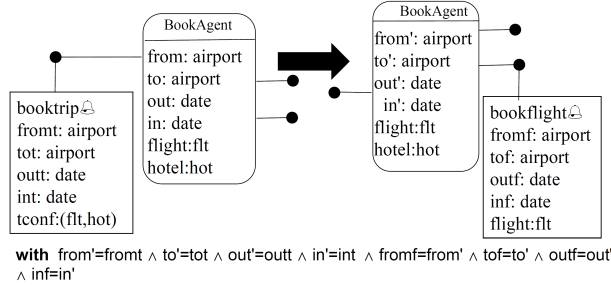
4

Figure 3: A symbolic rule

As usual, the application of a graph transformation rule to a given symbolic graph SG can be defined by a double pushout in the category of symbolic graphs. However, it can also be expressed in terms of a transformation of E-graphs.

As a remark, given a symbolic graph transformation rule $\langle \Phi_L, L \hookleftarrow K \hookrightarrow R, \Phi_R \rangle$ over a given data algebra D and a symbolic graph morphism $m : \langle L, \Phi_L \rangle \to \langle G, \Phi_G \rangle$, we have that $\langle G, \Phi_G \rangle \Longrightarrow_{p,m} \langle H, \Phi_H \rangle$ if and only if the diagram below is a double pushout in $\mathbf{E - Graphs}$ and $D \models \Phi_G \Rightarrow m(\Phi_L)$.

$$
\begin{array}{ccccc}
L & \longleftarrow & K & \longrightarrow & R \\
\downarrow m & (1) & \downarrow & (2) & \downarrow m' \\
G & \longleftarrow & F & \longrightarrow & H
\end{array}
$$

and, moreover, $\Phi_H = \Phi_G \cup m'(\Phi_R)$.

4   Hierarchical symbolic graphs

In this section we introduce our notion of hierarchical (symbolic) graph and we show that these graphs, together with their associated notion of morphism, form an M-adhesive category. Our notion of hierarchical graph is inspired in the notion of Petri Net refinement in [12]. According to that notion, in a a net refinement, a transition t can be replaced by another net, N_t, where some of its transitions are connected to the same places that n was connected. In our case, we consider that a hierarchical graph is a graph whose edges may include (hierarchical) graphs, that may be considered their refinement. As in the case of nets, the edges in the graph inside e may be connected to the same nodes that e is connected. This is done by means of a notion of hierarchical graph with interface, where the interface are part of the nodes of the graph (more precisely a sequence of nodes). In particular, if e is a hierarchical edge (i.e. e includes a graph) whose attachment is the sequence α, we assume that the graph inside e is a graph with interface α. Notice that, as a consequence, the nodes in the attachment of e maybe considered to be simultaneously inside and outside e. For instance, in Fig. 4 we can see a simple hierarchical graph. On the left, we can see the top level graph of that graph, i.e. the graph without seeing the contents of its edges. This graph has two nodes, n_1 and n_2 and two edges, e_1 and e_2. Edge e_1 is connected to n_1 and n_2 and e_2 is connected to n_1 and twice to n_2. This means that the attachment of e_1 may be $n_1 n_2$ and the attachment of e_2 may be $n_1 n_2 n_2$. The graph on the right shows the contents of the edges. In particular, e_1 has no contents, i.e. e_1 is non-hierarchical, or, to be more precise technically it includes the nodes in its interface (n_1 and n_2). The edge e_2 includes a graph with three edges e_3, e_4 and e_5, whose interface is $n_1 n_2 n_2$. In

particular e_3 and e_4 are connected to n_1 and n_2 (and to other internal nodes). Notice that, technically, we consider that nodes n_1 and n_2 belong simultaneously to the top level edge and to the graphs contained in e_1 and e_2.
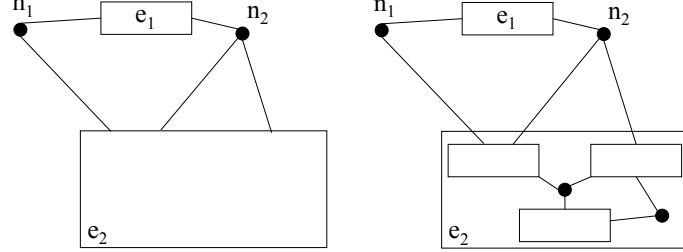

Figure 4: A hierarchical graph

Definition 4.1 (Symbolic graphs with interface and morphisms) *A symbolic graph with interface over a data algebra D is a triple $\langle G, \Phi_G, I_G \rangle$, where $\langle G, \Phi_G \rangle$ is a symbolic graph over D and I_G is the interface, a sequence of nodes from G, i.e. $I_G \in V_G^*$. A morphism between symbolic graphs with interface $h : \langle G, \Phi_G, I_G \rangle \to \langle G', \Phi_{G'}, I_{G'} \rangle$ is a symbolic graph morphism such that $h^*(I_G) = I_{G'}$, where h^* denotes the extension of h to sequences of nodes.*

In what follows, all our symbolic graphs are assumed to include an interface. As a consequence, symbolic graphs with interface will just be called symbolic graphs.

A hierarchical graph HG is a pair $\langle HG^{top}, cts^{HG} \rangle$, where HG^{top} is a symbolic graph, the top level graph, and cts^{HG} is the contents function that, for every edge in the top level graph, yields the graph included in that edge. For simplicity, we consider that if an edge is non-hierarchical then it includes the empty graph. Or, to be more precise, a non-hierarchical edge e is an edge that includes a graph consisting only of the nodes in its interface (i.e. the nodes in the attachment of e).

Hierarchical graphs are defined inductively as follows. For each natural number n, we define the class of hierarchical graphs of depth i, $\mathcal{HG}_i$, where $\mathcal{HG}_0$ consists of all hierarchical graphs whose top level graph has no edges, and $\mathcal{HG}_{i+1}$ is the class of all hierarchical graphs whose edges include hierarchical graphs of, at most, depth i.

Definition 4.2 (Hierarchical graphs) *The class $\mathcal{HG} = \bigcup_{i \geq 0} \mathcal{HG}_i$ of hierarchical symbolic graphs with interface is inductively defined as follows:*

- *$\mathcal{HG}_0$ is the class of all pairs $\langle HG^{top}, \emptyset \rangle$, where HG^{top} is a symbolic graph without edgesand $\emptyset$ is the empty function.*

- *$\mathcal{HG}_{i+1}$ is the class of all pairs $\langle HG^{top}, cts^{HG} \rangle$, where HG^{top} is a symbolic graph, and $cts^{HG} : E_{HG^{top}} \to \bigcup_{0 \leq j \leq i} \mathcal{HG}_j$, that maps each edge in HG^{top} into a graph of layer j smaller than $i + 1$.*

Hierarchical graphs in $\mathcal{HG}_i$ are called hierarchical graphs of depth i.

For instance, the graph on the right of Fig. 4 is a hierarchical graph of depth 1.

Hierarchical graphs can be flattened to form a standard symbolic graph replacing every hierarchical edge by its contents. More precisely:

Definition 4.3 (Flattened graph) *the flattening of a hierarchical graph, $Flat(HG)$ is inductively defined as follows:*

- *If $HG \in \mathcal{HG}_0$ then $Flat(HG) = HG^{top}$.*

- If $HG \in \mathcal{HG}_{i+1}$ then:

$$Flat(HG) = HG^{top} \cup \big(\bigcup_{e \in E_{HG^{top}}} cts(e) \big) \setminus \{e \in E_{HG^{top}} \mid E_{cts(e)} \neq \emptyset\}$$

Hierarchical graph morphisms are also defined inductively:

Definition 4.4 (Hierarchical graph morphisms) *Hierarchical graph morphisms are also defined inductively:*

- *If HG_0 is a graph in $\mathcal{HG}_0$, a hierarchical graph morphism $h : HG_0 \to HG_1$ is a symbolic graph morphism $h : HG_0^{top} \to HG_1^{top}$ between symbolic graphs with interface.*

- *If HG_0 is a graph in $\mathcal{HG}_{i+1}$, a hierarchical graph morphism $h : HG_0 \to HG_1$ is a pair $\langle h^{top}, h^{down} \rangle$, where $h^{top} : HG_0^{top} \to HG_1^{top}$, and $h^{down} = \{h^e : cts^{HG_0}(e) \to cts^{HG_0}(h^{top}(e))\}_{e \in E_{HG^{top}}}$ is a family including a hierarchical graph morphism for each edge in HG_0^{top}.*

In general, given a hierarchical morphism $h = \langle h^{top}, h^{down} \rangle$ we say that a symbolic graph morphism g is inside h if $g = h^{top}$ or g is inside any morphism in h^{down}.

This notion of graph morphism is quite restrictive with respect to graph transformation. In particular, transformation rules based on this notion of morphism can not produce transformations on the hierarchical structure of the given graph. We are currently studying different ways of making this definition more flexible.

It is routine to see that hierarchical symbolic graphs and morphisms over a data algebra D form a category, which we call **HSymbGraphs$_D$**. Moreover, we can see that this category is $\mathcal{M}$-adhesive, where $\mathcal{M}$ is the class of all monomorphisms h such that if g is inside h then g is an $\mathcal{M}$-morphism in **HSymbGraphs$_D$**. In Appendix 1 we may find a quite lengthy detailed proof. In particular, pushouts in this category are built by induction. Given the diagram below, if for all $i : 0 \leq i \leq 2$ HG_i is of depth 0 then HG_3 is essentially obtained as the pushout of the corresponding top level graphs of the diagram. If some graph is of depth greater than 0, then the top level graph of HG_3 is the pushout of the top level graphs of the graphs in the diagram and for every edge e_3 in HG_3^{top}, $cts^{HG_3}(e_3)$ is the colimit of the contents of each edge e_0 in HG_0^{top} such that $g_1^{top}(h_1^{top}(e_0)) = e_3$, of each edge e_1 in HG_1^{top} such that $g_1^{top}(e_1) = e_3$, and of each edge e_2 in HG_2^{top} such that $g_2^{top}(e_2) = e_3$.

$$
\begin{array}{ccc}
HG_0 & \xrightarrow{\ h_1\ } & HG_1 \\
\downarrow{\scriptstyle h_2} & & \downarrow{\scriptstyle g_1} \\
HG_2 & \xrightarrow{\ g_2\ } & HG_3
\end{array}
$$

5 Towards a semantics for $SRML$

Roughly speaking (for more detail, see e.g. [10]), states in $SRML$ are defined at two levels: as *state configurations* and as *business configurations*. A state configuration can be described as a graph whose nodes are the components that are active at a given moment and whose edges are the wires connecting them. Moreover, a state configuration also includes the values contained by the local variables of wires and components and the events that are pending to be executed. Then, in a business configuration, components and wires of a state configuration that correspond to the same business process are grouped into business activities, which are typed over activity modules. An activity module looks like a service module, but is created dynamically.

These states can evolve in two different ways. On the one hand, the execution of an event causes that this event is eliminated from the set of pending events and, moreover, it may cause that some local variables in the components involved in the event change their value, and some other events are triggered meaning that they are added to the set of pending events. For instance, the execution of the booktrip event, as specified in the BookAgent module in Sect. 2 would cause that the local variables of the BA component are assigned to the input arguments of the event and that a bookflight event is added to the set of pending events. On the other hand, when the requires interface of an activity module AM matches the provides interface of a service module SM the two modules are connected and the activity is bound to this new service. This implies that initialized instances of the components and wires of SM are added to the state configuration and also to the activity associated to AM. The activity module associated to the enriched activity would include the components and wires of that activity and, in addition, the remaining (non-matched) interfaces of AM and SM. For instance, if some customer is booking a trip, in the business configuration there may be an activity including instances of the components and wires of the BookAgent module (and perhaps some other components and wires). The activity module AM typing this activity may include a FlighAgent and a HotelAgent requires interfaces. If a service module SM is discovered in some repository including a provides interface matching the FlightAgent interface in AM then the two modules are connected, meaning that instances of the components and wires in SM would be added to the booking trip activity and to the state configuration. The activity module associated to that activity would still include the HotelAgent interface and, in addition, it would also include the non-matched interfaces of SM. In this paper, we consider only the first kind of state modification, i.e. state modifications associated to events execution, while the second kind of modification is left for future work.
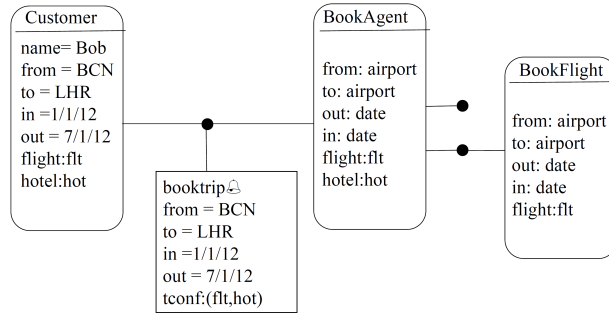

Figure 5: (Part of) a state configuration

In our approach, business and state configurations are represented by hierarchical graphs, whose hyperedges represent components and events at the lowest level and activities at the top level, and whose nodes represent wires. For instance, Fig. 5 represents part of a state configuration. This configuration includes three components. A customer component represents a customer called Bob that is booking a trip from Barcelona to London on Jan. 1, 2012, returning on Jan. 7. This component has sent a booktrip event to the BookAgent component, but this event is pending. At the business configuration level, all these components and wires would be included in one activity, which is not depicted due to lack of space.

Transitions specified in service modules are represented by transformation rules and the execution of an event is represented by the transformation defined by the associated rule. For instance, in Fig. 3 we depict the transformation rule associated to the transition described in the business role of the BookAgent component. In

particular, in that rule we specify that, if a BookAgent has a pending booktrip event, then we may transform the given state configuration into a new configuration where that event is not pending anymore, the local variables of the BookAgent component are updated by the arguments of the event and a new event bookflight is sent through one of the wires. Then, in Fig. 6. we can see the result of applying that rule to the state configuration depicted in Fig. 5.
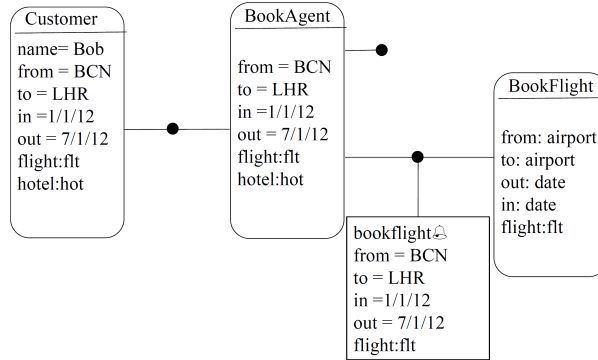

Figure 6: Updated configuration

6   Conclusion and Related Work

In this paper we have presented a new framework for dealing with hierarchical graphs and hierarchical graph transformation, showing that this framework is m-adhesive. Moreover we have shown how this approach can be used to define part of the semantics of the service modeling language $SRML$.

Our notion of hierarchical graph, as said in the previous section, is inspired in the notion of Petri net refinement in [12]. It is also inspired in the notion of hierarchical graph presented in [5]. However, in that notion the graphs inside a hyperedge cannot be connected to nodes outside the hyperedge. Moreover, their graphs are just labelled and do not support arbitrary attributes and attribute computation. Palacz, in [13], defines a much more general framework, where a hierarchical graph is a standard (non-attributted) graph plus a predecessor function that implicitly represents the hierarchy. In that way any element in the graph can be connect to any other element in the graph, independently of the hierarchy of the elements. Unfortunately, the approach is too general for DPO graph transformation. So the author restricts to certain classes of morphisms to ensure the existence of pushouts and the uniqueness of pushout complements. In both cases the main constructions (pushouts, pushout complements) are defined in an ad-hoc way for the specific class of graphs considered. Finally, in [4], the authors also propose a very general notion of hierarchical, without any restriction on the kinds of connections. However, they do not study graph transformation. Instead, they define a family of operations for building them, with the aim of using them for giving semantics to some process algebras.

The semantics of $SRML$ has been addressed in several papers by the group lead by Fiadeiro (e.g. see [8, 9, 10]). In this paper we replace the explicit ad-hoc computation associated to the semantics of interactions, by hierarchical symbolic graph transformation. The main differences of $SRML$ with respect to other approaches in the area of service oriented is that the language supports service binding at run time, in contrast with approaches like [15, 3, 1, 14].

In future work, we plan to study how to define more flexible notions of hierarchical graph morphisms so that we it is possible to perform transformations that change the hierarchical structure of a graph. In addition, we also plan to study how we can extend our semantics to cover service binding.

References

[1] B. Benatallah, F. Casati, and F. Toumani. Web service conversation modeling: a cornerstone for e-business automation. *IEEE Internet Computing*, 8(1):46–54, 2004.

[2] Artur Boronat and José Meseguer. An algebraic semantics for mof. In José Luiz Fiadeiro and Paola Inverardi, editors, *FASE*, volume 4961 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 2008.

[3] M. Broy, I. H. Krüger, and M. Meisinger. A formal model of services. *ACM Trans Softw Eng Methodol*, 16(1), 2007.

[4] R. Bruni, F. Gadducci, and A. Lluch-Lafuente. An algebra of hierarchical graphs. In M. Wirsing, M. Hofmann, and A. Rauschmayer, editors, *TGC*, volume 6084 of *Lecture Notes in Computer Science*, pages 205–221. Springer, 2010.

[5] F. Drewes, B. Hoffmann, and D. Plump. Hierarchical graph transformation. *Journal of Computer and System Sciences*, 64:249–283, 2002.

[6] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. Fundamental theory of typed attributed graph transformation based on adhesive HLR-categories. *Fundamenta Informaticae*, 74(1):31–61, 2006.

[7] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs of Theoretical Computer Science. Springer, 2006.

[8] J. L. Fiadeiro and A. Lopes. An algebraic semantics of event-based architectures. *Mathematical Structures in Computer Science*, 17(5):1029–1073, 2007.

[9] J. L. Fiadeiro, A. Lopes, and L. Bocchi. Algebraic semantics of service component modules. In *WADT*, pages 37–55, 2006.

[10] J. L. Fiadeiro, A. Lopes, and L. Bocchi. An abstract model of service discovery and binding. *Formal Asp. Comput.*, 23(4):433–463, 2011.

[11] F. Orejas and L. Lambers. Symbolic attributed graphs for attributed graph transformation. In *Int. Coll. on Graph and Model Transformation. On the occasion of the 65th birthday of Hartmut Ehrig*, 2010.

[12] J. Padberg. Categorical approach to horizontal structuring and refinement of high-level replacement systems. *Applied Categorical Structures*, 7(4):371–403, 1999.

[13] W. Palacz. Algebraic hierarchical graph transformation. *J. Comput. Syst. Sci.*, 68(3):497–520, 2004.

[14] W. Reisig. Modeling and analysis techniques for web services and business processes. In *FMOODS*, volume 3535 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2005.

[15] W.M.P. van der Aalst, M. Beisiegel, K. M. van Hee, D. König, and C. Stahl. A soa-based architecture framework. In *The role of business processes in service oriented architectures*, volume 06291 of *Dagstuhl seminar proceedings*. Schloss Dagstuhl, 2006.

A Proofs

In this appendix we prove that **HSymbGraphs$_D$** is M-adhesive.

Proposition A.1 $\mathcal{M}$*-morphisms in* **HSymbGraphs$_D$** *are closed under isomorphism, composition and decomposition*

Proof

If h is an $\mathcal{M}$-morphism and g is an isomorphism, then all the morphisms inside h are symbolic $\mathcal{M}$-morphisms and all the morphisms inside g are symbolic isomorphisms. Since symbolic $\mathcal{M}$-morphisms are closed under isomorphism, all the morphisms inside the composition of h and g are symbolic $\mathcal{M}$-morphisms, which means that this composition is also an $\mathcal{M}$-morphism.

To prove that $\mathcal{M}$-morphisms are closed under composition, we proceed by induction:

- If HG is a graph in $\mathcal{HG}_0$ and $h : HG_0 \to HG_1$, $g : HG_1 \to HG_2$ are $\mathcal{M}$-morphisms then $h : HG_0^{top} \to HG_1^{top}$ and $g^{top} : HG_1^{top} \to HG_2^{top}$ are symbolic graph $\mathcal{M}$-morphisms, implying that $g^{top} \circ h$ is also a symbolic graph $\mathcal{M}$-morphism. This means that $g^{top} \circ h : HG_0 \to HG_2$ is a hierarchical $\mathcal{M}$-morphism.

- If $h : HG_0 \to HG_1$, $g : HG_1 \to HG_2$ are $\mathcal{M}$-morphisms then $h^{top} : HG_0^{top} \to HG_1^{top}$ and $g^{top} : HG_1^{top} \to HG_2^{top}$ are symbolic graph $\mathcal{M}$-morphisms and, for all edges e_0 in HG_0^{top} and e_1 in HG_1^{top}, h^{e_0} and g^{e_1} are $\mathcal{M}$-morphisms. But this means that $g^{top} \circ h^{top}$ is a symbolic graph $\mathcal{M}$-morphism and for every edge e_0 in HG_0^{top}, $g^{e_1} \circ h^{e_0}$ is an $\mathcal{M}$-morphism, where $e_1 = h^{top}(e_0)$.

Finally, using again induction, we can see that $\mathcal{M}$-morphisms are closed under decomposition, meaning that if g and $g \circ h$ are $\mathcal{M}$-morphisms, then h is also an $\mathcal{M}$-morphism:

- If HG is a graph in $\mathcal{HG}_0$, given hierarchical morphisms $h : HG_0 \to HG_1$, $g : HG_1 \to HG_2$ such that g and $g \circ h$ are $\mathcal{M}$-morphisms, we have that g^{top} and $g^{top} \circ h$ are symbolic graph $\mathcal{M}$-morphisms. By the decomposition property of symbolic $\mathcal{M}$-morphisms, this means that h is also a symbolic $\mathcal{M}$-morphism, implying that h is a hierarchical $\mathcal{M}$-morphism.

- Given hierarchical morphisms $h : HG_0 \to HG_1$, $g : HG_1 \to HG_2$, such that g and $g \circ h$ are $\mathcal{M}$-morphisms, we have that g^{top} and $g^{top} \circ h^{top}$ are symbolic graph $\mathcal{M}$-morphisms, for every edge e_1 in HG_1^{top}, g^{e_1} is a hierarchical $\mathcal{M}$-morphism, and for every edge e_0 in HG_0^{top}, $g^{h^{top}(e_0)} \circ h^{e_0}$ is also a hierarchical $\mathcal{M}$-morphism. On the one hand, by the decomposition property of symbolic $\mathcal{M}$-morphisms, we have that h^{top} is a symbolic $\mathcal{M}$-morphism. On the other hand, by induction, he have that for every $e_0 \in HG_0^{top}$, h^{e_0} is a hierarchical $\mathcal{M}$-morphism. Therefore, h is a hierarchical $\mathcal{M}$-morphism. ∎

Let us now see that **HSymbGraphs$_D$** has pushouts and pullbacks. However, instead of proving directly the existence of pushouts, we will prove the existence of general colimits.

Proposition A.2 **HSymbGraphs$_D$** *has colimits.*

Proof

Given a diagram D consisting of a family of hierarchical morphisms $\{h_i : HG_{i_1} \to HG_{i_2}\}_{i \in D}$, we define its colimit by induction:

- If all the graphs involved are in $\mathcal{HG}_0$, the colimit in **HSymbGraphs$_\mathbf{D}$** essentially coincides with the colimit in **SymbGraphs$_\mathbf{D}$** .

- If each graph involved HG_j is in $\mathcal{HG}_{i_j}$, with $i_j \leq k+1$, the colimit of the diagram HG and the corresponding morphisms $g_j : HG_j \to HG$ are defined as follows:

  - HG_{top} and g_j^{top} are given by the colimit of the diagram $\{h_i^{top}\}_{i \in I}$ in **SymbGraphs$_\mathbf{D}$**:

  - For every edge e in HG^{top}, $cts^{HG}(e)$ is the colimit of the diagram including all the graphs $cts^{HG_j}(e')$, where $e = g_j^{top}(e')$, and all the morphisms $h_m^{e'} : cts^{HG_{m_1}}(e') \to cts^{HG_{m_2}}(e'')$, where $e'' = h_m^{top}(e')$ and $e = g_{m_2}^{top}(e'')$. By induction, we may assume that this colimit exists.

  - For every edge e_j in HG_j^{top}, $g_j^{e_j}$ is the canonical morphism defined by the colimit associated to the edge $g_j^{top}(e_j)$ in HG^{top} defined in the item above.

It is routine to prove that this construction is indeed a colimit. ∎

As a consequence, we have:

Corollary A.3 HSymbGraphs$_\mathbf{D}$ *has pushouts.*

Now, we prove the existence of pullbacks:

Proposition A.4 HSymbGraphs$_\mathbf{D}$ *has pullbacks.*

Proof
Given hierarchical morphisms $h_i : HG_i \to HG_3$, for $i \in \{1, 2\}$, we define its pullback by induction:

- If HG_1, HG_2, HG_3 are graphs in $\mathcal{HG}_0$, the pullback in **HSymbGraphs$_\mathbf{D}$** essentially coincides with the pullback in **SymbGraphs$_\mathbf{D}$** .

- If HG_1, HG_2 and HG_3 are graphs in $\mathcal{HG}_{i_1}$, $\mathcal{HG}_{i_2}$ and $\mathcal{HG}_{i_3}$, respectively, with $i_1, i_2, i_2 \leq i + 1$, the pullback:

$$
\begin{array}{ccc}
HG_0 & \xrightarrow{\ h_1\ } & HG_1 \\
{\scriptstyle h_2}\downarrow & & \downarrow{\scriptstyle g_1} \\
HG_2 & \xrightarrow[\ g_2\ ]{} & HG_3
\end{array}
$$

is defined as follows:

  - HG_0^{top}, h_1^{top} and h_2^{top} are given by the pullback in **SymbGraphs$_\mathbf{D}$**:

$$
\begin{array}{ccc}
HG_0^{top} & \xrightarrow{\ h_1^{top}\ } & HG_1^{top} \\
{\scriptstyle h_2^{top}}\downarrow & & \downarrow{\scriptstyle g_1^{top}} \\
HG_2^{top} & \xrightarrow[\ g_2^{top}\ ]{} & HG_3^{top}
\end{array}
$$

  - For every edge e_0 in HG_0^{top}, we have that $cts^{HG_0}(e_0)$ and $h_1^{e_0}$ and $h_2^{e_0}$ are given by the pullback:

$$cts^{HG_0}(e_0) \xrightarrow{h_1^{e_0}} cts^{HG_1}(e_1)$$

$$h_2^{e_0} \downarrow \qquad\qquad \downarrow g_1^{e_1}$$

$$cts^{HG_2}(e_2) \xrightarrow[c^{e_2}]{} cts^{HG_3}(e_3)$$

where $e_1 = h_1^{top}(e_0)$, $e_2 = h_2^{top}(e_0)$ and $e_3 = g_1^{top}(e_1) = g_2^{top}(e_2)$.

Again, it is routine to prove that this construction is indeed a pullback. ■

Pushouts and pullbacks preserve $\mathcal{M}$-morphisms:

Proposition A.5 *If the diagram below is a pushout and h_1 is an $\mathcal{M}$-morphism then g_2 is also an $\mathcal{M}$-morphism. Similarly, if the diagram below is a pullback and g_2 is an M-morphism then h_1 is also an $\mathcal{M}$-morphism.*

$$HG_0 \xrightarrow{h_1} HG_1$$

$$h_2 \downarrow \qquad\qquad \downarrow g_1$$

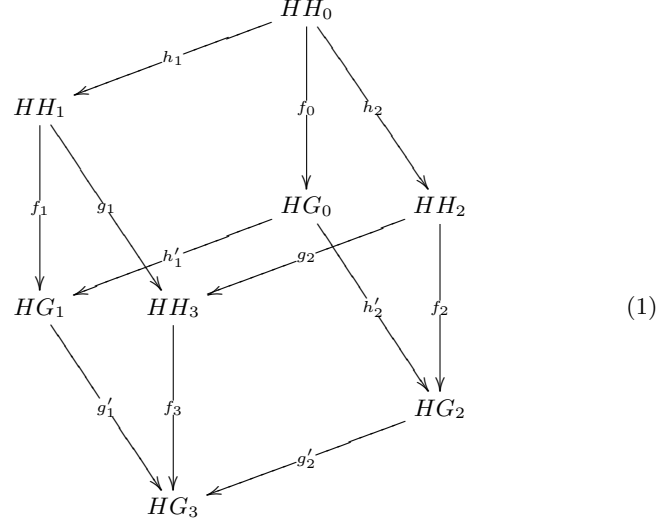$$HG_2 \xrightarrow[g_2]{} HG_3$$

Proof
Again by induction, the case base is trivial, since pushouts (resp. pullbacks) in **SymbGraphs$_\mathbf{D}$** preserve symbolic graph $\mathcal{M}$-morphisms. The general case for pullbacks is simple. In particular, it is enough to notice that the pullback of the top level morphisms preserve symbolic graph $\mathcal{M}$-morphisms, and pullbacks of the down level morphisms may be assumed, by induction, to preserve $\mathcal{M}$-morphisms. For pushouts, the general case is slightly more involved. First, as before, we know that the pushout of the top level morphisms preserve symbolic graph $\mathcal{M}$-morphisms. Then, considering that are graphs are assumed to be finite, it is enough to notice that each colimit of the down level morphisms can be defined as a combination of pushouts and pullbacks over $\mathcal{M}$-morphisms. Then, by induction we know that each of these pushouts preserves $\mathcal{M}$-morphisms and we also know that the composition of the resulting $\mathcal{M}$-morphisms is also an $\mathcal{M}$-morphism. ■

Proposition A.6 *Pushouts along hierarchical $\mathcal{M}$-morphisms are weak van Kampen squares.*
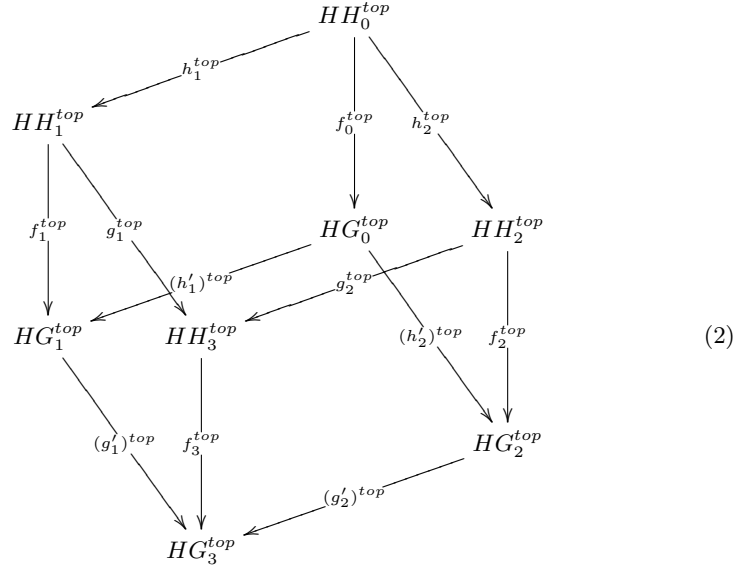
Proof
Let us consider the following commutative cube, where $h_1, h_1', g_2, g_2', f_1, f_2, f_3$ are $\mathcal{M}$-morphisms, the bottom square is a pushout and the back faces are pullbacks. We have

13

to show that the top square is a pushout if and only if the front faces are pullbacks.

$$
\begin{array}{c}
HH_0 \\
\xrightarrow{h_1} \quad f_0 \quad h_2 \\
HH_1 \\
f_1 \quad g_1 \qquad HG_0 \quad HH_2 \\
h_1' \qquad g_2 \\
HG_1 \quad HH_3 \qquad h_2' \quad f_2 \\
g_1' \quad f_3 \qquad HG_2 \\
g_2' \\
HG_3
\end{array}
\tag{1}
$$

Again, we proceed by induction. If all the graphs in the diagram are in $\mathcal{HG}_0$, then the property trivially holds, because the diagram would essentially be a weak van Kampen square in **SymbGraphs$_\mathbf{D}$**.

Let us consider the general case and suppose that the top square is a pushout, let us show that the two front faces are pullbacks. We know that the corresponding cube in terms of the top graphs and the top morphisms:

$$
\begin{array}{c}
HH_0^{top} \\
\xrightarrow{h_1^{top}} \quad f_0^{top} \quad h_2^{top} \\
HH_1^{top} \\
f_1^{top} \quad g_1^{top} \qquad HG_0^{top} \quad HH_2^{top} \\
(h_1')^{top} \qquad g_2^{top} \\
HG_1^{top} \quad HH_3^{top} \qquad (h_2')^{top} \quad f_2^{top} \\
(g_1')^{top} \quad f_3^{top} \qquad HG_2^{top} \\
(g_2')^{top} \\
HG_3^{top}
\end{array}
\tag{2}
$$

is a weak van Kampen square in **SymbGraphs$_\mathbf{D}$**, therefore its front faces are pullbacks in that category. Hence, we have to show that for every edge e_1 in HH_1^{top}, $cts^{HH_1}(e_1)$ is the pullback of $f_3^{e_3}$ and $(g_1')^{e_1'}$, where $e_3 = g_1^{top}(e_1)$ and $e_1' = f_1^{top}(e_1)$. Now, let $e_3' = f_3^{top}(e_3)$. We have two cases:

- If there is no edge e_0 in HH_0^{top}, such that $e_1 = h_1^{top}(e_0)$ then we know that there is also no edge e_0' in HG_0^{top}, such that $f_1^{top}(e_1) =$

14

$(h'_1)^{top}(e_0)$, since HH_0^{top} is the pullback of the back left square in (2). In addition, if we call $e_3 = g_1^{top}(e_1)$, we know that there is also no edge e_2 in HH_2^{top} such that $e_3 = g_2^{top}(e_2)$ since the top diagram in (2) is a pushout. Moreover, for similar reasons, we may also be sure that there is no edge e'_2 in HG_2^{top} such that $f_3^{top}(e_3) = (g'_2)^{top}(e'_2)$. This means, by the definition of pushouts of hierarchical graphs, that $cts^{HH_1}(e_1) = cts^{HH_3}(e_3)$ and $cts^{HG_1}(e'_1) = cts^{HG_3}(e'_3)$ and the morphisms $g_1^{e_1}$ and $(g'_1)^{e'_1}$ are identities.

Now, let us consider the diagram below:
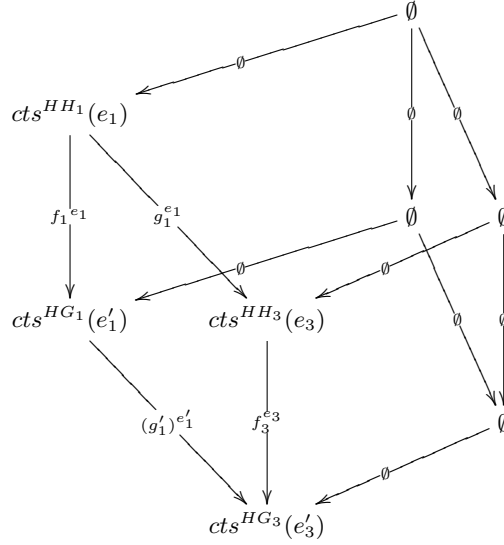
where $\emptyset$ denotes the empty graph or the empty morphism, depending on the context. Now, by construction, we may see that, in the above diagram, the bottom face and the top face are pushouts and the back faces are pullbacks. Therefore, by induction, the front left face is a pullback.

- If there is an edge e_0 in HH_0^{top} such that $e_1 = h_1^{top}(e_0)$, then this edge must be unique, since h_1 is an $\mathcal{M}$-morphism. However, if we call $e_3 = g_1^{top}(e_1)$, there may be several edges d_1 in HH_1^{top} such that $e_3 = g_1^{top}(d_1)$. Moreover, for each d_1 there must be exactly an edge d_0 such that $h_1^{top}(d_0) = d_1$, since h_1 is an $\mathcal{M}$-morphism and the top face diagram of (2) is a pushout. And for the same reasons, for all these edges d_0, $h_2^{top}(e_0) = h_2^{top}(d_0)$. Let us call e_2 the edge in HH_2^{top} such that $e_2 = h_2^{top}(e_0)$. This means that, in general, $cts^{HH_3}(e_3)$ is not the result of the pushout of $h_1^{e_0}$ and $h_2^{e_0}$, but it is the result of the colimit involving all the morphisms $h_1^{d_0}$ and $h_2^{d_0}$. Similarly, if we call $e'_i = f_i^{top}(e_i)$, for each such edges d_0 and d_1 there would be exactly two edges d'_0 and d'_1 in HG_0^{top} and HG_1^{top}, where $(g'_1)^{top}(d'_1) = e'_3$ and $(h'_1)^{top}(d'_0) = d'_1$. In particular, $d'_0 = f_0^{top}(d_0)$ and $d'_1 = f_1^{top}(d_1)$. Moreover, $cts^{HG_3}(e'_3)$ is the result of the colimit involving all the morphisms $(h'_1)^{d'_0}$ and $(h'_2)^{d'_0}$.

Now, we proceed by induction on the number of these edges, proving that for any number n of such edges d_{01}, d_{0n}, we can build a weak van Kampen square:
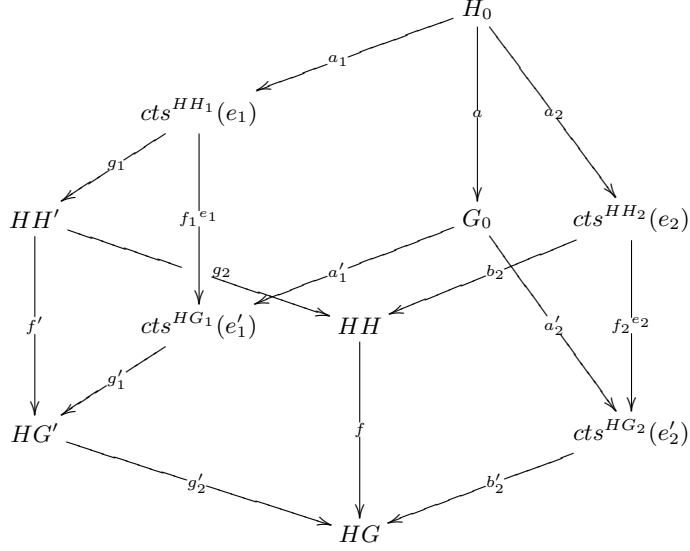
where $a_1, a_1', g_1,$ and g_1' are $\mathcal{M}$-morphisms, the top face and bottom face diagrams are pushouts (i.e. the top face is a pushout of the morphisms $g_1 \circ a_1$ and a_2 and the bottom face is a pushout of the morphism $g_1' \circ a_1'$ and a_2'), where all the vertical squares are pullbacks and where HH is the colimit of all the morphisms $h_1^{d_0}$ and $f_0^{d_0}$ and HG is the colimit of all the morphisms $h_1^{d_0'}$ and $f_0^{d_0'}$ and $f, g_2 \circ g_1$, and $g_2' \circ g_1'$ are morphisms induced by these colimits.

- If there is only one edge d_1 in HH_1^{top} such that $e_3 = g_1^{top}(d_1)$, i.e. $e_1 = d_1$, then we have that the cube below, by induction on the depth of the graphs, is a weak van Kampen square, where the top square is a pushout:
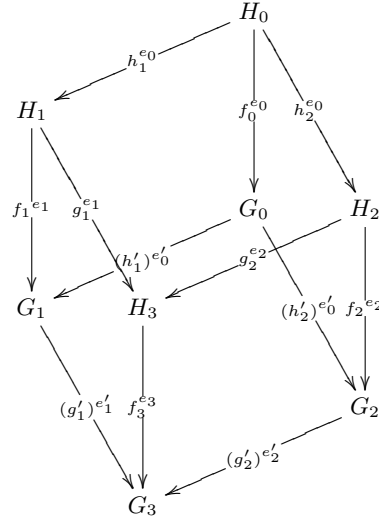

where, $e_2 = h_2^{top}(e_0)$ and, for every $0 \le i \le 3$, $e_i' = f_i(e_i)$, $H_i = cts^{HH_i}(e_i)$ and $G_i = cts^{HG_i}(e_i')$. Therefore, the this

cube satisfies the induction hypothesis when g_1 and g_1' are the identity morphisms.

– If there are $n + 1$ such edges, by induction we know that there is a weak van Kampen square associated to n edges:
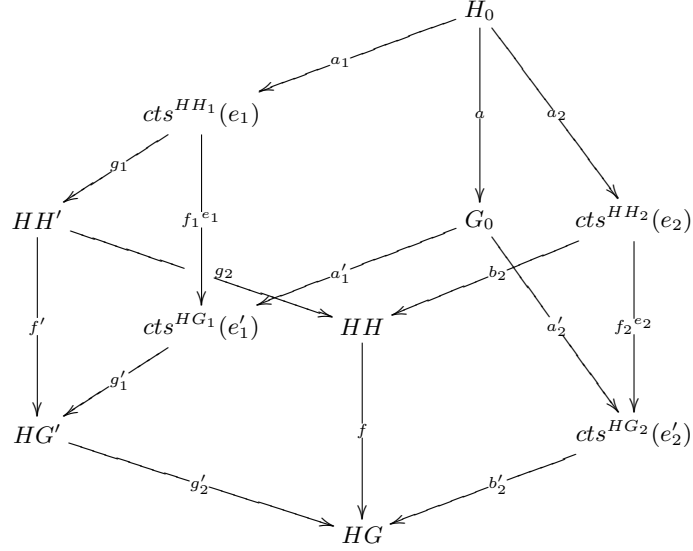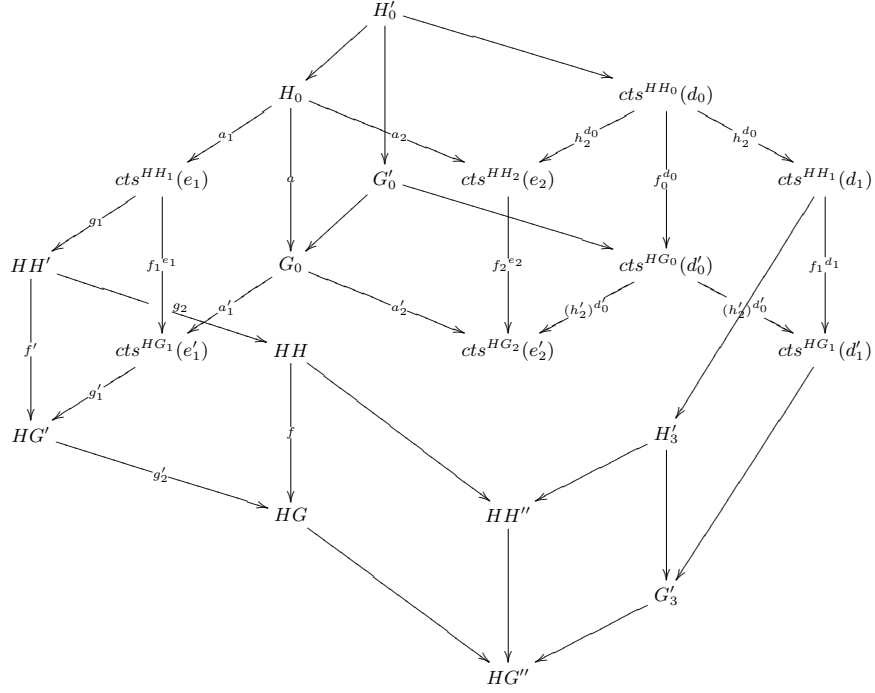

where the top face is a pushout and HH and HG are the colimit of the morphisms associated to the given edges. Let d_1 be the remaining edge and let us consider the following diagram:
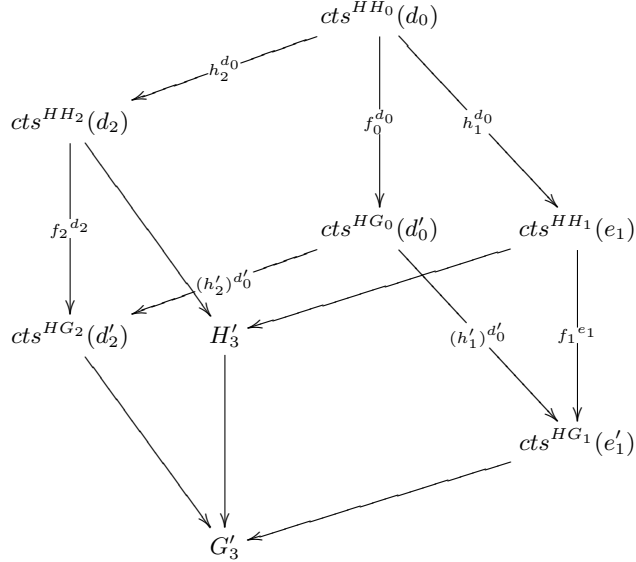
where, d_0 is the only edge in HH_0^{top} such that $d_1 = h_1^{top}(d_0)$, for every $i = 0, 1$, $d_i' = f_i^{top}(d_i)$, the top and bottom squares are pushouts and the (unnamed) morphism from H_3' to G_3' is the universal morphism associated to the top face pushout. By

17

induction on the depth of the graphs, this diagram is a weak van Kampen square where the top and bottom faces are pushouts and the rest of faces are pullbacks. Let us now put together (and extend) the two diagrams above, skipping some arrows which are not important now:

$$
\begin{array}{c}
H_0' \\
H_0 \qquad cts^{HH_0}(d_0) \\
a_1 \quad a_2 \quad h_2^{d_0} \quad h_2^{d_0} \\
cts^{HH_1}(e_1) \quad G_0' \quad cts^{HH_2}(e_2) \quad f_0^{d_0} \quad cts^{HH_1}(d_1) \\
g_1 \quad f_1^{e_1} \quad G_0 \quad f_2^{e_2} \quad cts^{HG_0}(d_0') \quad f_1^{d_1} \\
HH' \quad g_2 \quad a_1' \quad a_2' \quad (h_2')^{d_0'} \quad (h_2')^{d_0'} \\
f' \quad cts^{HG_1}(e_1') \quad HH \quad cts^{HG_2}(e_2') \quad cts^{HG_1}(d_1') \\
HG' \quad g_1' \quad f \quad H_3' \\
g_2' \quad HG \quad HH'' \quad G_3' \\
HG''
\end{array}
$$

where H_0' and G_0' are, respectively, the pullbacks of a_2 and $h_2^{d_0}$, and of a_2' and $(h_2')^{d_0'}$, HH'' is the pushout of the composed morphisms $H_0' \to HH$ and $H_0' \to H_3'$, similarly, HG'' is the pushout of the morphisms $G_0' \to HG$ and $G_0' \to G_3'$, and the rest of the arrows are part of or induced by these pushouts and pullbacks. Now, by induction of the depth of the graphs, this diagram is again a weak van Kampen square, where all the vertical diagrams are pullbacks by composition and decomposition of pullbacks and the top and bottom diagrams are pushouts by construction. Therefore, the front faces are also pullbacks. Moreover, by construction, HH'' and HG'' are the colimit of the morphisms associated to the given edges.

The proof that the front right face is also a pullback is similar to the previous proof.

Finally, we have to show that if the two front faces are pullbacks then the top face is a pushout. Again, we know that the corresponding cube in terms of the top graphs and the top morphisms is a van Kampen square in $\mathbf{SymbGraphs_D}$, therefore its top face is a pushout in that category. Hence, we have to show that for every edge e_3 in HH_3^{top} $cts^{HH_3}(e_3)$ is the colimit of all the morphisms of $h_1^{e_0}$ and $h_2^{e_0}$ for all edges e_0 such that $e_3 = g_1^{top}(e_1)$, where $e_1 = h_1^{top}(e_0)$. We proceed by induction on the number of edges e_0 such that $e_3 = g_1^{top}(e_1) = g_2^{top}(e_2)$, where $e_1 = h_1^{top}(e_0)$ and $e_2 = h_2^{top}(e_0)$. Notice that, for all these edges e_0, $e_2 = h_2^{top}(e_0)$ is always the same edge, since g_2^{top} is a monomorphism. In particular we prove that for any number n of such edges, given

graphs H_3 and G_3, if G_3 is the colimit of all the morphisms $(h'_1)^{e'_0}$ and $(h'_2)^{e'_0}$, where $e'_0 = f_0^{top}(e_0)$, and for each diagram:
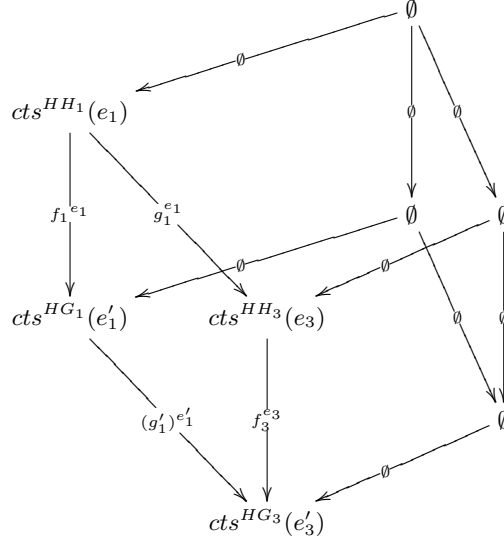
$$cts^{HH_0}(e_0)$$

(3)

where for every $i = 0, 2$, $e'_i = f_i^{top}(e_i)$, all the vertical faces are pullbacks, then we have that H_3 is the colimit of all the morphisms of $h_1^{e_0}$ and $h_2^{e_0}$ for all these edges e_0. In particular, since we assume that $cts^{HG_3}(e'_3)$ is the colimit of all the morphisms $(h'_1)^{e'_0}$ and $(h'_2)^{e_0}$, and if we replace H_3 and G_3 in diagram (3) by $cts^{HH_3}(e_3)$ and $cts^{HG_3}(e'_3)$, respectively, then all the vertical faces are pullbacks, this would imply that $cts^{HH_3}(e_3)$ is the colimit of all the morphisms of $h_1^{e_0}$ and $h_2^{e_0}$, as we want to prove.

- If there are no edges e_0 such that $e_3 = g_1^{top}(e_1)$, where $e_1 = h_1^{top}(e_0)$, this means that there must be either an edge e_1 in HH_1^{top} or an edge e_2 in HH_2^{top} such that $e_3 = g_1^{top}(e_1)$ or $e_3 = g_2^{top}(e_2)$. Let us assume that the existing edge is e_1 (in the case of e_2 the proof is similar). In this case, we have to prove that $cts^{HH_3}(e_3) = cts^{HH_1}(e_1)$, since this is equivalent to show that that the diagram below is a colimit:

$$\emptyset \xrightarrow{\emptyset} cts^{HH_1}(e_1)$$

Now, we can see that there is no edge e'_0 in HG_0^{top} such that $(h'_1)^{top}(e'_0) = f_1^{top}(e_1)$, since we know that diagram (2) above is a weak van Kampen square, where the back left face is a pullback, and this would have implied that there would have been an edge e_0 in HH_0^{top} such that $h_2^{top}(e_0) = e_2$. For similar reasons, i.e. the front right face of (2) is a pullback, we know that there does not exist an edge e'_2 in HG_2^{top} such that $(g'_2)^{top}(e'_2) = f_3^{top}(e_3)$. Then, let us now consider the following diagram:

where $e_1' = f_1^{top}(e_1)$ and $e_3' = f_3^{top}(e_3)$. By construction and knowing that diagram (1) is a weak van Kampen square where the front faces are pullbacks, the above diagram would also be a weak van Kampen square where the front faces are pullbacks. Hence, by induction, the top face would be a pushout, i.e. a colimit.

- Assume that there are $n+1$ edges e_0 and H_3 and G_3 are graphs such that G_3 is the colimit of all the morphisms $(h_1')^{e_0'}$ and $(h_2')^{e_0'}$, where $e_0' = f_0^{top}(e_0)$, and for each diagram:

(4)
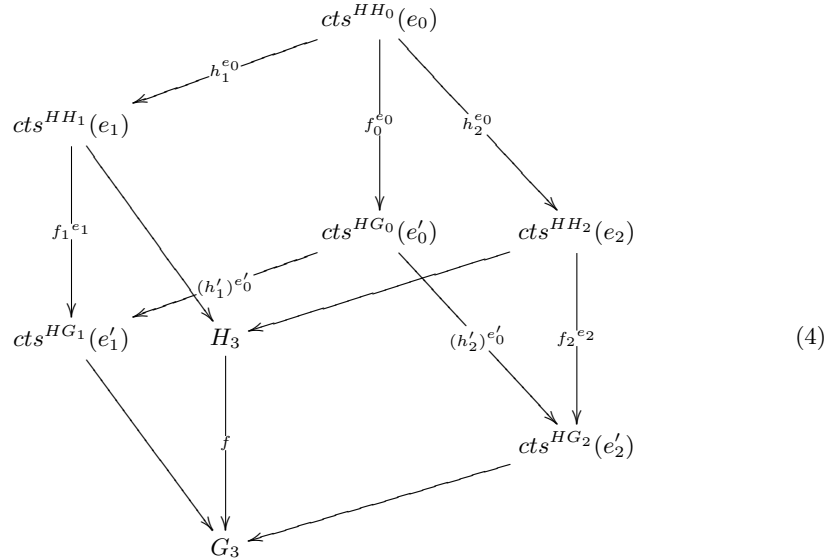
where for every $i = 0, 2$, $e_i' = f_i^{top}(e_i)$, all the vertical faces are pullbacks. Then, we have to prove that H_3 is the colimit of all the morphisms $h_1^{e_0}$ and $h_2^{e_0}$ for the $n+1$ edges e_0. Let d_0 be one of these $n+1$ edges, let $d_0' = f_0^{top}(d_0)$ and G_3' be the colimit of all

the morphisms $(h_1')^{e_0'}$ and $(h_2')^{e_0'}$ for the n remaining e_0' edges. We define the graph H_3' as the pullback of the diagram below:

$$
\begin{array}{ccc}
H_3' & \xrightarrow{\ g\ } & H3 \\
\downarrow{\scriptstyle f'} & & \downarrow{\scriptstyle f} \\
G_3' & \xrightarrow[\ g'\ ]{} & G_3
\end{array}
\qquad\qquad (5)
$$

where g' is the universal morphism given by the colimit property of G_3'. We can see that H_3' satisfies that for each edge e_0 different from d_0' we can build a diagram:
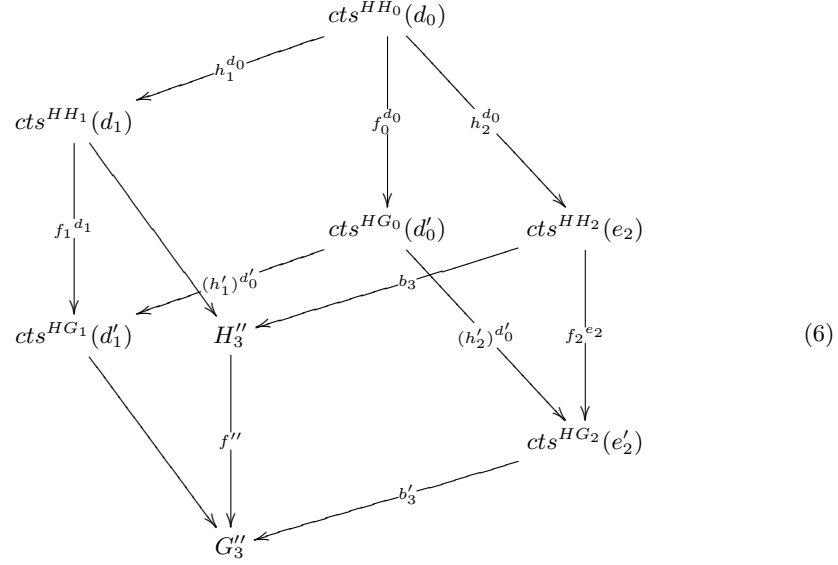
$$
\begin{array}{c}
cts^{HH_0}(e_0) \\
\end{array}
\qquad\qquad (6)
$$

where all its vertical faces are pullbacks. In particular, the back faces of diagram (6) coincide with the back faces of diagram (4) which are assumed to be pullbacks, therefore it is enough to build the front faces by pullback decomposition of the front faces of diagram (4) and diagram (5). This means that, by induction, H_3' is the colimit of all the morphisms $h_1^{e_0}$ and $h_2^{e_0}$ for all these edges e_0 different from d_0.

Similarly, if G_3'' is defined by the pushout below and H_3'' is the defined by the pullback below:
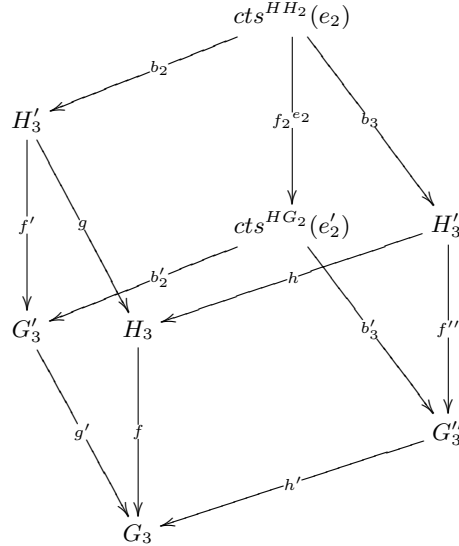
$$
\begin{array}{ccc}
cts^{HG_0}(d_0') & \xrightarrow{(h_1')^{d_0'}} & cts^{HG_1}(d_1') \\
\downarrow{\scriptstyle (h_2')^{d_0'}} & (PO) & \downarrow{\scriptstyle c_1'} \\
cts^{HG_2}(e_2') & \xrightarrow[\ c_2'\ ]{} & G_3''
\end{array}
\qquad
\begin{array}{ccc}
H_3'' & \xrightarrow{\ h\ } & H3 \\
\downarrow{\scriptstyle f''} & (PB) & \downarrow{\scriptstyle f} \\
G_3'' & \xrightarrow[\ h'\ ]{} & G_3
\end{array}
$$

where g'' is the universal morphism given by the colimit property of G_3'. Then, again, H_3'' satisfies that we can build a diagram:

21

$$cts^{HH_0}(d_0)$$

$$h_1^{d_0}$$

$$cts^{HH_1}(d_1)$$

$$f_0^{d_0} \qquad h_2^{d_0}$$

$$f_1^{d_1}$$

$$cts^{HG_0}(d_0') \qquad cts^{HH_2}(e_2)$$

$$(h_1')^{d_0'} \qquad b_3$$

$$cts^{HG_1}(d_1') \qquad H_3'' \qquad (h_2')^{d_0'} \qquad f_2^{e_2}$$

$$\tag{6}$$

$$f'' \qquad cts^{HG_2}(e_2')$$

$$b_3'$$

$$G_3''$$

where all its vertical faces are pullbacks. Moreover, by construction or by assumption, all the vertical arrows are $\mathcal{M}$-morphisms and so is $h_1^{d_0}$, and also by construction the bottom diagram is a pushout. Therefore, the diagram is a weak van Kampen square and, so, the top diagram is a pushout.

Finally, consider the following diagram:

$$cts^{HH_2}(e_2)$$

$$b_2$$

$$H_3'$$

$$f_2^{e_2} \qquad b_3$$

$$f' \qquad g$$

$$cts^{HG_2}(e_2') \qquad H_3''$$

$$b_2' \qquad h$$

$$G_3' \qquad H_3 \qquad b_3' \qquad f''$$

$$g' \qquad f \qquad G_3''$$

$$h'$$

$$G_3$$

In this diagram all the vertical faces are pullbacks by construction and it is routine to prove that the bottom diagram is a pushout, since G_3 is the colimit of all the morphisms $(h_1')^{e_0'}$ and $(h_2')^{e_0'}$, G_3' is the colimit of all these morphisms except $(h_1')^{d_0'}$ and $(h_2')^{d_0'}$, and G''_3 is the pushout of $(h_1')^{d_0'}$ and $(h_2')^{d_0'}$. This means that the above diagram is a weak van Kampen square and, as a consequence, the

top diagram is a pushout. But this means that H_3 is the colimit of all the morphisms $(h_1)^{e_0}$ and $(h_2)^{e_0}$

∎

So, as a consequence of Propositions A.1, A.3, A.4, A.5, and A.6 we have:

Theorem A.7 HSymbGraphs$_\mathbf{D}$ *is an $\mathcal{M}$-adhesive category.*