A graph-semantics of business configurations

José Luiz Fiadeiro[1], Nikos Mylonakis[2], and Fernando Orejas[2]

[1]Royal Holloway, University of London
[2]Department of Computer Science, UPC

September 10, 2014

Abstract

In this paper we give graph-semantics to a fundamental part of the semantics of the service modeling language *SRML*. To achieve this goal we develop a new graph transformation system for what we call 2-level symbolic graphs. These kind of graphs extend symbolic graphs with a simple 2-level hierarchy that can be generalized to arbitrary hierarchies. We formalize the semantics using this new graph transformation system using a simple example of a trip booking agent.

1 Introduction

SRML ([8, 10, 11]) is a service modeling language designed within the project SENSORIA. Its state model is considered at two levels 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 2-level graphs, based on the notion of symbolic graph and symbolic graph transformation [12], 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*. We believe that this new semantics has the advantage that an implementation could be relatively easy if we had a tool for the graph transformation system that we propose.

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 2-level graphs. Section 5 is dedicated to showing how we can define part of the semantics of $SRML$ using hierarchical graph transformation. Finally, in Section 6, we discuss some related work and conclude the paper.

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, 10, 11] for a detailed description of the language. Roughly speaking, a module can be seen as a graph of components that 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) that 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.

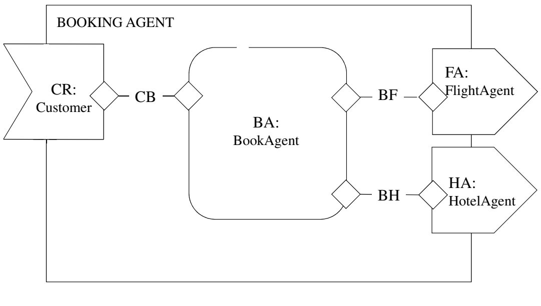


Figure 1: BOOKING AGENT service module

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.

```
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' ∧
. . .
```

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, destination, 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 *booktrip* 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. Another issue of $SRML$ that we do not treat for reasons of space is service level agreement (SLA).

As we saw in the definition of our service oriented architecture, business configurations 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 business configuration also includes the values contained by the local variables of wires and components and the events that are pending to be executed.

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. 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.

3 Symbolic graphs and symbolic graph transformation

Symbolic (hyper)graphs [12] 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 symbolic graph in Figure 2 specifies a class of attributed graphs, including distances in the edges, that satisfy the well-known triangle inequality.

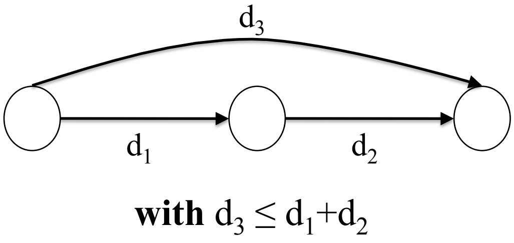


Figure 2: A symbolic graph

The notion of symbolic graph is based on the notion of a special kind of labeled graphs called E-graphs (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, for every graph G, instead of having source and target functions that map edges to nodes, we have an attachment function, $attach_G$, that maps each (hyper)edge to a sequence of nodes, i.e. the nodes connected by the edge.

Definition 3.1 *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 $\mathbf{SymbGraphs_D}$. In [12] it is shown that $\mathbf{SymbGraphs_D}$ is an adhesive HLR category, which means that all the fundamental results of the theory of graph transformations apply to this kind of graphs [7].

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 number of 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 *A* symbolic graph transformation rule *is a triple $\langle \Phi_L, L \hookleftarrow K \to R, \Phi_R \rangle$, 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 $X_L = X_K \subseteq X_R$, $L \hookleftarrow K \to R$ 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 $\Phi_L \subseteq \Phi_R$.*

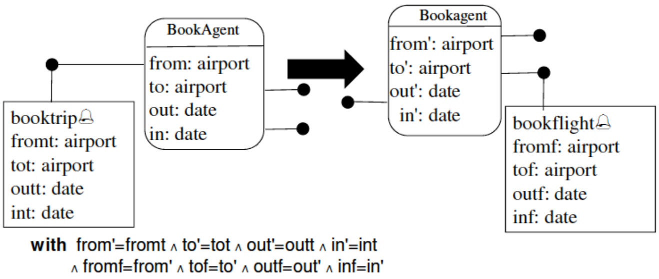


Figure 3: A symbolic rule

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.

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 $\mathbf{E}-\mathbf{Graphs}$ and $D \models \Phi_G \Rightarrow m(\Phi_L)$.

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

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

4 2-level symbolic graphs

In this section we introduce our notion of 2-level symbolic graph and it can be shown that these graphs, together with their associated notion of morphism, form an $\mathcal{M}$-adhesive category [7]. Our notion of 2-level graph is inspired by the notion of Petri Net refinement in [13]. According to that notion, in 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 t was connected. In our case, we consider that a 2-level graph is a graph whose edges include (standard) 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 graph with interface, where the interface consists of some nodes of the graph (more precisely a sequence of nodes). In particular, if e is an edge of a 2-level 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 may be considered to be simultaneously inside and outside e. For instance, in Fig. 4 we can see a 2-level 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_1n_2 and the attachment of e_2 may be $n_1n_2n_2$. The graph on the right shows the contents of the edges. In particular, e_1 has no contents 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 $n_1n_2n_2$. 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.

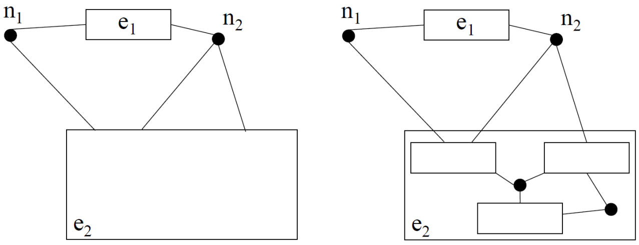


Figure 4: A 2-level graph

Definition 4.1 *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 and even if it is an abuse of notation, the class (category) of symbolic graphs with interface will also be denoted by $\mathbf{SymbGraphs_D}$.

Definition 4.2 *A 2-level graph TG is a pair $\langle TG^{top}, cts^{TG}\rangle$, where TG^{top} is a symbolic graph, the top level graph, and cts^{TG} is the contents function, $cts^{TG} : E_{TG^{top}} \to \mathbf{SymbGraphs_D}$ that, for every edge e in the top level graph, yields the graph included in that edge, such that the attachment of e coincides with the interface of the graph included in e, i.e. $attach_{TG^{top}}(e) = I_{cts^{TG}}(e)$.*

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

Definition 4.3 *The flattening of a 2-level graph, $Flat(TG)$ is defined as follows:*

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

Morphisms between 2-level graphs TG_1 and TG_2 can be seen as families of graph morphisms that map, on the one hand, the top level graph of TG_1 to the top level graph of TG_2 and, on the other, the contents of each edge e_1 of TG_1 to the contents of the corresponding edge of TG_2:

Definition 4.4 *A 2-level graph morphism $h : TG_1 \to TG_2$ is a pair $\langle h^{top}, h^{down}\rangle$, where $h^{top} : TG_1^{top} \to TG_2^{top}$ is a symbolic graph morphism between the top level graphs, and $h^{down} = \{h^e : cts^{TG_1}(e) \to cts^{TG_1}(h^{top}(e))\}_{e \in E_{TG_1^{top}}}$ is a family including a symbolic graph morphism for each edge in TG_1^{top}.*

In general, given a 2-level 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 included in h^{down}.

It is routine to see that 2-level symbolic graphs and morphisms over a data algebra D form a category, that we call $\mathbf{2SymbGraphs_D}$. 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 $\mathbf{SymbGraphs_D}$. In the appendix there is a quite lengthy detailed proof. In particular, pushouts in this category are built as follows. Given the diagram below, the top level graph of TG_3 is the pushout of the top level graphs and morphisms in the diagram and for every edge e_3 in TG_3^{top}, $cts^{TG_3}(e_3)$ is the colimit of the contents of each edge e_0 in TG_0^{top} such that $g_1^{top}(h_1^{top}(e_0)) = e_3$, of each edge e_1 in TG_1^{top} such that $g_1^{top}(e_1) = e_3$, and of each edge e_2 in TG_2^{top} such that $g_2^{top}(e_2) = e_3$.

$$\begin{tikzcd}
TG_0 \arrow[r, "h_1"] \arrow[d, "h_2"'] & TG_1 \arrow[d, "g_1"] \\
TG_2 \arrow[r, "g_2"'] & TG_3
\end{tikzcd}$$

5 A graph-semantics for business configurations

In our graph semantics business configurations are represented by 2-level symbolic graphs, whose hyperedges represent components and events at the lowest level and activities at the top level, and whose nodes represent wires. Additionally, in the semantics we will have two different graph transformation rules for these two ways of transforming the state: state transformation rules and reconfiguration rules.

Our graph semantics is presented in the next two subsections. In the first one, we present the graph semantics of business configurations, and in the second one its associated transformation system. The semantics that we propose is also more abstract than the original one [10] and the light version of [9], and therefore we do not use a concrete temporal logic in our provide and request specifications and we do not treat either the problem of name injections in wires.

5.1 Business configurations

In the first definition we present the basic concept of a state configuration:

Definition 5.1 *A state configuration is an E-graph with two types of hyperedges:*

- *Hyperedges that represent components with a positive number of nodes, an attribute with the name of the component and a set of attributes of the component. We will refer to them as component hyperedges.*
- *Hyperedges that represent events connected with just one node, an attribute with the name of the event, another with the type of the event and a set of attributes of the event. We will refer to them as event hyperedges.*

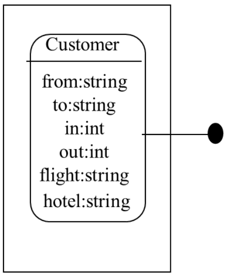
There are also two types of nodes: internal and interface nodes. Both types of nodes can be part of different component hyperedges and a set of event hyperedges. The main difference between these two types of nodes is that interface nodes are the ones with which subsystem binding is performed.

Next, we present the concept of business configuration:

Definition 5.2 *A business configuration is a 2-level graph where the top level is a graph without cycles such that:*

- *each hyperedge encapsulates one state configuration.*
- *the top level nodes can also optionally have a set of event hyperedges. If they are not connected to another node they are referred to as interface nodes. When these nodes have an event hyperedge, they triggered a process of selection of a reconfiguration rule package.*

For example, if a customer has developed an activity module that requests a booking agent to book a hotel and a flight, the 2-level symbolic graph that represents the initial business configuration with an instance of this activity module will consist of a hyperedge that represents the activity, and inside this hyperedge, another hyperedge that represents the customer component with a set of attributes for the flight and hotel reservation. The 2-level symbolic graph will also have a request specification in the with clause. A graphical representation is in figure 5 .



with from = “Barcelona” $\wedge$ to = “London” $\wedge$ in = 270712 $\wedge$ out = 150812 $\wedge$ Φ_{req}

Figure 5: Business configuration with just a customer activity

Additionally, in figures 7 and 9 we have two different stages of the initial business configuration of figure 5. In 7 we have a hyperedge encapsulating a customer subsystem with a set of attributes (from, to, in, out, ...). The hyperedge has an interface node with an event hyperedge. After triggering a process of selection of a reconfiguration rule package, the business configuration evolves to the one in figure 9, binding the interface node with an hyperedge with a booking agent subsystem inside. This new hyperedge has also two additional interface hierarchical nodes.

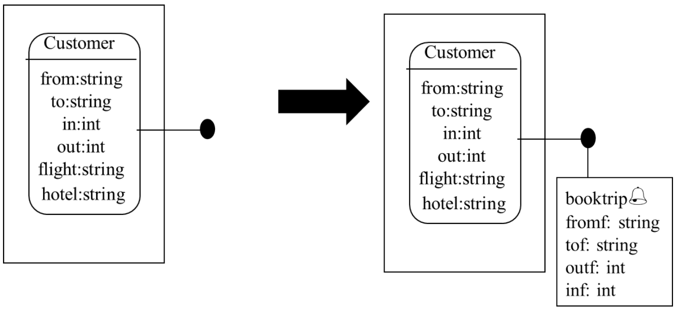
5.2 Transformation systems for business configurations

In this subsection we present first the two kinds of rules that we have in transformation systems for business configurations: state transformation rules and reconfiguration rules. After that we present reconfiguration rule packages that combine both kind of rules.

Definition 5.3 *A state transformation rule is a rule that can make the following transformations in one activity:*

- *process an event eliminating this event from a node of an hyperedge component;*
- *transform the values of the attributes of a component hyperedge using information of the processed events of its nodes;*
- *pend an event in the node of a hyperedge component.*

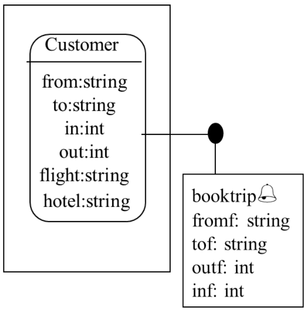
An example of state transformation rule is in figure 6: it pends an event in the hyperedge component of the customer.



with fromf = from $\wedge$ tof = to $\wedge$ inf = in $\wedge$ outf = out

Figure 6: Rule *initr* associated to the customer activity

Other possible rules are a rule for processing the information of the reply-event of the booking agent or a rule to start the payment.When the rule *initr* is applied to the business configuration, the initiating event is added to the business configuration. The resulting new business configuration is in figure 7.



with from = “Barcelona” $\wedge$ to = “London” $\wedge$ in = 270712 $\wedge$ out = 150812 $\wedge$ Φ_{req}

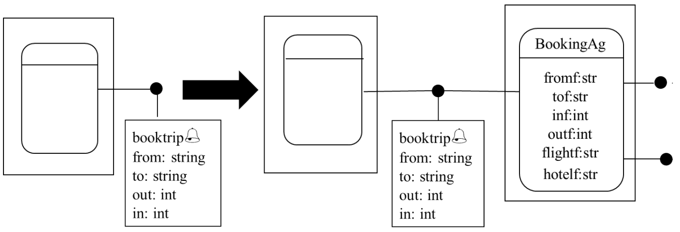
Figure 7: New business configuration with a trigger event

Definition 5.4 *A reconfiguration rule connects one business configuration with another.*

An example of a reconfiguration rule is in figure 8: it binds a business configuration with a customer component with a business configuration with a booking agent component. In the with clause we have the specification of what the booking agent provides (Φ_{prov}), and the specifications of what the booking agent requires of an hotel and a flight agent (Φ_{reqha} and Φ_{reqfa}).

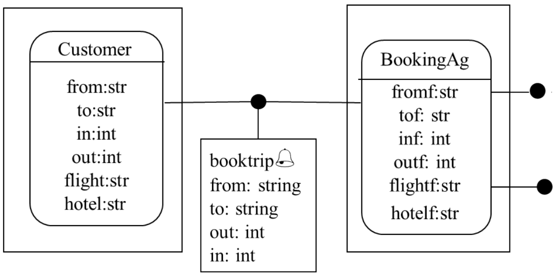
Definition 5.5 *A reconfiguration rule package contains one distinguished reconfiguration rule, and it additionally has an associated set of state transformation rules.*

The event in figure 7 triggers a process of selection of a reconfiguration rule together with a set of state transformation rules. The selected reconfiguration rule is in figure 8. After applying the reconfiguration rule, an instance of a booking agent module is connected to the instance of the customer activity module that is represented in figure 9.



with $\Phi_{\text{prov}} \wedge \Phi_{\text{reqfa}} \wedge \Phi_{\text{reqha}}$

Figure 8: A reconfiguration rule



with from = “Barcelona” $\wedge$ to = “London” $\wedge$ in = 270712 $\wedge$ out = 150812 $\wedge\ \Phi_{\text{req}}$
$\wedge\ m(\Phi_{\text{prov}}) \wedge m(\Phi_{\text{reqfa}}) \wedge m(\Phi_{\text{reqha}})$

Figure 9: Updated business configuration with a booking agent

A business repository contains all the possible services that are available at a certain time to make a binding in a process of selection of a reconfiguration rule package.

Definition 5.6 *A business repository is a set of reconfiguration rule packs.*

Now we present the concept of transformation systems for business configurations:

Definition 5.7 *A transformation system for business configurations consists of:*

- *a business configuration*
- *a business repository*
- *a set of state transformation rules.*

Finally we present the two different ways through which we can transform a transformation system for business configurations:

Definition 5.8 *A transformation step in a transformation system for business configuration can be one of the following:*

- *An application of a state transformation rule to the current business configuration. The result updates the business configuration.*
- *After a process of selection of a reconfiguration rule package by an interface node of an activity and at least an event hyperedge, the application of the distinguished rule of the selected reconfiguration package to the current business configuration. In this case we update again the subsystem configuration. The rest of the rules of the reconfiguration rule package are added to the current set of state transformation rules.*

In our running example, the initial business configuration of figure 5 has been transformed to the the business configuration of figure 7 by first applying the state transformation rule of figure 6. In a second step, after applying the distinguished rule of figure 8 of a reconfiguration rule pack to 7, we obtain the business configuration of figure 9. In order to successfully apply the rule one must prove that the specification of what the booking agent provides (Φ_{prov}) implies the specification of what the customer requires (Φ_{req}). The set of state transformation rules is then updated with the set of state transformation rules associated with the reconfiguration rule. These new set of rules will include rules to process the initiating event of the customer and generate two new initiating events to book a flight by a Flight Agent and to book a hotel by an Hotel Agent.

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 by the notion of Petri net refinement in [13]. It is also inspired by 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 labeled and do not support arbitrary attributes and attribute computation. Palacz, in [14], defines a much more general framework, where a hierarchical graph is a standard (non-attributed) 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 (e.g. see [8, 10, 11, 9]). In this paper we replace the explicit ad-hoc computation associated with the semantics of interactions by hierarchical symbolic graph transformation. The main difference 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 [16, 3, 1, 15].

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 and A. Lopes. A model for dynamic reconfiguration in service-oriented architectures. *Softw Syst Model*, pages 12:349–367, 2013.

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

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

[12] 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.

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

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

[15] 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.

[16] 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 $\mathbf{2SymbGraphs_D}$ is M-adhesive.

Proposition A.1 *$\mathcal{M}$-morphisms in* $\mathbf{2SymbGraphs_D}$ *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, let $h : TG_0 \to TG_1$, $g : TG_1 \to TG_2$ be 2-level $\mathcal{M}$-morphisms, then $h^{top} : TG_0^{top} \to TG_1^{top}$ and $g^{top} : TG_1^{top} \to TG_2^{top}$ are symbolic graph $\mathcal{M}$-morphisms and, for all edges e_0 in TG_0^{top} and e_1 in TG_1^{top}, h^{e_0} and g^{e_1} are also a symbolic graph $\mathcal{M}$-morphisms. But this means that $g^{top} \circ h^{top}$ is a symbolic graph $\mathcal{M}$-morphism and for every edge e_0 in TG_0^{top}, $g^{e_1} \circ h^{e_0}$ is also a symbolic graph $\mathcal{M}$-morphism, where $e_1 = h^{top}(e_0)$. Therefore, $g \circ h$ is a 2-level $\mathcal{M}$-morphism.

Finally, 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. Given 2-level morphisms $h : TG_0 \to TG_1$, $g : TG_1 \to TG_2$, 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 TG_1^{top}, g^{e_1} is a symbolic graph $\mathcal{M}$-morphism, and for every edge e_0 in TG_0^{top}, $g^{h^{top}(e_0)} \circ h^{e_0}$ is also a symbolic graph $\mathcal{M}$-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, he have that for every $e_0 \in TG_0^{top}$, h^{e_0} is a symbolic graph $\mathcal{M}$-morphism. Therefore, h is a 2-level $\mathcal{M}$-morphism. ■

Let us now see that **2SymbGraphs$_{\mathbf{D}}$** has pushouts and pullbacks.

2SymbGraphs$_{\mathbf{D}}$ has pushouts.

Proof

Given a diagram D consisting of a family of 2-level morphisms $\{h_i : TG_{i_1} \to TG_{i_2}\}_{i \in D}$, we define its colimit by induction:

- If all the graphs involved are in $\mathcal{TG}_0$, the colimit in **2SymbGraphs$_{\mathbf{D}}$** essentially coincides with the colimit in **SymbGraphs$_{\mathbf{D}}$** .
- If each graph involved TG_j is in $\mathcal{TG}_{i_j}$, with $i_j \leq k+1$, the colimit of the diagram TG and the corresponding morphisms $g_j : TG_j \to TG$ are defined as follows:
  - TG_{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 TG^{top}, $cts^{TG}(e)$ is the colimit of the diagram including all the graphs $cts^{TG_j}(e')$, where $e = g_j^{top}(e')$, and all the morphisms $h_m^{e'} : cts^{TG_{m_1}}(e') \to cts^{TG_{m_2}}(e'')$, 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 TG_j^{top}, $g_j^{e_j}$ is the canonical morphism defined by the colimit associated to the edge $g_j^{top}(e_j)$ in TG^{top} defined in the item above.

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

As a consequence, we have:

Proposition A.2 **2SymbGraphs$_{\mathbf{D}}$** *has pushouts.*

Proof

The pushout of the diagram:

$$\begin{tikzcd}
TG_0 \arrow[r, "h_1"] \arrow[d, "h_2"'] & TG_1 \arrow[d, "g_1"] \\
TG_2 \arrow[r, "g_2"'] & TG_3
\end{tikzcd}$$

is defined as follows:

- TG_3^{top}, g_1^{top} and g_2^{top} are given by the pushout in $\mathbf{SymbGraphs_D}$:

$$\begin{tikzcd}
TG_0^{top} \arrow[r, "h_1^{top}"] \arrow[d, "h_2^{top}"'] & TG_1^{top} \arrow[d, "g_1^{top}"] \\
TG_2^{top} \arrow[r, "g_2^{top}"'] & TG_3^{top}
\end{tikzcd}$$

- For every edge e_3 in TG_3^{top}, we have that $cts^{TG_3}(e_3)$ is the colimit in $\mathbf{SymbGraphs_D}$ of the diagram including all the graphs:
  - $cts^{TG_0^{top}}(e_0)$, where e_0 is an edge in TG_0 and $e_3 = h_1^{top}(g_1^{top}(e_0))$
  - $cts^{TG_1^{top}}(e_1)$, where e_1 is an edge in TG_1 and $e_3 = g_1^{top}(e_1)$
  - $cts^{TG_2^{top}}(e_2)$, where e_0 is an edge in TG_2 and $e_3 = g_2^{top}(e_2)$

  and all the morphisms $h_1^{e_0}$ and $h_2^{e_0}$, where e_0 is an edge in TG_0 and $e_3 = h_1^{top}(g_1^{top}(e_0))$.

It is routine to prove that this construction is indeed a pushout. ■

Now, we show the existence of pullbacks:

Proposition A.3 $\mathbf{2SymbGraphs_D}$ *has pullbacks.*

Proof
The pullback of the diagram:

$$\begin{tikzcd}
TG_0 \arrow[r, "h_1"] \arrow[d, "h_2"'] & TG_1 \arrow[d, "g_1"] \\
TG_2 \arrow[r, "g_2"'] & TG_3
\end{tikzcd}$$

is defined as follows:

- TG_0^{top}, h_1^{top} and h_2^{top} are given by the pullback in $\mathbf{SymbGraphs_D}$:

$$\begin{tikzcd}
TG_0^{top} \arrow[r, "h_1^{top}"] \arrow[d, "h_2^{top}"'] & TG_1^{top} \arrow[d, "g_1^{top}"] \\
TG_2^{top} \arrow[r, "g_2^{top}"'] & TG_3^{top}
\end{tikzcd}$$

- For every edge e_0 in TG_0^{top}, we have that $cts^{TG_0}(e_0)$ and $h_1^{e_0}$ and $h_2^{e_0}$ are given by the pullback in $\mathbf{SymbGraphs_D}$:

$$\begin{tikzcd}
cts^{TG_0}(e_0) \arrow[r, "h_1^{e_0}"] \arrow[d, "h_2^{e_0}"'] & cts^{TG_1}(e_1) \arrow[d, "g_1^{e_1}"] \\
cts^{TG_2}(e_2) \arrow[r, "c^{e_2}"'] & cts^{TG_3}(e_3)
\end{tikzcd}$$

  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.4 *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.*

$$\begin{tikzcd}
TG_0 \arrow[r, "h_1"] \arrow[d, "h_2"'] & TG_1 \arrow[d, "g_1"] \\
TG_2 \arrow[r, "g_2"'] & TG_3
\end{tikzcd}$$

Proof
For pullbacks, it is enough to notice that the pullback of the top level morphisms and the pullbacks of the down level morphisms preserve $\mathcal{M}$-morphisms. For pushouts, it 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 our 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 and pullbacks preserve $\mathcal{M}$-morphisms and we also know that the composition of the resulting $\mathcal{M}$-morphisms is also an $\mathcal{M}$-morphism. ■

Proposition A.5 *Pushouts along 2-level $\mathcal{M}$-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 to show that the top square is a pushout if and only if the front faces are pullbacks.

$$\begin{tikzcd}
 & & TH_0 \arrow[dll, "h_1"'] \arrow[dd, "f_0"] \arrow[ddr, "h_2"] & \\
TH_1 \arrow[dd, "f_1"'] \arrow[ddr, "g_1"] & & & \\
 & & TG_0 \arrow[dll, "h_1'"'] \arrow[ddr, "h_2'"] & TH_2 \arrow[dll, "g_2"'] \arrow[dd, "f_2"] \\
TG_1 \arrow[ddr, "g_1'"'] & TH_3 \arrow[dd, "f_3"] & & \\
 & & & TG_2 \arrow[dll, "g_2'"] \\
 & TG_3 & &
\end{tikzcd} \tag{1}$$

Let us suppose that the top square is a pushout and 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:

$$\begin{tikzcd}
 & & TH_0^{top} \arrow[dll, "h_1^{top}"'] \arrow[dd, "f_0^{top}"] \arrow[ddr, "h_2^{top}"] & \\
TH_1^{top} \arrow[dd, "f_1^{top}"'] \arrow[ddr, "g_1^{top}"] & & & \\
 & & TG_0^{top} \arrow[dll, "(h_1')^{top}"'] \arrow[ddr, "(h_2')^{top}"'] & TH_2^{top} \arrow[dll, "g_2^{top}"'] \arrow[dd, "f_2^{top}"] \\
TG_1^{top} \arrow[ddr, "(g_1')^{top}"'] & TH_3^{top} \arrow[dd, "f_3^{top}"] & & \\
 & & & TG_2^{top} \arrow[dll, "(g_2')^{top}"] \\
 & TG_3^{top} & &
\end{tikzcd} \tag{2}$$

is a weak van Kampen square in $\mathbf{SymbGraphs_D}$, therefore its front faces are pullbacks in that category. Hence, we have to show that for every edge e_1 in TH_1^{top}, $cts^{TH_1}(e_1)$ 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 TH_0^{top}, such that $e_1 = h_1^{top}(e_0)$ then we know that there is also no edge e_0' in TG_0^{top}, such that $f_1^{top}(e_1) = (h_1')^{top}(e_0)$, since TH_0^{top} is the pullback of the back left square in (2). In addition, we know that there is also no edge e_2 in TH_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 TG_2^{top} such that $f_3^{top}(e_3) = (g_2')^{top}(e_2')$. This means, by the definition of pushouts of 2-level graphs, that $cts^{TH_1}(e_1) = cts^{TH_3}(e_3)$ and $cts^{TG_1}(e_1') = cts^{TG_3}(e_3')$ and the morphisms $g_1^{e_1}$ and $(g_1')^{e_1'}$ are identities.

  Now, let us consider the diagram below:

$$\begin{tikzcd}
 & & \emptyset \arrow[dll, "\emptyset"'] \arrow[dd, "\emptyset"] \arrow[ddr, "\emptyset"] & \\
cts^{TH_1}(e_1) \arrow[dd, "f_1{}^{e_1}"'] \arrow[ddr, "g_1^{e_1}"] & & & \\
 & & \emptyset \arrow[dll, "\emptyset"'] \arrow[ddr, "\emptyset"] & \emptyset \arrow[dll, "\emptyset"'] \arrow[dd, "\emptyset"] \\
cts^{TG_1}(e_1') \arrow[ddr, "(g_1')^{e_1'}"'] & cts^{TH_3}(e_3) \arrow[dd, "f_3^{e_3}"] & & \\
 & & & \emptyset \arrow[dll, "\emptyset"] \\
 & cts^{TG_3}(e_3') & &
\end{tikzcd}$$

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 TH_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, there may be several edges d_1 in TH_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 TH_2^{top} such that $e_2 = h_2^{top}(e_0)$. This means that, in general, $cts^{TH_3}(e_3)$ 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 TG_0^{top} and TG_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, $cts^{TG_3}(e_3')$ 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:

$$\begin{tikzcd}
 & & & H_0 \arrow[dll, "a_1"'] \arrow[dd, "a"] \arrow[ddr, "a_2"] & \\
 & cts^{TH_1}(e_1) \arrow[dl, "g_1"'] \arrow[dd, "f_1{}^{e_1}"] & & & \\
TH' \arrow[drr, "g_2"] \arrow[dd, "f'"] & & & G_0 \arrow[dll, "a_1'"'] \arrow[ddr, "a_2'"] & cts^{TH_2}(e_2) \arrow[dll, "b_2"'] \arrow[dd, "f_2{}^{e_2}"] \\
 & cts^{TG_1}(e_1') \arrow[dl, "g_1'"'] & TH \arrow[dd, "f"] & & \\
TG' \arrow[drr, "g_2'"] & & & & cts^{TG_2}(e_2') \arrow[dll, "b_2'"'] \\
 & & TG & &
\end{tikzcd}$$

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 TH is the colimit of all the morphisms $h_1^{d_0}$ and $f_0^{d_0}$ and TG 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 TH_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:

$$\begin{tikzcd}
 & & H_0 \arrow[dll, "h_1^{e_0}"'] \arrow[dd, "f_0^{e_0}"] \arrow[ddr, "h_2^{e_0}"] & \\
H_1 \arrow[dd, "f_1{}^{e_1}"'] \arrow[ddr, "g_1^{e_1}"] & & & \\
 & & G_0 \arrow[dll, "(h'_1)^{e'_0}"'] \arrow[ddr, "(h'_2)^{e'_0}"'] & H_2 \arrow[dll, "g_2^{e_2}"'] \arrow[dd, "f_2{}^{e_2}"] \\
G_1 \arrow[ddr, "(g'_1)^{e'_1}"'] & H_3 \arrow[dd, "f_3^{e_3}"] & & \\
 & & & G_2 \arrow[dll, "(g'_2)^{e'_2}"] \\
 & G_3 & &
\end{tikzcd}$$

where, $e_2 = h_2^{top}(e_0)$ and, for every $0 \leq i \leq 3$, $e'_i = f_i(e_i)$, $H_i = cts^{TH_i}(e_i)$ and $G_i = cts^{TG_i}(e'_i)$. Therefore, 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:

$$\begin{tikzcd}
 & & & H_0 \arrow[dll, "a_1"'] \arrow[dd, "a"] \arrow[ddr, "a_2"] & \\
 & cts^{TH_1}(e_1) \arrow[dl, "g_1"'] \arrow[dd, "f_1{}^{e_1}"] & & & \\
TH' \arrow[drr, "g_2"] \arrow[dd, "f'"'] & & & G_0 \arrow[dll, "a'_1"'] \arrow[ddr, "a'_2"] & cts^{TH_2}(e_2) \arrow[dll, "b_2"'] \arrow[dd, "f_2{}^{e_2}"] \\
 & cts^{TG_1}(e'_1) \arrow[dl, "g'_1"'] & TH \arrow[dd, "f"] & & \\
TG' \arrow[drr, "g'_2"'] & & & & cts^{TG_2}(e'_2) \arrow[dll, "b'_2"] \\
 & & TG & &
\end{tikzcd}$$

where the top face is a pushout and TH and TG 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:

$$\begin{tikzcd}
 & & cts^{TH_0}(d_0) \arrow[dll, "h_2^{d_0}" description] \arrow[dd, "f_0^{d_0}" description] \arrow[ddr, "h_1^{d_0}" description] & \\
cts^{TH_2}(d_2) \arrow[dd, "f_2{}^{d_2}" description] \arrow[ddr] & & & \\
 & & cts^{TG_0}(d'_0) \arrow[dll, "(h'_2)^{d'_0}" description] \arrow[ddr, "(h'_1)^{d'_0}" description] & cts^{TH_1}(e_1) \arrow[dll] \arrow[dd, "f_1{}^{e_1}" description] \\
cts^{TG_2}(d'_2) \arrow[ddr] & H'_3 \arrow[dd] & & \\
 & & & cts^{TG_1}(e'_1) \arrow[dll] \\
 & G'_3 & &
\end{tikzcd}$$

where, d_0 is the only edge in TH_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 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:

$$\begin{tikzcd}
 & & & H'_0 \arrow[dl] \arrow[dd] \arrow[drr] & & & \\
 & & H_0 \arrow[dl, "a_1"'] \arrow[drr, "a_2"] \arrow[dd, "a"] & & & cts^{TH_0}(d_0) \arrow[dl, "h_2^{d_0}"'] \arrow[dr, "h_2^{d_0}"] \arrow[dd, "f_0^{d_0}"] & \\
 & cts^{TH_1}(e_1) \arrow[dl, "g_1"'] \arrow[dd, "f_1{}^{e_1}"] & & G'_0 \arrow[dl] \arrow[drr] & cts^{TH_2}(e_2) \arrow[dd, "f_2{}^{e_2}"] & & cts^{TH_1}(d_1) \arrow[dddl] \arrow[dd, "f_1{}^{d_1}"] \\
TH' \arrow[drr, "g_2"] \arrow[dd, "f'"] & & G_0 \arrow[dl, "a'_1"'] \arrow[drr, "a'_2"] & & & cts^{TG_0}(d'_0) \arrow[dl, "(h'_2)^{d'_0}"'] \arrow[dr, "(h'_2)^{d'_0}"] & \\
 & cts^{TG_1}(e'_1) \arrow[dl, "g'_1"'] & TH \arrow[dd, "f"] \arrow[ddrr] & & cts^{TG_2}(e'_2) & & cts^{TG_1}(d'_1) \arrow[dddl] \\
TG' \arrow[drr, "g'_2"] & & & & & H'_3 \arrow[dl] \arrow[dd] & \\
 & & TG \arrow[ddrr] & & TH'' \arrow[dd] & & \\
 & & & & & G'_3 \arrow[dl] & \\
 & & & & TG'' & &
\end{tikzcd}$$

where H'_0 and G'_0 are, respectively, the pullbacks of a_2 and $h_2^{d_0}$, and of a'_2 and $(h'_2)^{d'_0}$, TH'' is the pushout of the composed morphisms $H'_0 \to TH$ and $H'_0 \to H'_3$, similarly, TG'' is the pushout of the morphisms $G'_0 \to TG$ 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, TH'' and TG'' 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 TH_3^{top} $cts^{TH_3}(e_3)$ 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:

$$\begin{tikzcd}
 & & cts^{TH_0}(e_0) \arrow[dll, "h_1^{e_0}"'] \arrow[dd, "f_0^{e_0}"] \arrow[ddr, "h_2^{e_0}"] & \\
cts^{TH_1}(e_1) \arrow[dd, "f_1{}^{e_1}"'] \arrow[ddr] & & & \\
 & & cts^{TG_0}(e'_0) \arrow[dll, "(h'_1)^{e'_0}"'] \arrow[ddr, "(h'_2)^{e'_0}"'] & cts^{TH_2}(e_2) \arrow[dll] \arrow[dd, "f_2{}^{e_2}"] \\
cts^{TG_1}(e'_1) \arrow[ddr] & H_3 \arrow[dd] & & \\
 & & & cts^{TG_2}(e'_2) \arrow[dll] \\
 & G_3 & &
\end{tikzcd} \tag{3}$$

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 $cts^{TG_3}(e'_3)$ 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 $cts^{TH_3}(e_3)$ and $cts^{TG_3}(e'_3)$, respectively, then all the vertical faces are pullbacks, this would imply that $cts^{TH_3}(e_3)$ 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 TH_1^{top} or an edge e_2 in TH_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 $cts^{TH_3}(e_3) = cts^{TH_1}(e_1)$, since this is equivalent to show that that the diagram below is a colimit:

$$\begin{tikzcd}
\emptyset \arrow[r, "\emptyset"] \arrow[d, "\emptyset"'] & cts^{TH_1}(e_1) \arrow[d, "g_1{}^{e_1}"] \\
\emptyset \arrow[r, "\emptyset"'] & cts^{TH_3}(e_3)
\end{tikzcd}$$

Now, we can see that there is no edge e'_0 in TG_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 TH_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 TG_2^{top} such that $(g_2')^{top}(e_2') = f_3^{top}(e_3)$. Then, let us now consider the following diagram:

$$\begin{tikzcd}
 & & & \emptyset \arrow[dlll, "\emptyset"'] \arrow[dd, "\emptyset"] \arrow[ddr, "\emptyset"] & \\
cts^{TH_1}(e_1) \arrow[dd, "f_1{}^{e_1}"'] \arrow[ddrr, "g_1^{e_1}"] & & & & \\
 & & & \emptyset \arrow[dlll, "\emptyset"] \arrow[ddr, "\emptyset"] & \emptyset \arrow[dll, "\emptyset"'] \arrow[dd, "\emptyset"] \\
cts^{TG_1}(e_1') \arrow[ddrr, "(g_1')^{e_1'}"'] & & cts^{TH_3}(e_3) \arrow[dd, "f_3^{e_3}"] & & \\
 & & & & \emptyset \arrow[dll, "\emptyset"] \\
 & & cts^{TG_3}(e_3') & &
\end{tikzcd}$$

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:

$$\begin{tikzcd}
 & & cts^{TH_0}(e_0) \arrow[dll, "h_1^{e_0}"'] \arrow[dd, "f_0^{e_0}"] \arrow[ddr, "h_2^{e_0}"] & \\
cts^{TH_1}(e_1) \arrow[dd, "f_1{}^{e_1}"'] \arrow[ddr] & & & \\
 & & cts^{TG_0}(e_0') \arrow[dll, "(h_1')^{e_0'}"'] \arrow[ddr, "(h_2')^{e_0'}"'] & cts^{TH_2}(e_2) \arrow[dll] \arrow[dd, "f_2{}^{e_2}"] \\
cts^{TG_1}(e_1') \arrow[ddr] & H_3 \arrow[dd, "f"] & & \\
 & & & cts^{TG_2}(e_2') \arrow[dll] \\
 & G_3 & &
\end{tikzcd} \tag{4}$$

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:

$$\begin{tikzcd}
H'_3 \arrow[r, "g"] \arrow[d, "f'"] & H3 \arrow[d, "f"] \\
G'_3 \arrow[r, "g'"'] & G_3
\end{tikzcd} \qquad (5)$$

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:

$$\begin{tikzcd}
 & & cts^{TH_0}(e_0) \arrow[dll, "h_1^{e_0}"'] \arrow[dd, "f_0^{e_0}"] \arrow[ddr, "h_2^{e_0}"] & \\
cts^{TH_1}(e_1) \arrow[dd, "f_1^{e_1}"'] \arrow[ddr] & & & \\
 & & cts^{TG_0}(e'_0) \arrow[dll, "(h'_1)^{e'_0}"'] \arrow[ddr, "(h'_2)^{e'_0}"'] & cts^{TH_2}(e_2) \arrow[dll] \arrow[dd, "f_2^{e_2}"] \\
cts^{TG_1}(e'_1) \arrow[ddr] & H'_3 \arrow[dd, "f'"] & & \\
 & & & cts^{TG_2}(e'_2) \arrow[dll] \\
 & G'_3 & &
\end{tikzcd} \qquad (6)$$

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:

$$\begin{tikzcd}
cts^{TG_0}(d'_0) \arrow[r, "(h'_1)^{d'_0}"] \arrow[d, "(h'_2)^{d'_0}"'] \arrow[dr, phantom, "(PO)"] & cts^{TG_1}(d'_1) \arrow[d, "c'_1"] \\
cts^{TG_2}(e'_2) \arrow[r, "c'_2"'] & G''_3
\end{tikzcd}
\qquad
\begin{tikzcd}
H''_3 \arrow[r, "h"] \arrow[d, "f''"'] \arrow[dr, phantom, "(PB)"] & H3 \arrow[d, "f"] \\
G''_3 \arrow[r, "h'"'] & G_3
\end{tikzcd}$$

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:

$$\begin{tikzcd}
 & & cts^{TH_0}(d_0) \arrow[dll, "h_1^{d_0}"'] \arrow[dd, "f_0^{d_0}"] \arrow[ddr, "h_2^{d_0}"] & \\
cts^{TH_1}(d_1) \arrow[dd, "{f_1}^{d_1}"] \arrow[ddr] & & & \\
 & & cts^{TG_0}(d'_0) \arrow[dll, "(h'_1)^{d'_0}"'] \arrow[ddr, "(h'_2)^{d'_0}"] & cts^{TH_2}(e_2) \arrow[dll, "b_3"'] \arrow[dd, "{f_2}^{e_2}"] \\
cts^{TG_1}(d'_1) \arrow[ddr] & H''_3 \arrow[dd, "f''"] & & \\
 & & & cts^{TG_2}(e'_2) \arrow[dll, "b'_3"] \\
 & G''_3 & &
\end{tikzcd} \tag{6}$$

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:

$$\begin{tikzcd}
 & & & cts^{TH_2}(e_2) \arrow[dlll, "b_2"] \arrow[dd, "{f_2}^{e_2}"] \arrow[ddrr, "b_3"] & & \\
H'_3 \arrow[dd, "f'"] \arrow[ddr, "g"] & & & & & \\
 & & & cts^{TG_2}(e'_2) \arrow[dlll, "b'_2"] \arrow[ddrr, "b'_3"] & & H''_3 \arrow[dllll, "h"] \arrow[dd, "f''"] \\
G'_3 \arrow[ddr, "g'"] & H_3 \arrow[dd, "f"] & & & & \\
 & & & & & G''_3 \arrow[dllll, "h'"] \\
 & G_3 & & & &
\end{tikzcd}$$

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, and A.5 we have:

Theorem A.6 **2SymbGraphs$_{\mathbf{D}}$** *is an $\mathcal{M}$-adhesive category.*