Distributed Ambient Graphs with Business Configurations

Nikos Mylonakis

Universitat Politècnica de Catalunya,
C. Jordi Girona Salgado 1-3, 08034 Barcelona, Spain
nicos@cs.upc.edu

Abstract. In this paper we present distributed ambient graphs with business configurations to extend our original model of service oriented computing (*SOC*) with an ambient topology of locations. This ambient topology is based on the graph semantics of an adaptation of the ambient calculus. Thus, in this new service oriented model we can formalize all the external sites that take place in a service oriented application. Each site providing a service to a given service oriented application, will have its own business activity and it will have the possibility to transfer events with the service oriented application. Additionally, providing services can of course also require services to external sites. In this approach, we can also model several service oriented applications developed in different external sites. Thus, the resulting formalism allow us to perform choreographies.

Keywords: mobile calculus, graph transformation, service oriented computing (SOC)

1 Introduction

Service Oriented Computing (*SOC*) is a software paradigm that uses services provided by external sites distributed over the Internet to deliver services to client applications. Service-oriented programs can decide at run time which services to select after a process of discovery and ranking that takes into account how they meet required behavioural requirements and service-level constraints.

In [16] and then completed in [15], we present a graph transformation approach to formalize the service oriented model of the Sensoria Reference Modeling Language (*SRML*) [9]. *SRML* [9] is a service modeling language developed as part of the EU-FET project SENSORIA [20], whose aim was to develop a novel comprehensive approach to the engineering of software systems for service-oriented architectures where foundational theories, techniques and methods are fully integrated in a pragmatic software engineering approach. The language has supported basic research on fundamental concepts of *SOC* including a model for service-oriented interactions [8], an abstract model for service discovery and binding [7], and a model for dynamic reconfiguration [6].

In our graph semantics, business configurations are represented by symbolic graphs [17] whose hyperedges represent components and events. Each connected subgraph is a business activity whose nodes represent wires. Additionally, the semantics has two different graph transformation rules for these two ways of transforming the state: state transformation rules and reconfiguration rules.

A state transformation rule is a rule that can make the transformations in one activity like for example process an event, eliminating it from a node of a component, or transform the values of the attributes of a component using information of the processed events of its nodes. A reconfiguration rule connects one business activity with another. Each time one requires an external service and one is chosen, the business activity which requires the service is composed with the business activity of the service. Therefore, in [15] business activities are built in a single domain, location or ambient and one can compose several external services possibly from different locations in the same activity.

Symbolic graphs are especially adequate for business configurations because they are the most convenient graph formalism with attributes whose values have to be specified. Indeed, as shown in [17], symbolic graphs are more expressive than the standard approach [3]: for example, it allows us to specify arbitrary conditions on the attributes of a graph. On the other hand, with symbolic graphs, we may define different strategies for evaluating attributes when doing graph transformation, allowing for more flexibility [18]. Moreover, the extension with temporal formulas in [15] allow us to model de behavior of components and modules and to define requires and provides specifications in the *with* clause of symbolic graphs.

In this work, we develop the concept of distributed ambient graphs with business configurations to extend the original model of *SOC* with an ambient topology of locations using the basic ideas of the ambient calculus developed in [2] and adapted for our purposes giving a graph transformation semantics in [14]. Thus, in this new service oriented model we can model all the external sites that take place in a service oriented application. Each site providing a service to a given service oriented application, will have its own business activity and it will have the possibility to transfer events with the service oriented application. Additionally, providing services can of course also require services to external sites. In this approach, we can also model several service oriented applications developed in different external sites. Thus, our service oriented model goes beyond the modelling language presented in [9], and it can be seen as a choreographic formalism which we relate with other ones in the last section.

The paper is organized as follows. In Section 2 we present first-order and temporal symbolic graphs. In Section 3 we present the basic ideas of business configurations of [15], and in Section 4 we present distributed ambient graphs with business configurations. In Section 5, we present an example and in Section 6 we relate our work and give some conclusions and future work.

2 First-order and Temporal Symbolic Graphs

In this section, we first present symbolic graphs with first-order formulas, and then we define temporal symbolic graphs. Although the syntax of both are very similar, they have different semantics.

2.1 First-order Symbolic Graphs

Symbolic (hyper)graphs [17] 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 a set of formulas Φ specifies the values that the variables may take. We may consider that a symbolic graph SG denotes the class of all graphs obtained by replacing the variables in the graph by values that satisfy Φ. For example, the symbolic graph with a propositional formula in Figure 1 specifies a class of attributed graphs including distances in the edges that satisfy the triangle inequality.

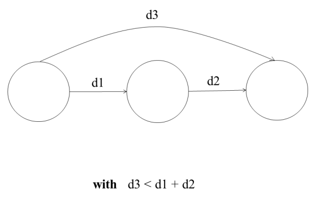

Fig. 1. A symbolic graph

Symbolic graphs are based on a special kind of labeled graphs, called E-graphs, where labels are variables (for more details, see [3,4]). The only difference between the notion of E-graph that we use and that in [3] is that we deal with hypergraphs. This means that, for every graph G, instead of having edges with just source and target graph nodes, we have hyperedges that are connected to a sequence of graph nodes. Additionally, nodes and hyperedges can have attributes.

Definition 1. *A* first-order 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.*

In Figure 2 we give an example of a symbolic rule with one symbolic graph on the left-hand side of the big black right arrow and another on the right-hand side. The meaning of the rule is explained below. The E-graph on the left-hand side has two hyperedges: one denotes an event that has one node and five attributes – the name of the event (*booktrip*) and four event parameters; the other hyperedge denotes a component that has three nodes and five attributes – the name of the component (*BookAgent*) and four variable components. These kind of graphs are part of business activities as defined in the next section.

Symbolic graphs over D together with their morphisms form the category $\mathbf{SymbGraphs_D}$. In [17] 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 [4].

As usual, typed symbolic graphs can be defined as morphisms from a given symbolic graph into a type graph.

Definition 2. *A typed labeled graph (AG,t) over a type graph ATG consist of a labeled graph (AG) together with a graph morphism $(t : AG \to ATG)$.*

As in [4], we consider that graph transformation rules consist of three parts, $L \hookleftarrow K \hookrightarrow R$, the left-hand side L, the right-hand side R, and K that is the common part of L and R, i.e. K is included in both L and R. Nevertheless, for simplicity, in our examples, only the left and right hand sides of the rules will be shown, leaving the common part implicit. Applying a rule $L \hookleftarrow K \hookrightarrow R$ to a given graph G means matching L to some subgraph of G using an injective morphism $m : L \to G$, then computing a graph F that includes all the elements in G that are not in the image of $L \setminus K$ and, finally, computing the result of the transformation H, obtained by adding to F all the elements that are in $L \setminus K$. Formally, this is equivalent to defining H in terms of the diagram below, where (1) and (2) are pushouts.

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

In the case of symbolic graph transformation, we consider that the left-hand side of the rules includes no conditions. As shown in [18] this is not a limitation but, on the contrary, it allows for additional flexibility. This means that symbolic graph transformation rules can be seen as standard graph transformation rules together with a set of conditions, i.e. the conditions of its right-hand side.

Definition 3. *A* symbolic graph transformation rule *is a tuple $\langle L \hookleftarrow K \hookrightarrow R, \Phi \rangle$, where L, K, R are E-graphs over the same set of variables X_R, $L \hookleftarrow K \hookrightarrow R$ is a standard graph transformation rule, and Φ is a set of formulas over X_R, and over the values in the given data algebra D.*

As an example, in Figure 2 we show a rule with generates an event by a Customer component. The rule states that when a Customer requests a flight, it must generate an event with the requested info, in order to start a process of service discovery. The formula below expresses that the origin, destination, and departure, return and requested dates are the ones requested by the customer. The intermediate graph K in general denotes the common subgraph between L and R. In our example this would be the Customer hyperedge; for simplicity, we do not depict it.

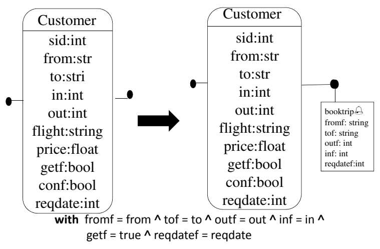

Fig. 2. A symbolic rule

In this context, the result of applying a transformation rule $\langle L \hookleftarrow K \hookrightarrow R, \Phi\rangle$ to a symbolic graph $\langle G, \Phi_G\rangle$ is equivalent to obtaining the symbolic graph $\langle H, \Phi_H\rangle$, where H is the result of applying the rule $L \hookleftarrow K \hookrightarrow R$ to G and $\Phi_H = \Phi_G \cup m'(\Phi)$. We may notice that applying a symbolic transformation rule reduces or narrows down 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, the result graph $\langle H, \Phi_H\rangle$ may be such that Φ_H includes the formula $x = 0$, expressing that 0 is the only possible value of x.

2.2 Temporal symbolic graphs

In this section we define temporal symbolic graphs and their semantics. The basic idea is that first-order symbolic graphs can be used to model the computation states of a service system, but the use of temporal formulas allow us to describe its behavior.

The temporal logic that we propose is very similar to LTL as presented in [12]. The main difference is that we work with expressions using the data algebra D, which include operators such as $<, >, =, \neq, \leq$ and $\geq$; we denote by $EXPR$ the set of correct expressions in the data algebra D. Boolean expressions are LTL formulas and the syntax of LTL formulas is as follows:

- If *exprb* is a correct boolean expression in $EXPR$, *exprb* is an LTL formula.
- If tf_1 and tf_2 are LTL formulas, then $true$, $false$, $\neg tf_1$, $(tf_1 \ \wedge \ tf_2)$, $\bigcirc tf_1$ and $(tf_1 \ \mathcal{U} \ tf_2)$ are also LTL formulas.

Note that in our formulation of LTL we have not included first-order formulas but propositional formulas plus the temporal operators of LTL. This is because we do not need the expressive power of first-order formulas to model service-oriented programs, and because our tool for temporal symbolic graph transformation is much more efficient if we work with a simple propositional logic.

LTL formulas are generally interpreted over a state transition system STS consisting a set of states S and a transition relation $\rightarrow$. An STS can be represented as a computational graph, where each path, formed by a sequence of states, corresponds to a possible run of the system. The intuitive semantics of the temporal formulas over such a path is as follows: the symbols $\neg$ and $\wedge$ have their usual meaning; the formula $\bigcirc tf_1$ intuitively means that tf_1 holds in the immediate successor of the current program state; $\mathcal{U}$ is the until operator – the formula $(tf_1 \ \mathcal{U} \ tf_2)$ intuitively means that there exists a prefix of the path such that tf_1 holds for every state of the prefix and tf_2 holds in the next state of the prefix. The abbreviation $\Diamond f = (true \ \mathcal{U} \ f)$ intuitively means that f will eventually hold, and $\Box f = \neg \Diamond \neg f$ means that f will always hold.

Now we present temporal symbolic graphs and their semantics.

Definition 4. *A* temporal symbolic graph SG *over the data algebra* D *is a pair* $SG = \langle G, \Phi_G\rangle$, *where* G *is an E-graph over a set of variables* X *and* Φ_G *is a set of LTL formulas over the operations and predicates in* D *including variables in* X *and elements in* D.

The semantics of temporal symbolic graphs is not a class of attributed graphs but a class of state transformation systems (STS) whose states are attributed graphs. For example, if we have a temporal symbolic graph with just one node and an attribute b with the temporal formula $\Diamond(b = 5)$, the semantics of this temporal symbolic graph is the class of all state transition systems $STSB$ where the attributed graphs of their states have arbitrary values for that attribute b, but with the particularity that at least in one state of all the possible paths of the $STSB$, we have an attributed graph with value 5 for b.

In the same way as typed symbolic graphs, we can define typed temporal symbolic graphs.

Syntactically, transformation rules for temporal symbolic graphs are similar to first-order symbolic ones, in the sense that they are also tuples $\langle L \hookleftarrow K \hookrightarrow R, \Phi\rangle$, but now Φ is a set of temporal formulas. Moreover, instead of using arbitrary variables in the formulas in Φ to denote the values of the attributes in the graphs in the rule, in this setting, if a variable x denotes the value of an attribute in R and if that attribute is also present in L, then its value in L will be denoted by the reserved name x_p. As before, the application of a graph transformation rule to a given temporal symbolic graph SG can be expressed in terms of a transformation of E-graphs.

Definition 5. *Given a temporal symbolic graph $SG = \langle G, \Phi_G\rangle$ and transformation rule $p = \langle L \hookleftarrow K \hookrightarrow R, \Phi\rangle$ over a given data algebra D and given a morphism $m : L \to G$, we define the application of p to SG by means of the matching m as the transformation $SG \Longrightarrow_{p,m} SH$, where $SH = \langle H, \Phi_H\rangle$ is defined as follows:*

1. *H is defined by the double pushout diagram of E-graphs depicted below:*

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

2. *$\Phi_H = T(\Phi_G) \cup m'(\Phi)$ where T is a transformation function on temporal formulas inductively defined as follows:*
   - $T(true) \Longrightarrow true$
   - $T(exprb) \Longrightarrow exprb'$
   - $T(\neg f) \Longrightarrow \neg T(f)$
   - $T(f_1 \wedge f_2) \Longrightarrow T(f_1) \wedge T(f_2)$
   - $T(\bigcirc f) \Longrightarrow f$
   - $T(f_1 \mathcal{U} f_2) \Longrightarrow f_2 \vee (f_1 \wedge f_1 \mathcal{U} f_2)$

   where $exprb'$ substitutes every variable x in $exprb$ by x_p.

We briefly justify the transformation of the most relevant cases:

- $T(exprb)$ After applying a rule the value or the possible values of the attribute x can be updated. In rules, we use the notation x to denote the current value of the attribute after applying a given rule and x_p to denote the previous value before applying a given rule. Thus, after applying a rule, the previous value of an attribute x still satisfies the proposition, if we use the previous values for all the attributes which appear in *expr*.
- $T(\bigcirc f)$ If we have in the with-clause of a symbolic graph this formula, after applying a rule f must hold. If it does not hold the transformation is not valid.
- $T(f_1 \mathcal{U} f_2)$ If we have in the with-clause of a symbolic graph this formula, after applying a rule one of the following must happen:
  - f_2 holds.
  - f_1 holds and therefore we still have to check the original temporal formula.
  - f_1 and f_2 do not hold and therefore the transformation is not valid.

In [15] one can find details of the semantics of temporal symbolic graphs.

3 Business Configurations and their Transformation Rules

Business configurations are represented by symbolic graphs whose hyperedges represent components and events. Each connected subgraph is a business activity whose nodes represent wires. Additionally, we present state transformation rules, which transform the state of a business activity.

We present this section in two subsections. First we present business configurations, and then its associated transformation rule.

3.1 Business configurations

First we introduce the basic concept of business activity. A business activity is a symbolic graph with components and events. Components have a positive number of nodes, an attribute with the name of the component and a set of attributes of the component. Events are connected to a component node and they have an attribute with the name of the event, another with the type of the event and a set of attributes of the event. We consider two types of component nodes: internal and interface nodes. Both types of nodes can be part of different components and events. The main difference between these two types of nodes is that interface nodes are the ones with which service discovery is performed. Next, we present the concept of business configuration. A business configuration is a symbolic graph that contains a set of business activities.

As mentioned in the definition of business activities, interface nodes are not connected to another node. When these nodes have an event, they triggered a process of selection of an external service in an external business repository. For example, if a customer has launched an activity module that requests a booking agent to book just a flight, the symbolic graph that represents the initial business configuration with an instance of this activity module consists of a customer component with a set of attributes for the flight. A graphical representation is in Figure 3. Since business activities are in general directed acyclic graphs, we have always a distinguished initial node with an attribute with two possible values: *busy* to indicate that there exists at least a business activity of *empty* to indicate there is no business activity. This attribute is necessary to define a transformation rule for distributed ambient graphs of business configurations which will be defined in next section.

Figure 4 show a different stage of the initial business configuration in Figure 3. Figure 4 has a customer subsystem with a set of attributes (from, to, in, out, ...). The component has an interface node with an event. After triggering a process of selection of an external service, the business configuration will send its event to the site where the reconfiguration rule package is, and a new business activity will be created there. We will explain details in next section after introducing distributed ambient graphs of business configurations.

We now present transformation rules for business configurations: state transformation rules.

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

– process an event, eliminating it from a node of a component;
– transform the values of the attributes of a component using information of the processed events of its nodes;
– publish an event in the node of a component.

An example of a state transformation rule is in Figure 5 which extends the one presented in the section of symbolic graphs with a temporal formula: it publishes an event in the interface node of the component of the customer.

Other rules can be used for processing the information of the reply-event of the booking agent or to start the payment. When the rule in Figure 5 is applied to the business configuration, the initiating event is added to the business configuration. The resulting new business configuration is in Figure 4. Note that the rule also has a temporal subformula in the with-clause, which requests always a confirmation from the chosen Booking Agent.

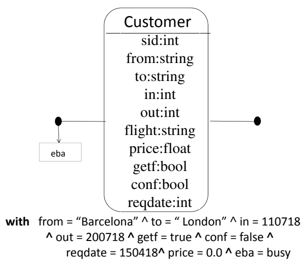
Customer
sid:int
from:string
to:string
in:int
out:int
flight:string
price:float
getf:bool
conf:bool
reqdate:int
eba
with from = "Barcelona" ^ to = " London" ^ in = 110718
^ out = 200718 ^ getf = true ^ conf = false ^
reqdate = 150418^ price = 0.0 ^ eba = busy

Fig. 3. Business configuration with just a customer activity.

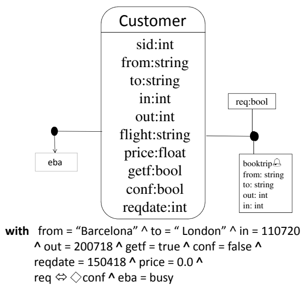

Fig. 4. New business configuration with a trigger event

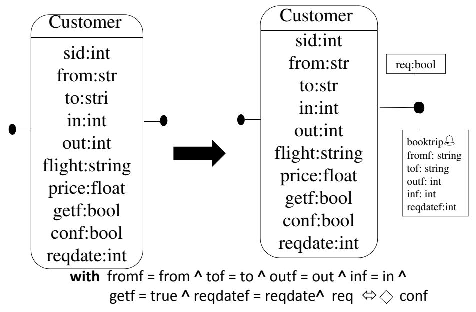

Fig. 5. Rule associated with the customer activity

4 Distributed Ambient Graphs with Business Configurations (*DAGBC*)

In this section we present distributed ambient graphs with business configurations and then their transformation system.

4.1 Description of *DAGBCs* and their types

The basic idea of these graphs is that they represent ambient topologies which will be defined as typed attributed graphs where the attributes will be *Names* to represent ambient names and business repositories *BREP* with an union $\cup$ operation. Additionally every ambient node can have a temporal symbolic subgraph which represent business configurations. Now we describe the type graph. In addition to the nodes and edges to represent attributes, we have five types of graph nodes: nodes to denote ambients with a name attribute, nodes to denote interfaces between ambient nodes, nodes to denote ambient visibility with name attributes, nodes of business configurations and one unique node per ambient to represent business repositories. Concerning the edges we also have four types: edges to define the hierarchy of ambients, edges to associate visibility nodes to ambient nodes, edges to associate business repository nodes to ambient nodes, and edges and hyperedges of business confiigurations.

Intuitively, our representation includes a node (and the corresponding attribute) for each ambient in the expression. Moreover, if an ambient a_1 is inside the ambient a_2 then we have an edge from the node associated to a_1 to an interface node and another edge from that interface node to the node associated to a_2 (we need these interface nodes for technical reasons). That is, the graph associated to an expression can be considered to embed the topology of the ambients involved in the expression. In addition, we have a visibility node associated to a given ambient and this visibility node can have a set of visible ambients as attributes, and a business repository node to attach a business repository to each ambient. Finally, we can have a business configuration associated to each ambient represented as a temporal symbolic subgraph.

Thus, ambient nodes will be represented in the graphs as $\mathbf{an_i}$ where i is an index, interfaces between ambient nodes as $\mathbf{in_i}$, visibility nodes as $\mathbf{vn_i}$, nodes of business configurations as $\mathbf{bc_i}$ and nodes for business repositories as $\mathbf{br_i}$. All type of edges will be represented in the same way as directed arrows.

A distributed ambient graph of business configurations has a distinguished data set $Names$ where a name denotes the name of an ambient. We have additionally a distinguished name *pub* to denote that the business repositories of an ambient is public to all ambients. Business repository nodes will have as attributes business repositories, which contain the set of services which an ambient provides. We will use the usual union operation on sets $\cup$ for union of business repositories. The data set will be referred as $BREP$. Each service is defined as a reconfiguration rule package which always include a distinguished reconfiguration rule which will be explained in detail in next subsection, appart from other transformation rules.

A distributed ambient graph of business configurations $DAGBC$ satisfies the following properties:

– for any pair nodes an_i and in_i there exists at most one edge between them.
– there are no cycles between nodes an_i and in_i.
– we have one visibility node $\mathbf{vn_i}$ associated to each ambient node with a set of ambient names as attributes. In particular, these names must be normal names or the distinguished name *pub*.
– Every ambient node has one and only one ambient name and every ambient name can be targeted by different edges but it can appear just once in the graph.
– We have one business repository node $\mathbf{br_i}$ associated to each ambient node with an attribute of type $BREP$.
– Business configurations are temporal symbolic subgraphs, each one with its own *with* clause. Since business configurations are directed acyclic graphs, we will have an initial business configuration node with a label with two distinguished values: *empty* to indicate that there is no business activity going on, and *busy* to indicate there is at least one active business activity.

Consider the following two examples of $DAGBC$ in Figure 6 and 7.

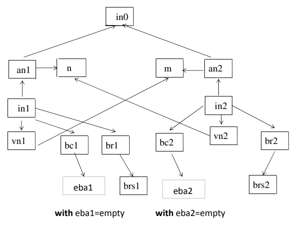

Fig. 6. A first example of $DAGBC$.

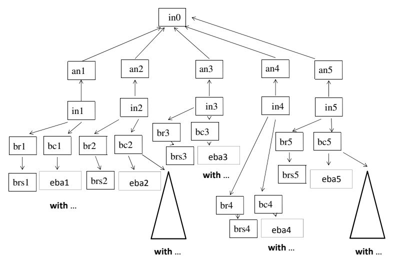

Fig. 7. A second example of $DAGBC$.

In the first example we have two ambients with names n and m, where both ambients n and m can access each other. Additionally, both of them have business repositories $brs1$ and $brs2$, and none of them have started a business activity. In the second example we have five ambients and, because of reasons of space, we have omitted the ambient names, considering the name of the ambient node, the name of the ambient. Additionally, we have also omitted the visibility nodes, and we consider that all ambients are public (*pub*). In this example all ambients have also business repositories, and two of them ($an2$ and $an5$) have started a business activity. Therefore, each of them has temporal symbolic subgraphs, and each of them has its local *with* clause. The other three ambients have no business activity and therefore $eba1 = eba3 = eba4 = empty$.

4.2 A transformation system for *DAGBCs*

The transformation system consists of two different kind of rules:

– State transformation rules, reconfiguration rules and transference rule to transform normally two business configurations. State transformation rules were explained in the previous section and we explain reconfiguration rules in this section, which differ from the reconfiguration rules presented at [15]. Transference rules are new and we will also explain them in this section.
– five rules to transform the ambient topology: a rule to move one ambient inside another ambient, a rule to move out an ambient from another ambient, a rule to open an ambient, and two rules to add and delete visible ambient names associated to an ambient. Because of simplicity, these rules are presented without the intermediate graph K of the double pushout approach.

Reconfiguration and transference rules

An example of a reconfiguration rule is in Figure 8: it has a business activity including a customer component and a business activity including a booking agent component in two different ambients. Note that the rule requires that the customer have at least seven attributes. Two of the attributes are $getf$, which is true because the customer requires information about a flight, and $conf$, which is false because the customer has not received confirmation yet from the booking agent. Two other three attributes are needed to define a temporal formula that provides a discount in the price of the flight. The sid attribute is needed to receive the service identifier from the Boo king Agent. This service identifier will be used in the transference rule to transfer in an event the chosen flight from the Booking Agent.

The Booking Agent has also two boolean attributes: $getfb$, which is true when the agent is treating a booking request, and $confb$, which is true when the agent has sent a confirmation of the request with or without information on the reservation. This reconfiguration rule has also temporal formulas in the with-clause. They express the provides specification of the Booking Agent. The first two conjunctions express that the agent will always receive the request after it has been sent, and that, if the agent receives a request, it will always send a confirmation. The last conjunction of the temporal formula expresses that if the request arrives 90 days before the flight departure, the customer will receive a discount of 10%. Thus, the reconfiguration rule adds a business activity with a booking agent component in the ambient which provides the service, and it also sends the requesting event from the ambient of the Customer to the ambient of the Booking Agent.

A reconfiguration rule package contains one distinguished reconfiguration rule and a set of state transformation rules and transference rules.

The event in Figure 4 triggers a process of selection of an external service including a reconfiguration rule together with a set of state transformation rules and transference rules. The selected reconfiguration rule is the one in Figure 8. After applying the reconfiguration rule, a business activity including a booking agent component is created in the ambient which provides the Booking Agent service, and the requesting event of the customer is also sent to the business activity of the Booking Agent. Additionally, a service identifier is sent to the Customer in an event. The two resulting business activities are in Figures 9 and 10 . Before applying the rule, we have to prove that the provides specification of the Booking Agent denoted by the boolean variable *prov* implies the requires specification of the customer denoted by the boolean variable *req*.

These two business configurations could be ubicated in the example of *DAGBC* given in Figure 7. Thus the Customer business configuration could be in ambient $an2$ and the business

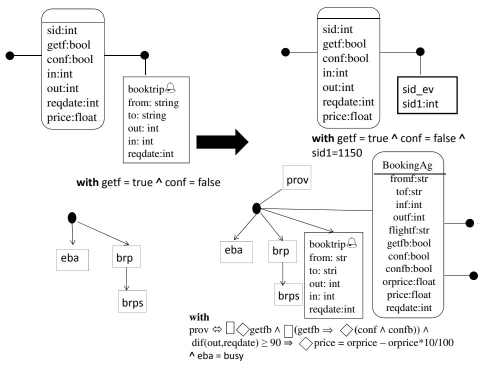

Fig. 8. A reconfiguration rule

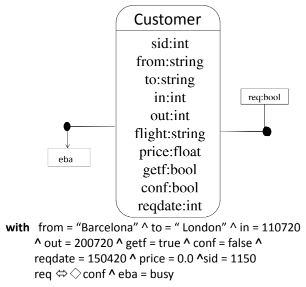
Customer
sid:int
from:string
to:string
in:int
out:int
flight:string
price:float
getf:bool
conf:bool
reqdate:int
req:bool
eba
with from = "Barcelona" ^ to = " London" ^ in = 110720
^ out = 200720 ^ getf = true ^ conf = false ^
reqdate = 150420 ^ price = 0.0 ^sid = 1150
req ⇔ ◇conf ^ eba = busy

Fig. 9. Updated business configuration of the Customer.

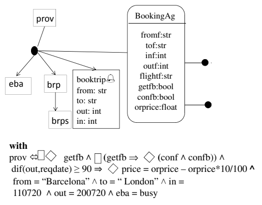
prov
BookingAg
fromf:str
tof:str
inf:int
outf:int
flightf:str
getfb:bool
confb:bool
orprice:float
eba
brp
brps
booktrip
from: str
to: str
out: int
in: int
with
prov ⇔□◇ getfb ∧ □(getfb ⇒ ◇ (conf ∧ confb)) ∧
dif(out,reqdate) ≥ 90 ⇒ ◇ price = orprice – orprice*10/100 ^
from = "Barcelona" ∧ to = " London" ∧ in =
110720 ∧ out = 200720 ∧ eba = busy

Fig. 10. Updated business configuration of the Booking Agent.

configuration of the Booking Agent could be in ambient $an5$. Note that in Figure 10 we have also the equations which define the attributes of the event which has been transferred.

In general, in reconfiguration rules normally two business configurations of different sites take part: the one which requests a service (bcr) and the one which provides a service (bcp). bcr has normally an event which is sent to bcp and therefore it is discarded from bcr. Additionally in bcp it is created a business activity with a provides specification $prov$ which has to satisfy the requirements specification req of bcr. These requirements req are not in the reconfiguration rule but in the $DAGBC$ which we will refer to DBC to which we are going to apply the rule. Additionally we can have in general a with clause associated to bcr with set of formulas Φ_r. So in the left $DAGBC$ of the reconfiguration rule we have bcr with an event and with its with clause, and a business configuration node from which the reconfiguration rule has been taken from a business repository $brps$. In the right $DAGBC$ of the reconfiguration rule we have bcr without the event, and pending from the initial node of the business configuration bcp the event from bcr, and a new business activity with a with clause with a set of formulas Φ_p including the provides specification $prov$. Reconfiguration rules will be applied to a $DAGBC$ which we have already referred as DBC. This DBC could be very big because it can have many ambients with their business configurations but only two business configurations of two ambients are transformed. The one which requires a service will be referred as BCR and it should include an event and a with clause referred as Φ_R. Φ_R should include the requirements including the initialization of the values of the attributes of the event. In order to apply successfully the reconfiguration rule to DBC one must proof that $\Phi_R \implies \Phi_r$ and $prov \implies req$. The business configuration which provides the service has the reconfiguration rule in its business repository set and it can be in general busy running several business activities different to the new one. Let us call these possibly existing business activities BCP with set of formulas Φ_P. To apply the reconfiguration rule to DBC we proceed in two steps:

– We calculate first the double pushout of the two business configurations which are going to be transformed without their with clauses.
– We add then the with clauses of both business configurations. The one which requires the service will include the formulas $T(\Phi_R) \cup \Phi_r$ where T is the transformation function defined to define rule application for temporal symbolic rules. The one which provides the service will include the formulas $T(\Phi_P) \cup \Phi_p \cup T(\Phi_{SR})$ where Φ_{SR} is the set of equations (possibly with auxiliary variables) which define the attributes of the event or events which are sent to the business configuration which provides the service.

As we have explained,a business repository contains all the possible services as reconfiguration rule packages that are available in a given ambient. As we have also mentioned, a reconfiguration rule package have state transformation rules and transference rules.

In transference rules, like in reconfiguration rule, two business configurations take part. They work like reconfiguration rules, and they normally just send an event from one business configuration to another. In our example, a transference rule is needed to send the information of the chosen flight from the Booking Agent to the Customer. In the transference rule, it has to be the service identifier of the Customer to identify the Customer which requested the service.

Rules to transform the ambient topology

To transform the ambient topology we do not need symbolic rules which transform the *with* clauses but parameterised rules over the dataset $Names$. For technical reasons, additionally, the morphism $r : K \to R$ going from the context to the right-hand side of a rule does not need to be a monomorphism.

Definition 6. *A production with parameterised names p consists of a set of names or labels SL, a DAGBC graph L with all the labels in SL, two more DAGBC graphs K and R together with a monomorphism $l : K \to L$ and an arbitrary morphism $r : K \to R$. The production p is represented as $p\ ll_1 \ldots ll_n : L \leftarrow K \to R$ where ll_i are the labels in SL.*

To perform a direct transformation $G \Rightarrow H$ via a left-linear production $p\ ll_1 \ldots ll_n :: L \leftarrow K \to R$ with a set of instantiation labels $il_1 \ldots il_n$ first we have to obtain the production with no labels $p' : L' \leftarrow K' \to R'$ by substituting every ll_i by its associated il_i. We also have to define the obvious induced morphisms l' and r' by l and r and the substitution. Then with a match m, the direct transformation is defined by the usual double pushout diagram.

Now we explain the five transformation rules to transform the ambient topology. As in previous rules, we will omit the intermediate graph K in the rules. The rule to move in one ambient n inside another ambient m is referred as **in** $n\ m$ and it is defined in Figure 11.

The rule to move out ambient n from the ambient which embeds n is referred as **out** n and it is defined in Figure 12. The rule **in** is the inverse of rule **out** and viceversa.

The rule to open an ambient n is referred as **open** n and it is defined in Figure 13. This rule is not easy to interpret and we give an explanation. The interface node $in2$ is identified with $in1$ and therefore it is added to $in1$ the ambient topology which was pending in $in2$. To apply this rule it is required that the business configuration of ambient n must be empty. On the other hand $vn2$ is identified also with $vn1$, and therefore all its visibility labels of $vn2$ are moved to $vn1$. Finally, the business repository of ambient n enriches the business repository of its superambient.

Finally, the rule to add the visibility name m to the visibility set of ambient n is referred as **addv** $m\ n$ and it is defined in Figure 14, the rule to remove the visibility name m from the visibility set of ambient n is referred as **rmv** $m\ n$ and it is defined in Figure 15.

As an example of parameterised rule application, we consider the first of five parameterised rules with which we transform the ambient topology. The first rule *in $n\ m$* has two parameters which are ambient names. If we apply this rule with instantiation names $an1$ and $an2$, then the

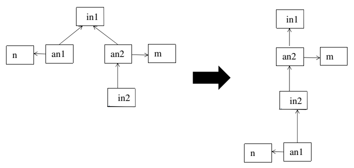

Fig. 11. Rule **in** $n\ m$.

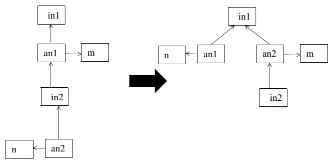

Fig. 12. Rule **out** n.

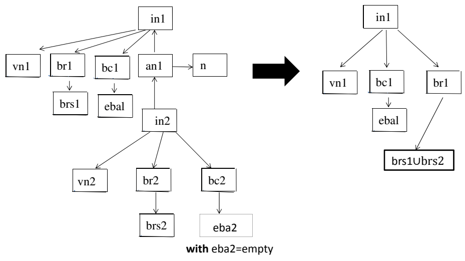

Fig. 13. The transformation rule **open** n

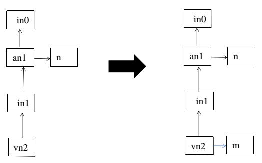

Fig. 14. Rule **addv** $m\ n$.

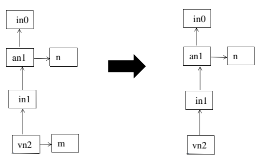

Fig. 15. Rule **rmv** $m\ n$.

target graph G must have these ambient names, and they have to have the same superambient. In order to compute the double pushout we first have to transform the parameterised rule *in n m* into a non-parameterised rule replacing every appearance of n and m in L, K and R by $an1$ and $an2$ respectively, obtaining L', K' and R' and its associated new morphisms l' and r'. Now with this non-parameterised rule we can compute the double pushout with no problems.

In order to define transformation systems for $DAGBC$ we need graph transformation units. Graph transformation units encapsulate a set of rules with a control unit. Graph transformation units have a transactional semantics in the sense that in order to produce a transformation all the rules of the transformation unit taking part in the transformation must be applied successfully. If some rule cannot be applied successfully no transformation is performed. The control unit specifies the order with which the rules must be applied. In our case the language of the control unit consists of a compositional sequential operator of the form $np1; \ldots ; npn$ where each name production has its set of instantiation names if necessary. It has also a parallel operator of sequences of rules rsi of the form $rs1 \mid ... \mid rsn$. For this parallel operator, if all possible sequentializations of the parallel operator yield the same result the application is correct. If not, the application returns the initial graph. See [14] for some examples using transformation units.

The transformation system

Now we present the concept of transformation systems for $DAGBCs$:

Definition 7. *A transformation system for distrbuted ambient graph of business configurations consists of:*

- *A distributed ambient graph of business configurations*
- *the five parameterised rules with names* **in**, **out**, **open**, **rmv**, **addv**
- *for each ambient a set of transformation rules associated to the business configuration of the ambient.*
- *a transformation unit.*

Finally we present the four different ways through which we can transform a transformation system for business configurations. Transformation steps can be one of the following:

- An application of a state transformation rule to a 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 of a busy business configuration of an ambient node *anr* and at least one event *ev*, the application of the distinguished rule of the selected reconfiguration package of possibly another ambient node *anp*. In this case we normally remove the event *ev* from the business configuration of *anr* and add it to *anp*. Additionally we add a business activity to *anp*.
  The rest of the rules of the reconfiguration rule package are added to the current set of state transformation rules of the business configuration.
- An application of a transference rule, which normally just transfer an event node from one business configuration to another.
- an application of one of the five parameterised rules with names **in**, **out**, **open**, **rmv**, **addv**

5 An Example

The initial ambient hierarchy of the running example is the one of Figure 7 with five ambients $an1, ..., an5$. We consider two independent customers requesting a loan of 500 euros by one customer and a flight to London by the other customer. The loan request is performed in ambient $an1$ and the flight request is performed in ambient $an2$. Initially the rule in Figure 5 would be applied to the flight customer in order to generate an event to search a Booking Agent, and in parallel a similar rule would be applied to the loan customer. Therefore, both customer sites start a process of service discovery, ranking and selection to find the requested services. Consider that the chosen Booking Agent service is in ambient $an4$ and the chosen loan service is in ambient $an5$. The ambient $an4$ which provides the flight service includes the reconfiguration rule of Figure 8 which adds a business activity in ambient $an4$ to search for a flight and transfers the booking request from ambient $an2$ to ambient $an4$. Other rules which should be applied in ambient $an4$ are:

– a rule to request a flight service. (The chosen flight service could be in ambient $an3$).
– rules to choose a flight to send to the customer and put the information in an event.
– a rule to tranfer the event with the flight info to the customer.

The business activity of ambient $an5$ which provides the loan service is simpler and it only has a reconfiguration rule similar to the one of Figure 8 an a transference rule to send in an event the requested loan info to the customer. This is a simple example of two customers requesting two independent services in parallel, but we do think that our model scales up well to an arbitrary number of services with the possibility to perform service composition.

We could have added rules to transform the ambient topology. See [14] for two examples on how to apply these rules.

6 Related work and conclusions

We can find several graph formalisms that have been used to model distributed computing.

In [19] she presents distributed graph transformation. She has two abstraction levels, the network and the local level, with the concept of synchronization by interface graphs into a new approach. It offers a clear and elegant description of dynamic networks, distributed actions as well as communication and synchronization based on graph transformation. She also uses the double-pushout approach.

Bigraphs [13] is focused in hierarchical graph models. They have two dimensions: place graphs and link graphs. Place graphs deal deal with structured design of processes and induce a tree-like hierarchy of nodes. Link graphs deal with interaction capabilities which can connect any tree nodes. Link graphs could be used to model service connections.

Gs-graphs [1] were not originally designed with these purposes, but in this paper they show that they are as convenient as bigraphs and they seem to offer some advantages over bigraphs.

None of the approaches integrates a formalism for service level agreements or a temporal logic for requires and provides specifications.

In [5] they present in a tutorial format a family of formalisms called Synchronised Hyperedge Replacement (SHR) and they show that is an adequate formalize to model service oriented computing. They present $SHReq$ which incorporates also c-semirings to deal with QoS requirements. On the other hand, they do not mention any use of a temporal logic.

In our work, we use temporal symbolic graphs to give a model of service oriented computing which goes beyond the language presented in [9]. Thus, our model represented by distributed ambient graphs with business configurations, allow us to represent different distributed sites, and model a choreography of service applications. Other formalisms to develop choreographies for service oriented computing are the following:

– The Web Services Choreography Description Language (WS-CDL [11]) is an XML-based language that describes peer-to-peer collaborations of participants by defining, from a global viewpoint, their common and complementary observable behaviour. The modelling of service interactions extensively rely on programming language constructs such as sequence, loops, and variable assignment. We strongly prefer our declarative approach using graph transformation.
– In [21] they propose a set of requirements that a language for choregraphies has to satisfy and propose a concrete one called Let's Dance. The language supports the description of both local and global views of service interactions (i.e. behavioural interfaces and choreographies respectively). In [10] they give a formal semantics using the π-calculus. We think that we can

express all the patterns of interaction that they propose using symbolic graph transformation. On the other hand, we do not know how to express our ambient hierarchy of distributed sites with visibility restrictions, and our temporal logic for requires and provides specifications in their formalism.

In this work we have not tackled the problems of working with persistent data in service oriented computing. We plan to do some future work on the area. Another area of interest is recommender systems for services.

Acknowledgment

This work has been partially supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under grant COMMAS (ref. TIN2013-46181-C2-1-R).

References

1. Roberto Bruni, Ugo Montanari, Gordon D. Plotkin, and Daniele Terreni. On hierarchical graphs: Reconciling bigraphs, gs-monoidal theories and gs-graphs. *Fundam. Inform.*, 134(3-4):287–317, 2014.
2. L. Cardelli and A. D. Gordon. Mobile ambients, 1998. In Maurice Nivat, editor, Proc. FOSSACS'98, International Conference on Foundations of Software Science and Computation Structures, volume 1378 of Lecture Notes in Computer Science, pages 140–155. Springer-Verlag.
3. 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.
4. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation.* EATCS Monographs of Theoretical Computer Science. Springer, 2006.
5. Gian Luigi Ferrari, Dan Hirsch, Ivan Lanese, Ugo Montanari, and Emilio Tuosto. Synchronised hyperedge replacement as a model for service oriented computing. In *Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures*, pages 22–43, 2005.
6. J. L. Fiadeiro and A. Lopes. A model for dynamic reconfiguration in service-oriented architectures. *Softw Syst Model*, pages 12:349–367, 2013.
7. J. L. Fiadeiro, A. Lopes, and L. Bocchi. An abstract model of service discovery and binding. *Formal Asp. Comput.*, 23(4):433–463, 2011.
8. José Luiz Fiadeiro, Antónia Lopes, and João Abreu. A formal model for service-oriented interactions. *Sci. Comput. Program.*, 77(5):577–608, 2012.
9. José Luiz Fiadeiro, Antónia Lopes, Laura Bocchi, and João Abreu. The sensoria reference modelling language. In Wirsing and Hölzl [20], pages 61–114.
10. M. Dumas G. Decker, J.M. Zaha. Execution semantics for service choreographies, 2006. In Proceedings of the 3rd International Workshop on Web Services and Formal Methods (WS-FM).
11. N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. Web services choreography description language version 1.0, w3c candidate recommendation, 2005. `http://www.w3.org/TR/ws-cdl-10`.
12. Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems - specification.* Springer, 1992.
13. Robin Milner. Pure bigraphs: Structure and dynamics. *Inf. Comput.*, 204(1):60–122, 2006.
14. Nikos Mylonakis. A graph semantics for a variant of the ambient calculus more adequate for modelling service oriented computing, 2017. Graph Computational Models (GCM).
15. Nikos Mylonakis, Fernando Orejas, and José Fiadeiro. Modeling service-oriented computing with temporal symbolic graph transformation systems. Research Report 2015 `upcommons.upc.edu` and submitted to International Journal of Services Computing (IJSC).
16. Nikos Mylonakis, Fernando Orejas, and José Fiadeiro. A semantics of business configurations using symbolic graphs, 2015. IEEE International Conference on Services Computing (SCC).
17. 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.
18. Fernando Orejas and Leen Lambers. Lazy graph transformation. *Fundam. Inform.*, 118(1-2):65–96, 2012.

19. Gabriele Taentzer. Distributed graphs and graph transformation. *Applied Categorical Structures*, 7(4):431–462, 1999.
20. Martin Wirsing and Matthias M. Hölzl, editors. *Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing*, volume 6582 of *Lecture Notes in Computer Science*. Springer, 2011.
21. Johannes Maria Zaha, Alistair Barros, Marlon Dumas, and Arthur ter Hofstede. *Let's Dance: A Language for Service Behavior Modeling*, pages 145–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.