Modelling Service-Oriented Computing with Temporal Symbolic Graph Transformation Systems

Nikos Mylonakis and Fernando Orejas
Department of Computer Science
Universitat Politècnica de Catalunya
Barcelona, Spain
Email: nicos@cs.upc.edu, orejas@cs.upc.edu

José Luiz Fiadeiro
Department of Computer Science
Royal Holloway University of London
Egham, UK
Email: jose.fiadeiro@rhul.ac.uk

***Abstract*—In this paper, we present a novel semantics for an essential aspect of service-oriented computing: the mechanism through which systems evolve through a symbiosis of state transformations and run-time service discovery and binding. The semantics is based on a new notion of temporal symbolic graph-transformation systems: in temporal symbolic graphs, interfaces can be specified using temporal logic, and service-level agreements can be specified in that logic's propositional fragment. An important advantage of our framework is that it can be supported by tools that implement temporal symbolic graph transformations, which would also provide a means of animating service-oriented systems evolution. We illustrate our semantics with a simple trip-booking service.**

***Keywords*-Service Oriented Computing (SOC); graph transformation systems;**

I. 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.

A possible state model for SOC can be considered at two levels of abstraction. At the lowest level, *state configurations* are graphs of interconnected components; at the highest level, *business configurations* are graphs of interconnected activities, where an activity is a graph of components. This definition at two levels of abstraction accounts for both state changes that result from computations performed by components and configuration changes that result from dynamic service discovery and binding. In this paper, we present a graph transformation approach to formalise both kinds of changes in a uniform way. This approach is used to give a semantics of the Sensoria Reference Modeling Language ($SRML$) [1], but it could also be used in connection with other approaches for modeling or specifying service systems.

$SRML$ [1] is a service modeling language developed as part of the EU-FET project SENSORIA [2], 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, which is based on those two levels of abstraction, has supported basic research on fundamental concepts of SOC including a model for service-oriented interactions [3], an abstract model for service discovery and binding [4], and a model for dynamic reconfiguration [5].

However, no operational semantics has been defined for $SRML$ that accounts for the symbiosis that exists between the two levels of abstraction, i.e., the way computation at the state level induces changes at the configuration level, and vice-versa. This has precluded a full implementation of the language and the development of tools that, like animation, can support service development.

The work reported in this paper develops such an operational semantics using graph transformation systems – a formalism for which several tools have been developed (for example, in Maude [6]). For this purpose, we define a graph transformation formalism that allows us to encode provides and requires interface specifications in temporal logic as well as service-level agreements (SLA). This formalism is based on symbolic graphs [7] – the most expressive graph formalism for specifying attribute values – which we extend with a temporal logic to obtain what we call 'temporal symbolic graphs'. Using this new formalism, we define a transformation system for business configurations. Our representation allows service modules to be connected with requires and provides specifications using the temporal logic LTL [8] and to express service-level constraints using the propositional fragment of that logic.

The paper is organized as follows. In Section II we present an overview of $SRML$. In Section III, we define symbolic graphs with temporal formulas. Section IV is dedicated to showing how we can define the crucial part of the semantics of $SRML$ (business configurations) using temporal symbolic graph transformations; a simple example of a service that makes flight and hotel reservations is used as an example. Finally, in Section V, we discuss some related work and conclude the paper.

II. Introduction to SRML

The essential concept of $SRML$ is the notion of module, which is inspired by the Service Component Architecture (SCA) [9]; see [1], [10], [3], [4], [5] for a detailed description of the language and some of its semantic aspects. A module can be seen as a graph of components that are connected by wires along which they communicate asynchronously; a module also includes provides and requires interfaces to allow a module to bind to other modules.

As an example of a module we present a booking agent. This module, which is graphically depicted in Figure 1, offers a service for booking trips (flight and hotel). It includes a single component (BookAgent) that takes care of the booking and three interfaces: a provides interface (Customer) for customer requests and two requires interfaces (FlightAgent and HotelAgent). The BookAgent receives requests for booking trips from customers that are connected to the Customer interface; it then binds to a FlightAgent and a HotelAgent to request a flight and a hotel, respectively; those services provide the corresponding reservation confirmations through a hotel and a flight code, which are then returned to the customer by the BookAgent. For our purposes in this paper, we focus only on how the BookAgent requests a flight to the FlightAgent.

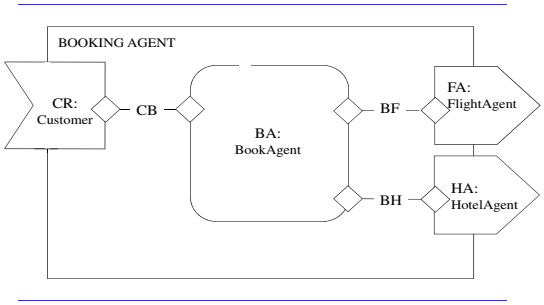


Figure 1. BOOKING AGENT service module

Components are specified by a 'business role' consisting of a signature and an *orchestration* part. A signature declares the events or messages in which a component may take part and the orchestration describes the behavior of the component. In Figure 2 we exhibit a fragment of the specification of the main component of the Booking Agent module. In this specification, we declare that BookAgent has an interaction called *booktrip* in which the component participates by receiving and then sending information (**r&s**), and another interaction called *bookflight* in which the component participates by sending and then receiving information (**s&r**). The interaction *booktrip* has four input parameters (*from, to, out* and *in*) and one output parameter (*tconf*).

In the orchestration part, first we declare the local variables of the component and, possibly, their initialization. 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 used for storing the basic data of the trip being booked (source, destination, departure and return dates, respectively), and *fconf* and *hconf* are used for storing the flight and hotel reservation codes that have been booked. We also declare the local effects of each interaction in which the component takes part. In the example, we show the transition *Torder*, which is triggered by the event *booktrip* that the component receives; the effects of the transition are to assign the local variables *from, to, in*, and *out* the contents of the corresponding parameters of *booktrip*. The transition also triggers the sending of *bookflight* with the corresponding input parameters instantiated with the updated values of the local variables.

In this way, components interact asynchronously with other components by exchanging events or messages. This form of interaction is essential for supporting the forms of loose binding that are typical in service oriented computing. Note that events or messages model the interactions that are exchanged between parties (service requesters and suppliers) or internally within a party, but not exchanges within the middleware infrastructure that implements service discovery, selection and binding; that is, the modeling takes place exclusively at the business level.

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'* ∧
...

Figure 2. BookAgent business role

External interfaces are specified through business protocols, which include a signature and a specification of the conversations that the module expects relative to each party. A fragment of the business protocol of Customer is depicted in Figure 3. Business protocols are used by the

middleware when discovering external parties that can bind to the interface. Wires bind the names of the interactions and specify the protocols that coordinate the interactions between two parties. For example, Booking Agent includes the wire CB that connects the business role BookAgent with the business protocol of the customer. For simplicity, we do not illustrate wire specifications, which in the examples used in the paper are simple one-to-one mappings between the signatures of the specifications that do not impose additional protocols (for example, encryption).

BUSINESS PROTOCOL Customer **is**
r&s *booktrip*
♦ *from,to*:string; *out,in,reqdate*:integer; *getfb*:bool;
✉ *tconf*: (fcode,hcode); *orprice,price*:float;
s *confirm*
♦ *info*:string, confb:bool;
...
BEHAVIOUR
$\Box\Diamond booktrip.getfb \wedge$
$\Box(booktrip.getfb \implies \Diamond confirm.confb)$
$dif(booktrip.out, booktrip.reqdate) \geq 90 \implies$
$\Diamond booktrip.price = 0.9 \times booktrip.orprice$
...

Figure 3. Customer business protocol

The specification of the Customer business protocol includes the interaction $booktrip$, which maps to the interaction with the same name of BookAgent, and the interaction $confirm$, which just sends a confirmation message to the customer after receiving the trip request. The behavior of the protocol is defined in the temporal logic LTL that we define in next section; the first formula guarantees that a confirmation message will always be sent after receiving a booktrip request. The second states that if the date of the request and the date of departure of the travel differs in 90 days or more, the final price of the travel has a 10% of discount.

Another aspect of $SRML$ that we will represent in our framework is service level agreements (SLA). Service modules are selected at run time to bind to interfaces subject to given SLA. An SLA is a set of constraints that have to be taken into account during discovery and selection. Every constraint involves a set of variables that includes both local parameters of the service being provided (e.g., the percentage of the cost of a trip that is refundable). $SRML$ adopts a framework for constraint satisfaction and optimization in which constraint systems are defined in terms of c-semirings [11]. These c-semirings contain a space of degrees of satisfaction with two distinguished values – 1 for maximal satisfaction and 0 for no satisfaction – a composition and a choice operation. The space of degrees of satisfactions could be, for example, the set $\{0, 1\}$ for crisp constraints (i.e., those that are either satisfied or not satisfied) or the interval $[0, 1]$ for intermediate degrees of satisfaction (soft constraints), A constraint system consists of a c-semiring, a totally ordered set of configuration variables, and a finite domain for each variable. A constraint consists of a function def_i that, given the values of some variables of the constraint system, returns a degree of satisfaction. An external configuration policy of a service module consists of a constraint system based on a fixed c-semiring, a set of constraints and an assignment of the configuration variables to the components of the module (Customer, Booking Agent, Flight Agent, etc).

In our example, we consider a constraint system over the interval $[0, 1]$, i.e., satisfaction levels are between worst (0) and best (1). We work with three configuration variables of type integer: pbd denotes the period before departure in which the customer makes the refund request, prc is the percentage of the refund that the Booking Agent decides to assign to the customer, and $bfee$ is the booking fee of the Booking Agent. The variable pbd is assigned to the customer and the variables prc and $bfee$ to the Booking Agent. We illustrate two constraints: the first requires that the percentage prc of the cost that is refundable is bounded by the least of 90 percent and a linear function of the period before departure pbd during which the deal can be revoked (the maximum refundable cost (100 percent) is obtained with 8 days or more):

$$def_1(pbd, prc) = \begin{cases} 1, & \text{if } 1 \leq pbd \text{ and } prc \leq 90 \\ & \quad \text{and } prc \leq 50 + 7 * pbd \\ & \quad \text{and } prc \leq 100. \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

The second constraint states that the degree of satisfaction of a booking agent is inversely proportional to the booking fee $bfee$:

$$def_2(bfee) = 1/(1 + bfee) \quad (2)$$

In [5] business configurations are defined using the notion of state configuration. A state configuration is a pair of a simple graph (undirected, without loops or multiple edges) of components and wires, and a configuration state. A configuration state is a mapping of states to the components and states to the wires. Thus, a state configuration also includes the values contained by the local variables of wires and components, and the events or messages that are on the components and wires waiting to be processed.

Configuration states can evolve in two different ways: execution steps and reconfiguration steps. In execution steps, the component states are changed by removing from the buffer the messages selected to be processed and adding those that are delivered to the component. The wire states are changed by removing from the buffer the messages that are delivered to the components and adding those that are published to the wire.

For reconfiguration steps, $SRML$ uses a typing mechanism through which so-called activity modules are used for typing the sub-configurations that, in a given state, execute

the business activities that are running. Such types are needed for capturing the business activities that perform in a configuration and determine how the configuration evolves. When a service module SM whose provides interface matches the requires interface of an activity module AM is selected, 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 with AM. The activity module associated with the enriched activity would include the components and wires of that activity and, in addition, the remaining (non-matched) interfaces of AM and SM.

The sequence diagram in Figure 4 captures part of the execution associated with a request from a customer. The execution involves four parties that are dynamically added to the execution: the customer, the booking agent, the flight agent and the hotel agent. Initially we just have the customer agent, which sends a request to the booking agent. After a process of service discovery and binding, one booking agent is chosen, which receives the travel requests. After processing the request, the booking agent sends in parallel two independent requests for hotel and flight booking. Again, after two independent processes of service discovery and binding, an hotel and a flight agent are chosen. They both send in parallel the chosen hotel and flight to the booking agent, and the booking agent sends this information to the customer. In the rest of the paper we consider only a flight request of a customer and represent as symbolic graphs with temporal formulas two rules to generate and process events, and the rule to connect the customer to the chosen booking agent.

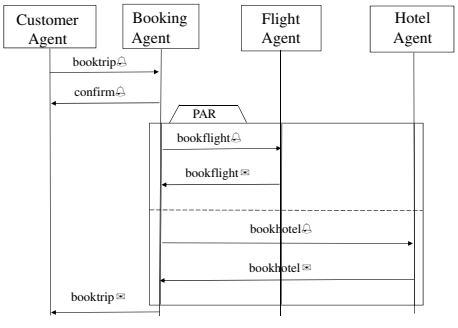


Figure 4. A sequence diagram capturing part of the execution associated with a request from a customer agent

III. 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. We present their semantics and, for temporal symbolic graphs, we also present the semantics of temporal transformation systems.

A. *first-order symbolic graphs*

Symbolic (hyper)graphs [7] 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 5 specifies a class of attributed graphs including distances in the edges that satisfy the triangle inequality.

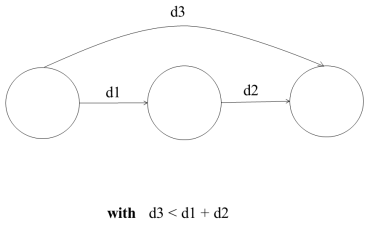


Figure 5. A symbolic graph

Symbolic graphs are based on a special kind of labeled graphs, called E-graphs, where labels are variables (for more details, see [12], [13]). The only difference between the notion of E-graph that we use and that in [12] 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.

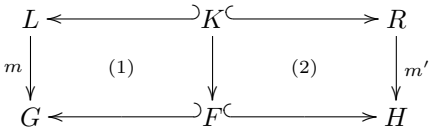
Definition 1: A *first-order symbolic graph* over the data algebra D is a pair $\langle G, \Phi_G \rangle$, where G is an E-graph over a set of variables X and Φ_G is a set of first-order formulas over the operations and predicates in D including variables in X and elements in D.

In Figure 6 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 [7] 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 [13].

As in [13], 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.



In the case of symbolic graph transformation, we consider that the left-hand side of the rules includes no conditions. As shown in [14] 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 2: 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 6 we show a rule with two events and a BookAgent component. The rule states that when a *booktrip* event arrives, the BookAgent registers it 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 event ($booktrip$) and in the outgoing event ($bookflight$). The intermediate graph K in general denotes the common subgraph between L and R. In our example this would be the BookAgent hyperedge; for simplicity, we do not depict it.

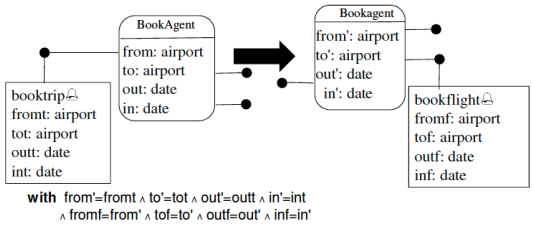


Figure 6. A symbolic rule

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.

B. 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 [8]. 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 $\to$. 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.

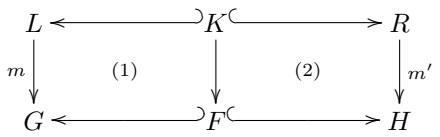
Definition 3: A *temporal symbolic graph* SG over the data algebra D is a pair $SG = \langle G, \Phi_G \rangle$, where G is an E-graph over a set of variables X and Φ_G is a set of LTL formulas over the operations and predicates in D including variables in X and elements in D.

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.

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 4: 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:



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

   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.

Next we give some definitions that are needed to define the semantics of transformation systems for temporal symbolic graphs.

Definition 5: A transformation system of symbolic graphs is a symbolic graph SG_0 together with a finite set of symbolic rules $\mathcal{SR}$.

Definition 6: A path of a transformation system $(SG_0, \mathcal{SR})$ is a possibly infinite sequence of symbolic graphs (SG_0, SG_1, SG_2, ...) such that, for all i, SG_{i+1} is obtained by applying a rule of $\mathcal{SR}$ to SG_i.

The representation of the semantics of a transformation system is again a state transition system where states are now symbolic graphs and the transition relation is defined by rule application of the transformation system. Temporal formulas restrict the structure associated with the semantics of the transformation system. For example, if a temporal symbolic graph TSG has the temporal formula $\Diamond(b = 0)$ requesting that the attribute b eventually has to have the value 0, the semantics of TSG must not include paths in which the attribute will never have the value 0.

Rules can add temporal formulas. When a temporal formula is added to a symbolic graph by a rule, then the semantics of the subtransformation system starting from this new graph has to include only paths that satisfy this new temporal formula. Therefore, the structure that defines the semantics of a transformation system with temporal symbolic graphs must include only the paths and subpaths that satisfy the temporal formulas that are in the initial symbolic graph and in all the symbolic graphs of the structure.

More precisely, we can define that a path of a transformation system satisfies a temporal formula as follows:

Definition 7: A path $P = (SG_i, SG_{i+1}, \dots)$, where $i \geq 0$, of a transformation system $(SG_0, \mathcal{SR})$ satisfies a formula with at least a temporal operator in the temporal symbolic graph SG_i if the following holds:

- $P \models \bigcirc tf$ if P has at least two symbolic graphs and $P' \models tf$ where $P' = (SG_{i+1}, ...)$.
- $P \models (tf_1 \,\mathcal{U}\, tf_2)$ if there exists a subpath $SG_i, ..., SG_j$ where $j \geq i$ such that tf_2 holds in SG_j and, for all $i \leq k < j$, $tf1$ holds in SG_k.

As a simple example, consider a symbolic graph with one node and an integer attribute b, and a $with$ clause containing the propositional formula $b \geq 2 \wedge b \leq 5$, which requires that the current value must be between 2 and 5, and the temporal formula $\Diamond(b = 0)$ meaning that b eventually becomes 0. Consider now a transformation rule that transforms the value of the attribute with the equation $b = b_p - 1$; the semantics of the transformation system would be just a path where the rule has been applied five times. In the last application of the rule, the propositional formula will be transformed into $b \geq -3 \wedge b \leq 0$ and, therefore, the temporal formula is still satisfied. After applying once more the rule, the temporal formula would not be satisfied anymore because b would never become 0. If we add another rule that now transforms the attribute with the equation $b = b_p - 2$, the semantics of the transformation system is a structure with paths of lengths $2, 3, 4$ and 5 rule applications, including all possible permutations of the application of these two rules while the temporal formula is satisfied.

As another example, consider the same initial symbolic graph, the same first rule, but a different second rule consisting of the propositional formula $b = b_p + 1$ and the temporal formula $\Box(b \leq 3)$, which states that b must always be lower than or equal to 3. If we did not have the temporal formula in the second rule, we could have non-bounded paths with just applications of the second rule; this is because we can always obtain the value 0 applying the first rule a similar number of times. With the temporal formula in the second rule, we have to guarantee that b has the value 3. Therefore, although the paths are again non-bounded, the interval of the values can not surpass $b \geq 3 \wedge b \leq 6$ because if we apply the rule again, we can not prove anymore that $b \leq 3$.

An interesting property of our semantics is that if we have two temporal symbolic graphs SG_1 and SG_2 such that SG_1 is transformed to SG_2 by a temporal symbolic rule p, then we can build a transformation function from the semantics of SG_2 to the semantics of SG_1. Basically for each state transformation system $STSG_2$ of the semantics of SG_2 we can build a state transformation system $STSG_1$ which belongs to the semantics of SG_1.

IV. A GRAPH-SEMANTICS FOR BUSINESS CONFIGURATIONS

In our graph semantics, 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, the semantics has two different graph transformation rules for these two ways of transforming the state: state transformation rules and reconfiguration rules.

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 [7], symbolic graphs are more expressive than the standard approach [12]: 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 [14]. Moreover, the extension with temporal formulas allow us to model de behavior of components and modules and to specify business protocols. In particular, as shown below, requires and provides specifications can be represented in a very natural way as a set of temporal formulas in the $with$ clause of symbolic graphs.

Our graph semantics is presented in the next two subsections. First we present the graph semantics of business configurations, and then its associated transformation system.

A. Business configurations

The first definition addresses the basic concept of business activity:

Definition 8: A business activity is a connected symbolic graph with two types of hyperedges:

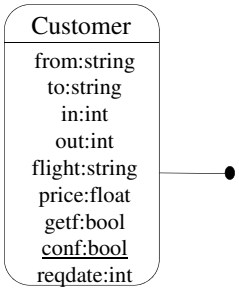
- Hyperedges that represent components with a positive number of nodes, an attribute with the name of the component and a set of attributes of the component. We refer to them as component hyperedges.
- Hyperedges that represent events connected with just one node, an attribute with the name of the event,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

We consider two types of nodes: internal and interface nodes. Both types of nodes can be part of different component hyperedges and 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 9: 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 hyperedge, they triggered a process of selection of a reconfiguration rule package. 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 hyperedge that represents the customer component with a set of attributes for the flight. A graphical representation is in Figure 7.



with from = "Barcelona" ∧ to = " New York" ∧ in = 240615
∧ out = 70715 ∧ getf = false ∧ conf = false ∧
reqdate = 150215 ∧ price = 0.0

Figure 7. Business configuration with just a customer activity

Figures 9 and 11 show two different stages of the initial business configuration in Figure 7. Figure 9 has 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 11, binding the interface node of a booking agent subsystem. This subsystem has also two additional interface hierarchical nodes. We further explain Figure 11 later in this section.

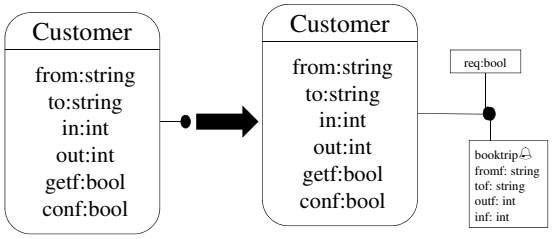
B. Transformation systems for business configurations

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 10: 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 hyperedge component;
- transform the values of the attributes of a component hyperedge using information of the processed events of its nodes;
- publish an event in the node of a hyperedge component.

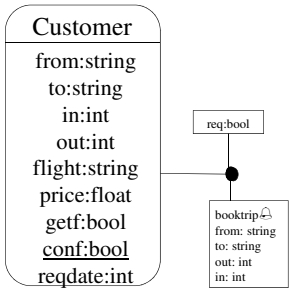
An example of a state transformation rule is in Figure 8: it publishes an event in the interface node of the hyperedge component of the customer.



with fromf = from ∧ tof = to ∧ outf = out ∧ inf = in ∧
getf = true ∧ req ⇔ ◇ conf

Figure 8. Rule $initr$ associated with the customer activity

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 $initr$ in Figure 8 is applied to the business configuration, the initiating event is added to the business configuration. The resulting new business configuration is in Figure 9. Note that the rule also has a temporal subformula in the with-clause, which requests always a confirmation from the chosen Booking Agent.



with from = "Barcelona" ∧ to = " New York" ∧ in = 240615
∧ out = 70715 ∧ getf = true ∧ conf = false ∧
reqdate = 150215 ∧ price = 0.0 ∧
req ⇔ ◇ conf

Figure 9. New business configuration with a trigger event

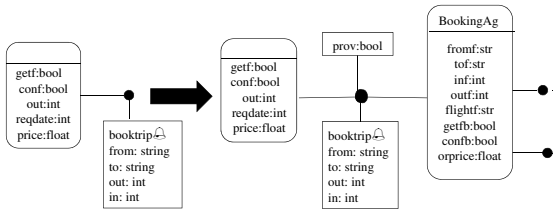
Definition 11: A reconfiguration rule connects one business activity with another.

An example of a reconfiguration rule is in Figure 10: it binds a business activity including a customer component with a business activity including a booking agent component. Note that the rule requires that the customer have at least five 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. The other three attributes are needed to define a temporal formula that provides a discount in the price of the flight.

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%.

Definition 12: A reconfiguration rule package contains one distinguished reconfiguration rule and a set of state transformation rules.

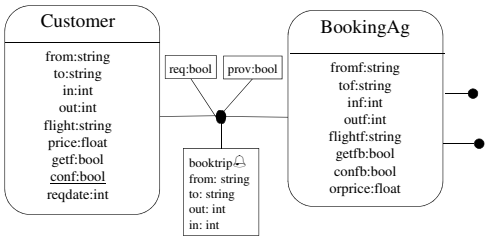
The event in Figure 9 triggers a process of selection of a reconfiguration rule together with a set of state transformation rules. The selected reconfiguration rule is the one in Figure 10. After applying the reconfiguration rule, an instance of



with getf = true ∧
prov ⇔ □ ◇ getfb ∧ □ (getfb ⇒ ◇ (conf ∧ confb)) ∧
dif(out,reqdate) ≥ 90 ⇒ ◇ price = orprice – orprice*10/100

Figure 10. A reconfiguration rule

a booking agent module is connected to the instance of the customer activity module as represented in Figure 11. Before making the connection, 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.



with from = "Barcelona" ∧ to = " New York" ∧ in = 240615
∧ out = 70715 ∧ getf = true ∧ conf = false ∧
reqdate = 150215 ∧ price = 0.0 ∧ getfb = false ∧
req ⇔ ◇ conf ∧
prov ⇔□◇getfb ∧ □ (getfb ⇒◇(conf ∧ confb))
dif(out,reqdate) ≥ 90 ⇒ ◇price = orprice – orprice*10/100

Figure 11. 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 13: A business repository is a set of reconfiguration rule packages.

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

Definition 14: 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 15: 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 business 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 in Figure 7 has been transformed to the business configuration in Figure 9 by first applying the state transformation rule in Figure 8. In a second step, after applying the distinguished

rule in Figure 10, we obtain the business configuration in Figure 11.

A process of service discovery and binding is needed to obtain a reconfiguration rule package. The set of state transformation rules is then updated with the set of state transformation rules associated with the reconfiguration rule. This 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 (in our case, just a Flight Agent).

To complete the execution of the service oriented program presented in the sequence diagram in Figure 4, we would need another reconfiguration rule to connect the Booking Agent with a Flight Agent and some state transformation rules to send the chosen flight from the Flight Agent to the Booking Agent, and from the Booking Agent to the Customer.

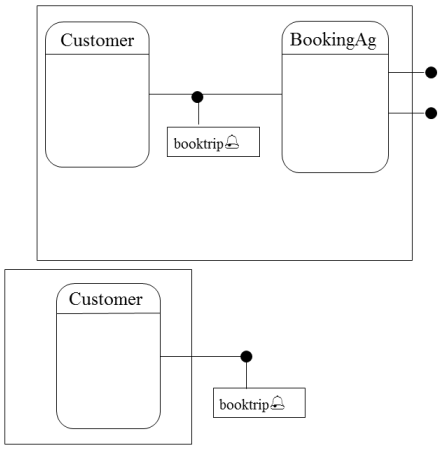


Figure 12. Updated business configuration with a booking agent

We now explain how service level agreements can be added to our framework. First the operations of c-semirings can be easily added to the data algebra D of symbolic graphs. The two constraints defined in Section II can be added in the reconfiguration rule of our example in the following way:

- Add the configuration variables as attributes of the attached components. In our case the attribute $pbd : int$ is assigned to Customer and the attributes $prc : int$ and $bfee : int$ are assigned to the Booking Agent.
- Add a variable for each constraint in the node that connects the Customer and the Booking Agent in the symbolic graph of the right-hand side of the reconfiguration rule. In that node there is already a boolean variable named $prov : bool$. In our case the variables $sla_1 : \{0, 1\}$ and $sla_2 : [0, 1]$ would be added.
- In the with-clause of the reconfiguration rule add a formula for each case which defines a constraint. For our example we have three formulas, two for the first constraint and one for the second:
  - $1 \leq pbd \wedge prc \leq 90 \wedge prc \leq 50 + 7 * pbd \wedge prc \leq 100 \implies sla_1 = 1$
  - $\neg(1 \leq pbd \wedge prc \leq 90 \wedge prc \leq 50 + 7 * pbd \wedge prc \leq 100) \implies sla_1 = 0$
  - $sla_2 = 1/(1 + bfee)$

In general, a business configuration can have several independent activities. In Figure 12 we have two independent activities of two different customers, one in the final state of the running example of the paper, and the other ready to trigger a process of discovery of another booking agent which has not to be the same as the chosen for the other customer. We omit the attributes of the components and events and the with-clause, and we encapsulate the two independent activities.

Finally, we relate our semantic framework with the one presented in [5]. We concentrate on the following concepts of their semantics:

- configuration steps
- business reflective configurations and reconfiguration steps

Configuration steps are related to our concept of state transformation rules. In [5], work configuration steps affect components and wires. Components and wires have buffers; in a configuration step, the messages selected to be processed are removed from every component buffer and those messages that are delivered to the component from the wire are added.

From the point of view of the wires, configuration steps change every state of the wires by removing from the buffer the messages that are delivered to the components and adding those that are published to the wire by the connected components. There are two constraints that configuration steps must satisfy: every wire delivers all messages to and only to the component it connects, and all the messages that are published to the wire come from the same connected components.

Our state transformation rules do not formalize this behavior exactly but in a similar way. Being more concrete, components do not have proper buffers and events only exist in the nodes of the component hyperedges that correspond to the concept of wires. A state transformation rule can transform the state of a component but an event can not inhabit it. A state transformation rule can also generate events to a wire. The two constraints must be satisfied also in state transformation rules and they must be checked before adding a reconfiguration rule package to a business repository.

The business configurations in [5] have activities typed by activity modules. These types are used for deciding how the configuration will evolve through events that trigger the discovery process. This information on types makes business configurations reflective, which makes the system adaptable to reconfiguration. In reconfiguration steps the business configurations evolve at the level of activities and at the level of types. In our case, we do not have this notion of type but we have information in the with-clause of the business configuration and in the with-clause of a reconfiguration rule. More concretely, in the with-clause of the business configuration we can have the requirements specification of an activity module, and in the with-clause of the reconfiguration rule we can have also the provides specification of a service module, using temporal formulas.

V. Concluding remarks

In this paper, we have presented a novel approach for describing in a uniform way the evolution of service systems by state transformations and run-time service discovery and binding, and we have used it to define a semantics for $SRML$, a language that was developed as part of the European project Sensoria [2]. The main difference between $SRML$ and other approaches in the area of service-oriented systems such [15], [16], [17], [18] is that $SRML$ supports service binding at run time.

Several semantic aspects of $SRML$ have already been addressed in several papers (e.g., [3], [4], [5]). In this paper, we replace the original semantics of business configurations with a transformation system that can be easily implemented with a tool for symbolic graph transformation. Additionally, our approach offers a formal semantics for binding that is independent of specific languages that might be adopted. It is also more expressive in relation to the conditions through which services can be selected, which could be used to enhance existing languages such as WSDL [19].

In this paper, we have also presented the novel notion of temporal symbolic graphs, which could play a relevant role in the specification of reactive systems. In the future we plan to study the foundations of this new kind of graphs, setting the basis for their use in different contexts.

We plan to study how to define hierarchical graph transformation with flexible notions of hierarchical graph morphisms so that it is possible to perform transformations that change the hierarchical structure of a graph. This approach would be a variation of the work reported in [20]. This would allow us to define a semantics for an ambient calculus with business configurations and business repositories.

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] J. L. Fiadeiro, A. Lopes, L. Bocchi, and J. Abreu, "The sensoria reference modelling language," in *Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing*, ser. Lecture Notes in Computer Science, M. Wirsing and M. M. Hölzl, Eds. Springer, 2011, vol. 6582, pp. 61–114. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-20401-2_5

[2] M. Wirsing and M. M. Hölzl, Eds., *Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing*, ser. Lecture Notes in Computer Science. Springer, 2011, vol. 6582. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-20401-2

[3] J. L. Fiadeiro, A. Lopes, and J. Abreu, "A formal model for service-oriented interactions," *Sci. Comput. Program.*, vol. 77, no. 5, pp. 577–608, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.scico.2011.12.003

[4] J. L. Fiadeiro, A. Lopes, and L. Bocchi, "An abstract model of service discovery and binding," *Formal Asp. Comput.*, vol. 23, no. 4, pp. 433–463, 2011.

[5] J. L. Fiadeiro and A. Lopes, "A model for dynamic reconfiguration in service-oriented architectures," *Softw Syst Model*, pp. 12:349–367, 2013.

[6] A. Boronat and J. Meseguer, "An algebraic semantics for mof," in *FASE*, ser. Lecture Notes in Computer Science, J. L. Fiadeiro and P. Inverardi, Eds., vol. 4961. Springer, 2008, pp. 377–391.

[7] 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.

[8] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems - specification*. Springer, 1992.

[9] "The Open Service Oriented Architecture collaboration," whitepapers and specifications available from www.osoa.org (see also oasis-opencsa.org/sca).

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

[11] S. Bistarelli, U. Montanari, and F. Rossi, "Semiring-based constraint satisfaction and optimization," *J. ACM*, vol. 44, no. 2, pp. 201–236, 1997.

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

[13] ——, *Fundamentals of Algebraic Graph Transformation*, ser. EATCS Monographs of Theoretical Computer Science. Springer, 2006.

[14] F. Orejas and L. Lambers, "Lazy graph transformation," *Fundam. Inform.*, vol. 118, no. 1-2, pp. 65–96, 2012. [Online]. Available: http://dx.doi.org/10.3233/FI-2012-706

[15] W. 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*, ser. Dagstuhl seminar proceedings, vol. 06291. Schloss Dagstuhl, 2006.

[16] M. Broy, I. H. Krüger, and M. Meisinger, "A formal model of services," *ACM Trans Softw Eng Methodol*, vol. 16, no. 1, 2007.

[17] B. Benatallah, F. Casati, and F. Toumani, "Web service conversation modeling: a cornerstone for e-business automation," *IEEE Internet Computing*, vol. 8, no. 1, pp. 46–54, 2004.

[18] W. Reisig, "Modeling and analysis techniques for web services and business processes," in *FMOODS*, ser. Lecture Notes in Computer Science, vol. 3535. Springer, 2005, pp. 243–258.

[19] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (WSDL)," 2001, 1.1. Technical report, W3C, available from www.w3.org/TR/wsdl/.

[20] F. Drewes, B. Hoffmann, and D. Plump, "Hierarchical graph transformation," *J. Comput. Syst. Sci.*, vol. 64, no. 2, pp. 249–283, 2002. [Online]. Available: http://dx.doi.org/10.1006/jcss.2001.1790