A semantics of business configurations using symbolic graphs

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

***Abstract*—In this paper we give graph-semantics to a fundamental part of the semantics of the service modeling language $SRML$: business configurations. To achieve this goal we use symbolic graph transformation systems. We formalize the semantics using this graph transformation system and illustrating it with a simple running example of a trip booking agent.**

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

I. Introduction

$SRML$ ([1], [2], [3]) is a service modeling language designed within the project SENSORIA [4]. Its state model is considered at two levels of abstraction. Roughly, at the lowest level, a *state configuration* is a graph consisting of interconnected components and, at the highest level, *business configurations* are graphs consisting of interconnected activities, where each activity is a graph of components. This definition at two levels of abstraction is needed to allow for dynamic service binding. Unfortunately, the operational semantics of SRML was not defined in a way that facilitates the animation of its models, making any such implementation too complex.

The goal of this work is to provide a graph transformation semantics for $SRML$ so that its models could be animated using a generic graph transformation tool, such as the Maude implementation of graph transformation [5]. For this semantics we use symbolic graphs and symbolic graph transformation [6], which have been shown that it is M-adhesive. Therefore we show how this framework can be used to define (part of) a graph transformation semantics of $SRML$. We believe that this new semantics has the advantage that an implementation could be relatively easy if we had a tool for the graph transformation system that we propose, because much of the implementation work would be delegated to the graph transformation tool.

The paper is organized as follows. In Sections 2 and 3, we present an overview of $SRML$ and symbolic graphs, respectively. Then, Section 4 is dedicated to showing how we can define part of the semantics of $SRML$ using symbolic graph transformation. Finally, in Section 5, we discuss some related work and conclude the paper.

II. Introduction to SRML

The essential concept of the Sensoria Reference Modeling Language ($SRML$) is the notion of module which is inspired by the constructions presented in Service Component Architecture (SCA). See [1], [2], [3], [7] for a detailed description of the language. Roughly speaking, a module can be seen as a graph of components that are connected by wires. Moreover a module also includes some provides and requires interfaces, which are also connected by wires to the components. As an example of a module we present a booking agent. This module, which is graphically depicted in Fig. 1, is supposed to offer a service for booking trips (flight and hotel). It includes a single component (BookAgent) that is supposed to take care of the booking and three interfaces: a provides interface (Customer) for customer requests and two requires interfaces (FlightAgent and HotelAgent). The BookAgent is supposed to receive trip reservation requests from customers that are connected to the Customer interface. Then, BookAgent is supposed to request a flight and a hotel to services connected to the FlightAgent and HotelAgent, respectively, which are supposed to provide the corresponding reservation confirmations through a hotel and a flight code. These codes will then be returned to the customer. For our purposes it is enough to see how the BookAgent requests a flight to the FlightAgent.

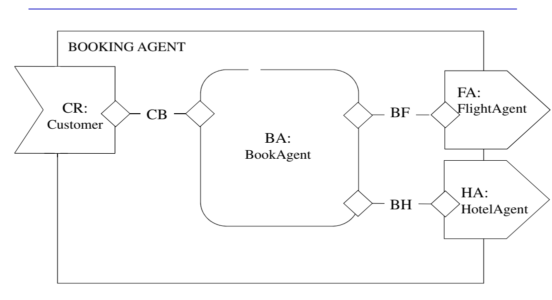

Figure 1. BOOKING AGENT service module

Components are specified by a business role consisting of a signature and an *orchestration* part. The signature declares the events or messages in which that component may take part and the orchestration describes the behavior of the component. For instance, below we can see a small part of the specification of the main component of the booking agent module.

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

In this specification we declare that BookAgent has an interaction called *booktrip* in which the component participates receiving and then sending information (**r&s**) and another interaction called *bookflight* in which the component participates sending and then receiving information (**s&r**). For example, *booktrip* has four input parameters (*from, to, out* and *in*) and one output parameter (*tconf*). Then, in the orchestration part, first we declare the local variables of the component and possibly their initialization, and then we specify the effects of the interactions in which the component may take part. For instance, in the example, the local variables *from, to, in*, and *out* are supposed to store the basic data of the trip being booked (source, destination, departure and return dates, respectively), and *fconf* and *hconf* are supposed to store the flight and hotel reservation codes that have been booked. In the example, we also declare the local effects of an interaction, called *Torder*, in which the component takes part. This interaction is triggered by the event *booktrip* that the component receives. The contents of the local variables *from, to, in*, and *out* after the interaction are the contents of the corresponding parameters of *booktrip*. Moreover, the interaction triggers an event or message *bookflight*, which is sent by the component with the corresponding input parameters. Thus, components can 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. Therefore, events or messages model the interactions that are exchanged between parties (service requesters and suppliers) or internally within a party, but not exchanges with the middleware infrastructure that implements the binding.

External interfaces are specified through business protocols. They also include a signature and they specify the conversations that the module expects relative to each party. It is the responsibility of the counterparty to adhere to these protocols. Finally, wires bind the names of the interactions and specify the protocols that coordinate the interactions between two parties. For instance, this module includes the wire CB that connects with two diamonds the business protocol of the customer of the BOOKING AGENT module and the business role BookingAgent of the same module. We do not include here an example of an interface or of a wire specification, since they are not relevant for this paper. Another issue of $SRML$ that we do not treat in this paper in detail is service level agreement (SLA).

In [7] business configurations are defined using the notion of state configuration. A state configuration is a pair of a simple graph (undirectecd, without self-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.

These 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 ared 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, $SRML$ defines business-reflective configurations. This concept is needed to capture the business activities that perform in a configuration and determine how the configuration evolves. Additionally, it is needed a typing mechanism of complex structures which are called activity modules. These activity modules type the sub-configurations that in a given state, execute the business activities that are running. Thus, when the requires interface of an activity module AM matches the provides interface of a service module SM the two modules are connected and the activity is bound to this new service. This implies that initialized instances of the components and wires of SM are added to the state configuration and also to the activity associated to AM. The activity module associated to the enriched activity would include the components and wires of that activity and, in addition, the remaining (non-matched) interfaces of AM and SM.

III. SYMBOLIC GRAPHS AND SYMBOLIC GRAPH TRANSFORMATION

Symbolic (hyper)graphs [6] can be seen as a specification of a class of attributed graphs (i.e. of graphs including values from a given data algebra in their nodes or edges). In particular, in a symbolic graph, values are replaced by

variables and, moreover, a set of formulas, Φ, specifies the values that the variables may take. Then, we may consider that a symbolic graph SG denotes the class of all graphs obtained replacing the variables in the graph by values that satisfy Φ. For instance, the symbolic graph in Figure 2 specifies a class of attributed graphs, including distances in the edges, that satisfy the well-known triangle inequality.

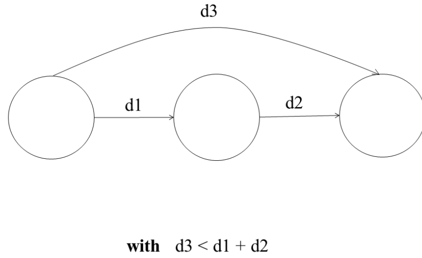

Figure 2. A symbolic graph

The notion of symbolic graph is based on the notion of a special kind of labeled graphs called E-graphs (for more details, see [8], [9]). The only difference of the notion of E-graph that we use with respect to the notion in [8] is that we deal with hypergraphs. This means that, for every graph G, instead of having source and target functions that map edges to nodes, we have an attachment function, $attach_G$, that maps each (hyper)edge to a sequence of nodes, i.e. the nodes connected by the edge. The definition is the following:

Definition 1: An E-graph G = $(V_1, V_2, HE_1, E_2, E_3, attach_1, (source_i, target_i)_{i=2,3})$ consists of sets

- V_1 and V_2 called graph nodes and data nodes respectively,
- HE_1, E_2, E_3 called graph hyperedges, node attribute edges and hyperedge attribute edges,

and attachment, source and target functions

- $attach_1 : HE_1 \to V_1^*$
- $source_2 : E_2 \to V_1, target_2 : E_2 \to V_2$
- $source_3 : E_3 \to HE_1, target_3 : E_3 \to V_2$

Definition 2: A *symbolic graph* over the data algebra D is a n-tuple $\langle G, \Phi_G \rangle$, where G is an E-graph over a set of variables X, Φ_G is a set of first-order formulas over the operations and predicates in D including variables in X and elements in D.

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

In Figure 3 we have an example of symbolic rule with one symbolic graph on the left side of the big black right arrow and another on the right side. The meaning of the rule will be explained later on this section and now we describe the E-graph on the left hand side. It has two hyperedges: one which denotes an event which has just one node and five hyperedge attribute edges for its five attributes: the name of the event ($booktrip$) and four event parameters. The other hyperedge denotes a component which has three nodes and five hyperedges attribute edges: the name of the component ($BookAgent$) and four variable components. These kind of graphs will be part of business activities which will be defined in the next section.

As we mentioned at the beginning of this section, symbolic graphs specifies a class of attributed graphs. Therefore next we also give the definition of attributed graphs:

Definition 3: An attributed graph AG over the data algebra D is an E-graph where V_2 is the disjoint union of all the values of every sorted set D_s of the data algebra D.

Symbolic graphs over D together with their morphisms form the category $\mathbf{SymbGraphs_D}$. In [6] 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 these kinds of graphs [9].

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

Definition 4: A *symbolic graph transformation rule* is a triple $\langle \Phi_L, L \hookleftarrow K \hookrightarrow R, \Phi_R \rangle$, where L, K are E-graphs over the same set of variables X_L, R is an E-graph over the set of variables X_R, with $X_L \subseteq X_R$, $L \hookleftarrow K \hookrightarrow R$ is a standard graph transformation rule, and Φ_L and Φ_R are sets of formulas over X_L and X_R, respectively, and over the values in the given data algebra D, with $\Phi_L \subseteq \Phi_R$.

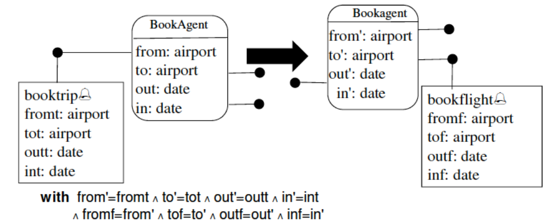

Figure 3. A symbolic rule

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

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

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

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

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

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, in the semantics we will have two different graph transformation rules for these two ways of transforming the state: state transformation rules and reconfiguration rules.

We believe that symbolic graphs are especially adequate because it is the most convenient graph formalism with attributes whose values have to be specified. For example, if we compared symbolic graphs with the most standard approach [8], there are two reasons for this. On the one hand, as shown in [6], the formalism of symbolic graphs is more expressive. 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 [10]. The original formalism specifies values with first order formulas but we plan to extend the $with$ clause of symbolic graphs with a more generic set of formulas including temporal ones. This is necessary to encode the requires and provides specification of business protocols. This change would not affect the semantics of business configurations because requires and provides specifications would be represented in a very natural way as a set of temporal formulas in the $with$ clause of symbolic graphs. We do not relate symbolic graphs because we are not aware of any graph transformation system with a similar expressive power.

Our graph semantics is presented in the next two subsections. In the first one, we present the graph semantics of business configurations, and in the second one its associated transformation system. The semantics that we propose is different than the original one [2] and the light version of [7], and therefore we do not use provide and request specifications and we do not treat either the problem of name injections in wires. In [7] one must handle with SLA (service level agreement) constraints too. We have not included SLA constraints in our paper either because they would be treated in a similar way as requires and provides specifications.

A. Business configurations

In the first definition we present the basic concept of business activity:

Definition 5: 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 will refer to them as component hyperedges.
- Hyperedges that represent events connected with just one node, an attribute with the name of the event, another with the type of the event and a set of attributes of the event. We will refer to them as event hyperedges.

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

Next, we present the concept of business configuration:

Definition 6: A business configuration is a symbolic graph where it can have in general a set of business activities.

As we 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 developed an activity module that requests a booking agent to book a hotel and 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 and hotel reservation. A graphical representation is in figure 4.

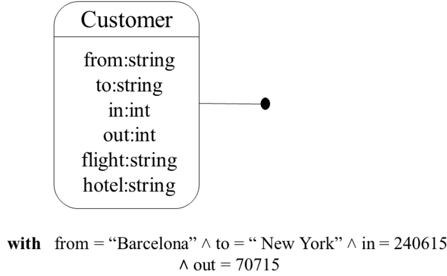

Figure 4. Business configuration with just a customer activity

Additionally, in figures 6 and 8 we have two different stages of the initial business configuration of figure 4. In 6 we have 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 8, binding the interface node of a booking agent subsystem. This subsystem has also two additional interface hierarchical nodes. We will explain further Figure 8 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 7: A state transformation rule is a rule that can make the following transformations in one activity:

- process an event eliminating this event from a node of 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 state transformation rule is in figure 5 which publishes an event in the interface node of the hyperedge component of the customer.

Other possible rules are a rule for processing the information of the reply-event of the booking agent or a rule to start the payment.When the rule $initr$ is applied to the business configuration, the initiating event is added to the business

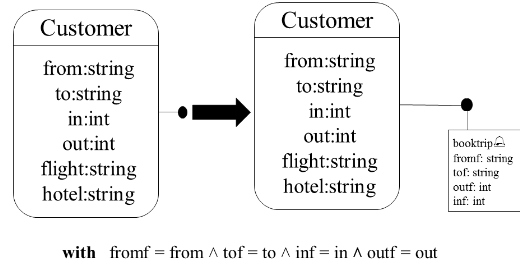

Figure 5. Rule $initr$ associated to the customer activity

configuration. The resulting new business configuration is in figure 6.

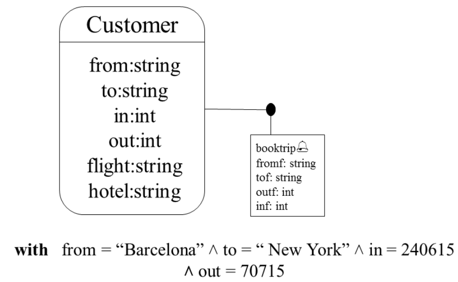

Figure 6. New business configuration with a trigger event

Definition 8: A reconfiguration rule connects one business activity with another.
An example of a reconfiguration rule is in figure 7: it binds a business activity with a customer component with a business activity with a booking agent component.

Definition 9: A reconfiguration rule package contains one distinguished reconfiguration rule, and it additionally has an associated set of state transformation rules.

The event in figure 6 triggers a process of selection of a reconfiguration rule together with a set of state transformation rules. The selected reconfiguration rule is in figure 7. After applying the reconfiguration rule, an instance of a booking agent module is connected to the instance of the customer activity module that is represented in figure 8.

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 10: A business repository is a set of reconfiguration rule packages.

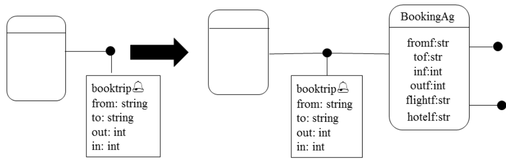

Figure 7. A reconfiguration rule

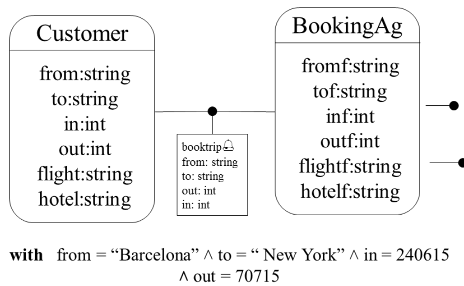

Figure 8. Updated business configuration with a booking agent

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

Definition 11: 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 12: 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 of figure 4 has been transformed to the the business configuration of figure 6 by first applying the state transformation rule of figure 5. In a second step, after applying the distinguished rule of figure 7 of a reconfiguration rule package to 6, we obtain the business configuration of figure 8. To obtain a reconfiguration rule package it is needed a process of service discovery and binding. The set of state transformation rules is then updated with the set of state transformation rules associated with the reconfiguration rule. These new set of rules will include rules to process the initiating event of the customer and generate two new initiating events to book a flight by a Flight Agent and to book a hotel by an Hotel Agent.

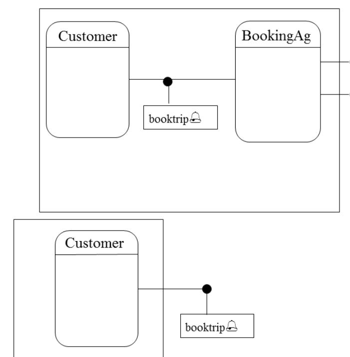

Figure 9. Updated business configuration with a booking agent

In general, a business configuration can have several independent activities. In the last figure 9 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 semantics with the one presented in [7]. 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 their work configuration steps affect components and wires. Components and wires have buffers and in a configuration step it is removed from every component buffer the messages selected to be processed and it is added those messages that are delivered to the

component from the wire. 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 which 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 these behavior exactly but it is similar. Being more concrete, components do not have proper buffers and events only exist in the nodes of the component hyperedges which correspond to the concept of wires. A state transformation rule can transform the state of a component but an event can not inhabit it. It 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.

Their business configurations have the notion of activities which are typed by activity modules. These types are used for deciding how the configuration will evolve through events that will trigger the discovery process. This information on the 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 could 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 could have the requirements specification of an activity module, and in the $with$ clause of the reconfiguration rule we could have also the provides specification of a service module. But for this, we need symbolic graphs with temporal formulas.

V. Conclusion and Related Work

The $SRML$ language was designed in the European project Sensoria (www.sensoria-ist.eu). Although a variety of tools were developed in the project to aid the creation and analysis of service-oriented software, a complete implementation of SRML has not been developed. We are not aware of any implementation of business configurations or of any alternative semantics for business configurations.

The semantics of $SRML$ has been addressed in several papers (e.g. see [1], [2], [3], [7]). In this paper we replace the original semantics of business configurations with a graph-semantics which can be easily implemented if we had a tool for symbolic graph transformation. The main difference of $SRML$ with respect to other approaches in the area of service oriented is that the language supports service binding at run time, in contrast with approaches like [11], [12], [13], [14]. Both semantics are related in the last part of the previous section. Although they are related, one should not think that a framework with our semantics would use the syntax of $SRML$. Instead, our designers would write reconfiguration rule packages with both a reconfiguration rule and a set of state transformation rules, using a graph transformation tool. 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.

In future work, we will extend the original formulation of symbolic graphs using first order formulas in the $with$ clause to include also formulas of temporal logics. To achieve this, as in [3] we will probably work with traces which are infinite sequences of set of actions. This is needed to define in the $with$ clause requires and provides specifications, which in $SRML$ are written in business protocols of external interfaces. An example of a request specification for the $BookAgent$ of the customer of our running example would include the following requirements:

- acknowledgment of the request
- acknowledgment of the payment

See [7] for a concrete requirements specification of a customer for a $MortgageAgent$ using temporal logic.

We also 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 will be a variation of the work of [15]. We think that this is necessary to define a semantics of an ambient calculus with business configurations and business repositories. Finally, we will also try to study how we can extend our semantics to cover complex service binding.

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

[3] ——, "An abstract model of service discovery and binding," *Formal Asp. Comput.*, vol. 23, no. 4, pp. 433–463, 2011.

[4] 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

[5] 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.

[6] 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.

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

[8] 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.

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

[10] 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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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