A Graph Semantics for a Variant of the Ambient Calculus more Adequate for Modeling SOC

Nikos Mylonakis

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

Abstract. In this paper we present a graph semantics of a variant of the well known ambient calculus. The main change of our variant is to extract the mobility commands of the original calculus from the ambient topology. Similar to a previous work of ours, we prove that our encoding have good properties. We strongly believe that this variant would allow us to integrate our graph semantics of our mobile calculus with previous work of us in service oriented computing (SOC). Basically, our work on SOC develops a new graph transformation system which we call temporal symbolic graphs. This new graph formalism is used to give semantics to a design language for SOC developed in an european project, but it could also be used in connection with other approaches for modeling or specifying service systems.

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

1 Introduction

Service Oriented Computing (SOC) is a software paradigm that uses services provided by external sites distributed over the Internet to deliver services to client applications. Service-oriented programs can decide at run time which services to select after a process of discovery and ranking that takes into account how they meet required behavioural requirements and service-level constraints. 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. First partially in [1] and then completed in [2], we present a graph transformation approach to formalize both kinds of changes in an uniform way. This approach is used to give a semantics of the Sensoria Reference Modeling Language (SRML) [3], but it could also be used in connection with other approaches for modeling or specifying service systems. The work reported in our two papers ([1] and [2]) develops such an operational semantics using graph transformation systems, a formalism for which several tools have been developed. 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 [5], 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 [4] and to express service-level constraints using the propositional fragment of that logic.

The running example of our two papers ([1] and [2]) is about a customer requesting a travel booking service which also requests two more services for flight and hotel booking. If we want to model a set of service companies providing and requiring services with the possibility to acquire one company by another, or partition a company attaching its parts to different companies, we need for example a topology with mobility similar to the ambient calculus which is a formalism

developed by Cardelli and Gordon [6]. The problem with this calculus is that it incorporates the orders to perform the transformations of the ambient topology inside the ambient expressions in forms of capability lists. In order to integrate our graph transformation approach for *SOC* with a graph semantics of the ambient calculus we have to remove capability lists and develop parameterised productions with labels and put them together in graph transformation units. Then graph transformation units would be one additional possible transformation of an ambient topology with service repositories and business configurations. You can have a look at [1] and [2] for a formal definition of these concepts. It is out of the scope of the paper to give an integration of the whole picture and therefore we just present a graph semantics of a variant of the ambient calculus as we explained, and incorporating service repositories and business configurations as just one black box associated to each ambient. These black boxes will be referred as business repositories. Additionally, we also remove the restriction operator and add a new visibility policy for ambients. Therefore, the relevance of this paper is not the very intuitive encoding of our variant of the ambient calculus but the variant itself. The modifications that we propose are necessary to make an application to our previous work on *SOC*. Although we appreciate all the work done on graphical encodings on the original ambient calculus, we have not been able to apply it to our *SOC* framework. Nevertheless, we relate our work with some work done on our area.

The representation of process calculi in terms of some form of graph transformation over different kinds of graphical structures has shown to be very successful for the study and analysis of these calculi (see e.g., [10,11,12,8,7,9,13,16,15,14]). In the specific case of the Ambient calculus, Gadducci and Montanari [11], on one hand, and Ferrari, Montanari and Tuosto [12], on the other, present a semantic definition of the Ambient calculus using graph transformation. In the former case, the ambient expressions are encoded in terms of ranked term graphs with interfaces and transformations are defined using the double-pushout (DPO) approach. In the latter case, Ambient expressions are encoded as hypergraphs, defined in terms of syntactic judgments, and transformations are defined as synchronized hyperedge replacements. Unfortunately, none of the two approaches described above is fully adequate for our purposes. The problem with the approach presented in [11] is the encoding of the Ambient expressions. In particular, in our view, ambient expressions should describe transformations of the given ambient (hierarchical) structure where the components are located. In this sense, we consider that the encoding of an Ambient expression should embed faithfully the ambient structure underlying that expression, and the transformations defined by the reduction of the expression should modify that structure accordingly. Unfortunately, in [11] the encoding of Ambient expressions does not satisfy these aims. In the lines of this work, there is a graphical implementation for finite processes of the original mobile ambient calculus in [16]. Additionally, in [15] and [14] they develop label transition systems (LTS) also for the original ambient calculus. In the case of the approach presented in [12] the situation is different. Ambient expressions are encoded according to our aims. In this case, the problem is related with the kind of transformations considered. In particular, we consider that synchronized hyperedge replacements do not enjoy the simple algebraic formulation and properties of double pushouts. In particular, we fear that using this kind of approach, our framework would be more involved. As a consequence, in this paper we reuse the approach which is based on this paper [13]. In our formalism, ambient expressions are encoded in terms of typed labeled graphs (with labels on the nodes) that, according to our aims, embed the ambient structure underlying the expression. Then, transformations are defined using the DPO approach. The original encoding is fully abstract and adequate and we use the basic ideas of the paper to develop a fully abstract and adequate encoding of our new variant. The encoded reduction relation in this paper is sound and complete.

The structure of the paper is as follows. In section 2 we present the variant of our ambient calculus and in section 3 we present typed label graph transformation systems. Then in section 4 we present the graph semantics of the ambient topology and in section 5 the graph transformation rule. Finally, in section 6 we give some examples and in section 7 we raise some conclusions and future work.

2 A Variant of the Ambient Calculus

The ambient calculus is a formalism developed by Cardelli and Gordon [6] for describing process mobility. Intuitively, ambients are the locations where the processes or the computation live and are hierarchically organized. The ambient topology can vary over time, having the possibility to move an ambient inside another ambient, to move an ambient out of another ambient and to dissolve or to open an ambient. These topological changes are performed by actions or capabilities associated to a given ambient where the action has to express the ambient to move in, the ambient to move out or the ambient to open. Additionally, in the calculus there exists a name restriction operator to restrict the topological space in which an action or capability with the restricted name can take place.

In the variant of the ambient calculus which we propose, we remove the capability lists associated to ambients but we can execute externally sequences of reduction rules with names and parameter names. For example if we had an ambient expression with ambients with names $an1$ and $an8$, a possible sequence of reduction rules would be $open\ an1; in\ an1\ an8$.These sequences of rules transform the ambient topology if all the rules can be applied correctly, and if any rule cannot be applied, they do not transform the ambient topology. Additionally, we do not have either the restriction operator. Instead each ambient n can have a set of ambient names.This set of ambients are the the ambients which can access ambient n. Alternatively an ambient n can be public to all the ambients. In this case we use the reserved name pub. Visibility restrictions can be considered. For example, we could consider that an ambient n can have a set of names which must be embedded in the same ambient and they must be at the same level of the ambient hierarchy as n.

Finally each ambient n has a name brn which represents a black box which contains a service repository with a set of services which provides n and some processors to request and use the different services which can access the ambient n in the ambient topology, and therefore perform service oriented programming. In [1] and [2] these concepts are referred as service repositories and business configurations.

The ambient calculus expressions are defined by the following grammar:

$$\begin{array}{l} P ::= P|Q \ \mid \ 0 \ \mid \ n[V; brn; P] \\ V ::= \{\} \mid vn.V \end{array}$$

In this definition n, m, brn and vn ranges over names, P and Q over processes and V over visibility sets. In the following, we will use the operator $\cup$ for the usual union of sets, and we will also use $-$ for the usual difference on sets.

The definition of the structural congruence between expressions of the calculus is the following:

$$\begin{array}{l} (1)\ P \equiv P \\ (2)\ P \equiv Q \Rightarrow Q \equiv P \\ (3)\ P \equiv Q,\ Q \equiv R \Rightarrow P \equiv R \\ (4)\ P \equiv Q \Rightarrow P \,|\, R \equiv Q \,|\, R \\ (5)\ P \equiv Q \Rightarrow n[V; brn; P] \equiv n[V; brn; Q] \\ (6)\ P|Q \equiv Q|P \\ (7)(P|Q)|R \equiv P|(Q|R) \\ (8)\ P|0 \equiv P \end{array}$$

The operational semantics of the ambient calculus consists first of the following five rules with names and parameters:

$$\begin{array}{l} (1) in\ n\ m\ n[V; brn; P] \,|\, m[W; brm; Q] \ \rightarrow m[W; brm; n[V; brn; P] \,|\, Q] \\ (2) out\ n\ m[W; brm; n[V; brn; P] \,|\, Q] \ \rightarrow \ n[V; brn; P] \,|\, m[W; brm; Q] \\ (3) open\ m\ n[V; brn; P \mid m[W; brm; Q]] \ \rightarrow \ n[V \cup W; brn; P \mid Q] \\ (4) addv\ vn\ n\ n[V; brn; P] \ \rightarrow \ n[V \cup \{vn\}; brn; P] \\ (5) rmv\ vn\ n\ n[V; brn; P] \ \rightarrow \ n[V - \{vn\}; brn; P] \end{array}$$

In these rules we have name parameters which can be used in the ambient expression. Therefore, the ambient expression depends on the names previously defined. The first three rules, like the original ambient calculus, are to move in an ambient n into m, to move out an ambient n from the ambient which embeds n, and to dissolve an ambient n in the ambient which embeds n. The main difference of these rule is in the *open* rule which after dissolution of ambient n, the ambient which embeds n is enriched with the business repository and the visibility set of n. The last two rules allows to add and remove a visibility name to a given ambient.

In order to define three more rules we have to define first sequences of rules which are a sequence of rules names with all the parameters except the ambient expression. We gave an example below with the rules *in*, *out* and *open* in this order. Sequences or rules can be applied to ambient expressions and they are applied from right to left in a compositional way. If some rules of the sequence can not be applied the result of the application is the initial ambient expression. If rule sequences ranges over RS, then the remaining rules are the following:

$$(6) RS\ P \rightarrow Q \Rightarrow n[V; brn; P] \rightarrow n[V; brn; Q]$$
$$(7)\ RS\ P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$$
$$(8)\ P' \equiv P,\ RS\ P \rightarrow Q,\ Q \equiv Q' \Rightarrow P' \rightarrow Q'$$

3 Typed labeled graph transformation systems

Ambient calculus expressions are going to be encoded in terms of typed labeled graphs, which are similar to the typed attributed graphs presented in [17], with the main difference that we do not have a Σ-algebra with a set of operations to define the data attributes, but a sorted set to define different sets of labels. In addition, in our graphs only nodes can have labels, i.e. we do not have labels on the edges. In particular, the intuition is that an attributed graph (in our case a labeled graph) is just a standard graph, where attributes (labels) are a special kind of nodes and where we also have a special kind of edges to bind a label to a (regular) node.

Definition 1. *A labeled graph* $AG = (V_1, V_2, E_1, E_2, (source_i, target_i)_{i=1,2})$ *consists of*

- *the set* V_1 *called graph nodes.*
- *the sorted set* V_2 *called label nodes.*
- *the sets* E_1,E_2 *called graph edges and node label edges, respectively.*
- *source functions* $source_1 : E_1 \rightarrow V_1$, $source_2 : E_2 \rightarrow V_1$.
- *and target functions* $target_1 : E_1 \rightarrow V_1$, $target_2 : E_2 \rightarrow V_2$.

Remark: *We denote by* $AG_{V1}, AG_{V2}, AG_{E1}, AG_{E2}, AG_{source_1}, AG_{source_2}, AG_{target_1}, AG_{target_2}$, *the different components of the labeled graph AG.*

Now we present labeled graph morphisms. Since in our case labeled graph morphism must preserve the values of the labels, the function f_{V2} presented in [17] is the identity.

Definition 2. *A labeled graph morphism* $f : AG1 \rightarrow AG2$ *is a tuple* $(f_{V1}, f_{V2}, f_{E1}, f_{E2})$, *with* $f_{V1} : AG1_{V1} \rightarrow AG2_{V1}$, $f_{V2} : AG1_{V2} \rightarrow AG2_{V2}$, $f_{E1} : AG1_{E1} \rightarrow AG2_{E1}$ *and* $f_{E2} : AG1_{E2} \rightarrow AG2_{E2}$, *such that f commutes with the* $source_1$, $source_2$, $target_1$ *and* $target_2$ *functions, and such that* f_{V2} *is the identity.*

As usual, typed graphs are defined as (standard) morphisms from a given graph into a type graph.

Definition 3. *A typed labeled graph (AG,t) over a type graph ATG consist of a labeled graph (AG) together with a standard graph morphism* $(t : AG \rightarrow ATG)$.

A typed labeled graph morphism $f : (AG_1, t_1) \rightarrow (AG_2, t_2)$ *is a labeled graph morphism* $f : AG_1 \rightarrow AG_2$ *such that* $t_2 \circ f = t_1$.

Typed labeled graphs together with typed labeled graph morphisms form the category of typed labeled graphs.

Our transformation rules are slightly more general than the standard rules for the double pushout approach. In particular, for technical reasons related with our encoding, the morphism $r : K \to R$ going from the context to the righ-hand side of a rule does not need to be a monomorphism. Additionally we will add label parameters.

Definition 4. *A production with labels p consists of a set of labels SL, a typed labeled graph L with all the labels in SL, two more typed label graphs K and R together with a monomorphism $l : K \to L$ and an arbitrary morphism $r : K \to R$. The production p is represented as $p\ l_1 \ldots l_n : L \leftarrow K \to R$ where l_i are the labels in SL.*

A transformation system of typed labeled graphs $GTS = (ATG, AG, P)$ consists of a labeled type graph ATG, a typed labeled graph AG, and a set of productions with labels.

A direct transformation $G \Rightarrow H$ via a left-linear production $p\ l_1 \ldots l_n :: L \leftarrow K \to R$ with a set of instantiation labels $il_1 \ldots il_n$, and a match m which additionally matches every l_i in L with its correspondent instantiation label il_i in G is defined by the double pushout diagram of figure 1.

Given a transformation system $GTS = (ATG, AG_0, P)$ typed labeled graph derivation is a sequence $AG_0 \Rightarrow \ldots \Rightarrow AG_n$ of direct transformations, written $AG_0 \Rightarrow_{*,GTS} AG_n$.

Graph transformation units encapsulates a set of rules with a control unit. Graph transformation units have a transactional semantics in the sense that in order to produce a transformation all the rules of the transformation unit taking part in the transformation must be applied successfully. If some rule cannot be applied successfully no transformation is performed The control unit specifies the order with which the rules must be applied. In our case the language of the control unit consists only of a compositional sequential operator of the form $np1; \ldots ; npn$ where eah name production has its set of instantiation names.

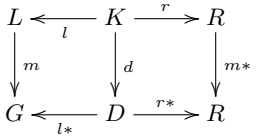

Fig. 1. Double pushout diagram

4 A graph semantics of our variant of the ambient calculus

In this section we present our encoding of our variant of the ambient calculus in terms of typed label graphs. First we describe the type graph.

In addition to the nodes and edges to represent labels, we have four types of graph nodes: nodes to denote ambients with a name label, nodes to denote interfaces between ambient nodes, nodes to denote ambient visibility with name labels and nodes to associate business repositories. Concerning the edges we also have three types: edges to define the hierarchy of ambients, edges to associate visibility nodes to ambient nodes and edges to associate a business repository node to each interface node.

Intuitively, our encoding includes a node (and the corresponding attribute) for each ambient in the expression. Moreover, if an ambient a_1 is inside the ambient a_2 then we have an edge from the node associated to a_1 to an interface node and another edge from that interface node to the node associated to a_2 (we need these interface nodes for technical reasons). That is, the graph associated to an expression can be considered to embed the topology of the ambients involved

in the expression. In addition, we have a visibility node associated to a given ambient and this visibility node can have a set of visible ambients as attributes. Finally, we have to include business repositories. It is out of the scope of the paper to give a representation of the business repository, and therefore we represent it as a black box node.

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

An ambient graph has as labels $Names$ where a name denotes the name of an ambient. We have additionally a distinguished name *pub* to denote that the business repositories of an ambient is public to all ambients embedded in the same ambient and at the same hierarchy level.

A symbolic graph of ambients SGA satisfies the following properties:

- for any pair nodes an_i and in_i there exists at most one edge between them.
- there are no cycles between nodes an_i and in_i.
- we have one visibility node $\mathbf{vn_i}$ associated to each ambient node with a set of ambient names as attributes. In particular, this name may be a normal name or the distinguished name *pub*.
- Every ambient node has one and only one ambient name and every ambient name can be targeted by different edges but it can appear just once in the graph.

An example of ambient graph is in Figure 2:

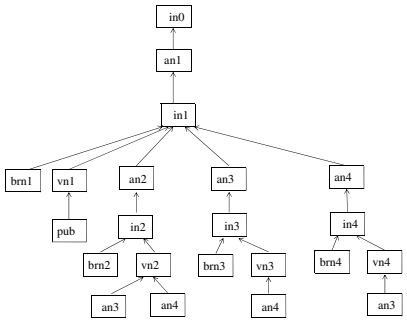

Fig. 2. An ambient graph

Because of reasons of figure space, we have not included ambient name attributes to the ambient nodes. We assume that they are implicit and they all have the same name of the ambient node an_i. Additionally we repeat visibility names in the ambient graph for readability but in the formal definition it is required that they have to appear just one in the whole ambient graph.

In this ambient graph we have just one top level ambient $an1$. This top level ambient has the business repository node $brn1$ and it is visible to all ambients of the ambient graph because its visibility node has the label *pub*. $an1$ has additionally three subambients $an2$, $an3$ and $an4$. There are no more ambients, and each of these three ambients have their business repository and the ambients which can access itself. For example, $an3$ and $an4$ can access $an2$.

In a similar and simpler way as in [13] we can define a semantic function $[\![]\!]$ which given a correct ambient expression returns a typed label ambient graph. In this variant of the calculus is very simple. The only important case which is different from [13] is the case $n[V; brn; P]$ which requires to differentiate two cases like in the original ambient calculus. If the name n has already appear in the encoding process in a visibility sequence we add an interface node in with an edge to the existing label n in the encoded graph so far. If the name n has not appeared yet we add a new interface node in with an edge to the new label n. The same happens with all the labels

of V. So additionally we have from the interface node *in* an edge to a visibility node with all the labels in V, an edge to a business repository node and an edge to an interface node *ins* which has the encoding of P. The formal definition is the following:

Definition 5. *The semantic function* $[\![\ ]\!]$ *which given a correct ambient expression returns a typed label ambient graph requires an auxiliary function with a graph* G *and a set of attribute nodes (names)* Γ *as parameters, and it returns a new graph, a set of names and a distinguished interface ambient node. It is inductively defined as follows:*

- $[\![P]\!] = [\![P]\!]^{(\emptyset,\emptyset)}$
- $[\![0]\!]^{G,\Gamma} = (G', \Gamma, iv)$ *where* G' *is* G *with an additional fresh interface graph node* iv.
- $[\![n[V; brn; P]]\!]^{G,\Gamma} = (G', \Gamma' \cup \{n\} \cup V, iv')$ *where* $(H, \Gamma', iv) = [\![P]\!]^{G,\Gamma}$. *The construction of* G' *has to add all the names of* $\{n\} \cup V$ *which are not in* H *and pend from* iv' *a fresh ambient node with name* n*, from which pends the interface node* iv *which has the encoding of* P. *Additionally, from* iv' *pends a business repository nodes* brn *and a fresh visibility node* vn *with all the visibility names of* V.
- $[\![P|Q]\!]^{G,\Gamma} = (G2', \Gamma 2, iv2)$ *where* $(G1, \Gamma 1, iv1) = [\![P]\!]^{G,\Gamma}$ *and* $(G2, \Gamma 2, iv2)[\![Q]\!]^{G1,\Gamma}$ *and the construction of* $G2'$ *is just* $G2$ *where the distinguished nodes* $iv1$ *and* $iv2$ *are identified as* $iv2$.

The associated ambient expression of the ambient graph of Figure 2 is the following:

$$an1[\{pub\}, brn1, an2[\{a3, an4\}, brn2, 0] \mid an3[\{an4\}, brn3, 0] \mid an4[\{a3\}, brn4, 0]]$$

Since the encoding of the expression 0 is an interface node, we should have put additional interface nodes inside $an2$, $an3$ and $an4$ in Figure 2 but we have omitted them because of presentation purposes.

Now it is easy to prove that our encoding is fully abstract with respect to the congruence relation:

Proposition 1. *If* $P \equiv Q$ *then* $[\![P]\!]$ *is isomorphic to* $[\![Q]\!]$

Actually, we can additionally prove that the encoding is adequate:

Theorem 1. *The semantic function that maps each congruence class of ambient expressions into its representation is injective and surjective.*

5 The transformation system

The transformation system will consist of five transformation rules on our variant of ambients: a rule to move one ambient inside another ambient, a rule to move out an ambient from another ambient, a rule to open an ambient, and to rules to add and delete visible ambient names associated to an ambient.

The rule to move in one ambient n inside another ambient m will be referred as **in** n m and it is defined in Figure 3.

The rule to move out ambient n from the ambient which embeds n will be referred as **out** n and it is defined in Figure 4. The rule **in** is the inverse of rule **out** and viceversa.

The rule to open an ambient n will be referred as **open** n and it is defined in Figure 5.

This rule is not easy to interpret and we give an explanation. The interface node $in2$ is identified with $in1$ and therefore it is added to $in1$ all what it was pending in $in2$. On the other hand $vn2$ is identified also with $vn1$, and therefore all its visibility labels of $vn2$ are moved to $vn1$.

The rule to remove the visibility name m from the visibility set of ambient n will be referred as **rmv** m n and it is defined in Figure 6.

Finally, the rule to add the visibility name m to the visibility set of ambient n will be referred as **addv** m n and it is defined in Figure 7.

Graph transformation units contains a sequence of different instantiation of the parameterised rules, and these rules are applied compositionally to the initial ambient expression. If all the rules

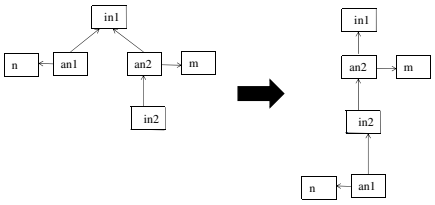

Fig. 3. The transformation rule **in** n m

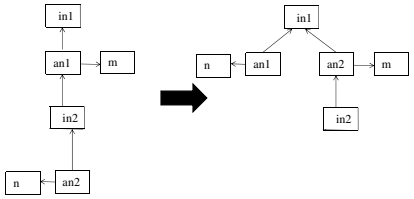

Fig. 4. The transformation rule **out** n

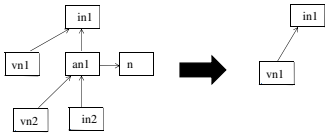

Fig. 5. The transformation rule **open** n

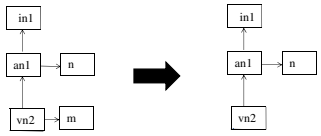

Fig. 6. The transformation rule **rmv** m n

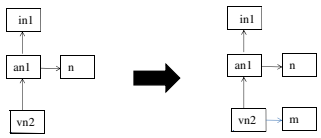

Fig. 7. The transformation rule **addv** m n

apply correctly we usually obtain a different ambient expression from the initial one, and if one rule does not apply correctly we obtain the initial ambient expression.

Now we present the concept of transformation system for our variant of the ambient calculus:

Definition 6. *A transformation system for business configurations consists of:*

- *an ambient expressions*
- *the five parameterised rules* **in**, **out**, **open**, **rmv**, **addv**
- *a graph transformation unit.*

Sequences of rules which apply to an ambient expression have the same syntax as graph transformation units. From now on, graph transformation units will range over *GTU*

It is not difficult to prove soundness and completeness of the reduction process via the encoding function of ambient expressions:

Theorem 2. *Given two ambient expressions P and Q, and the sequence of rules RS and the equivalent graph transformation unit GTU, RS $P \to Q$ if and only if GTU $[\![P]\!] \Rightarrow [\![Q]\!]$*

6 An example

In this example we give an ambient topology with visibility restrictions to access services in the internet. Services can have free access to the whole internet or can be accessed by a set of sites.

To show how the transformation system works, we extend the example of previous section in Figure 2 with three additional ambients at the top level: $an5$, $an8$ and $an9$. The extension is in Figure 8:

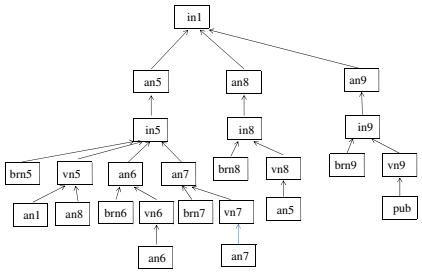

Fig. 8. An extension of the first ambient graph presented

So we have the initial ambient expression presented in two figures with top level ambients $an1$, $an5$, $an8$ and $an9$ and the graph transformation unit is the following sequence of rule applications:

$$in\ an4\ an9; out\ an4; open\ an1; rmv\ pub\ an1; in\ an1\ an8$$

After applying the first three rules of the sequences $open\ an1; rmv\ pub\ an1$; $an8$ acquires $an1$ removing its public access and dissolving it. The resulting top level ambient $an8$ is in Figure 9.

And the rest of the actual top level ambients $an5$ and $an9$ are now in Figure 10.

Finally, with the last two rules $an9$ acquires a partition of the dissolved $a1$ which is $a4$. The final ambient graph is in Figures 11 (top level ambients $an5$ and $an9$) and in Figure 12 (top level ambient $an8$).

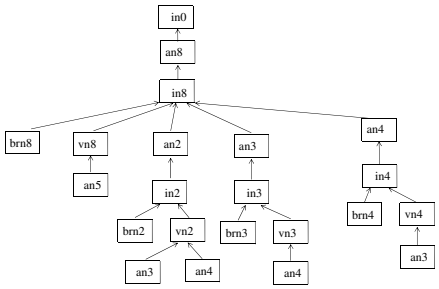

Fig. 9. The top level ambient $an8$ after removing a public visibility and dissolving $an1$

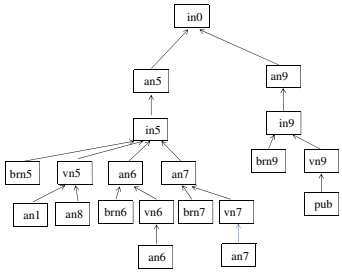

Fig. 10. The rest of the ambients after applying an *in* rule

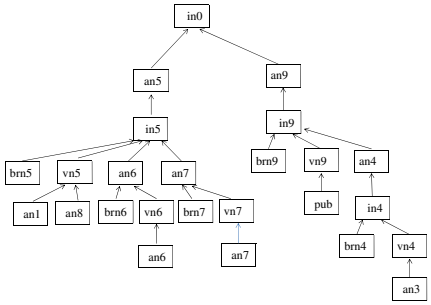

Fig. 11. Final ambient graph with just the top level ambients $an5$ and $an9$

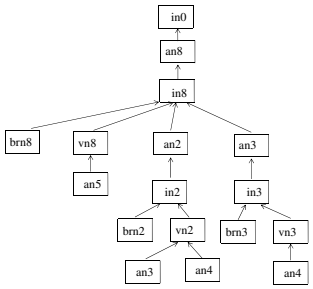

Fig. 12. Final ambient graph with just top level ambient $an8$

7 Conclusions and Future Work

In this paper we have presented a fully abstract and adequate graph semantics of a variant of the ambient calculus more adequate for an application to our model of SOC. Basically, we have removed the capability lists and the restriction operator of the ambient calculus, and we have added compositional application of parameterised rules, and a new visibility policy. We think that this would allow us to integrate our current work of SOC with the graph semantics of the variant of the ambient calculus developed in this paper. More precisely, we would develop a transformation system for an ambient topology with business configurations and service repositories. Then the transformations of the ambient topology would be one more of the possible transformation steps of the transformation steps. The other two possibilities were included in our previous work and they are also necessary in the integrated approach and they are the following:

- An application of a state transformation rule of an ambient to the current business activity of the same ambient. The result updates the business activity of the ambient.
- After a process of selection of a visible service by an ambient request, the binding of the chosen service to the current business configuration of the same ambient.

The most important goal of this work is starting with research and development of a real application on service oriented computing. Our work so far was independent of the middleware, to which the processes of service discovery, ranking and selection of services was delegated. From now on, I plan to do research and development to these delegated processes to the middleware, and work for example in recommender systems for services in the internet.

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). I would like to thank also Fernando Orejas for the idea of extracting capability lists from the ambient topology, and Hartmut Ehrig for hosting me during some months in Berlin during 2006, and for his help in our first graph semantics of the ambient calculus. RIP.

References

1. Nikos Mylonakis and Fernando Orejas and José Fiadeiro A semantics of Business Configurations Using Symbolic Graphs, IEEE International Conference on Services Computing (SCC 2015) New York (USA)

2. Nikos Mylonakis and Fernando Orejas and José Fiadeiro Modeling service-oriented computing with temporal symbolic graph transformation systems, Research Report 2015 and submitted to International Journal of Services Computing (IJSC)
3. José Luiz Fiadeiro and Antónia Lopes and Laura Bocchi and João Abreu, The Sensoria Reference Modeling Language, Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing, http://dx.doi.org/10.1007/978-3-642-20401-2_5,
4. Zohar Manna and Amir Pnueli, The temporal logic of reactive and concurrent systems - specification, Springer 1992
5. Fernando Orejas and Leen Lambers, Symbolic Attributed Graphs for Attributed Graph Transformation, Int. Coll. on Graph and Model Transformation. On the occasion of the 65th birthday of Hartmut Ehrig, Comm. of the EASST 2010
6. L. Cardelli and A. D. Gordon, Mobile Ambients, In Maurice Nivat, editor, Proc. FOSSACS'98, International Conference on Foundations of Software Science and Computation Structures, volume 1378 of Lecture Notes in Computer Science, pages 140–155. Springer-Verlag, 1998
7. H. Ehrig and B. Koenig, Deriving Bisimulation Congruences in the DPO Approach to Graph Rewriting, Foundations of Software Science and Computation Structures, FoSSaCS '04, LNCS 2987
8. S. Lack and P. Sobociński, Adhesive Categories, FOSSACS 2004, LNCS 2987
9. P. Baldan and A. Corradini and T. Heindel and B. Koenig and P. Sobociński, Processes for Adhesive Rewriting Systems, Foundations of Software Science and Computation Structures, FoSSaCS '06, lncs 3921, 2006
10. Ole Jensen and Robin Milner, Bigraphs and mobile processes, University of Cambridge, UCAM-CL-TR-57,
11. Fabio Gadducci and Ugo Montanari, A Concurrent Graph Semantics for Mobile Ambients, Electronic Notes of Theoretical Computer Science, 2001
12. Gian Luigi Ferrari and Ugo Montanari and Emilio Tuosto, A LTS Semantics of Ambients via Graph Synchronization with Mobility, ICTCS, 2001
13. Nikos Mylonakis and Fernando Orejas, Another fully abstract graph semantics for the ambient calculus, futur.upc.edu, presented at Graph Transformation for Verification and Concurrency(2007)
14. Filippo Bonchi and Fabio Gadducci and Giacoma Valentina Monreale, RPO semantics for mobile ambients, Mathematical Structures in Computer Science 2014
15. Julian Rathke and Pawel Sobocinski, Deriving Structural Labelled Transitions for Mobile Ambients, CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings,
16. Fabio Gadducci and Giacoma Valentina Monreale, A Decentralized Implementation of Mobile Ambients, Graph Transformations, 4th International Conference, ICGT 2008, Leicester, United Kingdom, September 7-13, 2008. Proceedings,
17. Hartmut Ehrig and Ulrike Prange and Gabriele Taentzer, Fundamental Theory for Typed Attributed Graph Transformation, ICGT 2004, LNCS 3256
18. H. Ehrig and K. Ehrig and U. Prange and G. Taentzer, Fundamentals of Algebraic Graph Transformation, EATCS Monographs of Theoretical Computer Science, Springer 2006