A Graph Semantics for a Variant of the Ambient Calculus More Adequate for Modeling Service Oriented Computing

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 has 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 [2] and then completed in [3], 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) [4], but it could also be used in connection with other approaches for modeling or specifying service systems.

The running example of our two papers ([2] and [3]) 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 [7]. 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 which are simpler than in [1], but ours have also a parallel operator allowing parallel execution similar to the ambient calculus. Then graph transformation units would be one additional possible transformation of an ambient topology with service repositories and business configurations. We strongly prefer not to have rule transformations inside our graph model of *SOC* and to have all rule transformations external to the graph model using the well known concept of transformation units. 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. It is not appropriate to include the name restriction operator because we can not use restricted names in graph transformation units because they are out of the scope of restricted names. To solve this loss of expressivity, we add a set of visibility names to ambients gaining in expressivity for our purposes. We belief that the restriction operator is not expressive enough to model visibility of services in the internet. For example, with our approach, we can model that an ambient n is visible only by a subambient of a brother of n. We do not know how to model it with the restriction operator. Our interests is an application of a variant of the ambient calculus for the design of a design language for SOC, which is not a programming language. Therefore, we are not interested in Turing completeness. Instead, we are interested in designing a calculus which can be integrated in our current graph transformation system for SOC . We leave open the problem to find an extension of our calculus which is Turing complete. To solve this problem, the new variant will probably not need visibility sets, which are certainly needed for the expressivity of our design framework. One way to prove this completeness result would be to adapt the work done by [20] which proves Turing completeness of a subfragment of the original ambient calculus without the restriction operator. Therefore, the relevance of this paper is not the encoding of our variant of the ambient calculus which the main ideas were developed in [14] but the variant itself, together with the incorporation of parameterised rules in graph transformation, to obtain soundness and completeness of the reduction processes. 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., [11,12,13,9,8,10,14,17,16,15]). In the specific case of the Ambient calculus, Gadducci and Montanari [12], on one hand, and Ferrari, Montanari and Tuosto [13], 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 [12] 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 [12] 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 [17]. Additionally, in [16] and [15] they develop label transition systems (LTS) also for the original ambient calculus. In the case of the approach presented in [13] 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 [14]. 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 [7] 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. The syntax of the calculus is the following:

$$\begin{array}{l} P ::= \ P|Q \ \mid \ 0 \ \mid \ n[P] \ \mid \ M.n[P] \ \mid \ M.0 \ \mid \ \nu n.P \\ M ::= \ \ \mathit{in}\, n \ \mid \ \mathit{out}\, n \ \mid \ \mathit{open} \ \ n \ \mid \ M.M' \end{array}$$

In this definition n ranges over names, P and Q over processes and M over capability lists. An example of an expression in the original calculus is the following:

$$n1[\mathit{in}\ n2.0] \mid n2[\mathit{open}\ n1.n3[0]]$$

The operational semantics of the calculus allows us to reduce the expression moving ambient $n1$ inside n and then open $n1$. The final expression after this reduction process is $n2[n3[0]]$ As we mentioned in the introduction, in graph transformation systems we have transformation units [1] which include a command language external to the graph which is going to be transformed by a set of rules. In the variant of the ambient calculus which we propose, we incorporate the idea of transformation units and we remove the capability lists associated to ambients. Therefore, we can execute externally sequences of reduction rules with names and parameter names. For example, the previous ambient expression of the original calculus is defined in our variant as follows:

$$n1[0] \mid n2[n3[0]]$$

And then we have the sequence of reduction rules $\mathit{open}\ n1; \mathit{in}\ n1\ n2$.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. After applying this sequence of rules the resulting ambient expression is $n2[n3[0]]$.

Once we have migrated capability lists to transformation units, the restriction operator can not be kept because restricted names can not be distinguished from normal names. Instead, we add a set of visibility names. Thus, each ambient n can have a set of ambient names.This set of ambients are 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.

Before comparing further our variant with the original calculus, we present the syntax of the variant:

$$P ::= P|Q \ \mid \ 0 \ \mid \ n[V;P]$$

In this definition n ranges over names, P and Q over processes and V over ambient name sets. Sets of names will be denoted as $\{n_1, \ldots, n_k\}$ where each n_i is a name. 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.

In the original calculus, visibility of ambient names can be restricted to a subexpression, and therefore a restricted ambient name is different from the same ambient name outside of the scope. For example, in the following ambient expression:

$$\nu n2.(n1[\mathit{in}\ n2.0] \mid n2[\mathit{open}\ n1.n3[0]]) \mid n2[0]$$

$n1$ can move inside the first ambient $n2$ starting from the left, but it can not move inside the second one $n2[0]$ because it is a different name from the restricted one. This restriction operator has some applications for security protocols with secret keys, but we do not how to apply it to model visibility of service sites in *SOC*. As a first example, consider the following expression:

$$an1[0] \mid an2[an3[0]]$$

We do not know how to express in the original calculus that only $an3$ has access to $an1$. If we use the restriction operator we have to include both. In our calculus we can easily express this with the expression:

$$an1[\{an3\}; 0] \mid; an2[\{pub\}; an3[\{pub\}; 0]$$

This expression states that only $an3$ has access to $an1$ and $an2$ and $an3$ are public. In general, we want to express that an ambient n has access to an arbitrary set of ambient nodes of the ambient expression. To express this, the name n must be included in all the visibility sets of the ambients to which it has access. As a final example, consider part of the ambient expression associated to the ambient graph of Section 6:

$$an1[\{pub\}; an2[\{an3, an4\}; 0] \mid an3[\{an4\}; 0] \mid an4[\{an3\}; 0]$$

In this ambient expression $an2$ can access $an1$, $an3$ can access $an2$ and $an4$, and $an4$ can access $an2$ and $an3$. Thus, in this variant, if ambients are considered as service sites, the service discovery algorithm which given an ambient name n and an ambient expression returns all the service sites to which n has access, can be performed easy by a traversal to the ambient nodes of the ambient expression.

Now we define the structural congruence between expressions of our calculus:

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

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

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

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 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 above with the rules *in* and *open* . Sequences of rules can be applied to ambient expressions and they are applied from right to left in a compositional way. Additional to the sequantial operator, we can also have a parallel operator on sequences of rules $rs1 \,|\, ... \,|\, rsn$. It allows to have several sequences applying in parallel. If some rules of the sequence can not be applied the result of the application is the initial ambient expression. If two different sequentializations of the parallel operator yield to different results, then the result of the application is also the initial ambient expression. In section 6, we have an example of different sequentializations of a parallel operator, and how to build the tree of all possible sequentializations of the given example. If rule sequences ranges over RS, then the remaining rules are the following:

$$\begin{array}{l}(6) RS\ P \rightarrow Q \Rightarrow n[V;P] \rightarrow n[V;Q]\\ (7)\, RS\, P \rightarrow Q \Rightarrow P \,|\, R \rightarrow Q \,|\, R\\ (8)\, P' \equiv P,\ RS\ P \rightarrow Q,\ Q \equiv Q' \Rightarrow P' \rightarrow Q'\end{array}$$

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 [18], 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 [18] 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 $r : K \rightarrow R$ going from the context to the right-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 $l : K \rightarrow L$ and an arbitrary morphism $r : K \rightarrow R$. The production p is represented as $p\ ll_1\ \dots\ ll_n$: $L \leftarrow K \rightarrow R$ where ll_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.

To perform a direct transformation $G \Rightarrow H$ via a left-linear production $p\ ll_1\ \dots\ ll_n :: L \leftarrow K \rightarrow R$ with a set of instantiation labels $il_1\ \dots\ il_n$ first we have to obtain the production with no labels $p' : L' \leftarrow K' \rightarrow R'$ by substituting every ll_i by its associated il_i. We also have to define the obvious induced morphisms l' and r' by l and r and the substitution. Then with a match m, the direct transformation is defined by the double pushout diagram of figure 1.

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

As an example of parameterised rule application, we consider the first of five parameterised rule which define the transformation system of ambient graphs. This first rule *in n m* has two parameters which are ambient names. In the first example of a transformation system defined in Section 6 this rule is applied with instantiation names $an1$ and $an8$. This application is correct because the target graph G has these ambient names. In order to compute the double pushout we first have to transform the parameterised rule *in n m* into a non-parameterised rule replacing every appearance of n and m in L, K and R by $an1$ and $an8$ respectively, obtaining L', K' and R' and its associated new morphisms l' and r'. In the following, we will omit the intermediate graph K in the rules. Now with this non-parameterised rule we can compute the double pushout with no problems.

Graph transformation units encapsulate a set of rules with a control unit. Graph transformation units have a transactional semantics in the sense that in order to produce a transformation all the rules of the transformation unit taking part in the transformation must be applied successfully. If some rule cannot be applied successfully no transformation is performed. The control unit specifies the order with which the rules must be applied. In our case the language of the control unit consists of a compositional sequential operator of the form $np1; \dots ; npn$ where each name production has its set of instantiation names. It has also a parallel operator of sequences of rules rsi of the form $rs1 \mid \dots \mid rsn$. For this parallel operator, if all possible sequentializations of the parallel operator yield the same result the application is correct. If not, the application returns the initial graph. In section 5, we give an example of some sequentializations of an use of the parallel operator.

$$\begin{tikzcd}
L' \arrow[d, "m"] & K' \arrow[l, "l'"'] \arrow[r, "r'"] \arrow[d, "d"] & R' \arrow[d, "m*"] \\
G & D \arrow[l, "l*"] \arrow[r, "r*"] & R
\end{tikzcd}$$

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 three types of graph nodes: nodes to denote ambients with a name label, nodes to denote interfaces between ambient nodes and nodes to denote ambient visibility with name labels. Concerning the edges we also have two types: edges to define the hierarchy of ambients and edges to associate visibility nodes to ambient nodes.

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.

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 $\mathbf{vn_i}$. 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.

An ambient graph *AG* satisfies the following properties:

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

As a first example, we give the representation of the following ambient expression:

$$n[\{m\};0] \mid m[\{n\};0] \mid w[\{pub\};0]]$$

The associated ambient graph is in Figure 12.

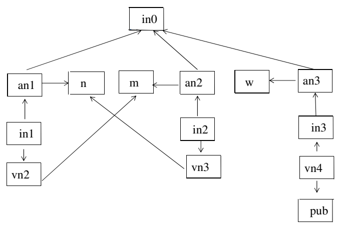

Fig. 2. A second example of the encoding of an ambient graph

In this ambient graph we have three ambients with name n, m and w. Ambients n and m can access each other and w is public.

In a similar and simpler way as in [14] 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 [14] is the case $n[V;P]$ 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 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 and requires an auxiliary function. This new semantic function, given a graph G and a set of attribute nodes (names) Γ as parameters, 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.*
- $[\![n[V;P]]\!]^{G,\Gamma} = (G', \Gamma' \cup \{n\} \cup V, iv')$ *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 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 $iv2$.*

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. This representation is unique up to isomorphism.*

5 The transformation system

The transformation system consists 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 two rules to add and delete visible ambient names associated to an ambient. Because of simplicity, these rules are presented without the intermediate graph K of the double pushout approach.

The rule to move in one ambient n inside another ambient m is 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 is 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 is 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 $vn1$.

Finally, the rule to add the visibility name m to the visibility set of ambient n is referred as **addv** m n and it is defined in Figure 6, the rule to remove the visibility name m from the visibility set of ambient n is referred as **rmv** m n and it is defined in Figure 7.

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

Definition 6. *A transformation system for an ambient graph consists of:*

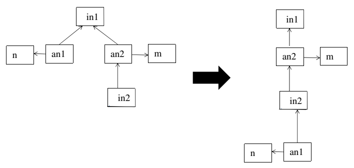

Fig. 3. Rule **in** n m .

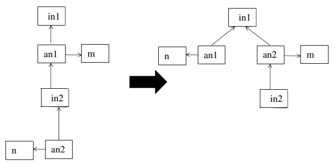

Fig. 4. Rule **out** n.

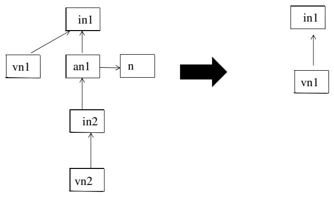

Fig. 5. The transformation rule **open** n

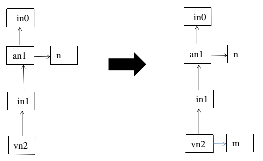

Fig. 6. Rule **addv** m n .

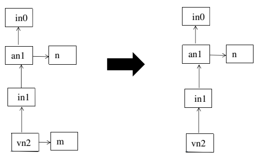

Fig. 7. Rule **rmv** m n.

– *an initial ambient graph*
– *the five parameterised rules with names* **in**, **out**, **open**, **rmv**, **addv**
– *a graph transformation unit.*

In next section, we give two examples of transformation systems with the same initial ambient graph and two different graph transformation units: the first one just sequential and then another one with the parallel operator.

Sequences of rules and parallel sequence 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 Two examples of transformation systems

In this example we present a transition system for an ambient graph with visibility restrictions, and one can think that each ambient inhabit a set of services. Services can have free access to the whole internet or can be accessed by a set of sites. The initial ambient graph is in two Figures and it is built identifying the top interface node $in0$. The two Figures are 8 and 9.

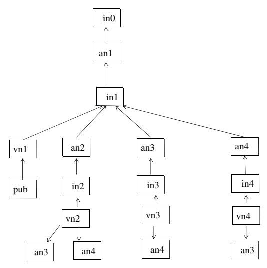

Fig. 8. An ambient graph.

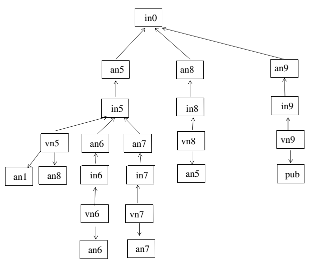

Fig. 9. An extension of the ambient graph on the left.

Because of reasons of figure space, in the following 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 once in the whole ambient graph.

In this ambient graph in two figures we have the top level ambients $an1, an5, an8$ and $an9$. In the following we just explain ambient $an1$ which is in Figure 8. This top level ambient $an1$ 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 subambients of $an1$, and each of these three ambients have the ambients which can access itself. For example, $an3$ and $an4$ can access $an2$.

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:

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

Now we explain how the transformation system works. After applying the first three rules of the sequences *open* $an1$; *rmv pub* $an1$; *in* $an1$ $an8$, $an8$ acquires $an1$ removing its public access and dissolving it. The resulting top level ambient $an8$ is in Figure 10. And the rest of the actual top level ambients $an5$ and $an9$ are now in Figure 11.

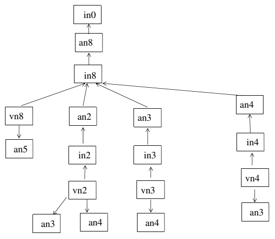

Fig. 10. The top level ambient $an8$ after removing a public visibility and dissolving $an1$

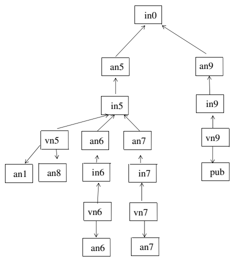

Fig. 11. The rest of the ambients after applying an *in* rule

Finally, with the last two rules $an9$ acquires a partition of the dissolved $a1$ which is $a4$. The final ambient graph is in Figures 12 (top level ambients $an5$ and $an9$) and in Figure 13 (top level ambient $an8$).

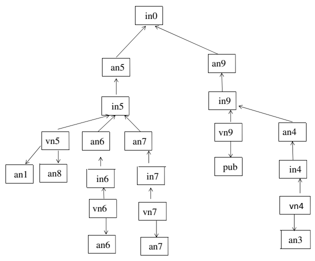

Fig. 12. Final ambient graph with just the top level ambients $an5$ and $an9$

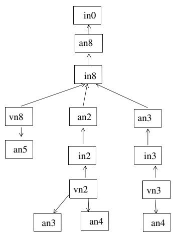

Fig. 13. Final ambient graph with just top level ambient $an8$

As a second example of transformation system, we consider the same initial graph of Figures 8 and 9. The graph transformation unit is three sequences in parallel and it is the following:

$$in\ an3\ an2\ ;\ out\ an4 \mid inan8\ an9 \mid open\ an5\ ;\ in\ an6\ an7$$

These parallel sequences of rules can be transformed in many possible sequences which have all the rules of the unit. The application is correct because all permutations give the same result because of lack of interferences. One possible way to find out all the possible sequential permutations is develop a tree of depth the total number of rules of the transformation unit (in our case 5), and each level of the tree has as nodes the rules that can be applied at each step. In our case, the first level has three possibilities: *in* $an6$ $an7$, *in* $an8$ $an9$ and *out* $an4$. At the second level *in* $an6$ $an7$ and *out* $an4$ have again three choices but *in* $an8$ $an9$ has only two choices. Following this way with the development of the permutational tree we obtain 31 possible permutations. Five possible permutations are the following:

$$\mathit{in}\ an3\ an2\ ;\ \mathit{out}\ an4\ ;\ \mathit{in}\ an8\ an9\ ;\ \mathit{open}\ an5\ ;\ \mathit{in}\ an6\ an7$$

$$\mathit{in}\ an3\ an2\ ;\ \mathit{in}\ an8\ an9\ ;\ \mathit{open}\ an5\ ;\ \mathit{in}\ an6\ an7;\ \mathit{out}\ an4$$

$$\mathit{in}\ an8\ an9\ ;\ \mathit{in}\ an3\ an2\ ;\mathit{open}\ an5\ ;\ \mathit{in}\ an6\ an7;\ \mathit{out}\ an4$$

$$\mathit{in}\ an3\ an2\ ;\mathit{open}\ an5\ ;\ \mathit{in}\ an8\ an9\ ;\ \mathit{in}\ an6\ an7;\ \mathit{out}\ an4$$

$$\mathit{in}\ an3\ an2\ ;\mathit{open}\ an5\ ;\ \mathit{in}\ an6\ an7;\ \mathit{out}\ an4\ ;\ \mathit{in}\ an8\ an9$$

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 than the original one for an application to our model of *SOC*. Basically, we have removed the capability of the ambient calculus, and we have added sequential application of parameterised rules with the possibility to parallelise sequences of rules. We have also added 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 work reported in our two papers ([2] and [3]) 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 [6], 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 [5] and to express service-level constraints using the propositional fragment of that logic. Therefore, we would have to integrate this previous work with the work presented in this paper. To achieve this, further research on temporal symbolic graph will have to be done. We also leave open the problem of extending our variant to make it Turing complete. The important design decision in the calculus is removing capability lists from ambient expressions. Visibility sets will probably not be needed to achieve Turing completeness, but they are certainly needed for the expressivity of our design framework. In [20] one can find a proof of Turing completeness of a subfragment of the original ambient calculus without the restriction operator.

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. Hans-Jörg Kreowski and Sabine Kuske Graph Transformation Units with Interleaving Semantics, Formal Asp. Comput., Vol. 11, 1999
2. 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)
3. Nikos Mylonakis and Fernando Orejas and José Fiadeiro Modeling service-oriented computing with temporal symbolic graph transformation systems, Research Report 2015 `upcommons.upc.edu` and submitted to International Journal of Services Computing (IJSC)
4. 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`,
5. Zohar Manna and Amir Pnueli, The temporal logic of reactive and concurrent systems - specification, Springer 1992
6. 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
7. 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
8. 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
9. S. Lack and P. Sobociński, Adhesive Categories, FOSSACS 2004, LNCS 2987
10. 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
11. Ole Jensen and Robin Milner, Bigraphs and mobile processes, University of Cambridge, UCAM-CL-TR-57,
12. Fabio Gadducci and Ugo Montanari, A Concurrent Graph Semantics for Mobile Ambients, Electronic Notes of Theoretical Computer Science, 2001
13. Gian Luigi Ferrari and Ugo Montanari and Emilio Tuosto, A LTS Semantics of Ambients via Graph Synchronization with Mobility, ICTCS, 2001
14. 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)
15. Filippo Bonchi and Fabio Gadducci and Giacoma Valentina Monreale, RPO semantics for mobile ambients, Mathematical Structures in Computer Science 2014
16. 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,
17. 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,
18. Hartmut Ehrig and Ulrike Prange and Gabriele Taentzer, Fundamental Theory for Typed Attributed Graph Transformation, ICGT 2004, LNCS 3256
19. H. Ehrig and K. Ehrig and U. Prange and G. Taentzer, Fundamentals of Algebraic Graph Transformation, EATCS Monographs of Theoretical Computer Science, Springer 2006
20. Nadia Busi and Gianluigi Zavattaro, On the expressive power of movement and restriction in pure mobile ambients, Theor. Comput. Sci., 322,3, pages 477–515, 2004