Another fully abstract graph semantics for the ambient calculus

Nikos Mylonakis and Fernando Orejas

June 11, 2007

Abstract

The long-term aim of this work is the definition of a framework for the modelling and development of distributed mobile component-based applications. As a first step we provide a fully abstract graph semantics for the ambient calculus which is more appropriate for our purposes than the existing ones. In particular, in our encoding, the graph representing an ambient calculus expression embeds faithfully the ambient structure underlying the given expression.

1 Introduction

The overall goal of this work is to provide a framework for the modelling of distributed mobile component-based systems. Components would be units as the ones described in [7, 6, 5] whose distribution and mobility would be described using the Ambient Calculus [2]. The idea is that components would live in ambients and that they would move using the operations of that calculus. We think that this kind of framework would be useful for systems that may be reconfigured in runtime or for the description of agent systems, where agents are some kind of components moving on the Internet.

In [6, 5] we defined the semantics of component-based architectures by means of graph transformation. In this context, we considered that extending the approach of [6, 5] to include mobility would be simpler if the semantics of the Ambient calculus would also be expressed in terms of graph transformation.

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, 10, 9, 12, 4, 3, 1]). In the specific case of the Ambient calculus, Gadducci and Montanari [10], on one hand, and Ferrari, Montanari and Tuosto [9], 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 judgements, 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 [10] 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 acordingly. Unfortunately, in [10] the encoding of Ambient expressions does not satisfy these aims.

In the case of the approach presented in [9] 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 present a new graph semantics for the ambient calculus. In our semantics, Ambient expressions are encoded in terms of typed labeled graphs (with labels on the nodes) that, according to ou aims, embed the ambient structure underlying the expression. Then, transformations are defined using the DPO approach. The encoding is fully abstract and adequate.

The paper is organized as follows. In section two we introduce typed labelled graphs, similar to [8]. In the next section we present the original definition of the ambient calculus [2] and our encoding in terms of typed labeled graphs. Then, in section 4 we give the transformation rules associated to the calculus. After that, we present a small example, and finally we present some conclusions.

2 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 [8], 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 an attribute to a (regular) node.

Definition 2.1 *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 $source_1 : E_1 \to V_1$, $source_2 : E_2 \to V_1$.*
- *and target functions $target_1 : E_1 \to V_1$, $target_2 : E_2 \to V_2$.*

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 [8] is the identity.

Definition 2.2 *A labeled graph morphism $f : AG1 \to AG2$ is a tuple $(f_{V1}, f_{V2}, f_{E1}, f_{E2})$, with $f_{V1} : AG1_{V1} \to AG2_{V1}$, $f_{V2} : AG1_{V2} \to AG2_{V2}$, $f_{E1} : AG1_{E1} \to AG2_{E1}$ and $f_{E2} : AG1_{E2} \to AG2_{E2}$, 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 2.3 *A typed labeled graph (AG,t) over a type graph ATG consist of a labeled graph (AG) together with a standard graph morphism ($t : AG \to ATG$).*

A typed labeled graph morphism $f : (AG_1, t_1) \to (AG_2, t_2)$ is a labeled graph morphism $f : AG_1 \to AG_2$ 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.

Definition 2.4 *A production p consists of three typed labeled graphs L, K and R together with a monomorphism $l : K \to L$ and an arbitrary morphism $r : K \to R$. As usual the production p is represented as $p : L \leftarrow K \to R$.*

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.

A direct transformation $G \Rightarrow H$ via a left-linear production $p : L \leftarrow K \to R$ and a match m 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 $AG_0 \Rightarrow_{*,GTS} AG_n$.

In our encoding we use negative application conditions to restrict the application of certain rules in certain circumstances.

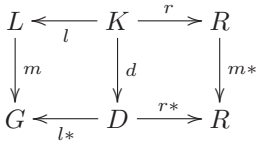


Figure 1: Double pushout diagram

Definition 2.5 *A negative application condition, or NAC for short, over a graph L is a finite set A of total morphisms* $\{c : L \to L_i\}_{i \in I}$ *called constraints*

A morphism $m : L \to G$ *satisfies a constraint* $c : L \to L_i$ *if there is no total morphism* $n : L_i \to G$ *such that* $n \circ c = m$*. A morphism* $m : L \to G$ *satisfies an application condition A over L if it satisfies all constraints in A.*

3 The ambient calculus and its representation as a typed labeled graph transformation system

3.1 The original ambient calculus

The ambient calculus is a formalism developed by Cardelli and Gordon [2] 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 ambient calculus expressions are defined by the following grammar:

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

In this definition n and m ranges over names, P and Q over processes and M and M' over capabilities, which are actions to make a move.

We have restricted the original definition of the ambient calculus in two senses. On one hand, we do not allow expressions of the form $M.(P|Q)$ because in our context these expressions make no sense. If we would allow them, some minor changes would be needed. On the other hand, we do not include the replicator operator ($!P$) either. The reason is that, to have a fully abstract representation of this operator, we would have to deal with infinite graphs.

Instead, we could include this operator in our semantic framework by encoding the following rule:

$$!P \rightarrow P \,|\, !P$$

The definition of the structural congruence between expressions of the calculus is the closure by α conversion of name restrictions of the rules in Fig. 2 and the operational semantics of the ambient calculus consists of the rules in Fig. 3:

$(1)\, P \equiv P$
$(2)\, P \equiv Q \Rightarrow Q \equiv P$
$(3)\, P \equiv Q,\ Q \equiv R \Rightarrow P \equiv R$
$(4)\, P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$
$(5)\, P \equiv Q \Rightarrow P \,|\, R \equiv Q \,|\, R$
$(6)\, P \equiv Q \Rightarrow !P \equiv !Q$
$(7)\, P \equiv Q \Rightarrow n[P] \equiv n[Q]$
$(8)\, P \equiv Q \Rightarrow M.P \equiv M.Q$
$(9)\, P|Q \equiv Q|P$
$(10)(P|Q)|R \equiv P|(Q|R)$
$(11)\, (\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$
$(12)\, (\nu n)(P|Q) \equiv P|(\nu n)Q \, if \, n \notin fn(P)$
$(13)\, (\nu n)(m[P]) \equiv m[(\nu n)P] \, if \, n \neq m$
$(14)\, P|0 \equiv P$
$(15)(\nu n)0 \equiv 0$

Figure 2: Structural equivalence for the Ambient Calculus

$(1)\, n[in\, m.P \,|\, Q] \,|\, m[R] \rightarrow m[n[P \,|\, Q] \,|\, R]$
$(2) m[n[out\, m.P \,|\, Q] \,|\, R]] \rightarrow n[P \,|\, Q] \,|\, m[R]$
$(3) open\, m.P \,|\, m[Q] \rightarrow P \,|\, Q$
$(4)\, P \rightarrow Q \Rightarrow (\nu n)P \rightarrow (\nu n)Q$
$(5)\, P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$
$(6)\ P \rightarrow Q \Rightarrow P \,|\, R \rightarrow Q \,|\, R$
$(7)\, P' \equiv P,\ P \rightarrow Q,\ Q \equiv Q' \Rightarrow P' \rightarrow Q'$

Figure 3: Operational semantics of the Ambient Calculus

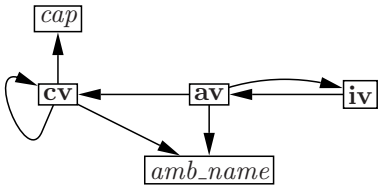
3.2 Ambient expressions as typed labels graphs

In this subsection we present our encoding of the ambient calculus in terms of typed labeled graphs and we prove that this encoding is fully abstract. First, we

give the definition of the type graph of ambients and then typed labeled graphs of ambients. In addition to the nodes and edges to represent the 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 capability units with a name label and a capability. Concerning the edges we also have three types: edges to define the hierarchy of ambients, edges to concatenate capability units and edges to associate capability units to ambient nodes.

Intuitively, our encoding includes a node (and the corresponding label) 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, if we have some capabilities associated to a given ambient then, on one hand, these capabilities are encoded as a list and, on the other hand, we have an edge from the corresponding ambient node to the first node in the list.

Definition 3.1 *The labeled type graph of ambients (LTGA) is defined in the following diagram:*



where the graph nodes are capability nodes (of type **cv***), ambient nodes (of type* **av***) and interfaces between ambient nodes (of type* **iv***), and where amb_name and cap are the graph nodes to represent the labels. In addition, we have edges between graph nodes of type* **av** *and* **iv** *whose type is denoted by ambient_edge, the type of the edge between graph nodes of type* **cv** *and* **av** *is denoted by amb_cap_edge and the type of the directed edges between graph nodes of type* **cv** *and itself is denoted by capability_edge.*

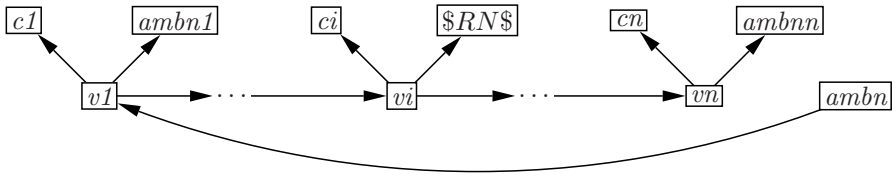
The rest of the edges are used to attach attributes to graph nodes.

Definition 3.2 *A labeled graph of ambients has the following data labels:*

- *Names where the name denotes the name of an ambient. Normal names do not have the $ symbol and we have additionally two distinguished names RN and NN to define restricted names and the null process.*
- *A capability set Cap with the capabilities in, out or open.*

Definition 3.3 *A typed labeled graph of ambients over LTGA consists of a labeled graph of ambients LGA and a graph morphism $t : LGA \to LTGA$ such that:*

- *for any pair nodes of V1 of type av or iv there exists at most one edge $e1$ of $E1$ such that the type $t(e1)$ is ambient_edge.*
- *there are no cycles on edges of type ambient_edge.*
- *the nodes of type* **cv***, which form sequences as the one in the figure below, have two kind of labels: a capability and an ambient name. In particular, this name may be a normal name or the distinguished name $\$RN\$$.*



- *Every ambient node has one and only one ambient name and every ambient name can be targeted by different edges $e2 \in E2$ but it can appear just once in the graph. Every capability node has one and only one capability and one and only one ambient name.*

Now, in what follows, ambient nodes will be denoted by $av, av0, av1, av2, \ldots$; capability nodes will be denoted by $cv, cv0, cv1, cv2, \ldots$ and interface nodes will be denoted by $iv, iv0, iv1, iv2, \ldots$

Now we will define the semantic function which, given an ambient expression *amb_expr*, returns its representation as a typed labeled graph. It requires an auxiliary function with two additional parameters: a graph G which is a graph already built and it has to be glued with the graph which represents *amb_expr* (sharing the common ambient names), and a set of ambient names Γ which are nodes of the graph G which might be targeted by arrows of the graph associated to *amb_expr*. We use the usual functions on sets $\cup$ (union) and $-$ (difference).

Before describing how this function is inductively defined, we give some examples of the result of this function, for some ambient expressions. First, in Fig. 4 we give the graph semantics of the expression

$$w[open\ k'.open\ k''.p[0]]]$$

In this case we just have a hierarchy of two graph nodes and the root node is av1 with ambient name w. In addition to this hierarchy we have the representation

of the list of capabilities *open* k'.*open* k''. For each capability we require a node with two labels: one for the kind of capability and one for the ambient name. Additionally the first node which represent a capability in the capability list is bound to its associated ambient node. In this case it is the ambient node av2. This is required for a correct representation of the in, out and open rule.

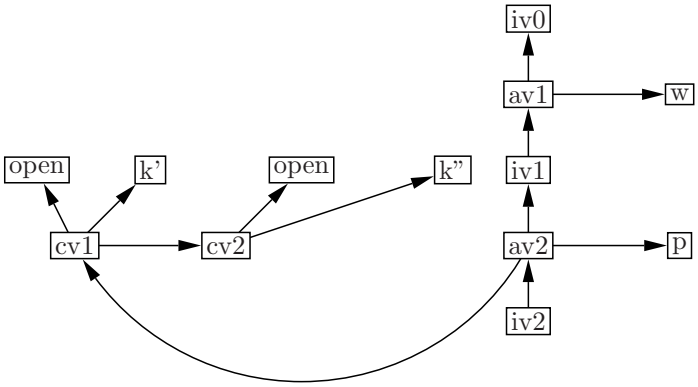


Figure 4: Representation of $w[open\ k'.open\ k''.p[0]]]$

Next, in Fig. 5 we give the representation of an ambient expression with a restricted name:

$$\nu w.w[k[0]]$$

Since w is a restricted name, the ambient node whose name is restricted, is labeled with $\$RN\$$ and not with w.

Finally, in Fig. 6 we represent an expression that is used in the example section of the paper. The ambient term has a subexpression with a restricted name which represents a firewall:

$$(\nu w)(w[k[out\, w.in\, k'.in\, w.0] \mid open\ k'.open\ k''.p[0]])$$

The term has also an agent in parallel which will enter the firewall:

$$k'[open\ k.k''[q[0]]]$$

In this case and throughout all the example we will represent the names k, k' and k'' several times in the graph for readability reasons. In the correct representation the names k and k'' must only appear once and be targeted twice, whereas the name k' must only appear once and be targeted three times. Note that the restricted name w is represented as $\$RN\$$ just once and it is targeted three times.

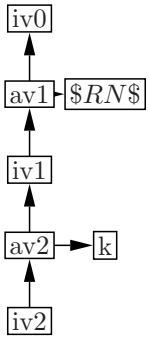


Figure 5: Representation of $\nu w.w[k[0]]$

Definition 3.4 *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.*
- $[\![n[P]]\!]^{G,\Gamma} = (G', \Gamma' \cup \{n\}, iv')$ *where $(H, \Gamma', iv) = [\![P]\!]^{G,\Gamma}$. The construction of G' requires to differentiate two cases: the case where $n \notin \Gamma'$ and the case where $n \in \Gamma'$. The case where $n \notin \Gamma'$ is represented in Fig. 7, and the case where $n \in \Gamma'$ is represented in Fig. 8.*
- $[\![M.n[P]]\!]^{G,\Gamma} = (I, \Gamma'', iv)$ *where* $(G', \Gamma', iv) = [\![n[P]]\!]^{G,\Gamma}$ *and* $(G'', \Gamma'', cvi, cvf) = [\![M]\!]^{G',\Gamma'}$. *The function $[\![M]\!]^{G,\Gamma}$, given a capability list, a graph and a set of name attributes of this graph returns a new graph, a new set of attribute nodes, and two capability nodes: the first and the last of the capability list. Each capability node will have two arrows: one to a capability attribute and the other one to a name attribute. We define inductively this function below. The representation of I is depicted in Fig. 9, where the subgraph H with the nodes iv, av, iv' and n and the edges among them is the encoding of G' and the subgraph CL with the nodes cvi, $capi$, ni, cvf, $capf$ and nf together with their associated edges among them is the encoding of M via the semantic function. Note that it could be the case that ni and nf were in Γ' and therefore they would be nodes in G'.*

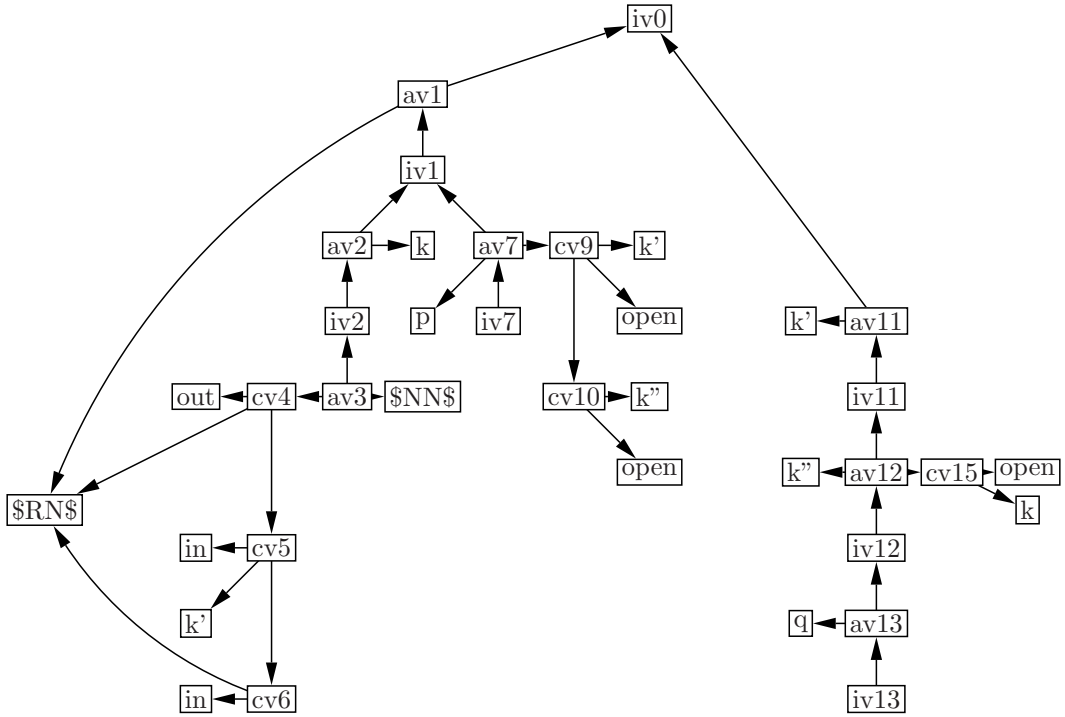


Figure 6: Representation of $(\nu w)(w[k[out\, w.in\, k'.in\, w.0] \mid open\, k'.open\, k''.p[0]])|k'[open\, k.k''[q[0]]]$

The function $[\![M]\!]^{G,\Gamma}$ *is inductively defined as follows:*

- $[\![cap\, n]\!]^{G,\Gamma} = (G', \Gamma \cup \{n\}, cv, cv)$*, where* cv *is in* G' *and its construction has two cases. If* $n \notin \Gamma$ *then the representation is depicted in Fig. 10. If* $n \in \Gamma$ *then the representation is depicted in Fig. 11.*
- $[\![M.M']\!]^{G,\Gamma} = (H, \Gamma'', cvi, cvf2)$ *where* $(G', \Gamma', cvi, cvf) = [\![M]\!]^{G,\Gamma}$ *and* $(G'', \Gamma'', cvi2, cvf2) = [\![M']\!]^{G',\Gamma'}$ *The construction of* H *is depicted in Fig. 12, where the subgraph* CL *with the nodes* cvi*,* $capi$*,* ni*,* cvf*,* $capf$ *and* nf *together with their associated edges among them is the encoding of* M *via the semantic function* $[\![M]\!]^{G,\Gamma}$ *and the subgraph* $CL2$ *with the nodes* $cvi2$*,* $capi2$*,* $ni2$*,* $cvf2$*,* $capf2$ *and* $nf2$ *together with their associated edges among them is the encoding of* M' *via the semantic function* $[\![M']\!]^{G',\Gamma'}$

- $[\![M.0]\!]^{G,\Gamma} = (I, \Gamma', iv)$ *where* $(G', \Gamma', cvi, cvf) = [\![M]\!]^{G,\Gamma}$*. In this case the*

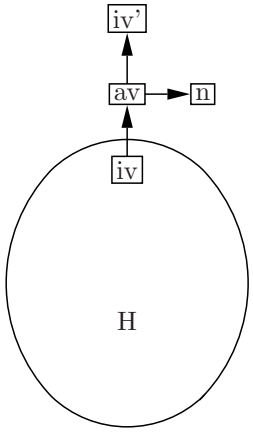


Figure 7: Adding an ambient whose name was not used previously

representation of I is depicted in Fig. 13, where the subgraph CL with the nodes cvi, $capi$, ni, cvf, $capf$ and nf together with their associated edges among them is the encoding of M via the semantic function $[\![M]\!]^{G,\Gamma}$

- $[\![(\nu n)P]\!]^{G,\Gamma} = (G'[\$RN\$/n], \Gamma' - \{n\}, iv)$ *where* $(G', \Gamma', iv) = [\![P]\!]^{G,\Gamma}$ *and in the resulting graph we replace in G' the attribute node with name n by $\$RN\$$.*
- $[\![P|Q]\!]^{G,\Gamma} = (G2', \Gamma 2, iv2)$ *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 $iv2$.*

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

Proposition 3.5 *If $P \equiv Q$ then $[\![P]\!]$ is isomorphic to $[\![Q]\!]$*

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

Theorem 3.6 *The semantic function that maps each congruence class of ambient expressions into its representation is injective and surjective.*

4 The representation of the reduction relation

The reduction relation will be defined as a set of typed labeled graph transformations. More precisely, the typed labeled graph transformation system is

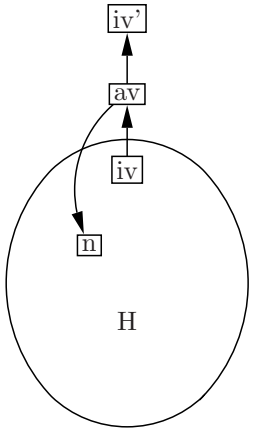


Figure 8: Adding an ambient whose name was used previously

defined as $AMBTS = (ATG, AMBTAG, REDP)$ where ATG is the labeled type graph of definition **3.1** and $AMBTAG$ a typed labeled ambient graph.

For the representation of the rules of the reduction relation we just have to encode the first three rules, because the rest hold in our graph transformation approach. For each rule of the ambient calculus we have three rules in the graph representation. Due to space limitations, we just give one of these three rules for the representation of the *in* rule and another rule for the representation of the *open* rule. In particular, the first graph transformation rule that encodes the in rule $(n[in\, m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R])$ is the following:

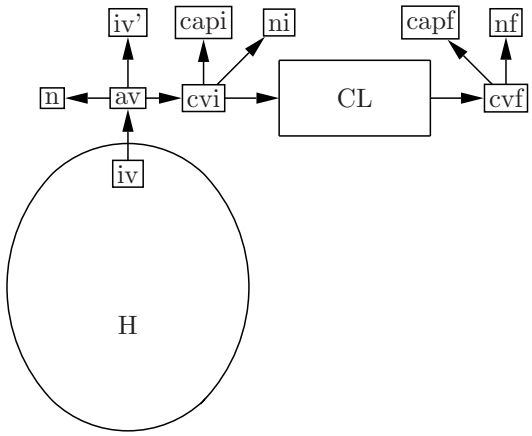
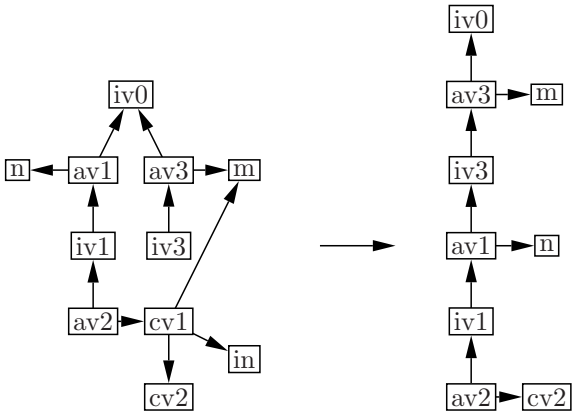


Figure 9: Capability list associated to the ambient n



This transformation rule encodes the case where the length of the capability list inside the ambient n is greater than one. In this case the ambient n is moved to ambient m and the in capability is removed from the list. The second case is when the length of the capability list is exactly one. We have an additional case when we have a capability list of length one associated to the null operator. In this case we have to remove the null node in the transformation. Note that these 3 cases work well for the case that n is a normal name and for the case

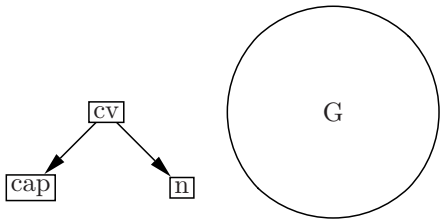


Figure 10: Adding a capability referring to an ambient not mentioned before

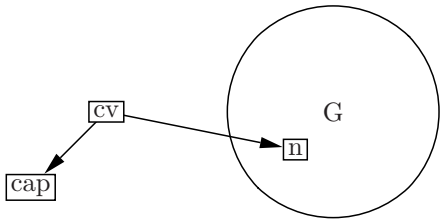


Figure 11: Adding a capability referring to an ambient already in the graph

that n is $\$RN\$$.

The rules to move an ambient with name m out of the ambient n are basically the reverse of the corresponding in rules.

In the case of the open rule, $(open\, m.P \mid m[Q] \rightarrow P \mid Q)$. we present the case where the capability list has length one and is associated to the null process. In this situation the encoding will be the rule below together with a NAC disallowing its application when the list of capabilities has length greater than one:

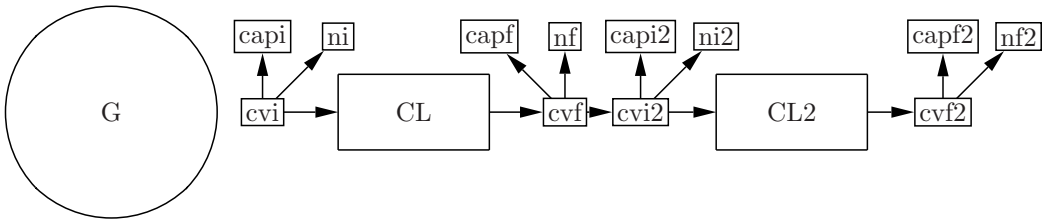


Figure 12: Concatenation of two capability lists

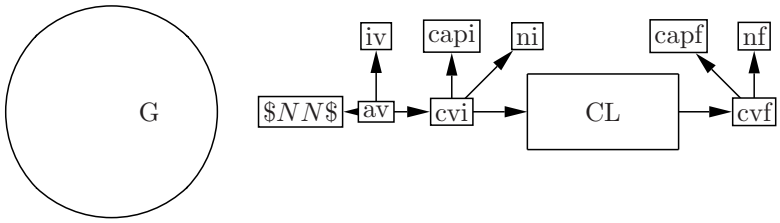
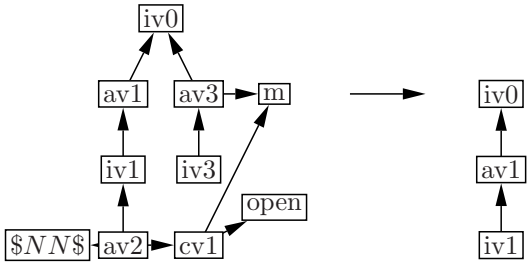
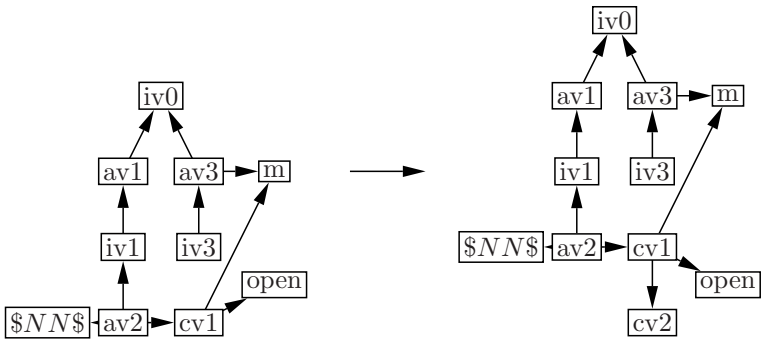


Figure 13: Capability list associated to 0



Its associated NAC is:



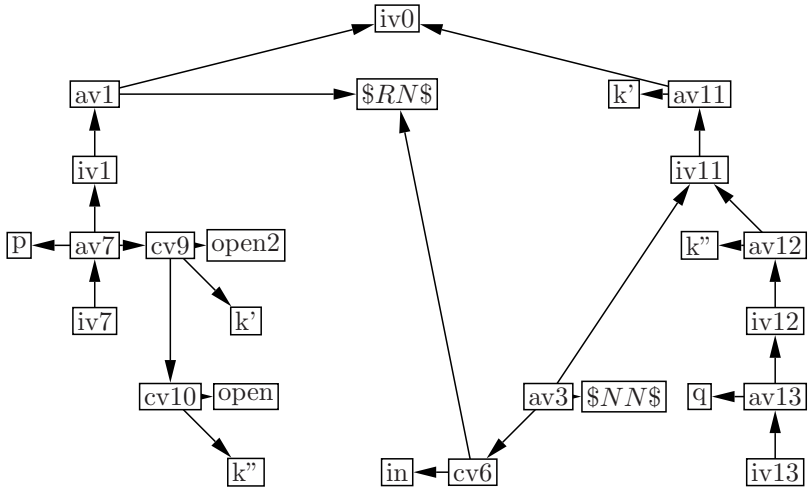
In this rule the interface node $iv3$ is not removed but identified with $iv0$. It is not difficult to prove the following theorem:

Theorem 4.1 *Given two ambient expressions P and Q, if $P \rightarrow Q$ then $[\![P]\!] \Rightarrow [\![Q]\!]$*

5 Example

As an example, we present a sequence of reductions so that the agent $k'[open\, k.k''[q[0]]]$ can enter the firewall $(\nu w)(w[k[out\, w.in\, k'.in\, w.0] \mid open\, k'.open\, k''.p[0]])$.

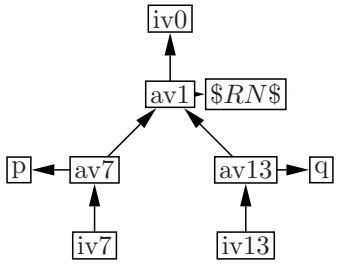
The representation of the expression is given in Fig. 6. After applying the transformations associated to an out and an in capabilities, respectively, and after applying the transformation associated to the rule to open ambient k, we obtain the following graph:



representing the expression:

$$(\nu w)(w[open\, k'.open\, k''.p[0]] \mid k'[in\, w.0 \mid k''[q[0]]])$$

Now, if we apply the transformation associated to the in rule to move the external agent inside the firewall and, finally, we apply twice transformations associated to the open rule we obtain the graph:



which is the encoding of the expression:

$$(\nu w)(w[p[0] \mid q[0]])$$

6 Conclusions and future work

As we mentioned in the introduction, the final aim of this work is the definition of a framework for the modelling of distributed mobile component-based systems.

In [13] we extended the generic approach to components presented in [7] to allow for the modelling of mobile systems. More precisely, we embedded the component approach from [7] into the ambient calculus [2]. However, the approach was too ad-hoc and, as a consequence, too involved.

Now we are approaching this task from a more systematic point of view. On one hand, in [6, 5] we have defined the semantics of component-based architectures in terms of graph transformation. On the other hand, in this paper we are also providing a graph transformation semantics for the ambient calculus. The future task will be the integration of the two approaches.

References

[1] P. Baldan, A. Corradini, T. Heindel, B. Koenig, and P. Sobociński. Processes for adhesive rewriting systems. In *Foundations of Software Science and Computation Structures, FoSSaCS '06*, volume 3921 of *lncs*, pages 202–216. Springer, 2006.

[2] L. Cardelli and A. D. Gordon. Mobile ambients. In *In Maurice Nivat, editor, Proc. FOSSACS'98, International Coference on Foundations of Software Science and Computation Structures, volume 1378 of Lecture Notes in Computer Science, pages 140–155. Springer-Verlag*, 1998.

[3] A. Corradini, F. Hermann, and P. Sobociński. Subobject transformation systems. *Applied Categorical Structures*, 2007. To appear.

[4] H. Ehrig and B. Koenig. Deriving bisimulation congruences in the DPO approach to graph rewriting. In *Foundations of Software Science and Computation Structures, FoSSaCS '04*, volume 2987 of *lncs*, pages 151–166. Springer, 2004.

[5] H. Ehrig, F. Orejas, J. Padberg, M. Klein, S. Perez, and E. Pino. A generic approach to connector architectures. Submitted, 2007.

[6] H. Ehrig, J. Padberg, B. Braatz, M. Klein, M. Piirainen, F. Orejas, S. Perez, and E. Pino. A generic freamework for connector architectures based on components and transformations. In *FESCA proceedings, Barcelona*, 2004.

[7] Hartmut Ehrig, Fernando Orejas, Benjamin Braatz, Markus Klein, and Martti Piirainen. A generic component framework for system modeling. In *FASE 2002 (LNCS 2306)*, 2002.

[8] Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamental theory for typed attributed graph transformation. In *ICGT*, number 3256 in LNCS, pages 161–177, 2004.

[9] Gian Luigi Ferrari, Ugo Montanari, and Emilio Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In *ICTCS*, pages 1–16, 2001.

[10] Fabio Gadducci and Ugo Montanar. A concurrent graph semantics for mobile ambients. *Electronic Notes of Theoretical Computer Science*, 45, 2001.

[11] Ole Jensen and Robin Milner. Bigraphs and mobile processes. Technical report, University of Cambridge, UCAM-CL-TR-57.

[12] S. Lack and P. Sobociński. Adhesive categories. In *FOSSACS*, number 2987 in LNCS, pages 273–288, 2004.

[13] N. Mylonakis and F. Orejas. A distributed and mobile component system based on the ambient calculus. In *Recent Trends in Algebraic Development Techniques*, 2004.