An abstract semantic framework for mobile component systems based on graph transformation (Extended abstract)

Nikos Mylonakis and Fernando Orejas

December 16, 2005

Abstract

The aim of this work is the definition of a generic framework for the modelling and development of component-based distributed applications with mobile processes in the internet. We describe component systems as typed attributed graphs, where nodes may represent localities, but also components, and where edges may represent links between localities, but also connections between components or bindings of components to localities. In this context, mobility is described in terms of graph transformation extending transformation rules to include (graph) variables.

1 Introduction

The aim of this work is the definition of a generic framework for the modelling and development of component-based distributed applications with mobile processes in the internet. These systems are in general heterogeneous, which means that they can be described or implemented using different languages. As a consequence, we believe that a generic approach, which is independent of any specific formalism, should provide the adequate setting for the study of this kind of systems. Using this framework, we show how the operations in the ambient calculus [1] can be represented as transformations.

In [6] we extended the generic approach to components presented in [3] to allow the modelling of mobile systems. More precisely, we embedded the component approach from [3] into the ambient calculus [1]. However, we found that the lack of symmetry in the way we dealt with components and mobility was not fully satisfactory. In particular, while components and component systems were approached in a very generic way we were using a very concrete formalism to describe mobility.

In order to avoid this shortcoming, in this work we describe component systems as typed attributed graphs, where nodes may represent localities (or ambients), but also components, and where edges may represent links between localities, but also connections between components or bindings of components
to localities. In this context, mobility is described in terms of graph transformation. In particular, we show how the operations in the ambient calculus can be represented as transformations in our framework. However, the standard (algebraic) theory of graph transformation ([4]) was insufficient to model these operations. We solved this problem by extending, in a simple way, the notion of transformation rule; our rules may include variables that, when applied to a graph \( G \), may be bound to a subgraph of \( G \).

The work presented in [8] is closely related to ours. In particular, they also represent component systems in terms of typed graphs and define graph transformations on them. However, the aim of these transformations is architecture reconfiguration and not the description of mobility and this makes both approaches quite different. In particular, on one hand, their graphs represent architectures where nodes are programs and arcs denote morphisms between programs structured by different connectors and, on the other, the transformations considered are very different.

Another closely related work, with a different aim, is the one by [5] on bigraphs. Instead of using typed attributed graphs with subgraph variables they use bigraphical reactive systems to encode different distributed and parallel calculus. For example, they encode a substantial fragment of the \( \pi \)-calculus and the encoding of the ambient calculus is not fully detailed. Basically, bigraphs are defined as two different graphs using the same set of nodes: link graphs and place graphs. Using the notion of controls with a certain arity and defined in a signature, link graphs are used to encode the communication channels of the \( \pi \)-calculus and place graphs are used to encode the acyclic ambient hierarchy of the ambient calculus.

The paper is organized as follows. In section two we introduce the graph transformation rules with variables needed in our approach. In the following section, we overview how we represent component systems in terms of graphs and how mobility can be represented by means of rules with variables. In particular, the rules describing some operations from the ambient calculus are presented here (the rest can be found in an appendix). Finally, in section 4, a small example is presented to show the applicability of the approach.

2 **Typed attributed graphs with graph variables and their transformation systems**

In this section we define graph transformation rules with graph variables and how they are applied to graphs. The intuitive idea is that a rule with variables is a kind of generator of standard graph transformation rules. In particular, when we want to apply a rule of that kind to a graph \( G \), first we have to bind each variable to a subgraph of \( G \). Then, replacing each variable by the corresponding subgraph can be seen as generating a standard rule.

Typed attributed graphs with graph variables are graphs as in [4] with, additionally, a set of graph variables and a relation binding (some) graph variables
to graph nodes. In particular, a graph variable may be not bound to any node or it may be bound to several nodes. If each variable is bound to at most one node then the graph will be called linear. The left-hand side of a rule will be assumed to be a linear graph.

**Definition 2.1** An attributed graph with graph variables \( G(X) \) is an attributed graph \( G = (V_1, V_2, E_1, E_2, E_3, (source_i, target_i)_{i=1,2,3}) \) with additionally

- a set \( X \) of graph variables.
- a relation \( rel_{\mathsf{gvar}} : X \times V_1 \) which can be partial in both arguments.

A linear attributed graph with graph variables is an attributed graph with graph variables where the relation \( rel_{\mathsf{gvar}} : X \times V_1 \) is a (partial) function from the set of graph variables to the graph nodes.

**Definition 2.2** A morphism of attributed graph with graph variables \( h : AG_1(X_1) \rightarrow AG_2(X_1) \) is a tuple \( (h_{V_1}, h_{E_1}) \) with \( h_{V_1} : AG_{1,V_1}(X_1) \rightarrow AG_{2,V_1}(X_1) \) and \( h_{E_1} : AG_{1,E_1}(X_1) \rightarrow AG_{2,E_1}(X_1) \) such that \( h \) commutes with the source and target functions and if \( rel_{\mathsf{gvar}}(z,v) \) is defined then \( rel_{\mathsf{gvar}}(x,h_{V_1}(v)) \) is also defined. Morphisms should preserve the values of the attributes. This is achieved by the following equations:

- \( target_2(v_1) = target_2(h_{V_1}(v_1)) \) for all \( v_1 \in V_1 \).
- \( target_3(e_1) = target_3(h_{E_1}(e_1)) \) for all \( e_1 \in E_1 \).

**Definition 2.3** A left-linear production \( p \) consists of two linear typed attributed graph with variables \( L(X_1) \) and \( K(X_1) \) and one typed attributed graph with variables \( R(X_1) \) with morphisms \( l : K(X_1) \rightarrow L(X_1) \) and \( r : K(X_1) \rightarrow R(X_1) \). As usual the production \( p \) is represented as \( p : L(X_1) \leftarrow K(X_1) \rightarrow R(X_1) \).

**Definition 2.4** A transformation system of typed attributed graphs with graph variables \( \mathsf{GTS} = (ATG, G, P) \) consists of an attributed type graph \( ATG \), a typed attributed graph \( G \), and a set of left-linear productions.

Now, to substitute a variable \( x \) in a graph \( G(X) \) by a graph \( G_x \) we need to know how to glue \( G_x \) to the rest of \( G(X) \). In order to do this we consider that a substitution associates to each variable \( x \) not only a graph \( G_x \), but also a distinguished vertex \( v_x \) in \( G_x \). Then, applying such a substitution to \( G(X) \) means gluing a copy of \( G_x \) to every vertex \( v \) bound to the variable \( x \), where the gluing identifies \( v \) with the corresponding copy of \( v_x \).

**Definition 2.5** A substitution of graph variables \( \sigma \) is a pair of functions \( (\sigma_{\mathsf{tg}}, \sigma_{\mathsf{da}}) \) where for every variable \( x \) of a set \( X \) \( \sigma_{\mathsf{tg}}(x) \) is a typed attributed graph without subgraph variables \( G_x \) and \( \sigma_{\mathsf{da}}(x) \) is a distinguished node of \( G_x \).
Definition 2.6 The application of a substitution of graph variables to a typed attributed graph with variables $G(X)$ is a typed attributed graph without graph variables $H$, also denoted as $G(X)$, where every occurrence of $x$ in $G(X)$ is replaced by a disjoint copy of $\sigma_{G}(x)$ and if $\text{rel}_{\text{sgvar}}(x, v)$ is defined $\sigma_{\text{tl}}(x)$ is replaced by $v$.

If $h : G_{1}(X_{1}) \rightarrow G_{2}(X_{1})$ is a morphism and $\sigma$ is a substitution of $X_{1}$ then we can define an unique morphism (up to natural isomorphism) $h \sigma : G_{1}(X_{1}) \rightarrow G_{2}(X_{2})$:

Proposition 2.7 If $h : G_{1}(X_{1}) \rightarrow G_{2}(X_{1})$ is a morphism between graphs with variables such that $G_{1}(X_{1})$ is linear, $\text{rel}_{\text{sgvar}}(x, v)$ is defined for every $x$ in $X_{1}$ or $G_{2}(X_{1})$ is also linear and $\sigma$ is a variable substitution of $X_{1}$, then there exists a unique morphism $h \sigma : G_{1}(X_{1}) \rightarrow G_{2}(X_{1})$ up to (natural) isomorphism satisfying:

- $\forall v \in V_{G_{1}(X_{1})} h \sigma(v) = h(v)$
- $\forall e \in E_{G_{1}(X_{1})} h \sigma(e) = h(e)$
- $\forall x \in X_{1} h \sigma(\sigma_{G}(x))$ is equal to any isomorphism to the disjoint copy of $\sigma_{G}(x)$ in $G_{2}(X_{1})$ where $\sigma_{G}(x)$ is identified with $h(v)$ in $G_{2}(X_{1})$ if $\text{rel}_{\text{sgvar}}(x, v)$ is defined in $G_{1}(X_{1})$. If for any $x \in X_{1}$ $\text{rel}_{\text{sgvar}}(x, v)$ is not defined then $h \sigma(\sigma_{G}(x))$ is equal to the unique disjoint copy of $\sigma_{G}(x)$ in $G_{2}(X_{1})$.

Note that in the case that we have in $X_{1}$ a variable $x$ such that $\text{rel}_{\text{sgvar}}(x, v)$ is not defined in $G_{1}(X_{1})$ and we have in $G_{2}(X_{1})$ that $\text{rel}_{\text{sgvar}}(x, v1)$ and $\text{rel}_{\text{sgvar}}(x, v2)$ then the morphism $h \sigma$ is not unique.

Definition 2.8 A match $m$ of a left linear production $p : L(X_{1}) \leftarrow K(X_{1}) \rightarrow R(X_{1})$ to a typed attributed graph without graph variables $G$ is defined by a pair of a substitution of graph variables $\sigma$ from $X_{1}$ and a typed attributed graph morphism $m_{\sigma} : L(X_{1}) \sigma \rightarrow G$ such that for every $x$ in $X_{1}, \sigma_{G}(x)$ and the image of $\sigma_{G}(x)$ by the morphism $m_{\sigma}$ are isomorphic.

Definition 2.9 A direct transformation $G \leftarrow D \rightarrow H$ via a left-linear production $p : L(X_{1}) \leftarrow K(X_{1}) \rightarrow R(X_{1})$ and a match $m$ is defined by the double pushout diagram of figure 1. As in normal graph transformation the match $m$ must satisfy an application condition called the gluing condition with two parts. To ensure that $D$ will have no dangling edges, the dangling condition requires that if $p$ specifies the deletion of a vertex of $G$, then it must specify also the deletion of all edges of $G$ incident to that node. The identification condition requires that every element of $G$ that should be deleted by the application of $p$ has only one pre-image in $L$. 

4
Now we may prove that for any left linear production and for any match \( m \) to a graph \( G \) there exists a unique graph \( R \) defined by the double pushout diagram of figure 1. This is true even for the case that the morphism \( l\sigma \) is not unique.

**Proposition 2.10** Given a left linear production \( p : L(X1) \leftrightarrow K(X1) \rightarrow R(X1) \) and a match \( m \) to a graph \( G \), the direct transformation \( G \leftrightarrow D \rightarrow R \) defined by the double pushout diagram of figure 1 is unique.

We use graph constraints to guarantee that every time a graph transformation is applied it satisfies all the conditions which must satisfy the typed attributed graph without variables defined in next section. There are negative and positive graph constraints but we just use the negative ones with the same definition as in [2]. We also use negative application conditions to restrict the application of certain rules in certain circumstances. We extend the usual definition of negative application conditions including negative application conditions which are applied in case a graph variable is instantiated. We also have a special condition which a variable has to satisfy when it is instantiated.

# 3 Interconnected forest of hierarchies and their transformations

In this section we give the main ideas of the instantiation of typed attributed graphs to define typed attributed graphs of interconnected forest of hierarchies. The instantiation of typed attributed graphs with graph variables to define typed attributed graphs of interconnected forest of hierarchies with graph variables is straightforward.

First we give the different types of nodes with their associated attributes that we have in typed attributed graph of interconnected forest of hierarchies:

- Ambient nodes with an attribute to denote the name of an ambient.
- Restriction nodes with an attribute to denote the restricted name.
- Replicator nodes to denote that the subforest associated to this kind of node can be replicated as many times as it is needed.
• Body component nodes, export component nodes and import component nodes with an attribute (Names, Specification) to define the body, export and import specification of the components associated to the ambient nodes.

The type of edges and their associated data attributes that we have in these graphs are the following:

• Edges to define the hierarchy of ambients with restriction and replication nodes. This hierarchy is acyclic.
• Edges to associate the body of components to ambient nodes.
• Edges to define the embeddings between a node which denotes the import of a component and a node which denotes the body of a component.
• Edges to define the transformation between a node which denotes the export of a component and a node which denotes the body of a component. There can also be transformations between a node which denotes an import of a component and a node which denotes an export of a component. The interconnections among component systems of the forest which we allow are from a component system to its immediate ancestor component system in the hierarchy. Additionally, we allow interconnections among component systems of the same hierarchy.

Additionally we present some graph transformation rules which are similar to the ambient calculus reduction rules and we explain the rest. In the appendix we present an important subset of these rules and we also present the graph constraints which are needed to guarantee that the graph transformation rules preserve all the conditions that interconnected forest of hierarchies have to satisfy.

The graphical notation that we use is the following:

• Ambient nodes are represented by its associated attribute name AMBN, AMBN1, ...
• Name restriction nodes are represented by pairs (RESTR, AMBN)
• IMPN, IMPN1,..., EXPN, EXPN1,..., BODN, BODN1 denote import specification, export specification and body specification respectively.
• A node denoted by REPL represents a node to replicate a typed attributed graph of interconnected forest of hierarchies.
• When the source or target node of an edge has no denotations, it represents three kind of nodes: an ambient node, a name restriction node or a node for replication.
• Graph variables which will denote interconnected forest of hierarchies are represented by triangles referenced by a letter (P,Q,R,...) with an edge to the associated node of the graph.

First we present the first two rules which are used to move an ambient node with name AMBN inside an ambient with name AMBN1 and to move an ambient with name AMBN out of the ambient AMBN1. Additionally, we present in the same figure a rule to open an ambient with name AMBN1. This rule will be applied if there are no components associated to the ambient AMBN1 and the subgraph variable P has been instantiated with the whole interconnected forest of hierarchies pending of AMBN1.

The representation of these rules is in figure 7.

![Figure 2: Rules to move in, move out and open an ambient](image)

We have also a set of rules to describe the autonomous behavior of component systems allowing to make new connections or make disconnections whenever is needed. These connections can be made within a given ambient node, among the neighbour component systems with a common parent ambient or from any given component associated to an ambient to another component associated to the parent ambient.

The next set of rules are to vary the scope of the restriction of use of new fresh names and the rule for replication. The basic idea underlying the rules for the restriction of names are that the scope of a restriction of a given name n can be extended and shrunk provided that the interconnected forest to be included or excluded does not contain an use of the given name n. In order to make sure that a given name does not appear in an interconnected forest of hierarchies, a constraint on a graph variable is needed. When we extend the restriction of a name the rule is not just the inverse of the rule to shrink the restriction
of a name. The case where the root of the interconnected forest of hierarchies is an ambient node with the same name as the restricted name has a different negative application condition. The rest of the cases is the inverse of the rule to shrink a name restriction with the same negative application condition. We also have a rule for the replication operator of the ambient calculus. This rule make possible an indefinite number of replications of a given interconnected forest of hierarchies. It has three negative application conditions to guarantee that the replication which is made is of the whole interconnected forest which is pending of the replicator operator. Additionally we have to guarantee that the variable P satisfies the condition $\text{Clos}(P)$ in order to make a copy of the whole interconnected forest of hierarchies pending of Repl.

4 Example

In this last section we present the same example as in [6] but now using a visual graphical notation instead of ambient calculus expressions. The example describes a server with a firewall together with two clients trying to obtain a software component.

The formalism which we use for the specification of components are algebra transformation systems as in [7]. Thus, specifications denote a class of computations where we have states which are represented by $\Sigma$-algebras and computation steps are partial functions from $\Sigma$-algebras to $\Sigma$-algebras.

One of the clients is represented in a complete way in figure 3.

![Figure 3: The interconnected forest of hierarchies of CLIENT1](image)

But we will use the visual abstract notation of figure 4 to represent the CLIENT represented above and the SERVER working in parallel.

One of the interconnections of the forest of hierarchies is from the import algebra transformation system of a component in the server agent to an export
algebra transformation system of a component in the SERVER. The import and export algebra transformation systems contain an operation to assign resources to agents.

CLIENT can access the SERVER and obtain the software component. In the following, we will see how the protocol works.

In the first sequence of transformations, a server agent with a component system enters the CLIENT. Now we analyze the dynamic reconfiguration of the interconnected forest of hierarchies. The hierarchy of SERVER has not got the server agent anymore, and now it is in the CLIENT hierarchy. Additionally, the component system of the server agent looses his connection with the global component system of the SERVER, and when entering the CLIENT, it establishes a new connection with the global component system of the CLIENT. This new connection will allow the agent of the server to gain resources in the client, which are shared with the resources which use the client agent.

An incomplete graphical representation of the resulting CLIENT ambient is in figure 5.

In the following sequence of reductions, the server agent enters the client agent generating a new agent with the capabilities of both.

Next, the new generated agent enters the server. Making this move, the component system of the new agent looses the connection with the component system of the CLIENT, and when entering the server, it establishes a connection with the global component system of the SERVER. This connection will allow the new agent to gain resources in the server.

Finally, the new generated agent which is now in the SERVER takes the component denoted in the figures by PROD and moves it to the CLIENT.
Figure 5: The client with the agent of the server

References


A An important subset of the transformation rules
Figure 6: Graph constraints
Figure 7: Rules to move in, move out and open an ambient
Figure 8: Rules to make and delete connections and to delete components
Figure 9: Rules for the restriction of names and replication