View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Scientific Publications of the University of Toulouse II Le Mirail

Generic approach for graph-based description of dynamically reconfigurable architectures

Cédric Eichler, Ismael Bouassida Rodriguez, Thierry Monteil, Patricia Stolf, Khalil Drira

To cite this version:

Cédric Eichler, Ismael Bouassida Rodriguez, Thierry Monteil, Patricia Stolf, Khalil Drira. Generic approach for graph-based description of dynamically reconfigurable architectures. Rapport LAAS n 12071. 2012. <hal-00674092>

HAL Id: hal-00674092

https://hal.archives-ouvertes.fr/hal-00674092

Submitted on 24 Feb 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Under consideration for publication in Math. Struct. in Comp. Science

Generic approach for graph-based description of dynamically reconfigurable architectures.†

CÉDRIC EICHLER[1,2,3], ISMAEL BOUASSIDA RODRIGUEZ, THIERRY MONTEIL[1,3]
PATRICIA STOLF[2,3], KHALIL DRIRA[1,3]

[1] *CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France*

[2] *IRIT; 118 Route de Narbonne, F-31062 Toulouse, France*

[3] *Univ de Toulouse, UPS, INSA, F-31400, UTM, F-31100 Toulouse, France*

Received

Architectural adaptation is studied for handling adaptation in autonomic distributed systems. It is achieved by implementing a model-based approach for managing reconfiguration of dynamic architectures. Describing such architectures includes defining rules for describing both architectural styles and theirs reconfiguration mechanisms. Within this research context, the work presented in this paper is conducted using formal specification based on graphs and graph rewriting appropriately for tackling architectural adaptation problems. A graph-based general approach for describing architectures and handling their dynamic reconfiguration is introduced. Our approach is illustrated in the context of a distributed hierarchical application. The formal models that allow the generation of a graph grammar for dynamic architecture description and the automatic definition of transformation rules for achieving intern self-protecting during the adaptation are elaborated.

1. Introduction

The description of evolving architectures cannot be limited to the specification of a unique static topology but must cover the scope of all the correct configurations. We develop, in this paper, the concept of characterization of architectural styles to achieve this goal. We elaborate and specify the architectural style for the design of applications. For this purpose we develop an appropriate formal framework using graph grammars. Our approach enables generating architectures in conformance with a given style.

Suitable description languages and formalisms for avoiding ambiguities are necessary for correct architectural design, management and analysis. Many architecture description languages were introduced providing rigorous syntax and semantic to define

† This paper has been produced in the context of SOP project ANR-11-INFR-001

architectural entities and relations. Several researches focuses on Architecture Description Languages (ADLs). p-Method (Oquendo, 2006), Rapid (Luckham et al., 1995), Wright (Allen and Garlan, 1997), and ACME (Garlan et al., 2000) provide modelling tools that help the designers to structure a system and to compose its elements. Often, ADLs allow to describe predefined dynamics. That is, they are interested in systems having a finite number of configurations known in advance. Few of them (Allen and Garlan, 1997; Garlan et al., 2000; Oquendo, 2006) allow various architectural styles to be distinguished. ADLs can be classified as language-oriented works, whereas this paper introduces a model-oriented approach which provides a more abstract view of a software architecture. ADLs rather offer an architecture view which is closer to the implementation. Additionally, ADLs allow to design architectural styles from the scratch, whereas what is proposed in this paper is a correct by design formal approach based on the pattern composition. It should be stressed that this article do not propose an alternative to ADLs approaches. The models presented in this paper can be integrated into p-Method (p-ADL) or into other ADLs. Functional languages have also been proposed. They introduce abstract notations allowing to describe dynamic software architectures in terms of properties. Semantic ADLs using ontologies (Zhou et al., 2007) and ADLs (Architecture Description Languages) uses XML deployment languages in (Dashofy et al., 2002). ADLs can be proprietary or implementing the formal and the semantic architecture description models. These ADLs are used to guarantee the architecture evolving and correctness during the different predictable and unpredictable changes in the systems environment. C2SADL (Taylor et al., 1995) is an architecture description language that allows the definition of architectural styles. A style is defined by declaring its component and connector types. C2SADL is based on a generic style called C2.

According to a past study (Kacem et al., 2005), we noted that ADLs (Medvidovic and Taylor, 2000) suffer from several insufficiencies for modeling and analyzing software architectures. We underlined that the majority of ADLs are concentrated on the structural description of architectures whereas the dynamic aspect of architecture is not well supported.

In ((de Paula et al., 2000), the authors developed a formal framework, specified in Z, to describe the dynamic configuration of software architectures. They did not address the design phase.

Designing and describing software models using UML (OMG, 2005) is a common practice in the software industry. UML descriptions of software architecture not only provide a standardized definition of system structure and terminology, but also facilitate a more consistent and broader understanding of the architecture (Selonen and Xu, 2003).

To specify the software architectural change and the architectural styles it is possible to use the formal approaches (Bradbury et al., 2004). The multi-Formalismes approaches (Loulou et al., 2004), (Kandé and Strohmeier, 2000) seek to combine different

formalisms with UML notations in order to describe the software architecture. They try to define relations between ADLs and UML (Roh et al., 2004), (Medvidovic et al., 2002) focusing on mapping the concepts of the former into the visual notation of the latter. Others, seek to combine UML notations and graph transformation in order to specify architectural change (Heckel et al., 2004). However, these researches do not offer a simple notation and metamodels for easily specifying architecture changes.

Other works (Hirsch et al., 1999; Le Métayer, 1998) are based on graph grammar techniques. Graph grammars consist in using graphs for representing software architectures. They are appropriate for formal modelling dynamic structures and software architectures. In this context, (Le Métayer, 1998) describes the software architectural style using a context-free graph grammar and verifies the conformity of an architecture to its style. Authors in (Chassot et al., 2006) present another model based method using graph grammars to adapt cooperative information systems to situation changes at the communication level.

2. Background: basic definitions

In the following, key concepts related to graph transformations will be presented. The definition of a homomorphism requires the definition of elements unification. To achieve such an unification, each attributes of the elements have to be unifiable. This produces a set of identification that has to be consistent.

2.1. *Graph*

The object manipulated in this paper are directed graph with multi-labelled vertices and multi-tagged edges. Each label or tag might be either constant or variable.

Definition 1. (**Graph with constant and variable multi-labelled nodes and edges**) A graph is defined by the system G = (V, E, Lab, Tag)

(i) V and E correspond to the set of vertices and edges of the graph,
(ii) |V| (resp. |E|) are the cardinality of V (resp. E),
(iii) Lab (resp. Tag) is a set of |V| (resp. |E|) sets Lab^v (resp. Tag^e). Lab^v (resp. Tag^e) represents the labels of the vertex v and their domain of definition (resp. the tags of the edge e), $|\text{Lab}^v|$ is the number of labels for the node v (resp. $|\text{Tag}^e|$ the number of tags for edge e),
(iv) Lab^v_i (resp. Tag^e_i) represents the i-th label of the vertex v (resp. the i-th tag of the edge e) and can be a constant or a variable,
(v) Dlab^v_i (resp. Dtag^e_i) is the set of possible values of Lab^v_i (resp. Tag^e_i): $\text{Lab}^v_i \in \text{Dlab}^v_i$ (resp. $\text{Tag}^e_i \in \text{Dtag}^e_i$),
(vi) Lab^v (resp. Tag^e) is the set of couples (Lab^v_i, Dlab^v_i) (resp. (Tag^e_i, Dtag^e_i)).

Convention 1.

(i) An attribute is a global term designing either a label or a tag.

(ii) To distinguish between constant and variable attributes, a constant attribute will be noted within quotation marks.

Convention 2. In order to lighten the notations, we adopt the following convention :

(i) A vertex v may be described as v(Lab^v).

(ii) An edge from v to v' may be noted $v \xrightarrow{Tag^{(v,v')}} v'$.

(iii) A graph G may be described by (V, E) where V and E respectively correspond to the set of its vertices and edges as described above.

Example 1. A graph G = (V, E, Lab, Tag) where V = {v_1, v_2}, Lab^{v_1} = {("1",ℕ), (x,ℕ), (z,ℕ)}, Lab^{v_2} = {(w,ℕ)}, E = {e_1=(v_1,v_2)} and Tag^{e_1} = {(a,{a,b})} can be described as G = (V, E) where E = {v_1(("1",ℕ),(x,ℕ), (z,ℕ)), v_2((w,ℕ))} and V = { $v_1 \xrightarrow{("a",\{a,b\})} v_2$}. Such a graph can be graphically represented as shown in the figure 1.

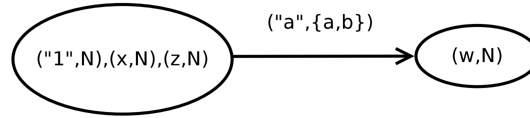

Figure 1. An example of graph

2.2. *Graph morphism*

Definition 2. (**Joker**) Two kinds of "jokers" attribute or super-variable are defined:

— v(*) is a vertex representing any vertex.

— $\xrightarrow{*}$ is an edge representing any edge.

Definition 3. (**Set of identification**) A set of identification I is a set of couple of attributes. I is said to be basic if ∀ (a_i, a_j) ∈ I, a_i is constant ∨ a_j is constant.

Definition 4. (**Attributes unification**) Two attributes a_i and a_j with the domains of definition Da_i and Da_j are unifiable if and only if

(i) they have the same type : $Da_i = Da_j$,

(ii) if they are both constant, then they have the same value.

If the attributes are unifiable and not both constant then the result of this unification is the set of identification {(a_i, a_j)} else it is empty.

Example 2. x and y with the domain of definition ℕ are unifiable and the result of this unification is {(x, y)}.

Definition 5. (**Consistent set of identification**) If I is a basic set of identification I is consistent if ∀ ((x_1, "$value_1$"),(x_2, "$value_2$")) ∈ I^2 one of the two following conditions is verified :

— x_1 and x_2 are two different variables,

— x_1 and x_2 correspond to the same variable and $value_1$ = $value_2$.

If I is a set of identification and I is not basic, I is consistent if for any couple (x,y) in I where x and y are variable I' = $\{ (a_i, a_j) \mid (a_i, a_j) \in I \wedge a_i \neq y \wedge a_j \neq y \} \cup \{ (x,a_i) \mid ((y,a_i) \in I \vee (a_i,y) \in I) \wedge a_i \neq x \}$ is consistent.

Example 3. The set of identification I = {(y, "1"), (x, y), (z, y) } is consistent as I' = {(x, "1"), (z, x)} is consistent as I" = {(x, "1")} is consistent.

Definition 6. (**Elements unification**) Two vertices v_1 and v_2 (resp. two edges e_1 and e_2) are unifiable if one of the two vertices is $n_i(*)$ (resp. $\xrightarrow{*}$) or if the three following conditions are verified :

(i) $|Lab^{v_1}| = |Lab^{v_2}|$ (resp. $|Tag^{e_1}| = |Tag^{e_2}|$).
(ii) These attributes are unifiable two at a time considering the order of their occurrences.
(iii) The union of the result of each unification of attributes is consistent.

Example 4. The vertex v_1(("1",$\mathbb{N}$), (x,$\mathbb{N}$), (z,$\mathbb{N}$)) from the example 1 and v_3((y,$\mathbb{N}$), (y,$\mathbb{N}$), (y,$\mathbb{N}$)) are unifiable as the set of identification I = {(y, "1"), (x, y), (z,y)} is consistent as seen in example 3.

Definition 7. (**Affectation**) For any consistent set of identification I, an affectation Aff_I is an application from the set of considered graphs to itself such as for any graph G = (V, E, Lab, Tag), Aff_I(G) is G integrating I :

— If I is basic,

(i) $\forall$ (v, i) $\in$ (V, $\mathbb{N}$), $Lab_i^v \in Lab^v$, if $\exists$ (Lab_i^v, "value") $\in$ I then Lab'^v_i = "value", else $Lab'^v_i = Lab_i^v$. Let $Lab'^v = \{(Lab'^v_1, Dlab^v_1),\ldots,(Lab'^v_{|Lab^v|}, Dlab^v_{|Lab^v|})\}$. Let Lab' = $\{Lab'^{v_1},\ldots, Lab'^{v_{|V|}}\}$.

(ii) $\forall$ (e, i) $\in$ (E, $\mathbb{N}$), $Tag_i^e \in Tag^e$, if $\exists$ (Tag_i^e, "value") $\in$ I then Tag'^e_i = "value", else $Lab'^e_i = Lab_i^e$. Let $Tag'^e = \{(Tag'^e_1, Dtag^e_1),\ldots,(Tag'^e_{|Tag^e|}, Dlab^e_{|Tag^e|})\}$. Let Tag' = $\{Tag'^{e_1},\ldots, Tag'^{e_{|E|}}\}$.

(iii) Aff_I(G) = (V, E, Lab', Tag').

— If I is not basic, for any couple (x, y) $\in$ I where x and y are variable,

(i) $\forall$ (v, i) $\in$ (V, $\mathbb{N}$), $Lab_i^v \in Lab^v$, if $\exists$ Lab_i^v = y then Lab'^v_i = x, else $Lab'^v_i = Lab_i^v$. Let $Lab'^v = \{(Lab'^v_1, Dlab^v_1),\ldots,(Lab'^v_{|Lab^v|}, Dlab^v_{|Lab^v|})\}$. Let Lab' = $\{Lab'^{v_1},\ldots, Lab'^{v_{|V|}}\}$.

(ii) $\forall$ (e, i) $\in$ (E, $\mathbb{N}$), $Tag_i^e \in Tag^e$, if $\exists$ Tag_i^e = y then Tag'^e_i = x, else $Lab'^e_i = Lab_i^e$. Let $Tag'^e = \{(Tag'^e_1, Dtag^e_1),\ldots,(Tag'^e_{|Tag^e|}, Dlab^e_{|Tag^e|})\}$. Let Tag' = $\{Tag'^{e_1},\ldots, Tag'^{e_{|E|}}\}$.

(iii) Let I' = $\{ (a_i, a_j) \mid (a_i, a_j) \in I \wedge a_i \neq y \wedge a_j \neq y \} \cup \{ (x,a_i) \mid ((y,a_i) \in I \vee (a_i,y) \in I) \wedge a_i \neq x \}$

(iv) $Aff_I(G) = Aff_{I'}((V, E, Lab', Tag'))$.

Example 5. With the set of identification I defined in example 3 and the graph G = (V, E) defined in example 1, Aff_I(G) = (V', E') where V' = {v_1(("1", ℕ), ("1", ℕ), ("1", ℕ)), v_2((w,ℕ))} and E' = { $v_1 \xrightarrow{("a",\{a,b\})} v_2$}

Definition 8. (**Graph homomorphism with variable label**) Two graphs G and G' such as G = (V, E, Lab, Tag) and G' = (V', E', Lab', Tag') are homomorph -noted G → G'- if and only if there is an affectation Aff_I and an injective function f : V → V', such as :

(i) For any couples of vertices (v_i, v_j) ∈ V^2 and (v'_i, v'_j) ∈ $(V')^2$ with f(v_i) = v'_i and f(v_j) = v'_j, if (v_i, v_j) ∈ E, then (v'_i, v'_j) ∈ E' and $\text{Tag}^{(v_i,v_j)}$ is unifiable with $\text{Tag}^{(v'_i,v'_j)}$.
(ii) Each vertices associated by f are unifiable.
(iii) The set of identification I resulting from each unification is consistent.

The resulting homomorphism is characterised by the couple (f, Aff_I). If f is bijective and (f^{-1}, Aff_I) is a homomorphism, then G and G' are isomorph.

Example 6. Let G' = ({v_3((y,ℕ), (y,ℕ), (y,ℕ))}, ∅) and G = (V, E) the graph defined in example 1. Let f : V' → V such as f(v_3) = v_1. As seen in example 4 these vertices are unifiable and the result of this unification is the consistent set of identification I, so that h = (f, Aff_I) is a homomorphism from G' to G.

Definition 9. (**Compatible graphs**) For two graphs G = (V, E, Lab, Tag) and G' = (V', E', Lab', Tag'), G and G' are said to be (f, Aff_I, V_S, V'_S)-compatible if and only if there exists V_S ⊆ V, V'_S ⊆ V', an affectation Aff_I and a bijective function f : V_S → V'_S such as :

(i) For any couples of vertices (v_1, v_2) ∈ $V_S{}^2$ and (v'_1, v'_2) ∈ $V'_S{}^2$ with f(v_1) = v_1' and f(v_2) = v_2', if (v_1, v_2) ∈ E and (v_1', v_2') ∈ E' then $\text{Tag}^{(v_1,v_2)}$ is unifiable with $\text{Tag}^{(v'_1,v'_2)}$.
(ii) Each vertices associated by f are unifiable.
(iii) The set of identification I resulting from each unification is consistent.

Remark 1. If G and G' are (f, Aff, V_S, V'_S) compatible, then G' and G are (f^{-1}, Aff, V'_S, V_S) compatible.

Example 7. Figure 2 shows an example of two compatible graphs. For readability sake, the tags of the edges have not been represented and will all be considered equals. Let V_S be the set of vertices named 1, 2 and 3 in the figure, V'_S be the set of vertices named 3', 2' and 4', I = {(a, x), (b, "2")} and f : V_S → V'_S associating the vertices named 1 to 2', 2 to 4' and 3 to 3'. As the edges (2,1) and (4',2') as well as (1,3) and (2',3') are unifiable, G and G' are (f, Aff_I, V_S, V'_S)-compatible.

Definition 10. (**Homomorphic common sub-graph**) A graph H is an h-homomorphic common sub-graph of two graphs G and G' if H is an induced sub-graph of G and there exists a homomorphisms h = (f, Aff) : H → G'.

Property 1. If H is a (f, Aff)-homomorphic sub-graph of G and G', then there exists f', V_S and V'_S such as G and G' are (f', Aff, V_S, V'_S)-compatible.

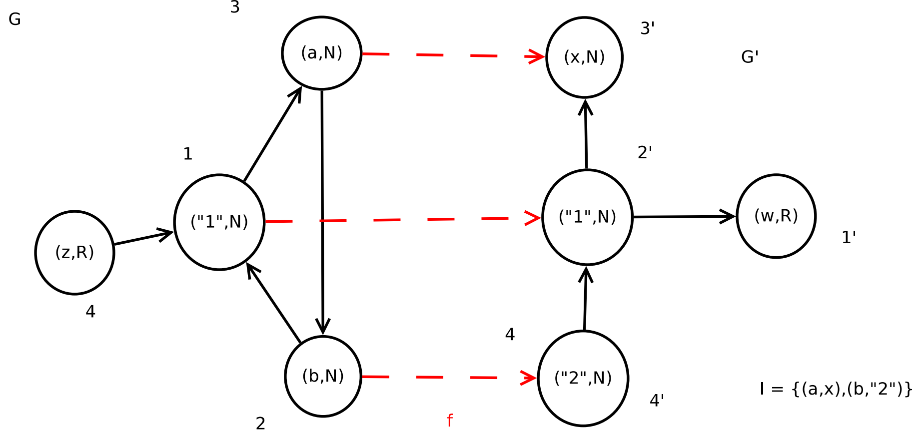

Figure 2. Two compatible graphs

Proof 1. Let f' : $V_G \rightarrow f(V_G)$ such as $\forall$ v $\in V_G$ f'(v) = f(v). By definition, f is injective, f' is thus bijective. By definition of a graph homomorphism, G and G' are (f', Aff, V_g, $f(V_G)$)-compatible.

2.3. *A new approach for graph transformation*

The approach used in this paper is based on the Double PushOut (DPO) (Ehrig, 1987) with multiple negative application conditions (NACs). The suspension condition is no longer considered and the dangling edges are handled as in the Single PushOut (SPO) method -i.e. suppression.

Definition 11. (**Graph rewriting rule**) A graph rewriting rule is a 4-tuple (L, K, R, NACs) where L and R are two graphs, K -called the Inv zone- is a sub-graph of both L and R and NACs is a set of graph specifying the negative application conditions such as $\forall$ NAC $\in$ NACs, L is a sub-graph of NAC. L\K is called the Del zone and R\K is called the Add zone.
A rule is applicable on a graph G if there is a homomorphism h : L $\rightarrow$ G and if $\forall$ NAC $\in$ NACs there is no homomorphism h' : NAC $\rightarrow$ G such as $\forall$ n $\in$ L h'(n)=h(n). Its application consist in erasing h(L\K), deleting the potential dangling edges and adding an isomorph copy of R\K integrating the affectation obtained with h.

Example 8. Figure 3 offers an example of how a transformation is handled in the approach previously defined approach. To lighten the figure, the tags of the edges have not been represented and will all be considered equals. The NAC is automatically valid because of NACs' emptiness. Moreover considering that L and G1 are homomorph, the transformation R can be applied to G1. The graph corresponding to the Del zone is removed leading to the apparition of an unique dangling edge -which used to link the

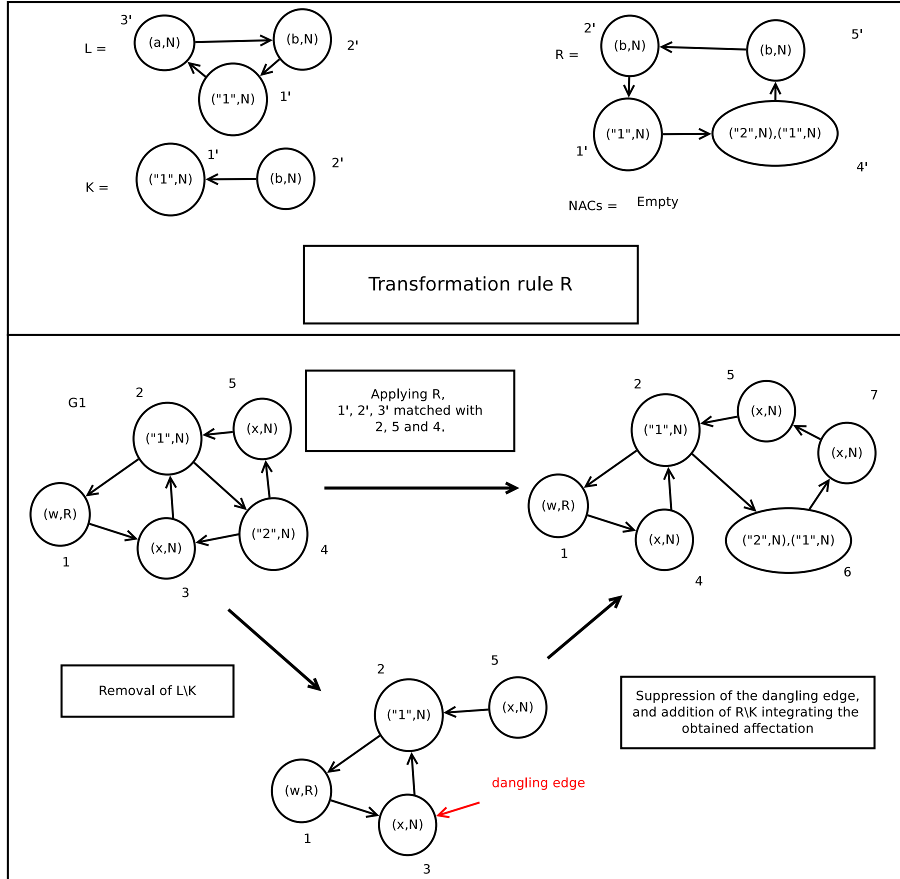

Figure 3. An example of graph transformation

node named 4 to the node named 3. This edge is suppressed and an isomorph copy of the Add zone is then added.

Convention 3. The following notations will be adopted :

(i) r_h(G) is the result of the application of a graph rewriting rule r to the graph G considering the homomorphism h : L → G.

(ii) $r_2_h_2.r_1_h_1(G)$ is the result of a the sequence r_2 applied to the result of r_1 applied to G with the matching h_1 in regard of the matching h_2.

Remark 2. Yet another graph transformation model

A notable characteristic of this model is its superior expressiveness regarding the common DPO approach.

The suspension condition can be expressed through NACs. Any rule (L,K,R) expressed in the DPO formalism can be expressed using the approach introduced in this paper with (L,K,R,NACs) where NACs is constructed as follow. Let $(V, E) = L$ and $m = |V|$. For each

$n_i \in V$, let NAC1(v_i)=($V \cup \{v_{m+1}(*)\}$, $E \cup \{(v_i \xrightarrow{*} v_{m+1})\}$ and NAC2(v_i)=($V \cup \{v_{m+1}(*)\}$, $E \cup \{(v_{m+1} \xrightarrow{*} v_i)\}$. NACs = $\bigcup_{v_i \in V} \{NAC1(v_i), NAC2(v_i)\}$.

Definition 12. (**Restriction** $\downarrow$) Let G and G' be two (f, Aff, V_S, V'_S)-compatible graphs. For any sub-graphs G_{sub} = ($V_{G_{sub}}$, $E_{G_{sub}}$, $Lab_{G_{sub}}$, $Tag_{G_{sub}}$), G'_{sub} = ($V_{G'_{sub}}$, $E_{G'_{sub}}$, $Lab_{G'_{sub}}$, $Tag_{G'_{sub}}$),

(i) let $V_{restric} = \{ v_i \mid v_i \in V_S \cap V_{G_{sub}} \wedge f(v_i) \in V_{G'_{sub}} \}$,
(ii) let $E_{restric} = \{ (v_i,v_j) \mid (v_i, v_j) \in V^2_{restric} \wedge (v_i, v_j) \in E_{G_{sub}} \wedge (f(v_i),f(v_j)) \in E_{G'_{sub}} \}$,
(iii) let $Lab_{restric} = \{ Lab^v_{G_{sub}} \mid v \in V_{restric} \}$,
(iv) and let $Tag_{restric} = \{ Tag^e_{G_{sub}} \mid e \in E_{restric} \}$,

The restriction relation is defined by $G_{sub} \downarrow_{(f,Aff,V_S,V'_S)} G'_{sub}$ = Aff(($V_{restric}$, $E_{restric}$, $Lab_{restric}$, $Tag_{restric}$)).

Example 9. With G, G', f, Aff, V_S and V'_S defined in the example 7, the result of $G\downarrow_{(f,Aff,V_S,V'_S)}G'$ is represented in figure 4.

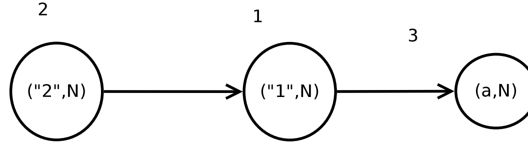

Figure 4. $G\downarrow_{(f,Aff,V_S,V'_S)}G'$

Definition 13. (**Expansion** $\uparrow$) Let G and G' be two (f, Aff, V_S, V'_S)-compatible graphs. For any sub-graphs G_{sub} = ($V_{G_{sub}}$, $E_{G_{sub}}$, $Lab_{G_{sub}}$, $Tag_{G_{sub}}$), G'_{sub} = ($V_{G'_{sub}}$, $E_{G'_{sub}}$, $Lab_{G'_{sub}}$, $Tag_{G'_{sub}}$),

(i) let $V_{expan} = \{ v_i \mid v_i \in V_{G_{sub}} \vee (v_i \in V_{G'_{sub}} \wedge v_i \notin V'_S) \}$,
(ii) let $E_{expan} = \{ (v_i,v_j)) \mid (v_i, v_j) \in E_{expan} \wedge ((v_i, v_j) \in E_{G_{sub}} \vee (f(v_i), f(v_j)) \in E_{G'_{sub}} \vee (v_i, v_j) \in E_{G'_{sub}} \vee (f(v_i), v_j) \in E_{G'_{sub}} \vee (v_i, f(v_j)) \in E_{G'_{sub}})\}$,
(iii) let $Lab_{expan} = \{ Lab^v_{G_{sub}} \mid v \in V_{G_{sub}} \} \cup \{ Lab^v_{G'_{sub}} \mid v \in V_{expan} \wedge v \in V_{G'_{sub}} \}$,
(iv) and let $Tag_{expan} = \{ Tag^e_{G_{sub}} \mid e \in E_{expan} \} \cup \{ Tag^{(v_i,v_j)}_{G'_{sub}} \mid (v_i, v_j) \in E_{expan} \} \cup \{ Tag^{(f(v_i),f(v_j))}_{G'_{sub}} \mid (v_i, v_j) \in E_{expan} \} \cup \{ Tag^{(f(v_i),v_j)}_{G'_{sub}} \mid (v_i, v_j) \in E_{expan} \} \cup \{ Tag^{(v_i,f(v_j))}_{G'_{sub}} \mid (v_i, v_j) \in E_{expan} \}$.

The expansion relation is defined by : $G_{sub} \uparrow_{(f,Aff,V_S,V'_S)} G'_{sub}$ = Aff((V_{expan}, E_{expan}, Lab_{expan}, Tag_{expan})).

Example 10. With G, G', f, Aff, V_S and V'_S defined in the example 7, the result of $G\uparrow_{(f,Aff,V_S,V'_S)}G'$ is represented in figure 5.

Remark 3. For any (f, Aff, V_S, V'_S)-compatible graphs G and G' and any sub-graphs G_{sub} and G'_{sub}, $G_{sub} \to G_{sub} \uparrow_{(f,Aff,V_S,V'_S)} G'_{sub}$ and $G'_{sub} \to G_{sub} \uparrow_{(f,Aff,V_S,V'_S)} G'_{sub}$

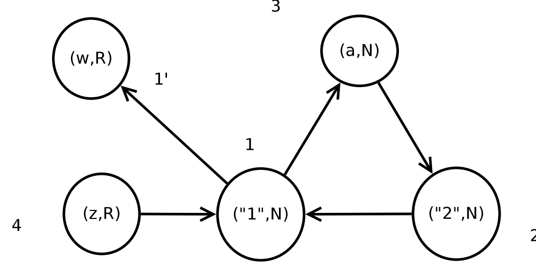

Figure 5. G$\uparrow_{(f,Aff,V_S,V_S')}$G'

2.4. *Graph rewriting rule composition*

For any pair (p,q) of graph transformation rules expressed in the previously exposed formalism two binary operators for composition are defined.

Definition 14. (**Graph rewriting rule composition considering a specific homomorphic common sub-graph**) For any graph G h-homomorphic common sub-graph of L_p and (V_{R_q}, $E_{R_q} \cup$ E', Lab_{R_q}, $Tag_{R_q} \cup$ Tag') where h = (f, Aff) and E' is a set of edges from V_{K_q} to V_{K_q} tagged by Tag', let r_1 = (G, G $\downarrow_{(f,Aff,V_G,f(V_G))}$ K_q, G $\downarrow_{(f,Aff,V_G,f(V_G))}$ K_q, $\emptyset$). If r_1_(id, $Aff_\emptyset$)(L_p) does not lead to the apparition of any dangling edge, then p$\circ_{(G,h)}$q is the rewriting rule described by

(i) Let M = r_1_(id,$Aff_\emptyset$)(L_p). M is L_p deprived of the part of G not identified with K_q via h which is the part of G added when q is applied.
$\mathbf{L}_{p\circ_{(G,h)}q}$ = M $\uparrow_{(f,Aff,V_{(G\downarrow_{f,Aff,V_G,f(V_G)}K_q)},V_{K_q}}$ L_q.

(ii) Let r_1' = ((G, G $\downarrow_{(id,Aff_\emptyset,V_G,V_G)}$ K_p, G $\downarrow_{(id,Aff_\emptyset,V_G,V_G)}$ K_p, $\emptyset$) and M' = r_1'_h(R_q). M' is Aff(R_q) deprived of the part of h(G) not belonging to h(K_p) which is the part of G suppressed when p is applied.
$\mathbf{R}_{p\circ_{(G,h)}q}$ = R_p $\uparrow_{(h,V_{K_p},V_{G\downarrow_{(id,Aff_\emptyset,V_G,V_G)}K_p})}$ M'.

(iii) $\mathbf{K}_{p\circ_{(G,h)}q}$ = $L_{p\circ_{(G,h)}q} \cap R_{p\circ_{(G,h)}q}$,

(iv) Let f' : $V_{L_q} \rightarrow V_{L_q} \backslash V_{K_q} \cup f^{-1}(V_{K_q})$, if v $\in V_{L_q} \backslash V_{K_q}$ f'(v) = v else, if v $\in V_{K_q}$ f'(v) = f^{-1}(v). Let $NACS_{qrelated}$ = { NAC | $\exists$ NAC_q $\in$ $NACs_q$, NAC = NAC_q $\uparrow_{(f',Aff,V_{L_q},V_{L_q}\backslash V_{K_q}\cup f^{-1}(V_{K_q}))}$ $L_{p\circ_{(G,h)}q}$}.
Let $NACs_{prelated}$ = { NAC$\uparrow_{(id,Aff,V_{L_p},V_{L_p})}$ $L_{p\circ_{(G,h)}q}$ | $\exists$ NAC_p $\in$ $NACs_p$, r_1_(id,$\emptyset$)(NAC_p) does not lead to the apparition of any dangling edge and NAC = r_1_(id,$\emptyset$)(NAC_p) }.
$\mathbf{NACs}_{p\circ_{(G,h)}q}$ = $NACs_{qrelated} \cup NACs_{prelated}$

E' and Tag' come from the graph on which the rule is going to be applied.

Definition 15. (**Graph rewriting rule composition**) The second binary operator $\circ$ is defined by p$\circ$q = {r |$\exists$ $\circ_{(G,h)}$, r = p$\circ_{(G,h)}$q }

Property 2. For any graph G, any pair of graph rewriting rules (p,q) and any couple of homomorphisms (h_p, h_q), $\exists$ r$\in$p$\circ$q, $\exists$ a homomorphism h_r, p_h_p.q_h_q(G) = r_h_r(G).

Proof 2. Let G = (V, E, Lab, Tag), h_p = (f_p, Aff_p), h_q = (f_q, Aff_q), q_h_q(G) = (V_q, E_q, Lab_q, Tag_q) and p_h_p.q_h_q(G) = (V_{pq}, E_{pq}, Lab_{pq}, Tag_{pq}). Let $V_{G'}$ = { v $\in$ V_{L_p} | f_p(v) $\notin$ V $\vee$ ($\exists$ v' $\in$ K_q, f_p(v) = f_q(v')) }. Let G' be the sub-graph of L_p induced by $V_{G'}$.

By construction of $V_{G'}$, a vertex of G' is either part of what has been added to G while applying q_h_q - $V_{R_q \setminus K_q}$- , or part of the part of G identified with V_{L_q} and still present in q_h_q(G) - h_q(V_{K_q}). Let f' : $V_{G'} \to V_{R_q}$, $\forall v \in V_{G'}$, f_p(v) $\in$ V $\implies$ f'(v) = f_q^{-1}(f_p(v))) $\wedge$ ($\forall v \in V_{G'}$, f_p(v) $\notin$ V $\implies$ f'(v) = v). By hypothesis L_p and q_h_q(G) are homomorph, thus - G' being an induced sub-graph of L_p - G' and q_h_q(G) are homomorph and G' is an homomorphic common sub-graph of L_p and q_h_q(G). Hence, there exists E' a set of edges from V_{K_q} to V_{K_q} - present in L_p q_h_q(G) but not in K_q - and tagged by Tag' such as G' is a (f', $Aff_p\circ$ Aff_q)-homomorphic common sub-graph of L_p and (V_{R_q}, $E_{R_q}\cup$ E', Lab_{R_q}, Tag_{R_q} $\cup$ Tag').

Let r = p $\circ_{(G',(f',Aff_p\circ Aff_q))}$ q. Let f_r : $V_{L_r} \to V_G$, $\forall v \in V_{L_r}$, ((v $\in$ L_p $\wedge$ f_p(v) $\in$ V_G) $\implies$ f_r(v) = f_p(v)) $\wedge$ ((v $\in$ L_p $\wedge$ f_p(v) $\notin$ V_G) $\implies$ f_r(v) = f_r(f_p(v))) $\wedge$ (v $\notin$ L_p $\implies$ f_r(v) = f_q(v)). h_r = (f_r, $Aff_p\circ$ Aff_q) is a homomorphism from L_r to G.

As no expansion of h_q is a homomorphism from a NAC in $NACs_q$ to G and no expansion of h_p is a homomorphism from a NAC in NAC_p to q_h_q(G) then by definition no expansion of h_r is a homomorphism from a NAC in $NACs_r$ to G. r is thus applicable to G.

The vertices associated by f_r are exactly the vertices associated by f_p and f_q and the affectation $Aff_p\circ$ Aff_q is exactly the application of Aff_q followed by the application of Aff_p. Besides considering the definition of R_r - R_p expanded to the part of R_q not deleted while applying p with the homomorphism h_p- p_h_p.q_h_q(G) = r_h_r(G).

Property 3. For any graph G and any sequence of application of graph rewriting rule r_n_h_n.(...).r_1_h_1(G), there exists a sequence of graphs $(G_m)_{m\in[|1,n-1|]}$ and a sequence of homomorphisms $(h'_l)_{l\in[|0,n-1|]}$ such as (r_n $\circ_{(G_{n-1},h'_{(n-1)})}$ (...($r_2\circ_{(G_1,h'_1)}r_1$)...)_h'_0(G) = r_n_h_n.(...).r_1_h_1(G)

Proof 3. By structural induction, let P(n) be the proposition "for any sequence of application of n graph rewriting rules r_n_h_n.(...).r_1_h_1(G), there exists a sequence of graphs $(G_m)_{m\in[|1,n-1|]}$ and a sequence of homomorphisms $(h'_l)_{l\in[|0,n-1|]}$ such as (r_n $\circ_{(G_{n-1},h'_{n-1})}$ (...($r_2\circ_{(G_1,h'_1)}r_1$)...)_h'_0(G) = r_n_h_n.(...).r_1_h_1(G)".

According to proposition 2, P(2) is true.

Suppose P(n) true. Then, for any sequence of application of n+1 graph rewriting rules $r_{n+1}_h_{n+1}.r_n_h_n.(\dots).r_1_h_1(G)$, as P(n) is true, there exists a sequence of graphs $(G_m)_{m\in[|1,n-1|]}$ and a sequence of homomorphisms $(h'_l)_{l\in[|0,n-1|]}$ such as $r_{n+1}_h_{n+1}.(r_n \circ_{(G_{n-1},h'_{n-1})} (\dots(r_2\circ_{(G_1,h'_1)}r_1)\dots)_h'_0(G) = r_{n+1}_h_{n+1}.r_n_h_n.(\dots).r_1_h_1(G)$.
According to proposition 2, there exists a couple of homomorphisms h'_n, $h"_0$ and a graph G_n such as $r_{n+1}\circ_{(G_n,h'_n)} (r_n \circ_{(G_{n-1},h'_{n-1})} (\dots(r_2\circ_{(G_1,h'_1)}r_1)\dots)_h"_0(G) = r_{n+1}_h_{n+1}.(r_n \circ_{(G_{n-1},h'_{n-1})} (\dots(r_2\circ_{(G_1,h'_1)}r_1)\dots)_h'_0(G) = r_{n+1}_h_{n+1}.r_n_h_n.(\dots).r_1_h_1(G)$. Thus, P(n+1) is true.

Hence, proposition 3 is true.

3. Characterizing architectural style

This section introduces both graph grammar foundations and the graph grammar based approach that is used for architectural application model. An architectural style will be characterized by a graph grammar, and each of its instance will then be represented by a graph. An example of graph grammar characterizing a collaborative application will finally be proposed.

3.1. *Generic case*

Graph grammars constitute an expressive formalism dynamic structure description. Following the commonly used conventions for standard graphical descriptions, one considers that vertices represent services or architectural components and edges correspond to their related interdependencies. The use of graphs is relevant since this paper addresses the specification of architectural styles where declarative aspects corresponding to the description of all the possible instances can be correctly specified by graph grammars. Moreover, theoretical work on this field provides formal means to specify and check structural constraints and properties (Rozenberg, 1997; Ehrig and Kreowski, 1991).
Inspired from Chomsky's generative grammars (Chomsky, 1956), graph grammars are defined, in general, as a classical system $< AX; NT; T; P >$, where AX is the axiom, NT is the set of the non-terminal vertices, T is the set of terminal vertices, and P is the set of graph rewriting rules, also called grammar productions. An instance belonging to the graph grammar is a graph containing only terminal vertices and is obtained starting from axiom AX by applying a sequence of productions in P. The following slightly different definitions will be considered.

Definition 16. (**Graph Grammar**)A graph grammar is defined by the 4-tuple (AX, NT, T, P) where

(i) AX is the axiom,
(ii) NT is the sets of non-terminal arch-vertices or archetypes of vertices,
(iii) T is the set of terminal arch-vertices or archetypes of vertices,

(iv) P is the set of graph rewriting rules belonging to the graph grammar.

Each vertex occurring in a graph rewriting rule in P or in a graph obtained by applying a sequence of productions $\in P$ to the axiom is then unifiable with at least one arch-vertex in NT or T. This means that for any of these vertices v that is not a joker, $\exists v' \in NT \cup T$, $|Lab^v| = |Lab^{v'}| \wedge \forall i \in [|1, \ldots, |Lab^v||], Dlab_i^v = Dlab_i^{v'}$.

Definition 17. (**Instance belonging to the graph grammar**) An instance belonging to the graph grammar (AX, NT, T, P) is a graph whose vertices and edges have only constant attributes and obtained by applying a sequence of productions in P to AX.

Definition 18. (**Consistent instance belonging to the graph grammar**) A consistent instance belonging to the graph grammar (AX, NT, T, P) - or consistent instance of the architectural style modelled by (AX, NT, T, P) - is an instance of (AX, NT, T, P) not containing any vertex unifiable with an arch-vertex from NT

To model the generation of the instances of a graph grammar, graph whose vertices represents the instances a graph grammar is introduced. Its edges represents the application of a graph rewriting rule.

Definition 19. (**Generation graph**) For any graph grammar GG = (AX, NT, T, P),

(i) Let Ins be the the set of instances of GG.
(ii) Let E be a set of edges tagged with Tag from Ins to Ins such as :
  (i) $\forall e \in E, |Tag^e| = 2 \wedge Dtag_1^e = P \wedge Dtag_2^e$ is the set of grah homomorphism $\wedge Tag_1^e$ and Tag_2^e are constant,
  (ii) $\forall (c, c') \in Ins^2, (c, c') \in E \Rightarrow Tag_1^{(c,c')}_Tag_2^{(c,c')}(c) = c'$.
(iii) The generation graph of GG is G = (Ins, E, $\emptyset$, Tag).

3.2. *A collaborative application used for further example*

DIET (Caron and Desprez, 2006) stands for Distributed Interactive Engineering Toolbox. It is a hierarchical load balancer for dispatching computational jobs over a distributed infrastructure (like a grid or cloud). DIET architecture consists of a set of agents: some Master Agents (MA) manage pools of computational SErver Deamons (SED) through none, one or several layers of Local Agents (LA). These servers can achieve specialized computational services. Communications between agents are driven by the omniORB system (OMNI). MAs listen to client requests and dispatch them through the architecture to the best SED that can carry out this service.
A description of this application using class diagrams (Sharrock et al., 2010) has been proposed; however such an approach lack of expressiveness. For example, the fact that a LA can manage another LA could not be taken into consideration whereas this is not an issue while employing graph grammars.

A simplified architecture with the following constraints will be considered :

(i) An instance of the architectural style comports exactly one MA and one OMNI.
(ii) While being deployed, each component record itself to the OMNI. This will be modelled by an edge labelled Keeps Track Of ("kto").
(iii) Each LA and each SED has a hierarchical superior.
(iv) The MA and each LA manage at least one LA or one SED - this condition could be trivially extended to any number of minimum managed entities.

Let IdMachine be a set of identifiers for each machine on which an architectural component might be deployed specifying both the machine and the component, and IdServices a set of identifiers for services that might be carried out by a SED.

Let T_{DIET} = {N((nature, {"MA", "LA"}), (id, IdMachine)),
N(("SeD",id), (id, IdMachine), (services, IdServices)),
N(("OmniNames", {"OmniNames"}))},
NT_{DIET} = {N("TempComponent", {"TempComponent"})},
P_{DIET} = {r_1,..., r_4 } where each graph rewriting rule is defined below and
GG_{DIET}={AX_{DIET},NT_{DIET}, T_{DIET}, P_{DIET}}.

Considering the arch-vertices defined for DIET and the absence of ambiguity for the domains of definition for the labels, they will be implied in the following section. Each terminal vertex represents an architectural component of the application. The non-terminal archetype is meant to represent either a LA or a SED. It is used to verify the fourth constraint and to handle the fact that a LA may manage another LA.

Initialisation, which consists in deploying the OMNI and the MA and thus validating the first constraint -as this production is the only one introducing and a MA or a OMNI and that it can not be applied more than once-, is realised by this first graph rewriting rule.

r_1 = (L={AX_{DIET}};
K={};
R\K={N1("OmniNames"), N2("MA", id), N3("TempComponent"),
N1 $\xrightarrow{\text{"kto"}}$ N2, N2 $\xrightarrow{\text{"manages"}}$ N3};
NACs=∅)

What has to be done now is offering the possibility to add a additional SED or LA on any agent MA or LA. As a "TempComponent" represents either a SED or a LA, this is equivalent to add such a non-terminal vertex on a MA or a LA.

r_2 = (L={N1(nature, id)}
K={N1(nature, id)};
R\K={N2("TempComponent"), N1 $\xrightarrow{\text{"manages"}}$ N2};
NACS=∅)

Finally, the graph grammar has to describe how a non-terminal vertex will be

instantiated into a LA or a SED. The case SED is simple, as it can not manage any other component and has no specific constraint except recording itself and having a manager.

r_3 = (L={N1("OmniNames"), N2(nature, id1), N3("TempComponent"),
N1 $\xrightarrow{\text{"}kto\text{"}}$ N2, N2 $\xrightarrow{\text{"}manages\text{"}}$ N3};
K={N1("OmniNames"), N2(nature, id), N1 $\xrightarrow{\text{"}kto\text{"}}$ N2};
R\K={N4("SED",id2), N2 $\xrightarrow{\text{"}manages\text{"}}$ N4, N1 $\xrightarrow{\text{"}kto\text{"}}$ N4 };
NACS=∅)

Concerning a LA, the graph rewriting rule is very similar, except for the fact that a LA has to be introduced with a managed entity, modelled by a non-terminal vertex.

r_4 = (L={N1("OmniNames"), N2(nature, id1), N3("TempComponent"),
N1 $\xrightarrow{\text{"}kto\text{"}}$ N2, N2 $\xrightarrow{\text{"}manages\text{"}}$ N3};
K={N1("OmniNames"), N2(nature, id), N1 $\xrightarrow{\text{"}kto\text{"}}$ N2,
N3("TempComponent")};
R\K={N4("LA",id2), N2 $\xrightarrow{\text{"}manages\text{"}}$ N4, N1 $\xrightarrow{\text{"}kto\text{"}}$ N4, N4 $\xrightarrow{\text{"}manages\text{"}}$ N3 };
NACS=∅)

Example 11.

An example of the generation of a consistent instance of a graph grammar is represented in figure 6. The domains of definition of the labels as well as the homomorphisms according to which the rules have been applied have not been represented as there is no ambiguity. Every tags have the same domain of definition {"kto", "manages"}.

4. Handling dynamicity : consistent reconfiguration

In this section, an approach to tackle the dynamic behaviour of applications previously characterized employing graph grammars will be introduced . Such an approach can be semi-automatized and guarantees some "good" properties. Besides the expressiveness, this method presents the notable advantage of being correct by construction whereas the approach described in (Sharrock et al., 2010) requires verification in runtime.

4.1. *Induced rules*

As seen previously in this paper, a graph grammar defined by the 4-tuple (AX, NT, T, P) models an architectural style. In order to take the dynamic aspect of an application into account, this graph grammar is extended to the 5-tuple (AX,NT,T,P,P_{reconf}) where P_{reconf} is the set of atomic graph transformations that represents the architectural evolution of the considered application during its execution.

Definition 20. (**Reciprocal rule**) A graph rewriting rule r^{-1} is the reciprocal of a graph rewriting rule r $K_{r^{-1}} = K_r \wedge R_{r^{-1}} = L_r \wedge L_{r^{-1}} = R_r \wedge NACs_{r^{-1}} = \emptyset$.

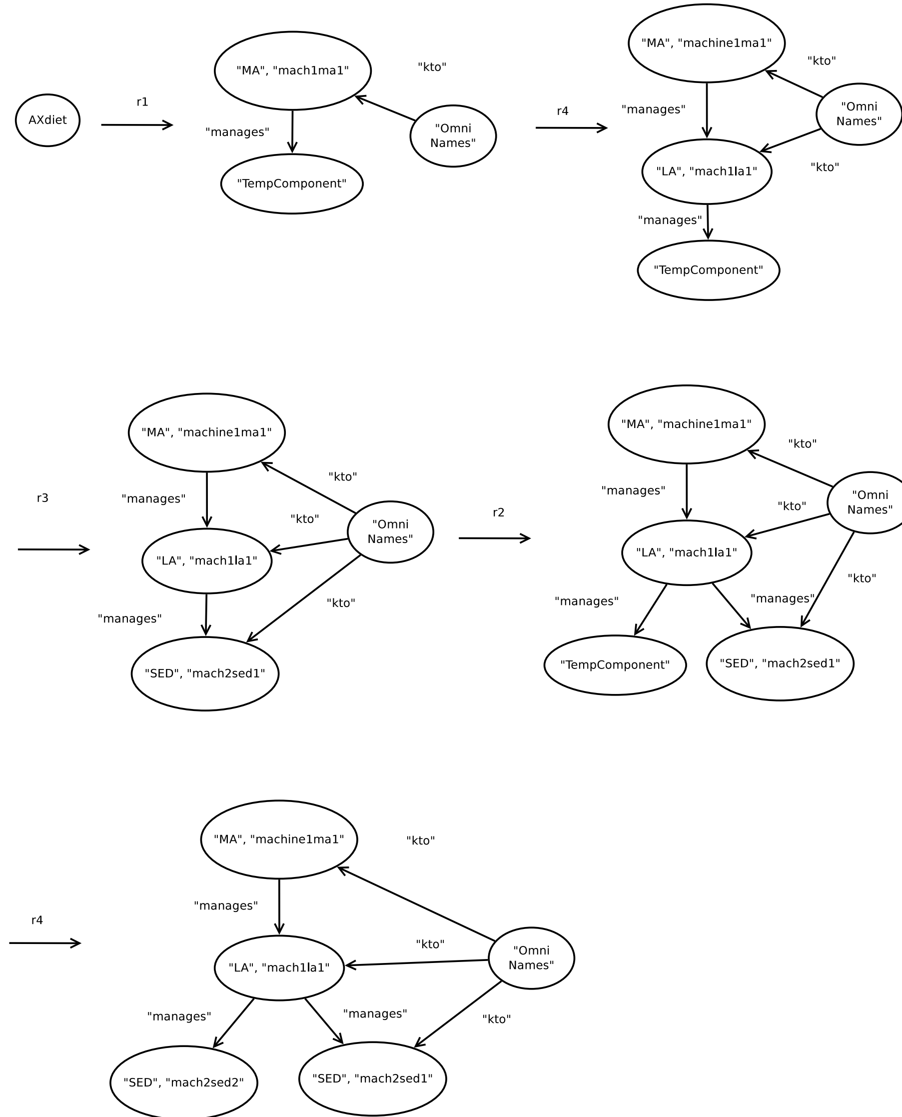

Figure 6. An example of graph generation using a graph grammar

Remark 4. Trivially, for any graph graph rewriting rule r, graph G and homomorphism h there exists a homomorphism h' such as r^{-1}_h'.r_h(G) = Aff_h(G) - h' is the canonical homomorphism associating $L_{r^{-1}} = R_r$ and the isomorph copy of R_r introduced while applying r_h on G.

Besides, if G is a graph whose vertices and edges have only constant attributes - such as a instance of a graph grammar-, for any graph graph rewriting rule r and homomorphism h there exists a homomorphism h' such as r^{-1}_h'.r_h(G) = G.

Example 12. The reciprocal rule of r_3^{-1} previously defined for the graph grammar modelling characterizing DIET is defined by :

r_3^{-1} = (L={N1("OmniNames"), N2(nature, id), N1$\xrightarrow{\text{"}kto\text{"}}$N2, N4("SED",id2), N2$\xrightarrow{\text{"}manages\text{"}}$N4, N1$\xrightarrow{\text{"}kto\text{"}}$N4};
K={N1("OmniNames"), N2(nature, id), N1$\xrightarrow{\text{"}kto\text{"}}$N2};
R\K={ N3("TempComponent"), N2$\xrightarrow{\text{"}manages\text{"}}$N3};
NACS=$\emptyset$)

Definition 21. (**Induced rule**) An induced graph rewriting rule of a graph grammar (AX, NT, T, P) is a rule r such as r $\in$ P $\vee$ $\exists$ r' $\in$ P, r is the reciprocal of r'.

The following definition shows how to navigate between consistent instances of any graph grammar GG = (AX, NT, T, P) without any non-consistent intermediate step considering the extended graph grammar $(AX, NT, T, P, P_{reconf})$ where P_{reconf} is the set of induced rules of GG.

Definition 22. (**Induced reconfiguration graph**) For any graph grammar GG = (AX, NT, T, P),

(i) Let CIns be the set of consistent instances of GG.
(ii) Let G_{gener} = (Ins, E_{gener}, $\emptyset$, Tag_{gener}) be the generation graph of GG.
(iii) Let E_{recip} = { $(v_i, v_j) \in Ins^2$ | $(v_j, v_i) \in E_{gener}$ }.
(iv) Let Tag_{recip} = { $(Tag_{recip})^{(v_i,v_j)}$ = { $((Tag_{recip})_1^{(v_i,v_j)}$, $P_{reconf})$, $((Tag_{recip})_2^{(v_i,v_j)}$, H)} where H is the set of graph homomorphism $|(v_i, v_j) \in E_{recip} \wedge$ $(Tag_{recip})_1^{(v_i,v_j)}$ is the reciprocal rule of $(Tag_{gener})_1^{(v_j,v_i)}$ $\wedge$ $(Tag_{recip})_1^{(v_i,v_j)}$_$(Tag_{recip})_2^{(v_i,v_j)}(v_i) = v_j$ }.
(v) Let $G_{induced}$ = (Ins, $E_{gener} \cup E_{recip}$, $\emptyset$, $Tag_{gener} \cup Tag_{recip}$).
(vi) Let E_{Cgener} = { $(v_i, v_j) \in CIns^2$ | $(v_i, v_j) \in E_{gener}$ } and E_{Crecip} = { $(v_i, v_j) \in CIns^2$ | $(v_i, v_j) \in E_{recip}$ }.
(vii) Let Tag_{Cgener} = { $(Tag_{gener})^e | e \in E_{Cgener}$ } and Tag_{Crecip} = { $(Tag_{recip})^e | e \in E_{Crecip}$ }
(viii) Let E_{comp} = { $(v_i, v_j) \in CIns^2$ | there exists in $G_{induced}$ a path from v_i to v_j with no vertex in CIns }.
(ix) Let Tag_{comp} = { $Tag_{comp})^{(v_i,v_j)}$ = { $((Tag_{comp})_1^{(v_i,v_j)}$, R, $((Tag_{recip})_2^{(v_i,v_j)}$, H } where H is the set of graph homomorphism and R the set of graph rewriting rule $|(v_i, v_j) \in E_{recip} \wedge$ $(Tag_{comp})_1^{(v_i,v_j)}$_$(Tag_{comp})_2^{(v_i,v_j)}(v_i) = v_j$ }.

The induced reconfiguration graph of GG is $G_{induced}$ = (CIns, $E_{comp} \cup E_{Cgener} \cup E_{Crecip}$, $\emptyset$, $Tag_{reconf} \cup Tag_{Cgener} \cup Tag_{Crecip}$).

Remark 5. Note that $\exists$ $Tag_{recip})_2^{(v_i,v_j)}$, $(Tag_{recip})_1^{(v_i,v_j)}$_$(Tag_{recip})_2^{(v_i,v_j)}(v_i) = v_j$ due to remark 4.

Besides $(Tag_{comp})_1^{(v_i,v_j)}$ and $Tag_{comp})_2^{(v_i,v_j)}$ exist. If $(v_i, v_j) \in E_{gener} \cup E_{recip}$, then they exist by definition and $(Tag_{comp})_1^{(v_i,v_j)}$ is an induced rule. Else there exists a sequence of

rules described by the tags of the edges on the path and they exists as a composition of said rules according to the property 3.

Property 4. An induced reconfiguration graph is strongly connected.

Proof 4. By definition, $\forall$ v $\in$ Ins, $\exists$ in G_{gener} a path from $v_0 = AX$ to v. Thus, by construction of $G_{induced}$, $\forall$ v $\in$ Ins, $\exists$ in $G_{induced}$ a path p_{gv} from v_0 to v and a path p_{rv} from v to v_0.
Let (pc^i_{gv}) and (pc^i_{rv}) be a pair of sequences of sequences of vertices $\in$ CIns and (pn^i_{gv}) and (pn^i_{rv}) be a pair of sequences of sequences of vertices $\in$ Ins $\setminus$ CIns so that $p_{gv} = ((pc^1_{gv})_1, \ldots (pc^1_{gv})_{|(pc^1_{gv})|}, (pn^1_{gv})_1, \ldots, (pn^1_{gv})_{|(pn^1_{gv})|}, (pc^2_{gv})_1, \ldots, pc_{gv}^{|(pc_{gv})|})_{|(pc_{gv}^{|(pc_{gv})|})|})$ and $p_{rv} = ((pc^1_{rv})_1, \ldots (pc^1_{rv})_{|(pc^1_{rv})|}, (pn^1_{rv})_1, \ldots, (pn^1_{rv})_{|(pn^1_{rv})|}, (pc^2_{rv})_1, \ldots, pc_{rv}^{|(pc_{rv})|})_{|(pc_{rv}^{|(pc_{rv})|})|})$ where |s| is the size of the sequence s.
By definition of E_{comp}, $\forall j$, $(pc^j_{gv} \in (pc^i_{gv}) \wedge pc^{j+1}_{gv} \in (pc^i_{gv})) \implies ((pc^j_{gv})_{|(pc^j_{gv})|}, (pc^{j+1}_{gv})_1) \in E_{comp}$. Hence $((pc^1_{gv})_1, \ldots (pc^1_{gv})_{|(pc^1_{gv})|}, (pc^2_{gv})_1, \ldots, pc_{gv}^{|(pc_{gv})|})_{|(pc_{gv}^{|(pc_{gv})|})|})$ is a path in $G_{induced}$ from v_0 to v. In a similar way, $((pc^1_{rv})_1, \ldots (pc^1_{rv})_{|(pc^1_{rv})|}, (pc^2_{rv})_1, \ldots, pc_{rv}^{|(pc_{rv})|})_{|(pc_{rv}^{|(pc_{rv})|})|})$ is a path in $G_{induced}$ from v to v_0.
Thus for any pair of vertices (v, v') of $G_{induced}$ there is in $G_{induced}$ a path from v to v_0 and from v_0 to v', as well as a path from v' to v_0 and from v_0 to v. Hence $G_{induced}$ is strongly connected.

The strong connection of the induced reconfiguration graph implies that any consistent instance of a graph grammar can be reached using induced rules composition from any other consistent instance, without any intermediate non-consistent intermediate step. Thus, any dynamic architectural style represented by a graph grammar may be reconfigured guaranteeing intern self-protecting by construction.

4.2. *Specifics rules*

Even though the induced rules are sufficient to navigate between every consistent instance of an architectural style, it might be desirable to specify additional application-specific reconfiguration rules. Such a set of rules - noted $P_{induced}$ - may either characterize some particularities of an application or can be used to achieve particular aim such as self healing. The extended graph grammar characterizing an architectural style is then $(AX, NT, T, P, P_{reconf})$ where $P_{reconf} = P_{induced} \cup P_{specific}$.

Example 13. Considering the previously defined architectural style representing a simplified architecture for DIET, if LAs are likely to break down, using induced rules it would be possible to suppress every components managed by a broken LA before redeploying the LA and the components. However, knowing that the managed entity are still in working order, a better solution is to define the following specific rules for self-healing triggered when a LA break down so as not to redeploy functional components.

After receiving an alert containing the identifier of the broken LA - noted "idBroken"

- the procedure is initialized by deploying a new LA.

sr$_1$ = (L={N1("LA", "idBroken")};
K={N1("LA", "idBroken")};
R\K={N2("LA",id), N2$\xrightarrow{\text{"}replaces\text{"}}$N1 };
NACS=∅)

Each component is then treated one by one by being linked to the new LA in the same way as it was with the broken LA.

sr$_2$ = (L={N1("LA", "idBroken"), N2("LA",id), N2$\xrightarrow{\text{"}replaces\text{"}}$N1,
N3(*), N1$\xrightarrow{*}$N3};
K={N1("LA", "idBroken"), N2("LA",id), N2$\xrightarrow{\text{"}replaces\text{"}}$N1,
N3(*)};
R\K={N2$\xrightarrow{*}$N3};
NACS=∅)

sr$_3$ = (L={N1("LA", "idBroken"), N2("LA",id), N2$\xrightarrow{\text{"}replaces\text{"}}$N1,
N3(*), N3$\xrightarrow{*}$N1};
K={N1("LA", "idBroken"), N2("LA",id), N2$\xrightarrow{\text{"}replaces\text{"}}$N1,
N3(*)};
R\K={N3$\xrightarrow{*}$N2};
NACS=∅)

Once that every component linked to the faulty LA have been treated, the LA is removed. This marks the end of the procedure.

sr$_4$ = (L={N1("LA", "idBroken"), N2("LA",id), N2$\xrightarrow{\text{"}replaces\text{"}}$N1,};
K={N2("LA",id)};
R\K={};
NACS={NAC1, NAC2};
NAC1={N3(*), N3$\xrightarrow{*}$N1};
NAC2={N3(*), N1$\xrightarrow{*}$N3};

A crucial issue concerning specific rules is to prove their correctness. This can be achieved either by classical means or by proving that for any sequence consisting in initializing, treating every vertices and terminating there is an equivalent sequence of induced rules.

5. Conclusion

In this paper, we formally defined the basic operators for graph manipulation including extension and restriction. Graph rewriting rules with multiple negative application conditions and their composition are formally defined. We defined the characterization rules

of an architectural style using Graph Grammars as well as the correct by design automatically generated characterization rules of consistent instance of an architectural style.

We defined the automated induction of reconfiguration rules from the set of generation rules guaranteeing the consistency of the set of generated configurations and the accessibility of any consistent configuration from any other.

We defined a formal framework for defining application specific reconfiguration rules and policies. Our approach is applied the example of a distributed hierarchical application, DIET, including generation, induced reconfiguration and specific reconfiguration.

References

Allen, R. and Garlan, D. (1997). A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.*, 6:213–249.

Bradbury, J. S., Cordy, J. R., Dingel, J., and Wermelinger, M. (2004). A survey of self-management in dynamic software architecture specifications. In *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, WOSS '04, pages 28–33, New York, NY, USA. ACM.

Caron, E. and Desprez, F. (2006). Diet: A scalable toolbox to build network enabled servers on the grid. *International Journal of High Performance Computing Applications*, 20(3):335–352.

Chassot, C., Guennoun, K., Drira, K., Armando, F., Exposito, E., and Lozes, A. (2006). Towards autonomous management of qos through model-driven adaptability in communication-centric systems. *ITSSA*, 2(3):255–264.

Chomsky, N. (1956). Three models for the description of language. *Information Theory, IEEE Transactions on*, 2(3):113–124.

Dashofy, E. M., van der Hoek, A., and Taylor, R. N. (2002). An infrastructure for the rapid development of xml-based architecture description languages. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 266–276, New York, NY, USA. ACM.

de Paula, V. C., Justo, G. R. R., and Cunha, P. R. F. (2000). Specifying and verifying reconfigurable software architectures. In *Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems*, pages 21–, Washington, DC, USA. IEEE Computer Society.

Ehrig, H. (1987). Tutorial introduction to the algebraic approach of graph grammars. In Ehrig, H., Nagl, M., Rozenberg, G., and Rosenfeld, A., editors, *Graph-Grammars and Their Application to Computer Science*, volume 291 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin / Heidelberg. 10.1007/3-540-18771-5-40.

Ehrig, H. and Kreowski, H.-J. (1991). *Graph Grammars and Their Application to Computer Science: 4th International Workshop, Bremen, Germany, March 5-9, 1990 Proceedings*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Garlan, D., Monroe, R. T., and Wile, D. (2000). Acme: Architectural description of component-based systems. In Leavens, G. T. and Sitaraman, M., editors, *Foundations of Component-Based Systems*, pages 47–68. Cambridge University Press.

Heckel, R., Cherchago, A., and Lohmann, M. (2004). A formal approach to service specification and matching based on graph transformation. *Electron. Notes Theor. Comput. Sci.*, 105:37–49.

Hirsch, D., Inverardi, P., and Montanari, U. (1999). Modeling Software Architectures and Styles with Graph Grammars and Constraint Solving. In Donohoe, P., editor, *Software Architecture (TC2 1st Working IFIP Conf. on Software Architecture, WICSA1)*, pages 127–143, San Antonio, Texas, USA. Kluwer.

Kacem, M. H., Jmaiel, M., Kacem, A. H., and Drira, K. (2005). Evaluation and comparison of adl based approaches for the description of dynamic of software architectures. In *ICEIS (3)*, pages 189–195.

Kandé, M. M. and Strohmeier, A. (2000). Towards a uml profile for software architecture descriptions. In *Proceedings of the 3rd international conference on The unified modeling language: advancing the standard*, UML'00, pages 513–527, Berlin, Heidelberg. Springer-Verlag.

Le Métayer, D. (1998). Describing software architecture styles using graph grammars. *IEEE Trans. Softw. Eng.*, 24:521–533.

Loulou, I., Kacem, A. H., Jmaiel, M., and Drira, K. (2004). Towards a unified graph-based framework for dynamic component-based architectures description in z. *Pervasive Services, IEEE/ACS International Conference on*, 0:227–234.

Luckham, D. C., Kenney, J. J., Augustin, L. M., Vera, J., Bryan, D., and Mann, W. (1995). Specification and analysis of system architecture using rapide. *IEEE Trans. Software Eng.*, 21(4):336–355.

Medvidovic, N., Rosenblum, D. S., Redmiles, D. F., and Robbins, J. E. (2002). Modeling software architectures in the unified modeling language. *ACM Trans. Softw. Eng. Methodol.*, 11:2–57.

Medvidovic, N. and Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *IEEE Trans. Softw. Eng.*, 26:70–93.

OMG (2005). *Unified Modeling Language Specification 2.0: Superstructure.* OMG doc. formal/05-07-04.

Oquendo, F. (2006). π-method: a model-driven formal method for architecture-centric software engineering. *SIGSOFT Softw. Eng. Notes*, 31:1–13.

Roh, S., Kim, K., and Jeon, T. (2004). Architecture modeling language based on uml2.0. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, APSEC '04, pages 663–669, Washington, DC, USA. IEEE Computer Society.

Rozenberg, G., editor (1997). *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations.* World Scientific.

Selonen, P. and Xu, J. (2003). Validating uml models against architectural profiles. *SIGSOFT Softw. Eng. Notes*, 28:58–67.

Sharrock, R., Monteil, T., Stolf, P., Hagimont, D., and Broto, L. (2010). Non-intrusive autonomic approach with self-management policies applied to legacy infrastructures for performance improvements. *International Journal of Adaptive, Resilient and Autonomic Systems*, 2:19.

Taylor, R. N., Medvidovic, N., Anderson, K. M., Whitehead, Jr., E. J., and Robbins, J. E. (1995). A component- and message-based architectural style for gui software. In *Proceedings of the 17th international conference on Software engineering*, ICSE '95, pages 295–304, New York, NY, USA. ACM.

Zhou, Y., Pan, J., Ma, X., Luo, B., Tao, X., and Lu, J. (2007). Applying ontology in architecture-based self-management applications. In *Proceedings of the 2007 ACM symposium on Applied computing*, SAC '07, pages 97–103, New York, NY, USA. ACM.