Correctness by Construction and Style Preserving Reconfigurations of Distributed Systems.

Cédric Eichler, Patricia Stolf, Thierry Monteil, Khalil Drira

Correctness by Construction and Style Preserving Reconfigurations of Distributed Systems.

Cédric Eichler[123], Patricia Stolf[13], Thierry Monteil[23], and Khalil Drira[23]

[1] IRIT; F-31062 Toulouse, France. eichler@irit.fr, stolf@irit.fr
[2] CNRS; LAAS; F-31077 Toulouse, France. eichler@laas.fr, monteil@laas.fr, drira@laas.fr
[3] Univ de Toulouse, UPS F-31400, INSA, F-31400, UTM F-31100, Toulouse, France.

Abstract. In distributed systems and dynamic environments, software architectures may evolve. A crucial issue when conducting system evolutions is to maintain the system in a consistent and functional state. Based on formal proofs in design-time, correctness by construction has recently emerged to efficiently guarantee system coherency.
This article proposes a new method for the construction and specification of correct by construction system reconfigurations. Such transformations are characterized by graph rewriting rules that necessarily preserve the coherency of a system. We firstly propose operators on graph transformations and show that they conserve their correctness. Given a system specified by a graph grammar, these operators then serve to construct and characterize a set of correct transformations. We show in particular that any correct configuration can be reached starting from any other one without inconsistent intermediate step, using these transformations only.

Keywords: Correctness by Construction, System Reconfigurations, Graph Rewriting, Dynamic Software Architecture

1 Introduction

Dynamic software architectures are studied in order to handle adaptation in autonomic distributed systems, coping with new requirements, new environments, and failures. By their very nature, 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. This scope is characterized by an architectural style, qualifying what is correct and what is not. Once this distinction made, system transformations themselves must be specified to depict their applicability conditions and effects. A crucial undesirable implication of these evolutions is a potential loss of correctness, the system withdrawing from the scope of consistency.

Formal unambiguous methods are necessary to study the consistency of a system at a given time (i.e., its compliance to a specified architectural style). Several ways of doing so have been developed in the literature. The most immediate approach, checking the consistency of the system at run-time, may lead to combinatorial explosions and the necessity of roll-backs if it is discovered that the system is in an inconsistent state. To efficiently tackle correctness in the scope of dynamic reconfiguration, correctness by construction [1] through formal approaches have emerged [2, 3]. *Based on formal proofs and reasoning in design-time, they guarantee the correctness of a system, requiring little or no verifications in run-time.* A way to achieve such proofs is to investigate the properties of transformations with regard to consistency preservation, so as to ensure that if a transformation is applicable on a correct configuration its result is another correct configuration. A transformation satisfying this property is then considered correct. Note that this notion of correctness is related to a certain style (a rule correct w.r.t. a style is not necessarily correct w.r.t. another). Conceptually, this means that any evolution characterized by a correct transformation can be safely triggered without worrying about the consistency of the resulting configuration.

Graph grammars constitute an expressive formalism for the characterisation of architectural styles. In particular, they offer a generative definition of the scope of correctness, where a set of graph rewriting rules called production rules axiomatically satisfy the criteria of correctness for the specified system. The approach presented in this paper consists in exploiting *three operators on graph rewriting rule preserving the properties of consistency preservation* (i.e., transformation correctness). Given some graph grammar, and thus a set of production rules, these operators can be used *to construct a set of correct transformations.* Evolutions specified by this set are sufficient to *reach any correct instance of the style starting from any other one without any inconsistent intermediate step.*

This paper present a new method for the construction and specification of correct transformations. To achieve this aim, the key contributions of this paper are :

1. The specification of two operators on graph transformations (specialization and composition).
2. The proof that these two operators conserve the correctness of transformations.
3. The stipulation of the hypothesises under which inversion of graph transformations preserve their correctness.
4. The characterization, for any given grammar (that satisfies the one hypothesis required for inversion to preserve correctness), of a set of correct transformations ensuring that :
   - Any instance of the style may be reached starting from any other.
   - Reconfigurations from a correct instance to another can be done without any inconsistent intermediate step.

Vocabulary and concepts from category theory is willingly ignored in this paper. Rather, a low-level, implementation-appropriate, view is adopted.

Section 2 introduces aspects of model transformations and approaches for correctness verification. Key concepts related to partially instantiated graphs, their relationships and transformations are introduced in Sec. 3. Based on those, said section presents the formal specification of architectural styles using graph grammars. Operators on transformations conserving their correctness are defined in Sec. 4. Section 5 shows that, starting from the set of correct transformations comprised in a given grammar, these operators can be used to generatively characterize a set of correct transformation. Besides, it establishes the property of configuration reachability using only correct transformation from the previously built set.

2 Related Works

Methodologies for correctness validation of evolving systems can be classified within three categories :

- *Model checking*, that consists in validating in run-time the whole system by verifying that some properties are met. This technique is very time-consuming and may lead to combinatorial explosions. Furthermore roll-backs may have to be applied if it is discovered that the system is in an inconsistent state.
- *Transformation checking* [4, 5], consists in verifying in run-time that a transformation do not introduce any inconsistency. While it is generally more effective than the previous solution, roll-backs are not dismissed. In real-life systems, roll-backs induce a loss of time, energy, and resources and should be avoided.
- *Correct-by-Construction Transformations* [2, 3, 6], consists in guaranteeing in design time that a transformation necessarily produce a correct state. Not only does this method implies few to no reasoning in design-time, but it also completely discard roll-backs.

There exists also various taxonomy for transformations themselves, each different kind having different purposes. They are classified :

- Depending on their impact on the model used to represent a transformation.
  - Exogenous transformations [2, 3, 6] imply a change of model but do not modify the system (or its properties). They are used to generate code or to transform a graphical model into a more formal one, for example.
  - Endogenous transformations [4, 5, 7] remains the model invariant but change the state of a system (e.g., its architecture, its behaviour...).
- Depending on their impact on the considered level of abstraction.
  - Vertical transformations [2, 3, 6, 7] symbolize refinement that modify the granularity and the level of precision with which the system is represented.

- Horizontal transformations [4, 5] are usually related to either model changes or evolutions of adaptive systems.

Endogenous horizontal transformations, called reconfigurations, typically represent system changes and adaptations.

In turn, the notion of correctness may vary depending on the transformations' nature. It may be the preservation of the system behaviour [2, 3], for code generation of model modification, for example. In some other case, correctness is linked to the presence, the absence or the the conservation of some property [5]. Architectural styles are particularly relevant when considering dynamic system. In this context, consistency is synonym of the compliance to a style [4].

Furthermore, and unlike [4], we consider transformations as rewriting rules solely and decorrelate them from graphs they are applied to. In this way, correctness is a property of the rule only, and transformations are valid for the whole range of valid graphs. To the best of our knowledge, the method presented in this paper is the first to guarantee correctness-by-construction for reconfigurations w.r.t. an architectural style allowing to reach any of its instance.

3 Preliminaries, Graph Background.

The following offer a quick overview of the formal approach adopted in this paper. Firstly, variable attributed graphs are introduced, as well as relationships between them based on patterns matching through attributed graph morphisms. These relations then serve as a basis for the definition of various graphs transformations and rewriting techniques. Transformations themselves intervene in the definition of a software architecture, as finally seen.

3.1 Attributed Graphs

The state of a system at a given time can be modelled by a conceptual **graph**. Following the commonly used conventions for standard graphical descriptions, one considers that vertices (V) represent services or architectural components and edges ($E \subseteq V^2$) correspond to their related connections, interdependencies or relationships. Each element el of the graph (i.e., its vertices and edges) is associated with an arbitrary number of attributes Att^{el} representing any relevant property or information. Each attribute might be either constant or variable. It is a couple composed by its value Att_i^{el} and its domain of definition $DAtt_i^{el}$ (the set of its possible values, $Att_i^{el}\ DAtt_i^{el}$). An **graph with constant and variable attributes** is noted $G = (V, E, Att)$.

To distinguish between constant and variable attributes in the examples, a constant attribute will be noted within quotation marks. Furthermore, we impose that two variable attributes from two disjoint graphs can not have the same name.

3.2 Pattern Matching and Relationships between Graphs

In order to find graph-like patterns in a context where attributes may be variable, the notion of element (vertices and edges) identification has to be defined. When trying to identified two elements, theirs attributes are matched two at a time w.r.t. the order of their occurrence (i.e., the i-th attributes of the first and second are associated with one another). Two elements are **unifiable** if (1) they have the same number of attributes and (2) matched attributes have the same domain of definition.

Element unifications thus induce attribute associations that can be seen as an equivalence relation (noted $\sim$)over the set of considered attributes. The resulting setoid is called a **set of identifications**. Such a set is considered **coherent** if each of the induced equivalence class contains at most one constant. It simply means that no variable has been directly or transitively identified with two different constants, and that the overall attribute matching is correct.

An **affectation** is the function impacting attributes identifications. Each occurrence of a variable is substituted with the representative of its equivalence class. If the class contains a constant, it is its representative, else the representative is chosen arbitrarily.

The existence of a homomorphism between two graphs formalizes the presence of a pattern similar to the first graph within the second. A **homomorphism** h between two attributed graphs G and $\overline{G}$ is defined as an injective function f from the vertices of G to those of $\overline{G}$ that preserves the edges [8] (if there is an edge between two vertexes in G, there is an edge between their images in $\overline{G}$). In addition, associated vertexes and edges have to be unifiable and the resulting set of identification $\tilde{I}$ has to be consistent. An homomorphism is characterized by f and I, a consistent set of identification such as $I \supseteq \tilde{I}$. Due to space shortage, the formal definition and an example of graph homomorphisms are detailed in appendix. By notational abuse, any function $\mathrm{f} : \mathrm{A} \to \mathrm{B}$ is assimilated to its canonic extension $\mathrm{f} : \mathrm{A} \cup \mathrm{A}^2 \to \mathrm{B} \cup \mathrm{B}^2$ such as $\forall\ (a,\tilde{a}) \in \mathrm{A}^2$, $\mathrm{f}((a,\tilde{a})) = (\mathrm{f}(a), \mathrm{f}(\tilde{a}))$.

In addition of graph homomorphisms, we consider the notion of graph compatibility. The idea is to consider two induced sub-graphs and to associate them, not through any morphism, but a weaker condition : if there exists an edge between two vertices of the first graph there does not need to be an edge between their images, but if there is one, then those two edges have to be unifiable. Due to space shortage, the definition of compatibility is presented in appendix.

Example 1. Figure 1 shows an example of two compatible graphs. For readability sake, the attributes of the edges have not been represented and will all be considered equals (and thus disregarded).

Let V_S be the set of vertices named 1, 2, and 3 in the figure and $\mathrm{V'}_S$ be the set of vertices named $2'$, $3'$ and $4'$. The function $f : \mathrm{V}_S \to \mathrm{V}'_S$ associating the vertices named 1 to $2'$, 2 to $4'$, and 3 to $3'$ induces the (coherent) set of identification $I = \{a, b, c, x, y, \text{“}1''\text{, “}2''\}$ with $a \sim x$ and $b \sim 2$. G and G' are (f,

I, V_S, V'_S)-compatible.

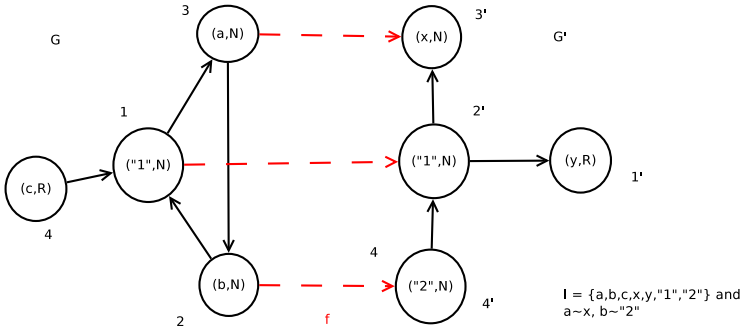


Fig. 1. two compatible graphs

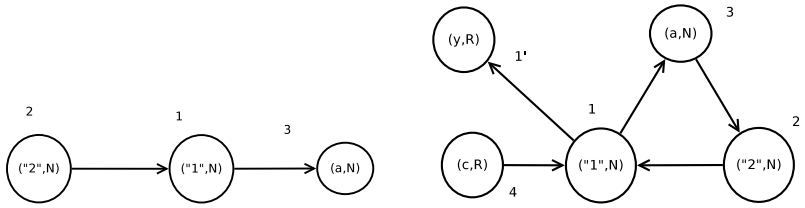
3.3 Graph Transformations

The occurrence of a pattern within a graph or a relation between two graphs, grant the possibility of applying graphs transformations. Since configurations of a system can be represented using graphs, graph transformations are used to model their evolutions.

Before introducing graph rewriting rules, let's consider two binary operator on graphs, **restriction** $\uparrow$ and **expansion** $\downarrow$. These lasts are similar to classical intersection and union, respectively. The difference arises from the fact that a similarity shall be found rather than a strict equality between elements of the graphs. Identifying analogous elements or sub-graphs is tackled by the notion of compatibility previously defined. Restriction and expansion thus depend on a graph compatibility, and are similarly characterized by an injective function between two sets of vertices and an affectation. Due to space shortage, formal definitions of these two operators are reported in appendix.

Example 2. With G, G', f, Aff, V_S and V'_S defined in the example 1, the result of $G\downarrow_{(f,I,V_S,V'_S)}G'$ is represented in Fig. 2(a). The result of $G\uparrow_{(f,I,V_S,V'_S)}G'$ is represented in Fig. 2(b).

Remark 1. For any (f, Aff, V_S, V'_S)-compatible graphs G and G' and any sub-graphs G_{sub} and G'_{sub},
$G_{sub}\downarrow_{(f,I,V_S,V'_S)}G'_{sub} \rightarrow G_{sub}$ and
$G_{sub}\downarrow_{(f,I,V_S,V'_S)}G'_{sub} \rightarrow G'_{sub}$. Similarly, $G_{sub} \rightarrow G_{sub}\uparrow_{(f,I,V_S,V'_S)}G'_{sub}$ and
$G'_{sub} \rightarrow G_{sub}\uparrow_{(f,I,V_S,V'_S)}G'_{sub}$.



(a) restriction, G$\downarrow_{(f,I,V_S,V'_S)}$G' (b) expansion, G$\uparrow_{(f,I,V_S,V'_S)}$G'

Fig. 2. graph restriction (intersection) and expansion (union)

Graph rewriting is a well-studied domain where a rule describes both a graph transformation and the circumstances under which it may be applied. The approach used in this paper is based on the Single Pushout (SPO) [9], where a rule is characterized by two graphs (L,R), respectively called left- and right-hand side, alongside a partial morphism m from L to R. For clarity sake, we consider rules satisfying $L \cap R \neq \emptyset$, so that m is implicit and induced by the identity function over $L \cap R$ (noted K). In addition, transformations are given the possibility to update the value of attributes of the graph on which they are applied. An example of a rule and its application is provided in appendix. We assume the canonic notations where for any set S, its cardinality is noted |S| and an interval of integer between a and b is noted $[|a, b|]$.

Definition 1. *(**Graph rewriting rule**) A graph rewriting rule is a 3-tuple (L, R, OPs) where L = (V_L, E_L, Att_L) and R = (V_R, E_R, Att_R) are two graphs. OPs is a set of triples OP = (el, i, op), where el ∈ $V_K \cup E_K$, i ∈ [| 1, |$(Att_K)^{el}$| |], and op is an unary invertible operation on $(Datt_K)_i^{el}$ under which $(Datt_K)_i^{el}$ has closure.*

A rule is applicable on a graph G = (V, E, Att) if there is a homomorphism h = (f, I) : L → G.

Its application consists in (1) erasing the image of L\K and deleting the potential dangling edges. (2) Adding an isomorph copy of R\K integrating the affectation obtained with h. (3) Conducting the specified updates of attributes.

The following notations will be adopted :

1. 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.
2. r_2_h_2.r_1_h_1(G) is the result of the sequence of rewriting consisting in applying r_2 in regard of the matching h_2 to the result of r_1 applied to G with the matching h_1.

3.4 Characterizing Architectural Style

Inspired from Chomsky's generative grammars, graph grammars [8] constitute an expressive formalism for describing dynamic structures. In this paper, architectural styles are characterized by such grammars. The correctness of the design

(i.e. of the grammar) is not questioned and defines the scope of acceptable configurations.

Definition 2. *(**Graph Grammar**)A graph grammar is defined by the 4-tuple (AX, NT, T, P) where AX is the axiom, NT is the sets of non-terminal arch-vertices or archetypes of vertices, T is the set of terminal arch-vertices or archetypes of vertices, and P is the set of graph rewriting rules (or productions) belonging to the graph grammar.*

Definition 3. *(**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. If it does not contain any vertex unifiable with an arch-vertex from NT, it is said to be **consistent**.*

We consider in the following that an instance of the style is a correct configuration whether it is consistent or not. Restricting the notion of correctness to consistent instances would only require to verify whether a correct rule introduces a non terminal vertex.

4 Three Operators Preserving Transformations Correctness

The generative definition of the architectural style is at the very core of the work proposed here. By very definition, any production rule is correct by construction. This means that the specification of a style also provides an initial set of correct transformations. Starting from this original set, we wish to build other correct transformations. To do so, this section introduces operations on transformations and show that they preserve transformation correctness.

4.1 Specialization

The first operation introduced in this paper is rule specialization. It consists in strengthening the applicability condition of a rule and/or narrowing the scope of its possible results. A possible use is to restrict the application of a rule to a particular context or to an entity with a specific identifier (e.g., a component that has been reported as faulty) or nature, for example.

Definition 4. ***Specialization*** *A rule q = (L_q, R_q, OPs_q) is said to be a specialization of p = (L_p, R_p,OPs_p) if and only if each of the following conditions is met.*

1. *There is a homomorphism h_L[1] = (f_L, I_L) : $L_p \xrightarrow{h_L} L_q$ such as the elements I_L are attributes of L_q and L_q is invariant for the affectation induced by I_L[2].*

[1] This homomorphism can be an identity alongside some set of identification.
[2] Hence, h(L_q) has necessarily less free variables than L_p.

2. *There is a homomorphism* $h_R = (f_R, I_R) : R_p \xrightarrow{h_R} R_q$ *such as the elements* I_R *are attributes of* R_q, R_q *is invariant for the affectation induced by* I_R *and* $\forall v \in V_{R_p}, v \in V_{L_p} \implies f_R(v) = f_L(v)$.
3. $\forall\ el \in V_{L_q} \cup E_{L_q}$, $el \notin f_L(V_{L_p}) \cup f(E_{L_p}) \implies el \in V_{R_q} \cup E_{R_q}$.[3]
4. $\forall\ el \in V_{R_q} \cup E_{R_q}$, $el \notin f_R(V_{R_p}) \cup f_R(E_{R_p}) \implies el \in V_{L_q} \cup E_{L_q}$.[4]
5. $OPs_q = OPs_p$.

Lemma 1. *For any graph G, any graph rewriting rule p and any specialization q of p, if there exist a homomorphism h such as* $L_q \xrightarrow{h} G$ *then there exists a homomorphism* $\bar{h}$ *such as* $L_p \xrightarrow{\bar{h}} G$ *and* $q\_h(G) = p\_\bar{h}(G)$.

Proof. Remember that a homomorphism is characterized among others by a consistent set of identifications that includes the identifications resulting from the actual element unifications. For any graph G, let's suppose that there exist a homomorphism $h = (f, I)$ such as $L_p \xrightarrow{h} G$.

$L_q \rightarrow G \implies L_p \rightarrow G$ since $L_p \rightarrow L_q$. In particular, let $\bar{h} = (f \circ f_L, I \cup I_L \cup I_R)$. Since I_L and I_R are integrated within L_q and R_q, $I \cup I_L \cup I_R$ can not be inconsistent. Hence, $\bar{h}$ is an homomorphism from L_p to G. Thanks to the third condition, the application of p to G w.r.t. $\bar{h}$ can not leads to the apparition of a dangling edge that would not have been deleted by the application of q (since any vertex deleted by p is deleted by q). It is immediate that $q\_h(G) = p\_\hat{h}(G)$.

Theorem 1. *A specialization of a correct (w.r.t. some architectural style) graph rewriting rule is correct (w.r.t. said style).*

Proof. Let G be a graph representing a consistent configuration of some architectural style, p a graph rewriting rule p correct w.r.t. said style, and q a specialization of p.

If q is applicable to G w.r.t. h, according to lemma 1 there exists a homomorphism $\bar{h}$ such as p is applicable to G and $q\_h(G) = p\_\bar{h}(G)$. By hypothesis and since p is correct, $p\_\bar{h}(G)$ is a correct instance of the style. Hence $q\_h(G)$ is consistent.

4.2 Composition

Compositionality of graph transformation depends on the formalism used for their specification [10]. It is usually employed to enable re-usability of rules and

[3] This means that any element deleted during an application of q is deleted during an application of p; i.e. any element of L_q that is not an image of an element of L_p by f_L is invariant w.r.t. the application of q.

[4] This means that any element added during an application of q is added during an application of p; i.e. any element of R_q that is not an image of an element of R_p by f_R is invariant w.r.t. the application of q.

to decompose rules, for better understanding and scalability [11, 12]. Remember that production rules can includes non-terminal vertexes with no "real-life" value. Consequently, composition can also be used in the context of this paper to skip inconsistent instances of the style. Yet, there exists, to the best of our knowledge, no low-level definition of composition for SPO-expressed rules. For rules expressed in the SPO formalism including variable attributes and operators, composition exists but is not unique and depends on compatibilities between parts of the rules, as defined below.

Definition 5. *(**Graph rewriting rule composition considering a specific compatibility**) For any couple of graph rewriting rules (p,q) and any compatibility C = (f, I, V $\subseteq$ V_{L_p}, V'$\subseteq$ V_{R_q}) such as L_p and R_q are C-compatible.*[5] *Let G be the sub-graph of L_p induced by V and let G' be the sub-graph of R_q induced by V'.*

If (H1) $\forall\ v \in V_G, \exists\ \tilde{v} \in V_{L_p}$ *such as* $(v, \tilde{v}) \in E_{L_p} \vee (\tilde{v}, v) \in E_{L_p} \implies f(v) \in K_q$ *and*
(H2) $\forall (v, v') \in V_G,\ f(v, v') \notin (K_q)^2 \implies ((v, v') \in E_{L_p} \implies f((v, v')) \in E_{R_q})$
then p and q can be composed w.r.t. C and $p \circ_C q$ *is the rewriting rule described by :*

1. *Let* $r_1 = (Aff_I(G),\ G \downarrow_{(f, Aff, V_G, f(V_G))} K_q, \emptyset)$ *and let* $M = r_1\_(id, Aff_I)(L_p)$.[6]
   $\boldsymbol{L}_{p \circ_C q} = M \uparrow_{(f, I, V_{(G \downarrow_{f, Aff, V, V'} K_q)}, V_{K_q}} L_q$.
2. *Let* $r_1' = (Aff_I(G'),\ K_p \downarrow_{(f, I, V, V')} G', \emptyset)$ *and* $M' = r_1'\_(id, Aff_I)(R_q)$.[7]
   $\boldsymbol{R}_{p \circ_C q} = R_p \uparrow_{(f, I, V_{(G \downarrow_{f, Aff, V, V'} K_q)}, V_{K_q}} M'$.

3. $\boldsymbol{OPs}_{p \circ_C q} = OPs_p \cup OPs_q$

Lemma 2. *For any graph G and any graph rewriting rule r such as there exists a couple of graph rewriting rule (p,q) and a compatibility C with* $r = p \circ_C q$, *if r is applicable to G w.r.t. h, then there exists a couple of homomorphism* $(\bar{h}, \tilde{h})$ *such as q is applicable to G w.r.t.* $\tilde{h}$, *p is applicable to* $q\_\tilde{h}(G)$ *w.r.t.* $\bar{h}$, *and* $r\_h(G) = p\_\bar{h}.q\_\tilde{h}(G)$.

Proof. Let $C = (f, I, V, V')$ and h be (f', I').

According to remark 1, there exists $\acute{h} = (\acute{f}, \acute{I})$ such as $L_q \xrightarrow{\acute{h}} L_r$. By hypothesis, $L_r \xrightarrow{h} G$. Hence $\tilde{h} = (f'|_{\acute{f}(V_{L_q})} \circ \acute{f},\ \acute{I} \cup I')$ is such as $L_q \xrightarrow{\tilde{h}} G$.

By definition of graph rewriting rules, there exists a homomorphism $\hat{h} = (\hat{f}, I')$ such as $R_q \xrightarrow{\hat{h}} q\_\tilde{h}(G)$. According to remark 1, there exists $\grave{h} = (\grave{f}, \grave{I})$ such

[5] Note that there exists at least one compatibility since V' can be empty (in which case the rule would be applied on independent parts of the graph).
[6] M is, modulo an affectation, L_p deprived of the part of G not identified with K_q via f (the part of G added when q is applied).
[7] M' is, modulo an affectation, R_q deprived of the part of G' not belonging to $f(K_p)$ (the part of G' suppressed when p is applied).

as $M \overset{\grave{h}}{\rightarrow} L_r$. Let $\bar{f} : V_{L_p} \rightarrow V_{q\_\tilde{h}(G)}$ be such as $\forall v \in V_{L_p}$, $\bar{f}(v) = f' \circ \grave{f}(v)$ if $v \in V_M$ and $\hat{f} \circ f(v)$ else. By construction, $\bar{h} = (\bar{f}, I \cup I' \cup \acute{I})$ is a homomorphism from L_p to $q\_\tilde{h}(G)$ if it preserves edges. Thanks to H1, $\forall (v, v') \in E_{L_p}$, (1) (v, v') $\in (V_M)^2 \vee$ (2) $((v, v') \in V_G \wedge f(v, v') \notin (K_q)^2)$.

(1) Since $M \overset{\grave{h}}{\rightarrow} L_r \overset{h}{\rightarrow}$, $(v, v') \in (V_M)^2 \implies \bar{f}((v, v')) \in E_G$.

(2) Thanks to H2, $(v, v') \in V_G \wedge f(v, v') \notin (K_q)^2 \implies f((v, v') \in E_{R_q}$. Since $\mathrm{R}_q \overset{\hat{h}}{\rightarrow} q\_\tilde{h}(G)$, $\bar{f}((v, v')) \in E_G$.

Hence, $\mathrm{L}_p \overset{\bar{h}}{\rightarrow} q\_\tilde{h}(G)$.

Due to space shortage, we do not report here the second part of the proof that states that with the appropriate homomorphisms defined in this proof and the construction of R_r, $r\_h(G) = p\_\bar{h}.q\_\tilde{h}(G)$.

Theorem 2. *The composition of two correct (w.r.t. some style) graph rewriting rules is correct (w.r.t. said style).*

This theorem is immediate considering lemma 2

4.3 Inversion

Inversion exploits the property of reversibility of graph rewriting rules [8]. It consists in defining an opposite transformation cancelling the effect of another. Intuitively, considering for example the deployment of a new server to absorb a load peak, inversion allows the characterization of its shut-down once the load goes back to normal. Inversion is classically conducted by swap the right and left hand-side of a rule. However, this would not be enough to guarantee correctness conservation. In addition, we have to verify that as long as a transformations that did require (i.e., that could be applied only in the presence of) the component has not been "cancelled", the component can not be suppressed. When using graph rewriting, this "require" relationship translates to the presence of the component in the image of the left hand-side of the rule considering the homomorphism linked with its application.

As a consequence, we assume that each vertex possesses an attribute that is a matching counter. This can be easily automated and hidden to the user by adding, for each element *el*, a concealed attribute ATT_0^{el} in $\mathbb{N}$ that is a mute free variable, except when initialized. It is initialized at 0 (i.e., for each production rule p, for all el element of $\mathrm{R}_p \backslash \mathrm{K}_p$, $\mathrm{ATT}_0^{el} = 0$). To each production rule is appended operators that increment the counter of each element in K (i.e., for each $el \in V_K \cup E_K$, OP = (el, 0, f : $\mathbb{N} \rightarrow \mathbb{N}$ such as f(x) = x+1) $\in$ OPs).

Definition 6. ***(Inverse rule)*** *A graph rewriting rule r^{-1} is the inverse of a graph rewriting rule r if : $R_{r^{-1}} = L_r$, $L_{r^{-1}} = R_r$, and $OPs_{r^{-1}} = \{$ OP = (el, i, f) : $\exists$ $\tilde{OP} \in OPs_r$, OP = (el, i, f^{-1}) $\}$.*

Noticeably, if the inversion of a production rule is applicable on a graph, then the matching counter of each vertex that would be deleted during its application is equal to 0. Moreover, its application decrements the matching counter of each vertex in its invariant part. In addition, for any graph rewriting rule r, $(r^{-1})^{-1} = r$.

Theorem 3. *The inversion of a correct (w.r.t. some style) graph rewriting rule r is correct (w.r.t. said style) if :*

- *(H1) : there exists a "matching counter" as described previously.*
- *(H2) : for any instance of the style G, the presence of a pattern isomorph to R_r in G implies that G can be obtained starting from the axiom by applying a sequence of correct rewriting rules, one of which being r (that has introduced said pattern).*

Please note that the second hypothesis is not met for any grammar. Intuitively, for the set of rules : $p_1 : AX \rightarrow a$, $p_2 : AX \rightarrow ab$, it is possible to have a pattern corresponding to the right-hand side of a rule (a) in a word that can not be derived using said rule (ab). It is also not a property of the style (i.e., the scope of consistency) itself, but rather of its definition (i.e., the grammar). Indeed, the previous grammar can be rewritten in a way that respects this property such as : $p_{\tilde{1}} : AX \rightarrow a$ and $p_{\tilde{2}} : a \rightarrow ab$.

Proof. By hypothesis (H2), if r^{-1} is applicable to G w.r.t. some homomorphism h, then there exists a sequence of rules and homomorphisms $((r_i, h_i))_{i\in[|1,n|]}$ such as $r_n\_h_n.\dots.r_1\_h_1(AX) = G$ and there exists $k \in [|1, n|]$ such as $r_k = r$.

It is immediate that for any graph graph rewriting rule r and any graph G, if r is applicable to G w.r.t a homomorphism $\bar{h}$ then there exists a homomorphism h' such as $r^{-1}\_h'.r\_\bar{h}(G) = G$. h' is the canonical homomorphism associating $L_{r^{-1}} = R_r$ with the isomorph copy of R_r introduced while applying r on G. Consequently, if k = n, the theorem is true since $r^{-1}\_h(G) = r_{n-1}\_h_{n-1}.(\dots).r_1\_h_1(AX)$ which is by definition an instance of the style. If k < n, the idea of the proof is as follows:

According to H2, $h(L_{r^{-1}}$ has been introduced by applying r_k (i.e., r_n applied w.r.t. h_n do not introduce anything required for the application of r^{-1} w.r.t. h). Hence r^{-1} is applicable on $r_{n-1}\_h_{n-1}.(\dots).r_1\_h_1(AX)$ w.r.t. h.

Since r^{-1} does not delete any element of G match with L_{r_n} through h_n (H1), (2.a) suppressing dangling edges can not affect h_n (since the suppressed extremity should also be matched through h_n), (2.b) r^{-1} do not invalidate h_nHence, r_n is applicable to $r^{-1}\_h.r_{n-1}\_h_{n-1}.(\dots).r_1\_h_1(AX)$ w.r.t. h_n. In particular, $r_n\_h_n.r^{-1}\_h.(\dots).r_1\_h_1(AX) = r^{-1}\_h(G)$.

By conducting this reasoning until getting $r^{-1}\_h(G) = r_n\_h_n.(\dots).r^{-1}\_h.r_k.h_k.(\dots).r_1\_h_1(AX)$, we obtain $r^{-1}\_h(G) = r_n\_h_n.(\dots).r_{k-1}\_h.r_{k+1}.h_k.(\dots).r_1\_h_1(AX)$, which is by definition an instance of the style.

5 Handling Dynamism with Correct Transformations

On one hand, we have seen in Sec. 3.4 that the definition of an architectural style through a graph grammar comprises a set of correct rules called production rules. On the other, section 4 introduces unary and binary operators on transformations that preserve their correctness. These two facts immediately bring up the following questions : what do we obtain if we apply introduced operators to the set of productions? Is a dynamic system manageable using correct transformations only?

5.1 Configuration reachability

The first worthwhile property that is studied here is the capacity of reaching a configuration given an initial state. Typically, a (potentially autonomic) manager identify a desirable configuration that better some optimization criterion. Can it necessarily be reached given the current state of the system? The focus of this paper is the correctness of rules, a property linked to their application on an instance of the style. In particular here, we aim at studying the reachability of configurations using correct rules only, and have few to no information on their behaviour w.r.t. states that do not conform the style.. Thus, the possible initial states are restricted to instances of the style.

Theorem 4. *Any instance of a given graph grammar can be reached starting from any other using only production rules and their inverses.*

The proof is summarized in the next sub-section.

5.2 Avoiding inconsistent instances of the style

Instances of the style can be correct or not, depending on the existence of non-terminal vertexes within them. Such vertexes are theoretic artefacts with no "real-life" value. Inconsistent instances of the style thus carry vertex(es) that are not matched with any component of the modelled system. To avoid this discrepancy, one may wish to remain in the scope of consistent configurations and avoid inconsistent instances.

Theorem 5. *For any graph grammar (AX, T, NT, P), let Trans be the smallest set of graph rewriting rules containing P for which inversion and composition have closure.*[8] *Any consistent instance of the grammar can be reached starting from any other without any inconsistent intermediary by applying a sequence of rules in Trans.*

Proof. The idea of the proof of theorems 4 and 5 is as follows :

[8] Note that $Trans$ is an infinite set.

Firstly, for a given grammar, let's consider a graph whose vertexes represent configurations of the style. An edge model the application of a production (a rule and a homomorphisms) such as the application of the symbolized transformation to its source result in its target. For each edge (v, v') symbolizing the application of p, one can introduce an edge (v', v) representing the application of p^{-1} cancelling the application of p.

Since there exists a path from the axiom to any vertex and a path from any vertex to the axiom, it is easy to see that there exists a path from any vertex to any other one. The fact that this graph is strongly connected directly implies theorem 4.

For each path from a consistence instance to another, each sub-path that contains only inconsistent instances can be by-passed by adding to the graph a vertex representing the composition of the transformation leading to or within the set of inconsistent instances. Finally, each inconsistent configuration can be deleted, and the resulting graph is still strongly connected, giving theorem 5.

6 Conclusion

Given a graph grammar specifying an architectural style, this paper introduces a method to build correct-by-constructions transformations that are style-preserving. It is important to notice that the style axiomatically defines the scope of correctness and that its faultlessness is not questioned. Correct transformations are particularly relevant in the management of dynamic distributed systems. Their use can assure the theoretical consistency of a system without requiring any checking in run-time.

The defined method originate from the fact that a graph grammar comprise a set of axiomatically correct transformations. The first contribution of this paper is the specification of correctness-preserving operators on system transformations. Alongside the initial correct transformations, they allow the characterization of a larger (infinite) set of correct transformations. We finally prove that any correct configuration can be reached starting from any other one without any inconsistent intermediate step using transformations from the previously defined set only.

However, the style-preserving property of one of the operator (inversion) is subject to an hypothesis on the grammar. Furthermore, there exists some grammars that do not satisfy hypothesis. In a short future, we plan on further investigating this property. We are particularly interested in grammar transformations that introduce the satisfaction of the required condition. On a more practical side, it is necessary to hide the intrinsic complexity of the formalism to future users. To this end, we wish to implement a graphical tool for the creation and manipulation of transformations.

Acknowledgement The work presented in this paper has been partially funded by the ANR in the context of the project SOP, ANR-11-INFR-001.

Bibliography

[1] Gössler, G., Graf, S., Majster-Cederbaum, M., Martens, M., Sifakis, J.: Program analysis and compilation, theory and practice. Springer-Verlag, Berlin, Heidelberg (2007) 201–224

[2] Baleani, M., Ferrari, A., Mangeruca, L., Sangiovanni-Vincentelli, A., Freund, U., Schlenker, E., Wolff, H.J.: Correct-by-construction transformations across design environments for model-based embedded software development. In: Design, Automation and Test in Europe, 2005. Proceedings. (2005) 1044–1049 Vol. 2

[3] Bonakdarpour, B., Bozga, M., Jaber, M., Quilbeuf, J., Sifakis, J.: Automated conflict-free distributed implementation of component-based models. In: 2010 International Symposium on Industrial Embedded Systems (SIES). (2010) 108–117

[4] Hirsch, D., Montanari, U.: Consistent transformations for software architecture styles of distributed systems. Electronic Notes in Theoretical Computer Science **28**(0) (2000) pp.4–25

[5] Percebois, C., Strecker, M., Tran, H.N.: Rule-level verification of graph transformations for invariants based on edges' transitive closure. In Hierons, R.M., Merayo, M.G., Bravetti, M., eds.: Software Engineering and Formal Methods, Madrid, Spain, 25/09/2013-27/09/2013. Volume 8137 of Lecture Notes in Computer Science., http://www.springerlink.com, Springer (September 2013) 106–121

[6] Tounsi, M., Mosbah, M., Méry, D.: From Event-B Specifications to Programs for Distributed Algorithms. In: 22th IEEE International Conference on Enabling Technologies: Infrastructures for Collaborative Enterprises.

[7] Oquendo, F.: pi-arl: An architecture refinement language for formally modelling the stepwise refinement of software architectures. SIGSOFT Softw. Eng. Notes **29**(5) (September 2004) 1–20

[8] Rozenberg, G., ed.: Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations. In Rozenberg, G., ed.: Handbook of Graph Grammars, World Scientific (1997)

[9] Löwe, M.: Algebraic approach to single-pushout graph transformation. Theoretical Computer Science **109**(12) (1993) 181 – 224

[10] Duval, D., Echahed, R., Prost, F.: Categorical abstract rewriting systems and functoriality of graph transformation. ECEASST **41** (2011)

[11] Rensink, A.: Compositionality in graph transformation. In Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P., eds.: Automata, Languages and Programming. Volume 6199 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2010) 309–320

[12] Balogh, A., Varr, D.: Pattern composition in graph transformation rules. In: European Workshop on Composition of Model Transformations, Bilbao, Spain (2006)

Appendix

Definition 7. *(**Graph homomorphism with variable label**) There exists an homomorphism h between two graphs G and G' such as G = (V, E, Att) and G' = (V', E', Att'), noted $G \to G'$ or $G \xrightarrow{h} G'$, if and only if there is a consistent set of identification I and an injective function $f : V \to V'$, such as :*

1. $\forall\ (v_i, v_j) \in V^2, \forall\ (v'_i, v'_j) \in (V')^2, f(v_i) = v'_i \wedge f(v_j) = v'_j \implies (v_i, v_j) \in E \implies (\ (v'_i, v'_j) \in E' \wedge (\ (v_i, v_j)$ *is unifiable with* $(v'_i, v'_j))$.
2. $\forall\ v \in V, \forall\ v' \in V', v' = f(v) \implies v$ *and* v' *are unifiable.*
3. $I \supseteq \bar{I}$, *where* $\bar{I}$ *is the set of identifications resulting from element unifications.*

The resulting homomorphism is characterised by the couple (f, I).

Definition 8. *(**Compatible graphs**) Two graphs G = (V, E, Att) and G' = (V', E', Att') are said to be compatible or (f, I, V_S, V'_S)-compatible if and only if there exists $V_S \subseteq V$, $V'_S \subseteq V'$, a consistent set of identification I and a bijective function $f : V_S \to V'_S$ such as :*

1. $\forall\ (v_1, v_2) \in {V_S}^2, \forall\ (v'_1, v'_2) = (f(v_1)\ , f(v_2)) \in {V'_S}^2 \implies ((v_1, v_2) \in E \wedge (v_1', v_2') \in E') \implies (v_1, v_2)$ *is unifiable with* (v_1', v_2').
2. $\forall\ v \in V_S, \forall\ v' \in V'_S, v' = f(v) \implies v$ *and* v' *are unifiable.*
3. $I \supseteq \bar{I}$, *where* $\bar{I}$ *is the set of identifications resulting from element unifications.*

Definition 9. *(**Restriction** $\downarrow$) For any G and G' (f, Aff, V_S, V'_S)-compatible graphs and any sub-graphs $G_{sub} = (V_{G_{sub}}, E_{G_{sub}}, Att_{G_{sub}})$, $G'_{sub} = (V_{G'_{sub}}, E_{G'_{sub}}, Att_{G'_{sub}})$,*

1. *let* $V = \{\ v \in V_S \cap V_{G_{sub}} \mid f(v) \in V'_S \cap V_{G'_{sub}}\ \}$,
2. *let* $E = \{\ (v,\tilde{v}) \in V^2 \cap E_{G_{sub}} \mid (f(v_i),f(v_j)) \in E_{G'_{sub}}\}$,
3. *let* $Att = \bigcup_{el \in V \cup E}\ (Att_{G_{sub}})^{el}$.

The restriction relation is defined by $G_{sub} \downarrow_{(f,Aff,V_S,V'_S)} G'_{sub} = Aff((V, E, Att))$.

Definition 10. *(**Expansion** $\uparrow$) For any G and G' (f, Aff, V_S, V'_S)-compatible graphs and any sub-graphs $G_{sub} = (V_{G_{sub}}, E_{G_{sub}}, Att_{G_{sub}})$, $G'_{sub} = (V_{G'_{sub}}, E_{G'_{sub}}, Att_{G'_{sub}})$,*

1. *let* $V = \{\ v \mid (v \in V_{G_{sub}}) \vee (v \in V_{G'_{sub}} \wedge v \notin f(V_S \cap V_{G_{sub}}))\ \}$,
2. *let* $E = \{\ (v,\tilde{v}) \in V^2 \mid (v,\tilde{v}) \in E_{G_{sub}} \cup E_{G'_{sub}} \vee (v \in V_S \wedge (f(v),\tilde{v}) \in E_{G'_{sub}}) \vee (\tilde{v} \in V_S \wedge (v,f(\tilde{v})) \in E_{G'_{sub}}) \vee ((v, \tilde{v}) \in V_S^2 \wedge (f(v),f(\tilde{v})) \in E_{G'_{sub}})\ \}$,

3. let $Att = \{\ Att^{el} \in Att_{G_{sub}} \cup Att_{G'_{sub}} \mid el \in V \cup E \wedge ($
$(\exists\ \tilde{el} \in V_{G_{sub}} \cup V_{G'_{sub}} \cup E_{G_{sub}} \cup E_{G'_{sub}},\ el = \tilde{el}) \vee ($
$el = (v, \tilde{v}) \in E \setminus (E_{G_{sub}} \cup E_{G'_{sub}}) \wedge ($
$(v \in V_S \wedge (f(v),\tilde{v}) \in E_{G'_{sub}} \wedge Att^{el} = (Att_{G'_{sub}})^{(f(v),\tilde{v})}) \vee$
$(\tilde{v} \in V_S \wedge (v,f(\tilde{v})) \in E_{G'_{sub}} \wedge Att^{el} = (Att_{G'_{sub}})^{(v,f(\tilde{v}))}) \vee$
$((v, \tilde{v}) \in V_S^2 \wedge (f(v),f(\tilde{v})) \in E_{G'_{sub}} \wedge Att^{el} = (Att_{G'_{sub}})^{(f(v),f(\tilde{v}))}$
$)))\}$.

The expansion relation is defined by :
$G_{sub} \uparrow_{(f,Aff,V_S,V'_S)} G'_{sub} = Aff((V, E, Att))$.

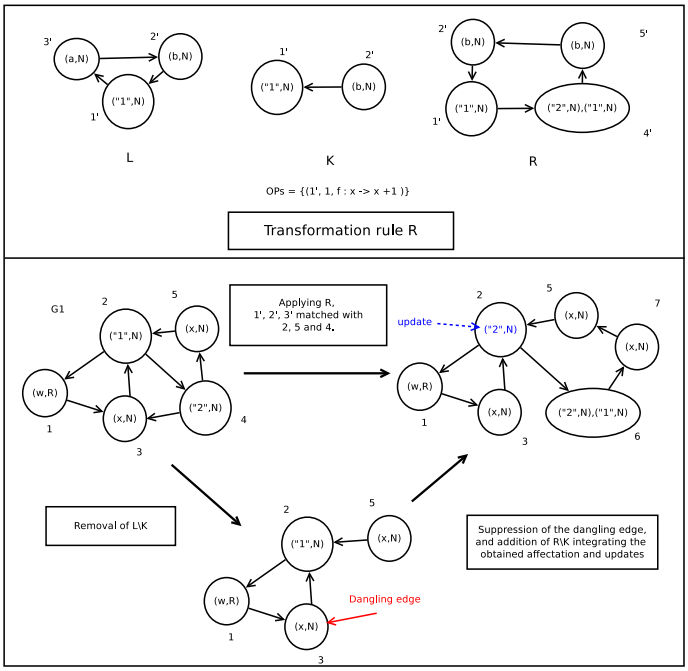


Fig. 3. An example of graph transformation

Example 3. Figure 3 offers an example of how a transformation is handled in this paper. To lighten the figure, the attributes of the edges have not been represented and will all be considered equals. Considering that L and G1 are homomorph and that the suppression of the edge 4 would not introduce any dangling edge, the transformation R can be applied to G1. The graph corresponding to the Del zone is removed and an isomorph copy of the Add zone is then added.