Correctness by construction and style preserving reconfigurations of system of systems

Cédric Eichler, Khalil Drira, Thierry Monteil, Patricia Stolf

Correctness by Construction and Style Preserving Reconfigurations of System of Systems

Cédric Eichler
LIFO EA 4022, INSA Centre Val de Loire
Bourges, France
cedric.eichler@insa-cvl.fr

Khalil Drira
LAAS-CNRS
Toulouse, France
drira@laas.fr

Thierry Monteil
LAAS-CNRS, INSA Toulouse
Toulouse, France
monteil@laas.fr

Patricia Stolf
IRIT, Université Jean Jaurès
Toulouse, France
stolf@irit.fr

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. As system complexity rises, manual checking or exhaustive model checking may be too time- and resource-consuming, lacking in scalability. This is particularly true with system of systems. 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 can be leveraged to construct correct transformations. We show in particular that any correct configuration can be reached starting from any other one without inconsistent intermediate step, using such transformations only.

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.

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. 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.* 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.*

The key contributions of this paper are :

(1) The specification of two operators on graph transformations (specialization and composition) that conserves correctness of transformations.

(2) The study of inversion and the stipulation of hypotheses under which it preserves transformation correctness.
(3) The characterization, for any given grammar (for which inversion preserve correctness), of a set of correct transformations ensuring that any instance of the style may be reached starting from any other 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, transformations, and grammars are introduced in Sec. 3. Operators on transformations conserving their correctness are defined in Sec. 4. Section 5 shows that these operators can be used to generatively characterize a set of correct transformations allowing to reach any configuration from any other.

2 RELATED WORKS

There exists various kind of transformations each having different purposes. They can be loosely classified depending on their impact on the model and the level of abstraction.
Exogenous transformations [2–4] 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* [5–7] remains the model invariant but change the state of a system. *Vertical transformations* [2–4, 7], e.g. refinements, modify the granularity and the level of precision with which the system is represented. *Horizontal transformations* [5, 6] are usually related to model changes or system evolutions.
In this paper, we are interested in endogenous horizontal transformations, called reconfigurations. They typically represent adaptations in self-adaptive systems. We assume that these adaptations are guided and aim at guaranteeing the preservation of system consistency.

Methodologies for correctness validation of evolving systems can be classified within three categories. *Configuration 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* [5, 6], 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. *Correct-by-Construction Transformations* [2–4], 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.

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 [6]. Architectural styles are particularly relevant when considering dynamic system. In this context, consistency is synonym of the compliance to a style [5].

Furthermore, and unlike [5], 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 and their relationships through graph morphisms are introduced. These relations then serve as a basis for the definition of graphs transformations and rewriting techniques and, finally graph grammars.

3.1 Attributed Graphs

At a given time, the state of a system, be it a System of System (SoS) or not, can be modelled by a conceptual **graph**. Classically, vertices (V) represent services or architectural components. In SoSs, vertices represent the systems composing the SoS. 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 and is characterized by its value Att_i^{el} and its domain of definition $DAtt_i^{el}$. 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., their i-th attributes 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 induce attribute associations that can be seen as an equivalence relation (noted $\sim$) over the set of attributes. The resulting setoid is called a **set of identifications**. It is considered **coherent** if each of the induced equivalence class contains at most one constant. This means that no variable has been directly or transitively identified with two different constants.
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 that $I \supseteq \tilde{I}$. By notational abuse, any function $f : A \to B$ is assimilated to its canonic extension $f : A \cup A^2 \to B \cup B^2$ such that $\forall (a,\tilde{a}) \in A^2$, $f((a,\tilde{a})) = (f(a), f(\tilde{a}))$.

Graph compatibility associate two induced sub-graphs through 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.

Definition 1. ***(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 that :*

(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) $I \supseteq \tilde{I}$, where $\tilde{I}$ is the set of identifications resulting from element unifications.

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 be disregarded.*

Let V_S be the set of vertices named 1, 2, and 3 in the figure and V'_S be the set of vertices named $2'$, $3'$ and $4'$. The function $f : V_S \dashrightarrow V'_S$ 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.

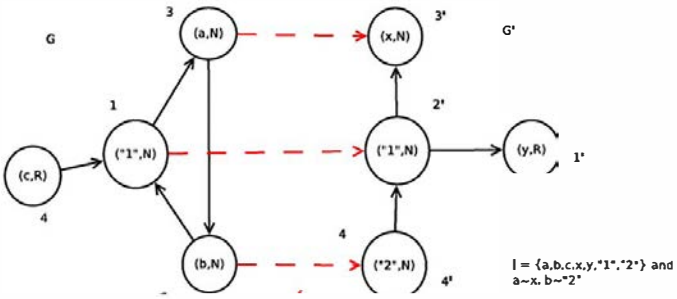

Figure 1: two compatible graphs

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.

Definition 2. ***(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 3. ***(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 :
$G_{sub} \uparrow_{(f,Aff,V_S,V'_S)} G'_{sub} = Aff((V, E, Att))$.

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},*
$G_{sub} \downarrow_{(f,I,V_S,V'_S)} G'_{sub} \to G_{sub}$ *and*
$G_{sub} \downarrow_{(f,I,V_S,V'_S)} G'_{sub} \dashrightarrow G'_{sub}$. *Similarly,* $G_{sub} \dashrightarrow G_{sub} \uparrow_{(f,I,V_S,V'_S)} G'_{sub}$ *and*
$G'_{sub} \to G_{sub} \uparrow_{(f,I,V_S,V'_S)} G'_{sub}$.

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

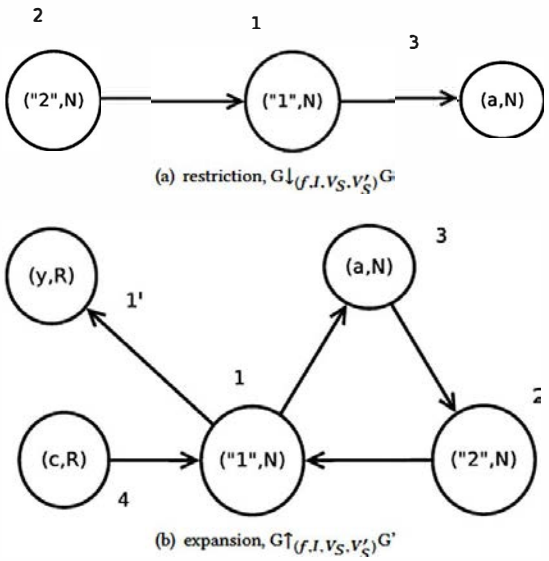

Figure 2: graph restriction (intersection) and expansion (union)

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.

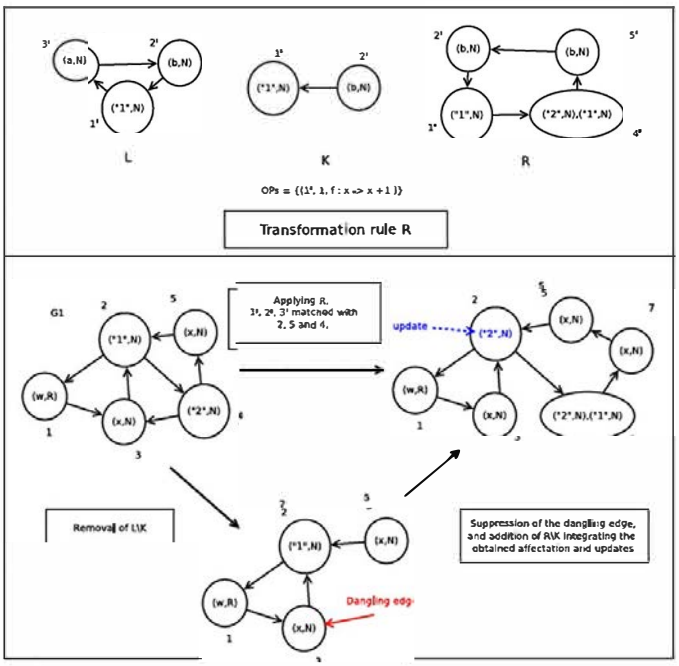

Figure 3: An example of graph transformation

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.*

DEFINITION 4. ***(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 $(Datt_K)^{el}_i$ under which $(Datt_K)^{el}_i$ 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 5. ***(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 6. ***(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 our proposal. By very definition, any production rule is correct. 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 7. ***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 This homomorphism can be an identity alongside some set of identification. = (f_L, I_L) : $L_p \xrightarrow{h_L} L_q$ such that the elements I_L are attributes of L_q and L_q is invariant for the affectation induced by I_L. 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 that 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) $\forall\, el \in V_{L_q} \cup E_{L_q}, el \notin f_L(V_{L_p}) \cup f(E_{L_p}) \implies el \in V_{R_q} \cup E_{R_q}$. 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) $\forall\, el \in V_{R_q} \cup E_{R_q}, el \notin f_R(V_{R_p}) \cup f_R(E_{R_p}) \implies el \in V_{L_q} \cup E_{L_q}$. 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.

(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 that $L_q \xrightarrow{h} G$ then there exists a homomorphism $\bar{h}$ such that $L_p \xrightarrow{\bar{h}} G$ 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 that $L_p \xrightarrow{h} G$.

$L_q \to G \implies L_p \to G$ since $L_p \to L_q$. 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 2. *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 that 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 to decompose rules, for better understanding and scalability [11, 12]. Production rules can include purely theoretical non-terminal vertexes. Consequently, composition can also be used in the context of this paper to skip inconsistent instances of the style. 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 8. *(**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 that L_p and R_q are C-compatible. 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 that $(v, \tilde{v}) \in E_{L_p} \vee (\tilde{v}, v) \in E_{L_p} \implies f(v) \in K_q$ 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). 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).
$\boldsymbol{L}_{p \circ_C q}$ = $M \uparrow_{(f, I, V_{(G \downarrow_{f, Aff, V, V'} K_q)}, V_{K_q}} L_q$.

(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). 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).
$\boldsymbol{R}_{p \circ_C q}$ = $R_p \uparrow_{(f, I, V_{(G \downarrow_{f, Aff, V, V'} K_q)}, V_{K_q}} M'$.

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

Lemma 3. *For any graph G and any graph rewriting rule r such that 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 ($\tilde{h}$, $\tilde{h}$) such that 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 that $L_q \xrightarrow{\acute{h}} L_r$. 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

that $L_q \xrightarrow{\tilde{h}} G$.

By definition of graph rewriting rules, there exists a homomorphism $\hat{h} = (\hat{f}, I')$ such that $R_q \xrightarrow{\hat{h}} q\_\tilde{h}(G)$. According to remark 1, there exists $\dot{h} = (\dot{f}, \dot{I})$ such that $M \xrightarrow{\dot{h}} L_r$. Let $\bar{f} : V_{L_p} \to V_{q\_\tilde{h}(G)}$ be such that $\forall v \in V_{L_p}$, $\bar{f}(v) = f' \circ \dot{f}(v)$ if $v \in V_M$ and $\hat{f} \circ f(v)$ else. By construction, $\bar{h} = (\bar{f}, I \cup I' \cup \dot{I})$ 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 $M \xrightarrow{\dot{h}} L_r \xrightarrow{h}$, $(v, v') \in (V_M)^2 \implies \bar{f}((v, v')) \in E_G$.
(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 $R_q \xrightarrow{\hat{h}} q\_\tilde{h}(G)$, $\bar{f}((v, v')) \in E_G$.

Hence, $L_p \xrightarrow{\bar{h}} q\_\tilde{h}(G)$.

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 4. *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 3

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 $R_p \setminus K_p$, $ATT_0^{el} = 0$). 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 : $\mathbb{N} \to \mathbb{N}$ such that f(x) = x+1) ∈ OPs).

Definition 9. ***(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) : ∃ $\tilde{OP} \in OPs_r$, 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 5. *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).*

Note that the second hypothesis is not met for any grammar. Intuitively, for the set of rules : p_1 : AX → a, p_2 : AX → 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 that : $p_{\tilde{1}}$: AX → a and $p_{\tilde{2}}$: a → 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 that $r_n\_h_n.\ldots.r_1\_h_1(AX)$ = G and there exists $k \in [|1, n|]$ such that 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 that 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) = $r_{n-1}\_h_{n-1}.(\ldots)\ .r_1\_h_1(AX)$ 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 w.r.t. h on $r_{n-1}\_h_{n-1}.(\ldots).r_1\_h_1(AX)$.

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_n.
Hence, r_n is applicable to $r^{-1}\_h.r_{n-1}\_h_{n-1}.(\ldots).r_1\_h_1(AX)$ w.r.t. h_n. In particular, $r_n_h_n.r^{-1}_h.(\ldots).r_1_h_1(AX) = r^{-1}$_h(G).

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

5 GUIDING EVOLUTIONS WITH CORRECT TRANSFORMATIONS

On one hand, we have seen in Sec. 3.4 that a graph grammar comprises a set of production rules correct by definition. On the other, section 4 introduces 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?

5.1 Configuration reachability

The first worthwhile property considered is the capacity of reaching a configuration given some initial state. Typically, a (potentially autonomic) manager identify a desirable configuration; can it necessarily be reached given the current state of the system?

Theorem 6. *Any instance of a given graph grammar can be reached starting from any other using only production rules and their inverses.*

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. To avoid this discrepancy, one may wish to remain in the scope of consistent configurations and avoid inconsistent instances.

Theorem 7. *For any graph grammar (AX, T, NT, P), let $Trans$ be the smallest -potentially infinite- set of graph rewriting rules containing P for which inversion and composition have closure. 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 6 and 7 is as follows. For a given grammar, let's consider a new kind of graph with a higher level of abstraction representing the generation process of instances of the grammar. Vertexes of such graphs represent instances of the grammar -the previous graphs- and edges model the application of a production. The existence of an edge from v to v' means that v' can be obtained from v by applying some production.

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 -by definition- 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 6.

For each path from a consistence instance to another, each subpath 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 7. □

6 CONCLUSION

Given a graph grammar specifying a system or a SoS, this paper introduces a method to build correct transformations that necessarily preserves the conformance to the grammar. Correct transformations are particularly relevant in the management of dynamic (system of) systems. Their use can ensure theoretical consistency w.r.t. guided adaptations 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. 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.

REFERENCES

[1] G. Gössler, S. Graf, M. Majster-Cederbaum, M. Martens, and J. Sifakis, "Ensuring properties of interaction systems," in *Program analysis and compilation, theory and practice*. Springer, 2007.

[2] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, U. Freund, E. Schlenker, and H.-J. Wolff, "Correct-by-construction transformations across design environments for model-based embedded software development," in *Design, Automation and Test in Europe*, 2005.

[3] B. Bonakdarpour, M. Bozga, M. Jaber, J. Quilbeuf, and J. Sifakis, "Automated conflict-free distributed implementation of component-based models," in *International Symposium on Industrial Embedded Systems*, 2010.

[4] M. Tounsi, M. Mosbah, and D. Méry, "From Event-B Specifications to Programs for Distributed Algorithms," in *22th IEEE International Conference on Enabling Technologies: Infrastructures for Collaborative Enterprises.*, 2013.

[5] D. Hirsch and U. Montanari, "Consistent transformations for software architecture styles of distributed systems," *Electronic Notes in Theoretical Computer Science*, vol. 28, pp. 4–25, 2000.

[6] C. Percebois, M. Strecker, and H. N. Tran, "Rule-level verification of graph transformations for invariants based on edges' transitive closure," in *Software Engineering and Formal Methods*, 2013, pp. 106–121.

[7] F. Oquendo, "pi-arl: An architecture refinement language for formally modelling the stepwise refinement of software architectures," *SIGSOFT Softw. Eng. Notes*, vol. 29, no. 5, pp. 1–20, 2004.

[8] G. Rozenberg, Ed., *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[9] M. Löwe, "Algebraic approach to single-pushout graph transformation," *Theoretical Computer Science*, vol. 109, no. 1âĂŞ2, pp. 181 – 224, 1993.

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

[11] A. Rensink, "Compositionality in graph transformation," in *Automata, Languages and Programming*, 2010, pp. 309–320.

[12] A. Balogh and D. VarrÃş, "Pattern composition in graph transformation rules," in *European Workshop on Composition of Model Transformations*, 2006.