Electronic Communications of the EASST
Volume 58 (2013)

Proceedings of the 12th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2013)

Annotations on Complex Patterns

Paolo Bottoni, Francesco Parisi Presicce

14 pages

Guest Editors: Matthias Tichy, Leila Ribeiro
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Annotations on Complex Patterns

Paolo Bottoni[1], **Francesco Parisi Presicce**[1]

[1] Dipartimento di Informatica, "Sapienza" Università di Roma, Italy

Abstract: Modelers of systems often want to isolate specific parts of a model to be treated as a whole, for example to protect them from accidental changes, to constrain them to specific policies, or to identify them as instances of a general pattern. In particular, we study here the case in which these parts are annotated with information from some external model. In a previous paper, we have discussed the use of annotations on individual model elements, represented as nodes in a graph; in this paper we model annotation processes involving also annotations themselves or whole configurations. To address the latter problem, we enrich the notion of graph by introducing a third sort of elements, called boxes, encompassing subgraphs, and associate them with annotations, too. We show how annotations on boxes support the modeling of complex policies, adapting the previous constructions for notation-aware rewriting to include boxes. The paper illustrates these concepts on the concrete modeling scenario of an organisation with security and temporal annotations.

Keywords: Annotations, graph configuration, box.

1 Introduction

In system modeling, the need often arises to identify specific configurations within a complex model, and refer to them as a whole. In many cases, such configurations may involve a number of model elements which is not known a priori and which may vary dynamically. Such is the case with pattern-based modeling, especially of software systems, where several software patterns, usually relying on polymorphism, may be realised by arbitrary numbers of instances, as for example in the *Strategy*, *Observer* or *Decorator* patterns. While instances of patterns can be identified on the basis of the typical relations among their elements, constructs are needed to manipulate them as distinct individual model elements, persisting beyond their usage.

The basic model of graphs, composed of nodes and directed edges, does not support in a natural way the notion of configuration, in particular where the extent of the configuration is not definable a priori. Hypergraphs provide a way to deal with such a problem, by allowing multiple tentacles to touch all the elements involved in a configuration [DKH97]. Triple graphs have been used to manage instances of patterns, where a specific node in the correspondence graph is connected to all the nodes establishing the correspondence between a model element in the model graph and an element representing the element role in the pattern graph [BGL10]. Both solutions, however, do not scale up to the need for composing configurations into more complex ones, so as to form configuration hierarchies. Indeed, for the hypergraph-based solution this would require the ability to define edges between hyperedges, whereas for the pattern-based solution, this requires that the intended composition of patterns be defined before-hand, while

the need for composition could be restricted to a specific model. Such a mechanism has been defined in [BGL10], but is not immediately extendable to configurations defined on the fly.

This problem is particularly relevant in model-to-model transformation, in situations where the interest is in annotating configurations as a whole, so as to constrain the use of particular transformations only to elements in some configuration. We argue that the limitations of the graph-based modeling derive from the use of a single type of diagrammatic relation, *connectedness*, which is not a transitive one. On the contrary, the *containment* relation, being transitive, naturally supports the representation of hierarchies, with the important property that removing a level of containment maintains the relation between the levels adjacent to the one removed.

In this line, we adopt an extension of the notion of graph by introducing *boxes*, which are elements including subgraphs and other boxes as well. Boxes are naturally organised in containment structures, but can also be the source and target of edges. It is to be remarked, however, that boxes are not organised in a proper hierarchy, as box containment does not induce a partial order, since anti-symmetry is not required. This allows for configurations where boxes can be mutually contained into one another, without being identified.

Boxes were originally proposed in an informal way in [PP00], where they were called *loops*. We give here a formalisation of them and set them in the framework of model annotation and transformation, by making them the target of annotation edges. In this way, we extend the definition of annotation in [BP12], and provide a formal characterisation of annotations of configurations, which, though hinted at there, was not completely developed.

Paper organisation. In Section 2 we formally define the required extension of graphs to contain boxes allowing structured arrangements of nodes and edges, as well as reference to entire subgraphs, while Section 3 introduces the running example of an organisational model annotated with temporal and security information. Section 4 presents the metamodel for annotation processes and formalises the notion of domain, and Section 5 illustrates the impact on the rewriting process of the combination of annotations and boxes referring to the running example. The paper ends with Sections 6 and 7 discussing related work and some concluding remarks, respectively.

2 Preliminaries

We adopt the framework of typed graphs, and we extend the usual notion of graph with elements of a new sort, called *boxes*, allowing a nested structuring of graphs.

Definition 1 A (directed) *graph with boxes* is a tuple $G = (V, E, B, s, t, cnt)$, where: (1) V and E are sets of nodes and edges as in usual graphs; (2) B is a set of boxes, such that $B \cap (V \cup E) = \emptyset$; (3) the *source* and *target* functions s and t extend their codomains to $V \cup B$; (4) $cnt : B \to \wp(V \cup B)$ is a function associating a box with its *content*[1] with the property that if $x \in cnt(b_1)$ and $b_1 \in cnt(b_2)$, then $x \in cnt(b_2)$.

In the rest of the paper we refer to graphs with boxes as *B-graphs* or just graphs unless it is necessary to distinguish them.

[1] Here and elsewhere $\wp$ denotes the powerset.

Definition 2 A *morphism* $f : G_1 \to G_2$ between *B-graphs* $G_i = (V_i, E_i, B_i, s_i, t_i, cnt_i)$ is a triple $(f_V : V_1 \to V_2, f_E : E_1 \to E_2, f_B : B_1 \to B_2)$ that preserves source, target and content function, i.e., $f_{V \cup B} \circ s_1 = s_2 \circ f_E$, $f_{V \cup B} \circ t_1 = t_2 \circ f_E$, and if $x \in cnt_1(b)$, then $f_{V \cup B}(x) \in cnt_2(f_B(b))$ for all $x \in V \cup B$ and $b \in B$, where $f_{V \cup B}$ is the (disjoint) union of f_V and f_B.

In a *type B-graph* $TG = (V^T, E^T, B^T, s^T, t^T, cnt^T)$, V^T, E^T and B^T are sets of node, edge and box types, respectively, while the functions $s^T : E^T \to V^T \cup B^T$ and $t^T : E^T \to V^T \cup B^T$ define source and target node- and box- types for each edge type, and the function $cnt^T : B^T \to \wp(V^T \cup B^T)$ associates each type of box with the set of types of elements it can contain.

A B-graph G is *typed on* a type B-graph TG if there is a graph morphism $type : G \to TG$, with $type_V : V \to V^T$, $type_B : B \to B^T$ and $type_E : E \to E^T$ s.t. $type_V(s(e)) = s^T(type_E(e))$ and $type_V(t(e)) = t^T(type_E(e))$. Moreover, given $b \in B, x \in V \cup B$, we have: $x \in cnt(b) \implies type_X(x) \in cnt^T(type_B(b))$, where X is V or B, depending on $x \in V \cup B$. A morphism $f : G_1 \to G_2$ between TG-typed graphs preserves the type, i.e. $type^2 \circ f = type^1$. Any type graph morphism $f^T : TG_1 \to TG_2$ induces two functors, $Incl_{f^T}$ from the category of TG_1 typed graphs to that of TG_2 typed graphs, and $Forget_{f^T}$ in the opposite direction, in the obvious way. The result in Theorem 1 is already stated in [PP00] without proof.

Theorem 1 *[Category of B-graphs] All the B-graphs, typed over the same type B-graph TG, and all the type-preserving B-morphisms form a category* $\mathbf{GraphL_{TG}}$ *closed under pushouts and pullbacks.*

Proof sketch. It is sufficient to deal with the last component *cnt* of a *B-graph*, as the rest can be viewed as a graph where the set of nodes is distinguished into V and B. The composition $f^3 = f^1 \circ f^2 : G_1 \to G_3$ satisfies the additional property about *cnt*: if $x \in cnt(b_1)$ then $f^1_{V \cup B}(x) \in cnt_2(f^1_B(b))$ since f^1 is a morphism, and therefore $f^3_{V \cup B}(x) = (f^1_{V \cup B} \circ f^2_{V \cup B})(x) \in cnt_3(f^3_B(b) = (f^1_B \circ f^2_B(b))$ since f^2 is also a morphism and the content function *cnt* is 'transitive' by definition.

Given a span of morphisms $G_1 \xleftarrow{f^1} G_0 \xrightarrow{f^2} G_2$, we construct its pushout $G_1 \xrightarrow{g^1} G_3 \xleftarrow{g^2} G_2$ in the usual way for the first five components, with morphisms $g^i : G_i \to G_3$ preserving the source s_i and target t_i functions for $i = 1, 2$. To define cnt_3 for G_3, let $b \in B_3$. Then, by construction, either $b = g^1(b_1) = g^1(f^1(b_0))$ and $b = g^2(b_2) = g^2(f^2(b_0))$ for some $b_i \in B_i$ and $b_0 \in B_0$, or $b = g^1(b_1)$ for some $b_1 \in B_1$ and $b \neq g^2(b_2)$ for any $b_2 \in B_2$ (the third possibility reversing the roles of 1 and 2 is similar). In the first case, $cnt_3(b) = g^1(cnt_1(b_1)) \cup g^2(cnt_2(b_2))$, while in the second case $cnt_3(b) = g^1(cnt_1(b_1))$. The 'transitivity' property of cnt_3 follows from that of cnt_1 and cnt_2, while the universal property follows from the universal property of the union in **Sets**.

Given a cospan of morphisms $G_1 \xrightarrow{g^1} G_3 \xleftarrow{g^2} G_2$, we construct its pullback $G_1 \xleftarrow{f^1} G_0 \xrightarrow{f^2} G_2$ in the usual way for the first five components. For $b_0 \in B_0$, we have $g^1(f^1(b_0)) = g^2(f^2(b_0))$ by construction and define $cnt_0(b_0) = \{x \in G_0 \mid f^1(x) \in cnt_1(f^1(b_0)) \wedge f^2(x) \in cnt_2(f^2(b_0))\}$. □

We adopt the DPO (Double PushOut) approach to graph transformation [EEPT06], extending it to allow rewriting on B-graphs. A DPO rule consists of three (B-)graphs, called left- and right-hand side (L and R), and interface graph K. Two injective morphisms $l : K \to L$ and $r : K \to R$ model the embedding of K (i.e. the sub B-graph preserved by the rule) in L and R. Figure 1 (left) shows a DPO direct derivation diagram. Square (1) is a pushout (i.e. G is the union of L and D

through their common elements in K), modeling the deletion of the elements of L not in K, while pushout (2) adds the new elements, present in R but not in K. Figure 1 also illustrates the notion of *negative application condition* (NAC), of the form $n\colon L \to N$ that a match $m\colon L \to G$ should satisfy. A rule is applicable if there is no morphism $q\colon N \to G$ such that $q \circ n = m$.

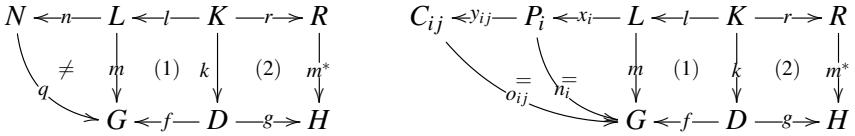

Figure 1: DPO Direct Derivation Diagram for rules with NACs (left) and ACs (right).

We denote the application of the rule p on a match $m : L \to G$ by $G \Rightarrow_p^m H$ and write $G \Rightarrow_p H$ if H can be derived from G by applying p with respect to some match m for L in G. As with the traditional DPO approach to graph transformation, the existence of a morphism m from L to G is not sufficient to guarantee the applicability of the rule. The match $m : L \to G$ must satisfy the Gluing Conditions, which extend naturally the original ones: the Dangling Conditions must be satisfied by all edges, including those with a box as source/target entity, and the Identification Condition must be satisfied not only by edges and vertices, but also by boxes.

Theorem 2 (Gluing Conditions) *Given a rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ (a span of B-graphs and morphisms), and a morphism $m : L \to G$, let $ID_{m_X} = \{x \in X_L \mid \exists y \in X_L[x \neq y \wedge m_X(x) = m_X(y)]\}$ with $X = V,B,E$, and $DANG_m = \{x \in V_L \cup B_L \mid \exists e \in E_G \setminus m_E(E_K)[m_{V\cup(B)}(x) = s_G(e) \vee m_{V\cup(B)}(x) = t_G(e)]\}$. Then the pushout complement D exists* iff*: (1) $DANG_m \subseteq l(K)$, (2) $ID_{m_X} \subseteq l_X(K)$.*

Just as the traditional Gluing Conditions can be extended to B-graphs, it is easy to verify, by choosing as distinguished class $\mathscr{M}$ the injective B-morphisms, that $\mathbf{GraphL_{TG}}$ is an *HLR1-category*, thus enjoying all the usual properties related to the Church-Rosser and Parallelism Theorems [EHKP90]. It is still open whether the equivalent of the Amalgamation and Concurrency Theorems hold. The proof that the *HLR*1 properties hold can be reconstructed by starting with the fact that typed graphs form an *HLR*1-category, and then completing it by noticing that the added component, the function *cnt*, for the pushouts and pullbacks in $\mathbf{GraphL_{TG}}$ is constructed using set union and set intersection and that the category **Sets** is *HLR*1 as well [EHKP90].

Given TG-typed rules $p_i : L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i$, $i = 1,2$, a *rule morphism* $h : p_1 \to p_2$ is a triple $h_X : X_1 \to X_2$, with $X = V,B,E$, of TG-typed morphisms with $h_R \circ r_1 = r_2 \circ h_K$ and $h_L \circ l_1 = l_2 \circ h_K$.

An *atomic constraint* is a total morphism between typed attributed graphs $c\colon P \to C$. A graph G *satisfies* c, noted $G \vDash c$, if for each *match morphism* $m : P \to G$ there exists a morphism $y : C \to G$ s.t. $y \circ c = m$. If $c : P \to C$ is an atomic constraint, then $\neg c$ is also an atomic constraint, and $G \vDash \neg c$ *iff* $G \nvDash c$. We call *negative atomic constraint*[2] an atomic constraint of the form $n_P = \neg i_P : P \to P$, where i_P is the identity morphism, and $G \vDash n_P$ *iff* $\nexists m_P\colon P \to G$. We call $M(c) = \{G \mid G \vDash c\}$ the set of *models* for c, and we work with consistent sets of constraints $\mathscr{C}$, i.e. $\bigcap_{c_i \in \mathscr{C}} M(c_i) \neq \emptyset$. We use different types of morphisms depending on the domain and the usage, both for constraints and rules. A *constraint morphism* $k\colon ac_1 \to ac_2$ is defined in a way

[2] In this paper, we deal only with positive atomic constraints.

similar to a rule morphism, i.e. it is a pair $k_X : X_1 \to X_2$, $X \in \{P, C\}$, of TG-typed morphisms s.t. $h_C \circ r_1 = r_2 \circ h_P$. Figure 1 (right) shows that an atomic constraint can be associated with a rule as an *application condition* AC, of the form $\{x_i : L \to P_i, \{y_{ij} : P_i \to C_{ij}\}_{j \in J_i}\}_{i \in I}$, for a match $m : L \to G$ of the LHS of a rule, where I and J_i are index sets for each $i \in I$. An AC is satisfied by m if, for each $n_i : P_i \to G$ s.t. $n_i \circ x_i = m$, there exists some $o_{ij} : C_{ij} \to G$ s.t. $o_{ij} \circ y_{ij} = n_i$. A NAC can also be seen as a particular case of AC where $\{y_{ij} : P_i \to C_{ij}\}_{j \in J_i}\} = \emptyset$. A *general application condition* (GAC) is a composition of nested constraints, together with a formula on their matches, built with the operators $\exists$, $\forall$, $\wedge$ and $\vee$. From this point on, a *rule* is a pair $p = (r, ac)$ consisting of a span of morphisms $r = L \xleftarrow{l} K \xrightarrow{r} R = \pi_1(p)$ and a general application condition $ac = \pi_2(p)$.

3 Running example: security annotations on teams

We present the running model for this paper, discussing the use of boxes and annotations in the rewriting process with reference to an organisational domain in which members can access areas, and teams can be formed. Teams are represented as boxes which can include both members and areas. If a member m and an area a belong to the same team t, the access of m to a is considered to be within the scope of t as well. We consider the annotation of this domain with contextual information from two domains. The first, a security domain, is defined for simplicity as a collection of security levels which can be compared via a reflexive and transitive relation, called *dominates*. The second domain provides temporal information, which is defined by *periods* expressed with reference to a calendar model of time [BBF01]; a period corresponds to some recurrence in the calendar at some granularity level (e.g. *dayTime*, *nightTime*, or *weekDays*). Again, a period p_1 can dominate a period p_2, if p_1 completely encompasses p_2.

We consider annotation processes where security levels are associated with areas, members, or teams to establish constraints on access or on inclusion in a team, while calendar periods are associated with a model configuration, as represented by a box encompassing a graph, to give information on the current period. Calendar periods can also be used to annotate security annotations, denoting their periods of validity, thus realising a nested form of annotation.

The setting for the annotation process is summarised by the type graph in Figure 2, presented as a UML class diagram, where stereotypes have been used to distinguish between types of box and types of nodes. Another set of stereotypes is used to identify the domain (organisational, security or temporal) from which the types originate, or to indicate that a node indicates the presence of an annotation. The cnt^T function is such that $cnt^T(\texttt{Conf}) = \{\texttt{Member}, \texttt{Area}, \texttt{Team}\}$ and $cnt^T(\texttt{Team}) = \{\texttt{Member}, \texttt{Area}\}$. The edges from the two annotation nodes indicate that nodes of the organisation domain are annotated with domain elements from the security and temporal domains. Moreover, nodes representing security annotations can be in turn annotated with information from the temporal domain, to indicate the periods in which the annotation is valid. Indeed, the type graph depicted in Figure 2 is the outcome of a complex annotation process, where the organisational domain is first annotated with security information, and then the resulting domain is annotated with temporal information.

A number of constraints define the acceptable relations among annotations on the organisational domain. For example, the constraint `isAccessSecure` in Figure 3 (left) states that if an area is accessed by a member and both area and member are annotated with some security level,

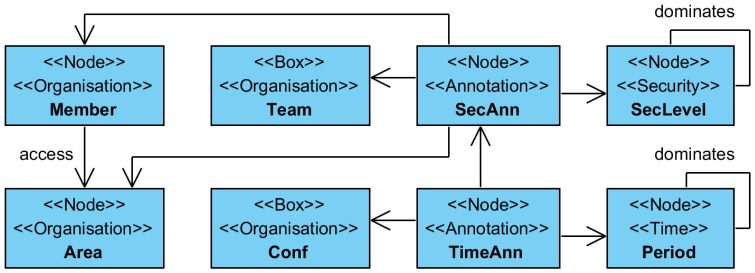

Figure 2: The type graph for the running example.

then the security level of the member must dominate that of the area. Similarly, the constraint `isTeamSecure` in Figure 3 (right) requires that an area can be assigned to a team if the team has the necessary authorisations to use the resources in the area, i.e. has a higher security level. An analogous constraint is defined for inclusion of members in a team.

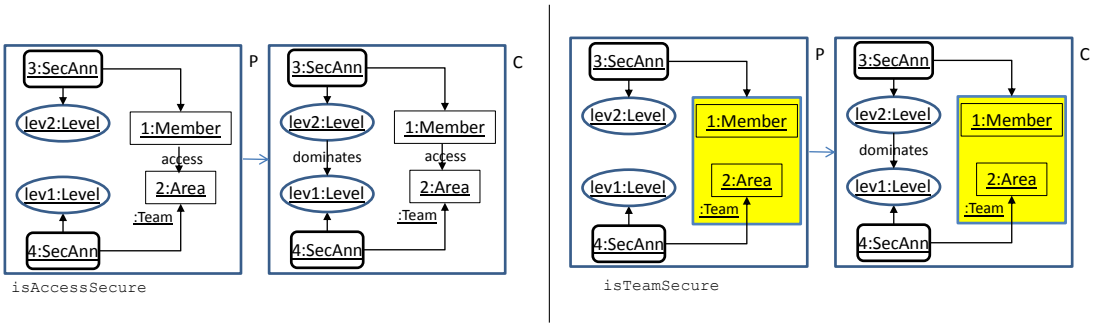

Figure 3: Security constraints on access (left) and teams (right).

We adopt a concrete representation derived from UML instance diagrams, where instances of the box sort are defined as rectangles surrounding their content, nodes from the organisation domain are usual instance rectangles, nodes representing annotations are rectangles with rounded corners, and nodes representing elements from the security and temporal domains are ovals. A box of type `Team` has a yellow background, and a box of type `Conf` has a pale green background[3]. When morphisms are involved, they are represented by the matching of numbers in the names of elements. Edges appearing in both graphs and associated with identified elements in the morphism are considered to be identified as well. We only show the types of edges from the various domains, the types of the annotation edges being easily inferrable.

We also show some of the rules by which graphs in the organisational domain can be formed, leaving details on the annotation process for Section 5. Rule `simpleAccessWithoutTeams` in Figure 4 (left) grants members access to areas when both members and areas are outside the scope of any team, while rule `teamAccess` in Figure 4 (right) allows access within the scope of a single team. In this paper, since we deal only with non-deleting rules defined by injective morphisms, we only indicate the L and R components of the rules, the K component being equal

[3] Lighter gray and darker gray in a greytone print, respectively.

to L. Constants define concrete values for annotations. For the remaining rules, we assume that a NAC is always present excluding membership in some team, unless this is required, and that NACs prevent the application of rules where security information is required.

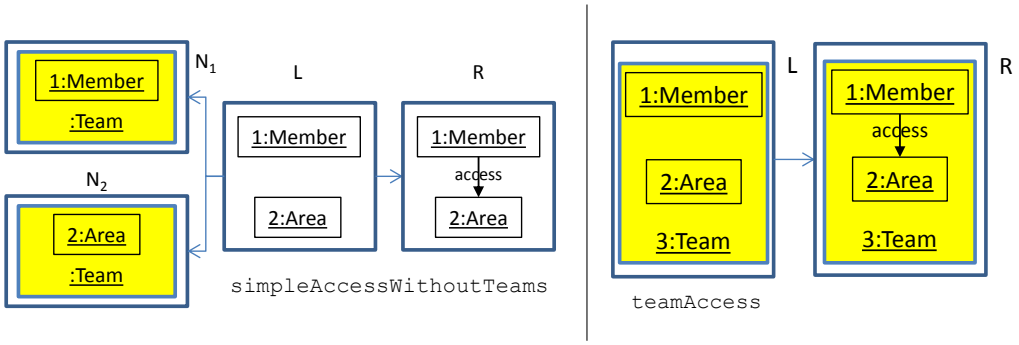

Figure 4: Rules for access for non-teamed elements (left) and within a team (right).

Figure 5 presents a snapshot of an organisation as a graph G, together with some annotations, with three members and two areas. Two members and one area are within a team, and security levels are associated with the area and with the team as a whole. Another area is annotated with two security levels, each valid during a different period. The whole graph is annotated to indicate that this snapshot is taken at daytime. We do not explicitly present the edges representing the *dominates* relation between levels, which can be inferred from the names of the levels.

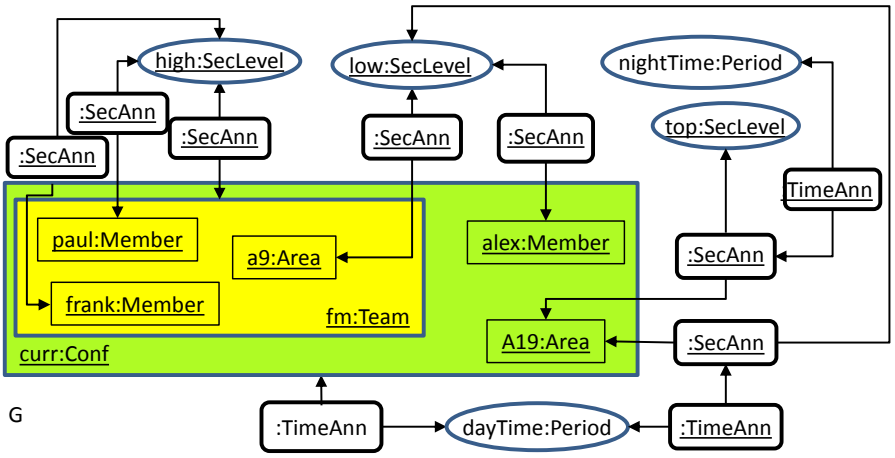

Figure 5: A snapshot of an organisation with security and temporal annotations.

4 Annotatable elements

Figure 6 presents the metamodel at the basis of the proposed extension, refining the one presented in [BP12]. Annotations represent dynamical relations established between annotatable

elements and domain entities; annotatable entities are either elements representing annotations (`AnnotationTypeNode`) or elements representing some domain notion (`DomainConcept`). We consider three types of domain concepts: entities, relations and configurations, recursively composed of concepts. Constraints on `AnnotationTypeNode` and the `DomainConcept` maintain the notions of domain and of annotation consistent with one another. In particular, a domain configuration is composed of domain concepts, all belonging to the same domain, while an annotation relates a domain concept with an entity from a different domain. Note that a domain concept can be annotated only with domain entities, not with configurations. The notion of domain is induced from the notion of type graph, as per Definition 3.

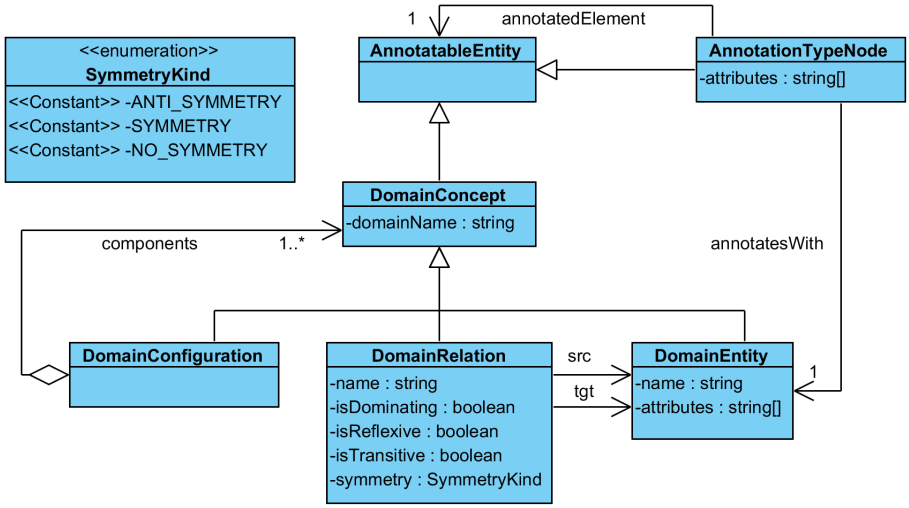

Figure 6: The extended metamodel for complex annotations.

Definition 3 (Domain) Given a type graph TG and a set $\mathscr{C}$ of constraints on it, a *domain* is the set of graphs typed, by TG, that satisfy all of the constraints in $\mathscr{C}$.

This organisation allows the flexible annotation of subgraphs, besides individual nodes or edges, with domain elements. Moreover, the `DomainConfiguration` meta-type realises a special form of the *Composite* pattern, one which admits cycles. With reference to the type graph of Figure 2, we observe that `<<Node>>` types are instances of `DomainEntity`, while `<<Box>>` types are instances of `DomainConfiguration` and `<<Annotation>>` types are instances of `AnnotationTypeNode`. Edge types in the type graph conform in the obvious way with the edge meta-types in the metamodel.

Given a graph $G_1 = (V_1, E_1, B_1, s_1, t_1, cnt_1)$ in a domain D_1, an annotated version of G_1 on the domain D_2 is constructed as $G' = (V', E', B', s', t', cnt')$, where $V' = V_1 \cup A \cup V_2$, $E' = E_1 \cup E_A \cup E_2$, with: A the set of nodes whose type is an instance of `AnnotationTypeNode`, V_2 a set of nodes typed in TG_2, E_A edges relating annotation nodes with elements in $V_1 \cup B_1$ or V_2, E_2 edges typed on TG_2 and relating nodes in V_2, s' and t' suitable extensions of s_1 and t_1 to include $E_A \cup E_2$ and

$V_A \cup V_2$ in their domains and codomains, respectively. Hence, we allow only the use of nodes, and not of edges or boxes as values for the annotations, while any type of element from the application domain graph can be annotated (we do not consider edge annotation in this paper). Moreover, B'=$B_1 \cup B^a$, where B^a is a new set of nodes containing subgraphs of G_1, and associated with some node in A, i.e. boxes which are not part of the original model, but which are created to support some annotation. We do not present examples of boxes in B^a in this paper. The previous definitions of t' and cnt' are therefore enriched accordingly.

Since `AnnotationTypeNode` is a type of `AnnotatableEntity` annotations can be nested. In particular, an annotation node an_1, connecting an element x of a domain D with some entity v_1 of a domain D_1, can be annotated, through an annotation node an_2, with an element v_2 of a third domain D_2. We require that nested annotations do not form cycles on domains. Such an annotation is interpreted as constraining the validity of an_1 in the context denoted by v_2.

5 Rewriting with annotations

We now discuss the impact of annotation processes on the original rules and constraints in the organisational domain by presenting a collection of rules derived from rules `teamAccess` and `simpleAccessWithoutTeams` (see Figure 4), as well as an extension of constraint `isAccessSecure` (see Figure 3). Since we have rules with $K = L$, we use the construction given in [BP12] for the SPO approach, that we summarise here. Hence, for a rule $r\colon L \to R$ and a constraint $c\colon P \to C$, where R has a non-empty intersection Z with P, we build two graphs X and Y and morphisms $x\colon L \to X$ and $y\colon X \to Y$. X contains the elements in L and in P, without considering elements added by R to L, with potential duplicate elements identified through r and Z, while Y integrates the elements in C, again with all the necessary identifications. The morphisms x and y are derived by requesting all compositions to commute. As an example, composing constraint `isTeamSecure` of Figure 3 (right) and rule `simpleAccessWithoutTeams` above produces rule `securedAccess` in Figure 7, where we have left the NACs understood.

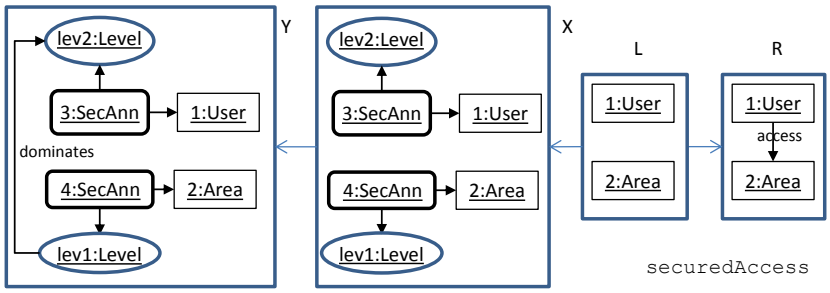

Figure 7: Composing `simpleAccessWithoutTeams` and `isAccessSecure`.

In a similar way, constraint `isTimeConsistent` in Figure 8 states that an area annotated with some period information can be accessed only during that period (i.e. in a configuration annotated with that period). Its composition with rule `simpleAccessWithoutTeams` produces rule `timedAccessArea` in Figure 9.

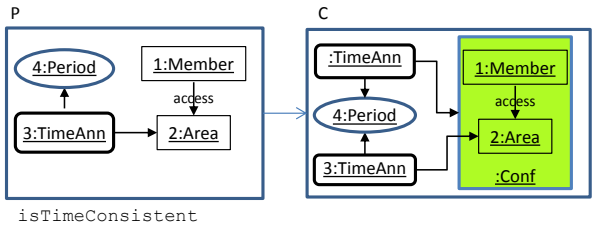

Figure 8: A temporal constraint on access.

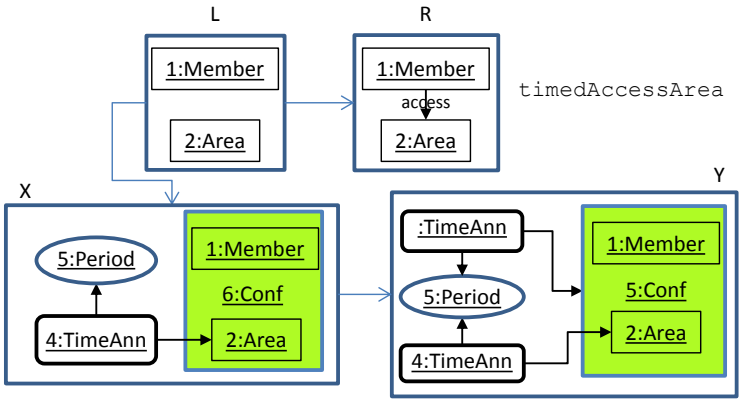

Figure 9: Composing `simpleAccessWithoutTeams` and `isTimeConsistent`.

The combination of the two constraints above on one area gives rise to a conjunction of the corresponding application conditions, as exemplified by rule `securedTeamTimedAccess` in Figure 10, where $ac1$ and $ac2$ stand for the whole application conditions in Figures 9 and 7.

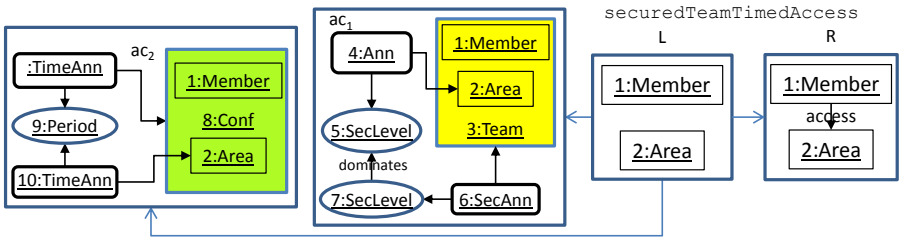

Figure 10: Combining application conditions.

We now discuss how the progressive nesting of applications causes the modification of the rules derived from constraints. Figure 11 shows constraint `isTimeSecurityConsistent`, which extends constraint `isAccessSecure` with a temporal annotation on the security annotation of an area, analogous to the one used to derive rule `timedAccessArea`.

Then, we need to extend the application condition for rule `securedTeamAccess`. Let $ac\colon X \to Y$ be the morphism in the application condition derived from `isAccessSecure` (de-

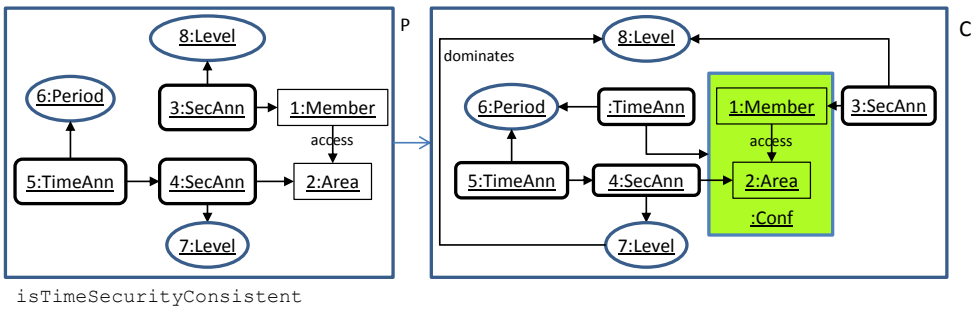

Figure 11: Extending security constraint `isAccessSecure` with a temporal annotation.

fined by $c_1 : P_1 \to C_1$) and $ac_2 : Y_1 \to Y_2$ be the morphism in the application condition derived from `isTimeSecurityConsistent`. The intersection between P_1 and Y_1 is exactly X. Then we transform the original atomic application condition into a general application condition resulting from the conjunction of the original case and the extended case, as expressed by the formula $\forall m_X [\exists m_Y \wedge \forall m_{Y_1} [\exists m_{Y_2}]]$, where m_Z, $Z = X, Y, Y_1, Y_2$, indicates the existence of a match for the graph Z which extends, according to the morphisms in the application condition, a match for L in the host graph G. The resulting rule, `timedAccess`, is shown in Figure 12.

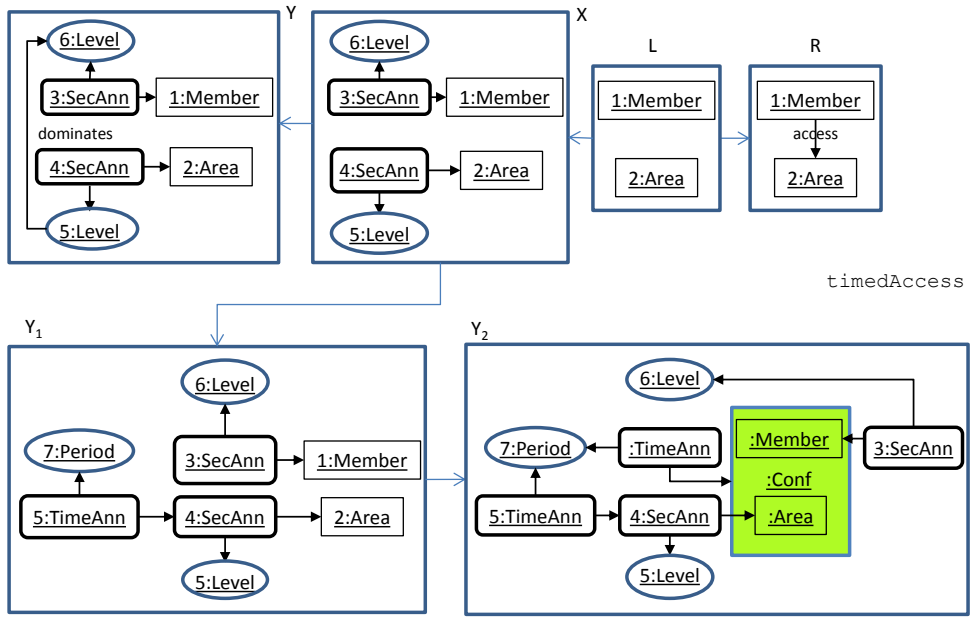

Figure 12: Extending rule `securedAccess` after `isTimeSecurityConsistent`.

For the host graph in Figure 5 rule `timedAccess` is applicable on a match formed by member Alex and area 19, while rule `securedAccess` is applicable twice, to both members Frank and Paul, for area 9. Thanks to the transitivity of *cnt*, also rule `securedTeamTimedAccess` can be applied in this case. As they belong to a team, they cannot access area 19 outside of it,

nor can Alex access area 9. Figures 13 and 14 show the resulting graphs.

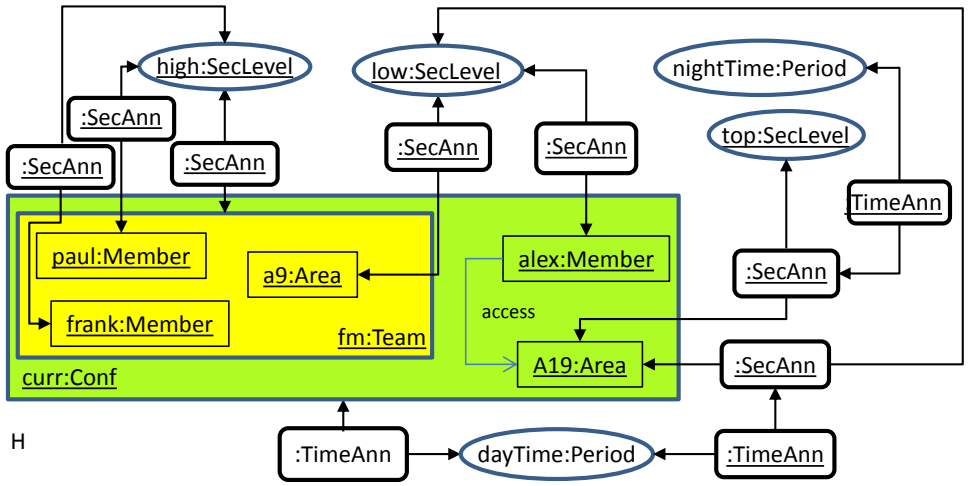

Figure 13: The graph H generated by applying the rule of Figure 12 to the graph G.

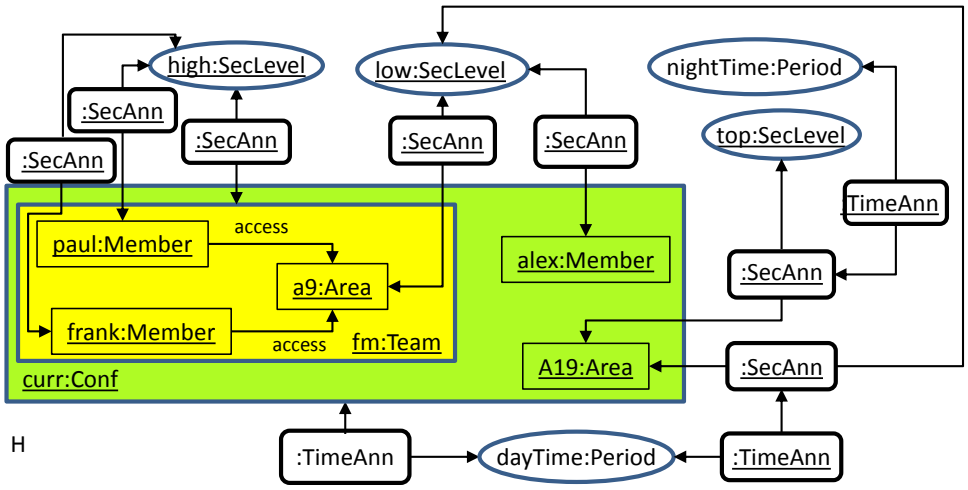

Figure 14: The graph H generated by applying twice the rule of Figure 7 to the graph G.

6 Related work

We discuss here literature relative to modeling with boxes. For a discussion of work related to annotations, see [BP12]. In [PP00], the notion of box was introduced (called there *loop*), where a box encloses a subgraph or other graphs recursively, and an extension of the notion of graph rewriting was proposed to encompass rewriting of graphs with boxes. The proposal in this

paper capitalises on this notion, allowing boxes to be annotated, thus constraining the possible transformations in graphs with annotations.

The study of families of diagrammatic relations and of their adequacy to modeling domains exhibiting specific relations has been conducted in [BG04] and [FB05].

The motivation for boxes is analogous to that for introducing Hierarchical Graphs in [DHP02]. These identify some specific types of hyperedges as containing entire graphs, and of multi-level graphs [PP95], where some nodes may hide some part of a graph at some level of abstraction. In this case, the resulting structure is not a strictly hierarchical one. The notion of views, realised through distributed graphs [GMT99], allows the composition of partial specifications of a model, not necessarily in a nested way, but considering levels of abstraction separately.

Boxes as proposed in this paper allow both the definition of hierarchies and the composition of different views, with elements which may belong to different hierarchies.

7 Concluding Remarks

We have presented an approach to enriching models of application domains with constraints coming from contextual domains through the use of annotations relating values from the latter one to model elements of the former one.

By extending the notion of graph to include boxes which can be source or target of edges (in particular, target of annotation edges), we extend the definition of annotation provided in [BP12], and provide here a formal characterisation of annotations of configurations, which was not completely developed there but only hinted at. Boxes can be nested, allowing the construction of complex configurations. This extension allows modelers to express and reason about complex interplays among annotations exploiting elements from different domains. It is important to notice that boxes are a first-class modeling construct, independent of their content. Hence, two boxes can have the same exact content, without being identified or without being contained into one another. Conversely, two boxes can be mutually contained into one another, again sharing the same content, but maintaining possible independent evolutions. Finally, boxes can be used as place-holders for collections of elements yet to be defined. For example, teams can be formed and annotated before assigning members to them.

Among several aspects still to be investigated in details is the question of maintaining consistency with respect to annotations. In particular, if an annotated element is removed, than the corresponding annotations should be removed as well. This could be accomplished through units, with a preliminary removal of the annotation followed by a simple DPO rule. Also under investigation is the possibility of defining boxes within the SPO approach. In this case, repair actions could be used after removing an annotated node (and the annotation edge which had that node as a target) to remove the annotating node.

Bibliography

[BBF01] E. Bertino, P. A. Bonatti, E. Ferrari. TRBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.* 4(3):191–233, 2001.

[BG04] P. Bottoni, A. Grau. A Suite of Metamodels as a Basis for a Classification of Visual Languages. In *Proc. VL/HCC'04*. Pp. 83–90. IEEE CS, 2004.

[BGL10] P. Bottoni, E. Guerra, J. de Lara. A language-independent and formal approach to pattern-based modelling with support for composition and analysis. *Information & Software Technology* 52(8):821–844, 2010.

[BP12] P. Bottoni, F. Parisi-Presicce. Modeling context with graph annotations. *ECEASST* 47, 2012.

[DHP02] F. Drewes, B. Hoffmann, D. Plump. Hierarchical Graph Transformation. *J. Comput. Syst. Sci.* 64(2):249–283, 2002.

[DKH97] F. Drewes, H.-J. Kreowski, A. Habel. *Hyperedge Replacement, Graph Grammars*. Pp. 95–162. World Scientific, 1997.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. Fundamental Theory for Typed Attributed Graphs and Graph Transformation based on Adhesive HLR Categories. *Fundam. Inform.* 74(1):31–61, 2006.

[EHKP90] H. Ehrig, A. Habel, H.-J. Kreowski, F. Parisi-Presicce. From Graph Grammars to High Level Replacement Systems. In *Graph-Grammars and Their Application to Computer Science*. Pp. 269–291. 1990.

[FB05] F. Fondement, T. Baar. Making Metamodels Aware of Concrete Syntax. In *Proc. ECMDA-FA*. LNCS 3748, pp. 190–204. Springer, 2005.

[GMT99] M. Goedicke, T. Meyer, G. Taentzer. ViewPoint-Oriented Software Development by Distributed Graph Transformation: Towards a Basis for Living with Inconsistencies. In *Proc. RE'99*. Pp. 92–99. IEEE CS, 1999.

[PP00] F. Parisi-Presicce. Which Graphs for Visual Modeling? In *ICALP Satellite Workshops*. Pp. 383–386. 2000.

[PP95] F. Parisi-Presicce, G. Piersanti. Multilevel Graph Grammars. In *Proc. WG'94*. Lecture Notes in Computer Science 903, pp. 51–64. Springer, 1995.