Electronic Communications of the EASST
Volume 47 (2012)

Proceedings of the
11th International Workshop on Graph Transformation and
Visual Modeling Techniques
(GTVMT 2012)

Modeling context with graph annotations

Paolo Bottoni, Francesco Parisi Presicce

14 pages

Guest Editors: Andrew Fish, Leen Lambers
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Modeling context with graph annotations

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

Department of Computer Science, ”Sapienza” University of Rome[1]

Abstract: Organisational policies are often formed of declarational (defining constraints on functional services) and operational (realising functionalities via simple activities) aspects. However, when several perspectives are involved, constraints and operations can comprise different aspects, without identifying the origin of some details. We propose the use of annotations as a way to flexibly add and remove application conditions on rules, while maintaining an indication of their origin. We use graph transformations to model operations in some application domain, graph constraints to model conditions imposed by some external domain, and annotations to combine domains. We explore the problem of failure of transactions due to the additional constraints imposed by the contextual domain, and describe a way to redefine the success conditions for transactions employing the modified rules.

Keywords: Contextual domain, Graph annotation, Graph constraints

1 Introduction

Separation of concerns leads to modeling systems under different perspectives, for example, the control-flow, data, resource, and operational ones for workflows [AH02]. High-order views of service architectures are based on the composition of abstract business models with security, distribution, or communication protocols [LWF03]. In the field of programming languages, especially Java, aspects and annotations are ways to enrich behaviours or structure of programs through the action of specific engines. In particular, aspect-oriented programming is based on weaving management of non-functional concerns with operational functionalities [KLM$^+$97].

In general, organisational policies are formed of declarative and operational aspects. The declarative part defines the constraints regulating the overall delivery of the functional services, while in the operational part, these services are realised by the coordination of simple activities. However, when several perspectives are involved, constraints and operations can comprise different aspects, without the ability to identify the origin of some details. A typical example of this is the construction of application conditions constraining the application of rules in a graph transformation approach to process modeling [EEHP06, KP06].

We propose the use of annotations as a way to flexibly add and remove application conditions on rules, while maintaining an indication of their origin. In particular, we consider that rules are defined to model operations in some application domain, while constraints are imposed according to the context in which the process has to operate, modeled as a separate domain. Moreover, while this domain can be in turn modeled via constraints and rules, we assume that its relevance as context for the execution of the operations can be modeled through some designated property and relation. Hence, a specific form of annotated constraint is introduced and the construction of application conditions for rules operating on annotated domains is presented.

Besides showing how annotated constraints can be employed to defined application conditions, in quite a standard way, we explore the problem of how the execution of transactions can be affected by the context. We observe that a typical transactional pattern is of the form *inception-execution-termination*, where inception is modeled by a rule setting the process in motion and typically producing a token to identify the transaction instance, execution is modeled by some sequence of rules (we consider here iteration of a single rule), and termination ensures that the result is collected and consumes the instance token. When contextual application conditions place additional constraints on the execution rule, it is possible that a transaction fails, which would otherwise terminate correctly, as the original conditions for termination can be no longer satisfied, even if some iteration has been performed. In this case, the typical constructions for modifying a single rule based on constraints are insufficient, and we present a method to modify both the inception and the termination rule based on the modification of the execution rule. The involved techniques are illustrated on a running example based on the domain of distributed document management, with contextual annotations deriving from security and locality domains.
Paper organisation. We give the graph transformation framework in Section 2 and illustrate the running example in Section 3. A formal definition of annotation and the involved constructions are given in Section 4 and Section 5 applies the construction to annotations on the contextual domains. Finally, Section 6 discusses related work and Section 7 conclusions and future work.

2 Preliminaries

We adopt the framework of typed graphs. A graph $G = (V, E, s, t)$ consists of a set of *nodes* $V = V(G)$, a set of *edges* $E = E(G)$, and *source* and *target* functions, $s, t : E \to V$. For a set S, its cardinality is denoted by $|\, S\,|$. In a *type graph* $TG = (V_T, E_T, s^T, t^T)$, V_T and E_T are sets of node and edge types, while the functions $s^T : E_T \to V_T$ and $t^T : E_T \to V_T$ define source and target node types for each edge type. For G a typed graph on TG, there is a graph morphism $type : G \to TG$, with $type_V : V \to V_T$ and $type_E : E \to E_T$ s.t. $type_V(s(e)) = s^T(type_E(e))$ and $type_V(t(e)) = t^T(type_E(e))$.

We use graph transformations according to the Single PushOut (SPO) approach [Löw93]. An SPO rule has the form $p : L \xrightarrow{r} R$ where L and R are called left-hand and right-hand side graphs, respectively, and p is a type-preserving partial morphism. The left of Figure 1 shows an SPO direct derivation diagram, modeling the application of a rule $p : L \xrightarrow{r} R$ to a *host* graph G to produce a *target* graph H, via a match m for L in G. Basically, H contains all $g \in G$ s.t. there is no $x \in L \setminus dom(r)$ with $g = m(x)$ and all $z \in R$ except the elements $r(y)$ s.t. $m(y) = m(x)$ for $x \in L \setminus dom(r)$ and $y \in dom(r)$. Notice that m' can be partial even if m is total. 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.

An *atomic constraint* is a morphism of typed attributed graphs $c : X \to Y$. A graph G *satisfies* c, noted $G \vDash c$, if for each *match morphism* $m : X \to G$ there exists a morphism $y : Y \to G$ s.t. $y \circ c = m$. For $c : X \to Y$ an atomic constraint, $\neg c$ is an atomic constraint, and $G \vDash \neg c$ *iff* $G \nvDash c$. An atomic constraint of the form $n_X : X \to \emptyset$ is called a *negative atomic constraint*. We call $M(c)$ $= \{G \mid G \vDash c\}$ the set of *models* for c, and assume that a constraint system C is consistent, i.e. $\bigcap_{c_i \in C} M(c_i) \neq \emptyset$. We use different types of morphisms depending on the domain and the usage,

$$\begin{tikzcd}
L \arrow[r, "r"] \arrow[d, "m"] & R \arrow[d, "m'"] \\
G \arrow[r] & H
\end{tikzcd}
\qquad
\begin{tikzcd}
Y_{ij} \arrow[drr, bend right, "o_{ij}"', "="] & X_i \arrow[l, "y_{ij}"] \arrow[dr, bend right, "n_i"', "="] & L \arrow[l, "n"] \arrow[r, "r"] \arrow[d, "m"] & R \arrow[d, "m'"] \\
 & & G \arrow[r] & H
\end{tikzcd}$$

Figure 1: SPO Direct Derivation Diagram for simple rules (left) and with AC (right).

both for constraints and rules.

The right of Figure 1 shows that an atomic constraint can be associated with a rule as an *application condition* AC, of the form $\{x_i : L \to X_i, \{y_{ij} : X_i \to Y_{ij}\}_{j \in J_i}\}_{i \in I}$, 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 $n_i : X_i \to G$ s.t. $n_i \circ x_i = m$, there exists some $o_{ij} : Y_{ij} \to G$ s.t. $o_{ij} \circ y_{ij} = n_i$. A *negative application condition* (NAC) derives from a negative constraint: for the rule to be applicable, X_i must not be present. A *general application condition* (GAC) is a composition of nested constraints according to the operators $\exists$, $\forall$, $\wedge$ and $\vee$.

We use control expressions over identifiers of rules to govern rule application for transactional processes. Expressions are built on a grammar allowing single rules, the sequential construct ';', and the iteration construct r^*, where r is a single rule identifier.

3 The running example

To ilustrate our techniques, we use an example of production of, and access to, distributed documents, where the application domain is described by the typed graph of Figure 2. Types of relations are concretely described as edge types between node types.

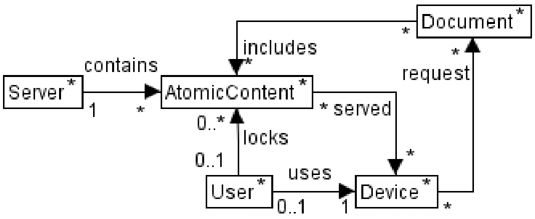


Figure 2: The type graph for the document access domain.

A *document* is composed of some *atomic content*, distributed over some *server*. A *user* obtains access to documents via some controlled *device*. By considering multiplicities, one notices that users are always in control of exactly one device, while devices can be available for the next user. Hence, the processes of user and device creation differ in that a user is added to the system only if a device is available for him/her. In particular, new devices can always be added while a user can be added only if a device is available (we omit presenting the corresponding rules, due to their simplicity). Users can switch from using a device to another, provided that a request is not being served by the old device, and that the new device is not in use (see Figure 3).

Users can add content to a document or use a device to request one. For the first rule (not shown here), a NAC prevents the same content to be added twice to the same document (but

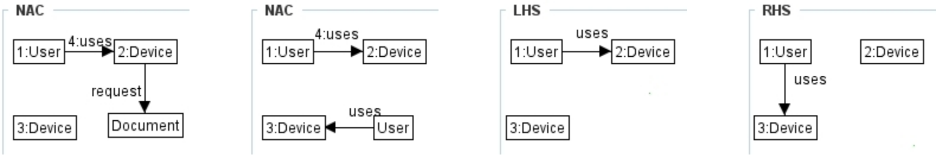


Figure 3: The rule for changing the user of a device.

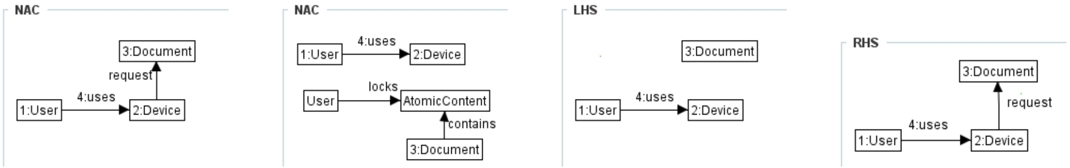


Figure 4: The rule for starting a request.

it can be shared by several documents). The rule `startRequest` activates a process which guarantees that all content from the document is served on the device. As shown in Figure 4, two NACs are defined, one preventing the replication of the same request, the other checking that no content included in the document is locked by a user. For all of these rules we consider partial, type preserving morphisms, and we allow non-injective matches. Atomic contents which are parts of the document are served on the requesting device if not served already to that device and if they are not locked, (see rule `serveContent` in Figure 5). The process is completed when all contents have been served, as defined by rule `requestEnded` in Figure 6. This rule (represented as $L \xrightarrow{r} R$ in the diagram on the right) is equipped with a GAC expressed by the graphs g_0 and g_1 on the left of Figure 6 and the formula $\forall m^{g_0}.\exists m^{g_1}$. Here m^{g_0} and m^{g_1} are matches for g_0 and g_1, respectively, such that the diagram commutes, with m^L a match of L in G. The expression `startRequest`; `serveContent`*; `requestEnded` governs the process.

$$\begin{tikzcd}
g_1 \arrow[drrr, "m^{g_1}"'] & g_0 \arrow[l] \arrow[drr, "m^{g_0}"] & & L \arrow[ll] \arrow[r, "r"] \arrow[d, "m^L"] & R \arrow[d, "m'"] \\
 & & & G \arrow[r] & H
\end{tikzcd}$$

4 Annotations

Figure 7 presents a metamodel for annotation of patterns from some application domain with values from some other domain, providing the annotation content. A *Domain* is therefore com-

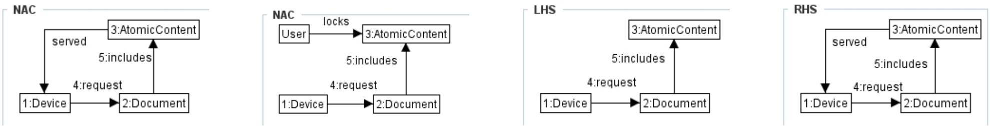
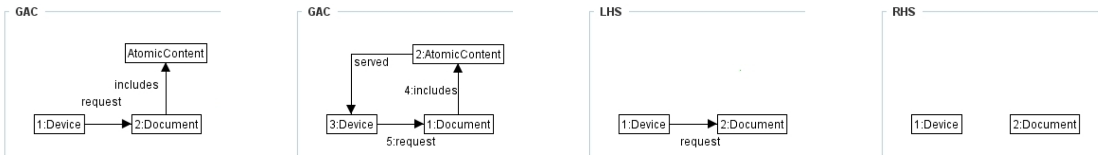


Figure 5: The rule for serving content from a requested document.



(a) g_0 (b) g_1

Figure 6: The rule for signaling completion of a request.

posed of at least one metatype of *domain elements* and some metatype of *domain relationships*. Relationships are here considered to be binary and directed.

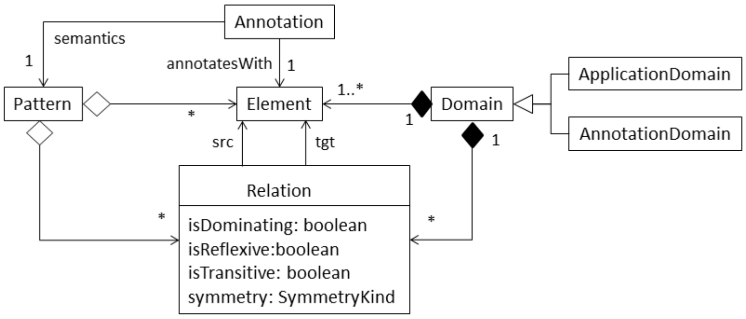


Figure 7: The metamodel for the annotation activity.

We consider one domain to be the *application domain*, and annotate it with elements from a number of *contextual domains*. The *Annotation* metatype is used to indicate the *semantics* relation by which some domain *Pattern* is *annotated with* some element in a contextual domain, where a pattern is formed by any aggregation of elements and relationships. We express invariants on the metamodel using OCL. For navigation expressions, if the association does not have a name, we use the name of the navigation target with a lowercase initial. As annotations relate elements in different domains, the following invariants hold in the context of an *Annotation*:

(1): `self.annotatesWith.domain.isOclKindOf(AnnotationDomain).`

(2): `(self.semantics.element.domain->union(semantics.relation))`
`->forall(d|d.isOclKindOf(ApplicationDomain)).`

For each annotation domain, exactly one special relation is taken as the *dominating* relation, whereby some structure can be imposed on the elements of the domain. Therefore, in the context of a *Relation*, the following invariant holds:

(3): `(self.isDominating=true)implies`
`(self.domain.isOclKindOf(AnnotationDomain)),`

while in the context of an *AnnotationDomain* we have:

(4): `self.relation->select(r|r.isDominating=true)->size()=1.`

This relation can be *reflexive*, *transitive*, *symmetric*, or else, depending on modeling purposes. The different axiomatic properties can be expressed through graph constraints. Figures 8 and 9

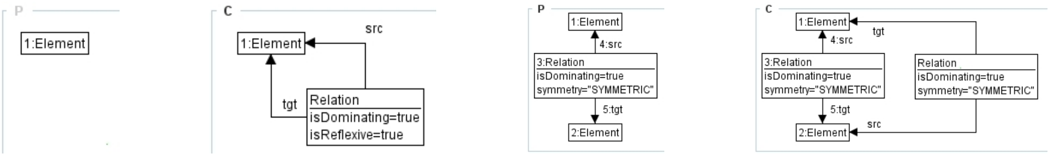


Figure 8: Constraints expressing the reflexivity and symmetry axioms.

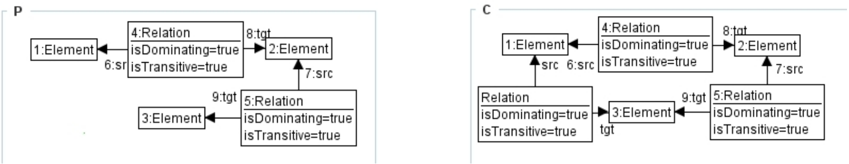


Figure 9: Constraint expressing the transitivity axiom.

show three meta-constraints, using injective morphisms, to be specialised to the types of elements and relations used for annotation. They describe reflexivity and symmetry (Figure 8 left/right, resp.), and transitivity (Figure 9). Antisymmetry can be expressed by composing symmetry and reflexivity, using non-injective morphisms.

Finally, patterns are closed w.r.t. relations. Hence, in the context of a *Pattern* we have:

(5): `self.relation->forall(r|self.element->contains(r.src) && self.element->contains(r.tgt)).`

We discuss here the case in which the annotation domain is restricted to a single type of element used for annotating and to the use of only the dominating relation. The annotation process is employed to place additional constraints on the application domain. Moreover, we consider only annotations on nodes, leaving the treatment of the general case of annotation of relations and patterns for future work. In particular, we show how to lift rules on the application domain to their annotated versions, and how to place additional conditions (of positive, negative or general form) as a result of adding constraints employing elements from the annotation domain to rules in the application domains. We summarise the approach in the following definitions.

Definition 1 (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 and satisfying all of the constraints in $\mathscr{C}$.

Definition 2 (Annotation type graph) Let $\mathscr{D}$ and $\mathscr{A}$ be two domains and TG_D and TG_A their type graphs. An *annotation type graph* is formed by designating in TG_D a subset $S_V \subset V_T(TG_D)$ and incrementing TG_D with: a node type $l \in V_T(TG_A)$, an edge type $r \in E_T(TG_A)$, s.t. $s^T(r) = t^T(r) = l$, a new node type $a \notin V_T(TG_D) \cup V_T(TG_A)$, and a set of edge types L, $L \cap (E_T(TG_D) \cup E_T(TG_A)) = \emptyset$ s.t. $\forall v \in S_V . \exists! l \in L.((s^T(l) = a) \wedge t^T(l) = v)$.

Definition 3 (Annotated graph) Given an *annotation type graph* TG, a graph G typed on TG is called an *annotated graph*.

Definition 4 (Annotated constraint) For TG an *annotation type graph*, with $\mathscr{A}$, $\mathscr{D}$ and S_V as in Definition 2, an *annotated constraint* is an injective morphism $p : P \hookrightarrow C$ s.t. $P \in \mathscr{D}$, C is an annotated graph on TG, and $\forall \chi \in V(P).((type_T(\chi) \in S_V) \Rightarrow ((\exists! \alpha \in V(C), \rho \in E(C)).type_T(\alpha) = a \wedge type_E(\rho) \in L \wedge s(\rho) = \alpha \wedge t(\rho) = p(\chi)))$.

The most important construction for adapting domain rules $p : L \overset{r}{\to} R$ to (atomic, positive) contextual constraints $c : P \to C$ is shown in Figure 10. Here, X is a maximal overlap between R and P and Q is their pushout with respect to X; the graph W obtained from Q by applying the inverse rule $p^{-1} : R \overset{r^{-1}}{\to} L$ represents the part of the premise P of the constraint already present in L. The graph Z is obtained as the pushout of R and C with respect to X, and the result of applying r^{-1} to it (via the morphism induced by the pushout) is denoted by M. This graph represents the part of the conclusion C of the constraint already present in L. The morphisms $Q \to Z$ and $W \to M$ exist and are unique because of the universal property of the pushouts for Q and W. The morphisms $h : L \to M$ and $k : W \to M$ are then used to build application conditions, as will be discussed in Section 5. An analogous construction builds negative application conditions in the case of negative constraints. Note that there may be several morphisms $W \to M$ for different choices of X. In the rest of the paper, we consider only situations in which the annotations in Z only refer to elements which are not produced by R. For the case that also elements produced by R are annotated in C, that we do not discuss here, one would need to activate some repair action, after applying $p : L \to R$, to produce proper annotations for these elements.

$$\begin{tikzcd}
L \arrow[rr, "r"] \arrow[dd, "h"'] \arrow[dr, "w"] & & R \arrow[ll, bend right, "r^{-1}"'] \arrow[dr] \arrow[dd] & X \arrow[l, hook'] \arrow[r, hook] & P \arrow[dl] \arrow[dd, "c"] \\
& W \arrow[dl, "k"'] & & Q \arrow[ll] \arrow[dl] & \\
M \arrow[rr] & & Z & & C \arrow[ll]
\end{tikzcd}$$

Figure 10: Construction of graph M for application conditions.

5 Applying annotations

We now present two cases of annotation for document distribution, one from the security domain, ensuring that content is served only in a way complying with security levels, and one from the localisation domain, where distribution is restricted to accessible locations. In the first case the dominating relation is anti-symmetric, while in the second case we do not demand anything with respect to symmetry. In both cases the relation is also reflexive and transitive.

Figure 11 shows the introduction of security annotations on the document domain. Each type in the domain, except *Document*, can be annotated with a *security level* and a number of constraints can be imposed on the existence of edges in the application domain. For example, the constraint of Figure 12 states that an atomic content can only be kept by a server with a higher security level than the content level, while Figure 13 constrains an atomic content to be served only to more secure devices. Analogous constraints (not shown here) require users to only use devices with a higher security level and to lock content with lower security level.

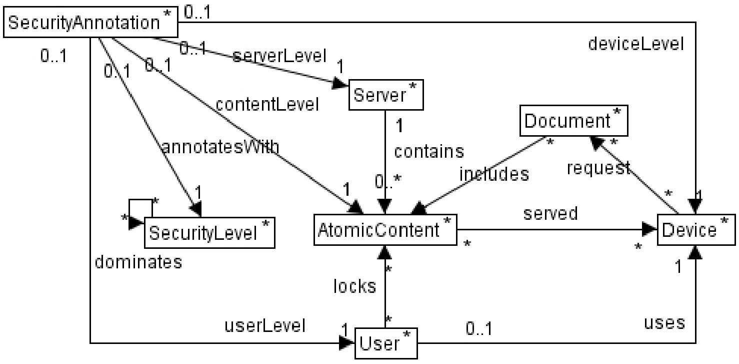


Figure 11: Introducing security annotations on the domain model.

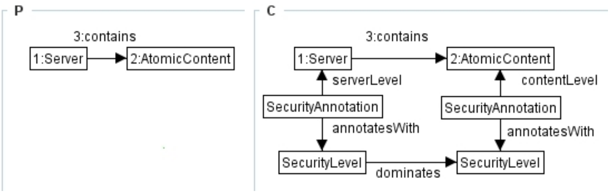


Figure 12: Constraining distribution of content over servers.

Due to this additional constraint on service of content, particular care must be given to the roles of the rules in the control expression for document service, in order to ensure termination of the process. More precisely, one needs to make the conditions on starting and running the process more stringent, and the conditions on concluding the process more relaxed, acting on the application conditions of the corresponding rules. To this end we need different constructions.

(a) For the starting rule, one has three choices with respect to the security level

- (a1) For an *existential* condition at least one content can be served.
- (a2) For a *universal* condition all of the contents can be served.
- (a3) A *contextual* condition relaxes the universal condition so that the security level is not checked on contents excluded from service based on other conditions.

(b) For an intermediate rule, it must be applied only in conformity with the security constraint.

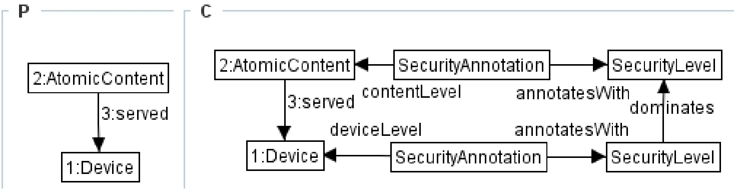
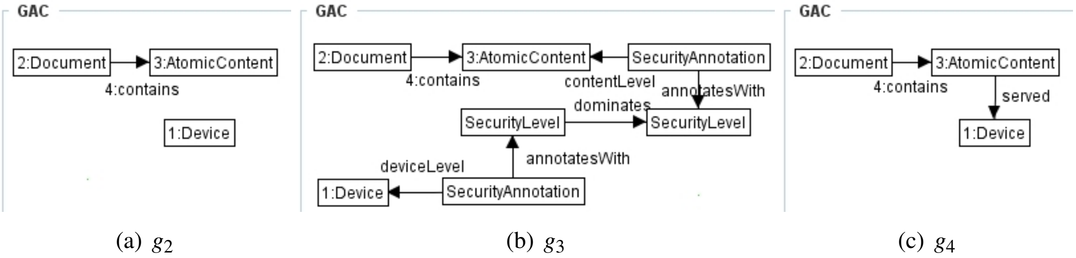


Figure 13: Constraining access to secured documents.

(c) For the concluding rule, depending on choice made for (a), a different GAC is required. For (a1) and (a3) all elements which were not served based on some criterion can also be present, for (a2) all elements have had been served.

We deal with case (a) by considering the nested GACs of Figure 14. The graph g_2 on the left describes a situation in which the document contains some content, the graph g_3 in the middle describes a situation where content is servable in a secure way, while g_4 on the right depicts the situation where content has already been served. For the case (a1) the associated formula is $\exists m^{g_2}$, for case (a2) we have $\forall m^{g_2}.\exists m^{g_3}$, while for case (a3) the formula is $\forall m^{g_2}.(\exists m^{g_3} \vee \exists m^{g_4})$.



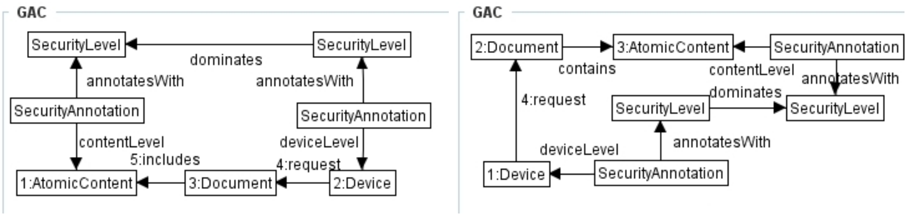
(a) g_2 (b) g_3 (c) g_4

Figure 14: Application conditions for starting the content service process.

Figure 15 shows on the left the graph g_5 used in the GACs for dealing with case (b), and on the right the graph g_6 in the GACs for case (c). For (b) we simply add an application condition requiring the existence of a morphism m^{g_5} while for case (c), to deal with subcase (a1) and (a3) we extend the formula of Section 3 so that it becomes $\forall m^{g_0}.(\exists m^{g_1} \vee \exists m^{g_1})$, with the same convention on the name of matches. To deal with subcase (a2) the original GAC built on graphs g_0 and g_1 in Section 3 can be employed. All of the graphs g_3, g_5, and g_6 are constructed as the graph M of Figure 10, starting from the corresponding rules and the constraint in Figure 13.

The resulting version of the transaction is correct in the sense of Proposition 1.

Proposition 1 *Let $\Gamma = r_1; r_2^*; r_3$ be the control expression for the transaction above and let $\Gamma^a = r_1^a; r_2^{a*}; r_3^a$ be its annotated version. Then, for any graph G in which Γ would succeed, Γ^a would succeed on the annotated version G'*



(a) g_5 (b) g_6

Figure 15: Application condition for serving content and terminating a request.

Proof. In general, the only case in which Γ does not succeed is one where some element has not been processed by r_2 due to the failure of some application condition. In the case of document serving, there is no such case. For the annotated version, in case (a1), r_1^a is applied if at least a content can be served. r_2^a will serve this content, plus any other legitimate content. r_3^a can then fail only if some content which was servable has not been served, but this cannot happen due to the iteration of r_2^a. For case (a2) all contents were servable and iterating r_2^a has served them, while for case (a3) a content is non- servable if already served. □

In order to detect violations of security, one can modify the GAC for rule r_3^a to check that all contents are served. For the cases (a1) and (a3) the failure of this rule would signal the violation.

Figure 16 presents the type graph for annotating the document distribution domain with localisation values. The dominating relation, called *accesses* is reflexive and not transitive and indicates direct accessibility of locations. We are not interested here in whether it is symmetric or not. The constraint in Figure 17 (left and middle) states that a user can only use a device located in some position accessible from the user location. Introducing this constraint modifies the rule for changing a device (see Figure 3) by introducing a GAC based on graph g_7 on the right of Figure 17 associated with the morphism $g_L^7 : L \to g_7$. In a similar way, constraints on location add GACs to the rules managing the request process, following the construction described above.

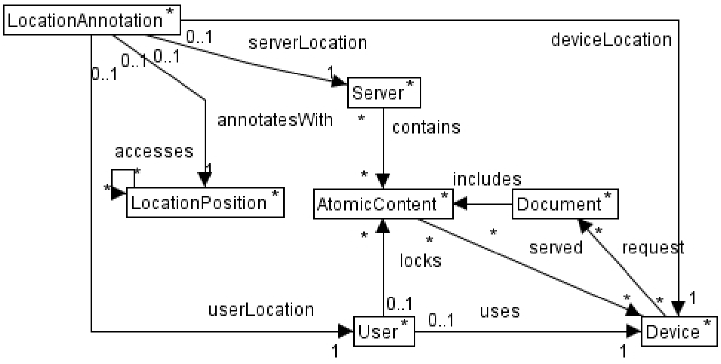
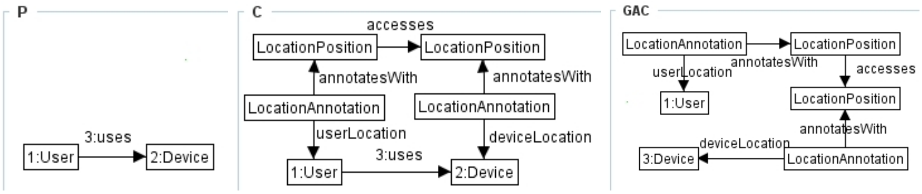


Figure 16: Introducing locality annotations on the domain model.

The effect of combined annotations with security and localisation context is readily expressable by taking the co-limits of the resulting application conditions. Alternatively, one can construct layers of nested annotations, each time redefining as application domain the resulting of applying an annotation layer. This opens the way to unifying several models for role-based access control. As an example, one can first annotate an access domain with role information, then the resulting domain with temporal information, then with geographical information, etc.



(a) (b) (c) g_7

Figure 17: Constraining use of a device based on location.

5.1 Modeling GeoRBAC

In this section we discuss how the approach can be applied also in the case when the constraints involve not simply the existence of an instance of the dominating relation between values of the annotation domain, but require that specific values be assigned to elements. We show this extension by considering the addition of location annotations to the Role Based Access Control model [SFK00]. The players in this model are users, roles, sessions and permissions. For this example we ignore the permissions, which play no part in the extensions with localities. In order for a user to access objects with permissions, the user opens a session (which becomes *owned* by the user) that is associated with some role authorized for the user (the system then checks that the permissions requested belong to the role, but this is irrelevant in our example). The simple rule corresponding to this operation is shown in Figure 18.

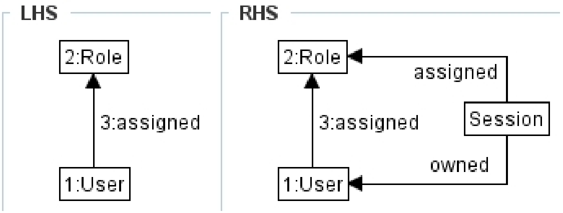


Figure 18: Creating a session based on role.

Following [DBCP07] we are given a set of features $F = F_s \cup F_{ns}$ distinguished in spatial features F_s and non-spatial features F_{ns}, and a function $LocObj : F \to (GEO \cup \{\perp\})$ to associate features with geographical locations. Features are typed, and a partial order on these types represents the usual notion of containment (a room part of a building, a town inside a region, etc.). The type graph for this extended model of RBAC with annotations is shown in Figure 19.

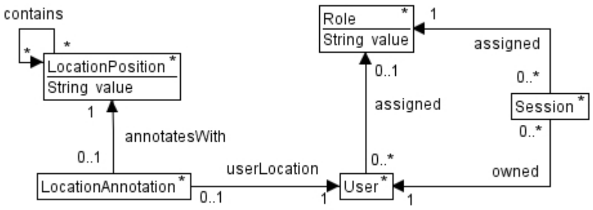
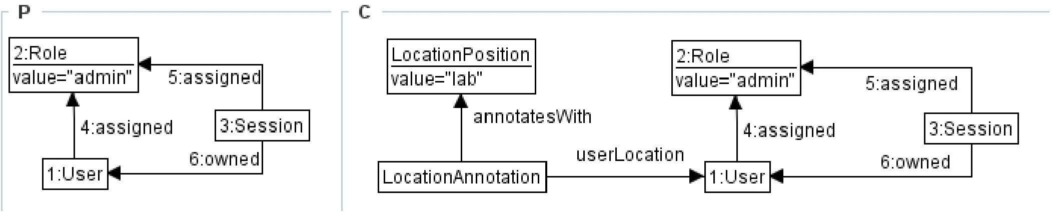


Figure 19: The type graph for annotating RBAC with geographical information.

In this extended model, we want to add a constraint, shown in Figure 20: a user can open a session in the role of *administrator* only if the user is present in a certain location, say a *laboratory*. The rule in Figure 18 is no longer safe as it would allow the session to be opened in the administration role even when the user is elsewhere. General application conditions need to be added. The two components resulting from of the application of the procedure in Section 4 are shown in Figure 21. By calling L the LHS of Figure 18 and g_8, g_9 the two components of the GAC from left to right in Figure 21, we obtain the formula $\forall g_L^8 : L \to g_8.\exists g_8^9 : g_8 \to g_9$. Note that in this case g_8 corresponds to W in Figure 10, while g_9 corresponds to M.



(a) g_8

(b) g_9

Figure 20: The location constraint for creating a session as administrator.

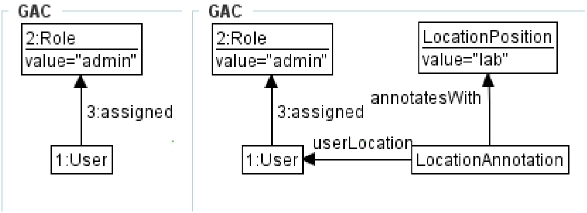


Figure 21: The elements of the nested GAC for the rule in Figure 18.

6 Related work

The problem of adapting organizational policies to context conditions is not new. Several papers extend well known security models to incorporate context conditions, e.g. the location of the requester or of the resource requested, or the time and duration of the request.

In the standard Role Based Access Control model RBAC [SFK00], users have access to a resource only if an appropriate role, authorized for the user and associated to the requested operation, has been activated. In the Geo-RBAC [DBCP07] extension, such a role is activated only if the context is right, viz. if the user location is correct. Spatial and location-based information is modelled as spatial entities and the whole hierarchy of RBAC models is extended with this addition using role schema. The framework used is the usual set-theoretic one. Another important extension is Temporal RBAC [BBF01], where roles are made available to users only at certain times. The organizational policies are modified depending on an external context with periodic role enabling and disabling, role triggers and temporal dependencies among actions. Again, the formalism is set-theoretical and the approach is ad hoc. Despite both Geo-RBAC and GTRBAC being extensions of RBAC taking into account an external context for the activation of roles, one approach is not easily adapted to the other.

Koenig [Koe05] develops an extensive theory of annotated hypergraphs, where a hypergraph annotation is a morphism from a (typed) hypergraph to a (complete) lattice and morphisms can also be annotated by functions, in such a way that the composition of annotated morphisms results into a morphism annotated by the composition of the annotating functions. In this paper we embed annotations in the typing system, relaxing conditions on the domain and providing a different mechanism for composition of annotations.

Ehrig *et al.* [EEHP06] provide a general approach to the construction of application conditions on rules, while Koch and Parisi [KP06] present a method for composition of security policies through the addition of application conditions. In both cases, the analysis deals with individual rules or sets of rules, but not with their composition into transactional specifications.

Lambers *et al.* [LET08] explore conditions for success of sequences of DPO rules with NACs, but they do not describe how these conditions change when new constraints are added on rule application. However, their analysis can be used to discover causes for failure in modified rules.

7 Conclusions

We have presented an approach to enriching models of some application domains with constraints coming from contextual domains through the use of annotations relating values from the latter to model elements of the first. In particular, we have focused on the generation of application conditions on model rules, and on the management of transactions following a typical inception-execution-pattern, where execution is modelled as iteration of a single rule.

The study has been conducted for the case when only nodes can be annotated, but it can be readily extended to the annotation of edges, by considering them at the metamodel level, and of patterns, by introducing a special type of annotation elements and considering groups of elements associated with them through annotation edges. Contextual constraints can then be used to coordinate the annotations of several model elements belonging to the same pattern.

In this paper, we have considered only increasing or decreasing rules in the application domain and positive atomic constraints for the contextual domain. The case of more general rules and of negative or nested contextual constraints should also be investigated. Moreover, we need to properly formalise, along the lines discussed in Section 4, the construction of repair actions for situations in which elements which in a rule have also to be constrained. Finally, while the use of general sequences in a transaction can be, under certain circumstances, reduced to that of a single rule [LET08], more general control expressions could require more complex constructions.

Bibliography

[AH02] W. M. P. van der Aalst, K. M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002.

[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.

[DBCP07] M. L. Damiani, E. Bertino, B. Catania, P. Perlasca. GEO-RBAC: A spatially aware RBAC. *ACM Trans. Inf. Syst. Secur.* 10(1), 2007.

[EEHP06] H. Ehrig, K. Ehrig, A. Habel, K.-H. Pennemann. Theory of Constraints and Application Conditions: From Graphs to High-Level Structures. *Fundam. Inform.* 74(1):135–166, 2006.

[KLM$^+$97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, J. Irwin. Aspect-Oriented Programming. In *ECOOP*. Pp. 220–242. 1997.

[Koe05] B. Koenig. A general framework for types in graph rewriting. *Acta Informatica* 42(4/5):349–388, 2005.

[KP06] M. Koch, F. Parisi Presicce. UML specification of access control policies and their formal verification. *Software and System Modeling* 5(4):429–447, 2006.

[LET08] L. Lambers, H. Ehrig, G. Taentzer. Sufficient Criteria for Applicability and Non-Applicability of Rule Sequences. *ECEASST* 10, 2008.

[Löw93] M. Löwe. Algebraic Approach to Single-Pushout Graph Transformation. *TCS* 109(1&2):181–224, 1993.

[LWF03] A. Lopes, M. Wermelinger, J. L. Fiadeiro. High-order architectural connectors. *ACM Trans. Softw. Eng. Methodol.* 12(1):64–104, 2003.

[SFK00] R. S. Sandhu, D. F. Ferraiolo, D. R. Kuhn. The NIST model for role-based access control: towards a unified standard. In *Proc. ACM Wks. on RBAC*. Pp. 47–63. 2000.