Electronic Communications of the EASST
Volume 73 (2016)

Graph Computation Models
Selected Revised Papers from GCM 2015

Conditions, constraints and contracts: On the use of annotations for policy modeling

Paolo Bottoni, Roberto Navigli, Francesco Parisi Presicce

20 pages

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

Conditions, constraints and contracts: On the use of annotations for policy modeling

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

[1]Università di Roma "Sapienza"

Abstract: Organisational policies express constraints on generation and processing of resources. Application domains, however, rely on transformation processes, which are in principle orthogonal to policy specifications, so that domain rules and policies may evolve in a non-synchronised way. In previous papers, we proposed annotations as a flexible way to model aspects of some kinds of policy. Annotations could be used to impose *constraints* on domain configurations, and we showed how to derive application *conditions* on transformations, and how to annotate complex patterns. We extend the approach here in different directions: we allow domain model elements (individual resources or collections thereof) to be annotated with collections of elements; we propose an original construction to solve the problem of *orphan* annotations, when annotated resources are consumed; we introduce a notion of *contract*, used by a policy to impose additional pre-conditions and post-conditions on rules for deriving new resources. We also show how contracts for refined rules can be derived from contract schemes defined on some rule kernel. We discuss a concrete case study of linguistic resources, annotated with information on the licenses under which they can be used. The annotation framework allows forms of reasoning such as identifying conflicts among licenses, enforcing the presence of licenses, or ruling out some modifications of a licence configuration.

Keywords: Contracts, Annotations, Licenses, Policies

1 Introduction

Organisational policies express constraints on generation and processing of resources which are accepted by agents subject to, or anyway acknowledging, the organisation authority. Agents, however, retain the ability to operate on such resources according to their own strategies, as long as the results of these operations conform to the policy, or are used in areas not subject to it.

A typical example is that of licenses, which define ways in which software resources can be manipulated and made available for usage by third parties. The diffusion of open data [ABK+07] and of public repositories allows a dissemination of resources which can be employed in different ways, ranging from their simple replication for integration in local pools, to the creation of sophisticated services. While non proprietary resources can usually be accessed without restrictions, access to the results of resource manipulations could be subject to restrictions related to the safeguard of intellectual property; however, using certain resources in the construction of a service may prevent the possibility of imposing such restrictions, or force forms of access compatible with those for the used resource.

In [BP13a, BP13b] we proposed the usage of annotations as a flexible way to indicate the aspects for which domain model elements can fall under some policy, and showed how annotations can be used to impose *constraints* on configurations of domain resources, how to derive application *conditions* on their transformations, and how to annotate complex patterns. In particular, this allows the management of situations in which access policies change, or different policies have to be applied simultaneously. Annotations are indeed a way of flexibly associating elements of different domains, and a number of techniques have been developed to express the constraints imposed on transformations modeling the evolution of elements in an *application* domain, when they are subject to constraints depending on annotations with elements of a *contextual* domain.

We extend the notion of annotation in four directions: first we propose an original construction for transformations of annotated models, which solves the problem of *orphan annotations*, i.e., dangling annotation when annotated resources are consumed. Second, we allow domain model elements to be annotated with *collections* of elements from the annotations domain, which can be collectively applied to individual elements or collections thereof. Third, we consider violations of constraints induced by the creation of elements for which an annotation must be provided and define *constraint repair actions* to solve such situations. Finally, we introduce a notion of *contract*, by which a policy imposes additional pre- and post-conditions on rules for deriving new resources. Contracts are naturally generalised to be viewed as *contract schemes*, which allows the derivation of contracts on rules refining a kernel rule to which the original contract applied.

Again, the use of contracts and contract schemes leads to an original notion of transformation under contracts. In all the considered cases, the definition of transformations relies on the specific features of annotations and classical categorial constructions.

Paper organisation. After recalling fundamental notions in Section 2, we provide a construction for avoiding orphan annotations in Section 3 and introduce a motivational case study, concerning linguistic resources annotated with licenses, in Section 4. This is used in Section 5 to illustrate the model for rewriting under annotation constraints and contracts. Section 6 presents the extension to contract schemes. After revising related work in Section 7, Section 8 concludes the paper.

2 Preliminaries

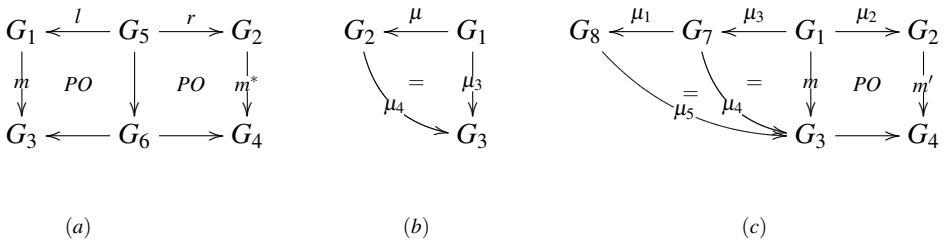
Graphs and morphisms. We recall classical notions from graph transformation theory (see [EEPT06]).

Definition 1 (Graph) A (directed) *graph* is a construct $G = (V_G, E_G, s, t)$, where V_G is a set of *nodes*, E_G is a set of *edges*, $s\colon E_G \to V_G$ and $t\colon E_G \to V_G$ are the *source* and *target* functions.

Definition 2 (Graph morphism) Given two graphs G_1 and G_2 a *morphism* $\mu\colon G_1 \to G_2$ is a pair of functions $\mu_V\colon V_{G_1} \to V_{G_2}$, $\mu_E\colon E_{G_1} \to E_{G_2}$ such that μ_E preserves the images of sources and targets, i.e. for $e \in E$ we have: $s_2(\mu_E(e)) = \mu_V(s_1(e))$ and $t_2(\mu_E(e)) = \mu_V(t_1(e))$.

We use morphisms for a number of purposes, some of which exemplified through Figure 1.

Typing. For G_1 a graph, and G^T a *type graph* (such that $V_{G_1} \cap V_{G^T} = \emptyset$, $E_{G_1} \cap E_{G^T} = \emptyset$), $\mu_T\colon G_1 \to G^T$ is a *typing morphism* if μ is total. Given G_1 and G_2 typed on the same G^T, $\mu : G_1 \to G_2$ is a *type preserving morphism* if $\forall x \in V_{G_1} \cup E_{G_1}\, \mu_t(\mu(x)) = \mu_T(x)$. Typed graphs are



(a) (b) (c)

Figure 1: (a) DPO Derivation, (b) satisfaction of constraint, (c) SPO derivation with AC.

extended to attributed typed graphs defining specific sets of attributes for them. A node of a certain type will be associated with a value for each of the attributes defined by the type.

Transformation rule. One (as in the Single Pushout Approach, SPO, see Figure 1 (c)) or two (as in the Double Pushout Approach, DPO, see Figure 1 (a)) type-preserving morphisms are used to define rules. In either case, a rule p identifies a graph G_1 and the applicability of p to a graph G_3 depends on the existence of a total type-preserving morphism $m\colon G_1 \to G_3$. If a rule p is applied to a graph G_3 to produce graph G_4, we write $(G_3, G_4) \in \Longrightarrow_p$.

Constraint. $\mu\colon G_1 \to G_2$ defines a *constraint* together with a satisfaction relation $\models$. For any graph G_3, $G_3 \models \mu$ *iff* for each morphism $\mu_3\colon G_1 \to G_3$, there exists a morphism $\mu_4\colon G_2 \to G_3$ such that the triangle of Figure 1(b) commutes[1]. A *negative* constraint, denoted by $\neg\mu\colon G_1 \to G_2$, is satisfied by G_3 if no such morphism μ_4 exists for some μ_3. The particular case $\neg\mu\colon G_1 \to G_1$, defines a *forbidden graph* and is represented by the single graph G_1.

Application condition. Given a (DPO or SPO) rule p with identified graph G_1, $\mu_1\colon G_7 \to G_8$ is an *application condition* (AC) for p if it restricts the relation $\Longrightarrow_p$ to pairs (G_3, G_4) for which there exist morphisms $\mu_3\colon G_1 \to G_7$, $\mu_4\colon G_7 \to G_3$ and $\mu_5\colon G_8 \to G_3$ such that the triangles in the diagram of Figure 1(c) commute (in the SPO approach the square in it is a pushout, analogously for the two squares in Figure 1(a)). The requirement that no such μ_4 exists is called a *negative application condition* (NAC).

All of the above is naturally extended to sets of rules and to attributed type graphs, so that graph constraints and application conditions can include constraints on the values that attributes of the matched nodes must have. Rules can require updates on the values associated with preserved nodes, or assignment of values for the created nodes.

Annotations. Annotations of elements of a domain $\mathscr{D}_1$ with nodes of a domain $\mathscr{D}_2$ are defined via nodes of types derived from `AnnotationNodeType`. We call $\mathscr{A}$ the domain of such annotation nodes. Each annotation node $a \in \mathscr{A}$ participates in exactly one instance of the pattern $\pi_a = x \xleftarrow{e_1} a \xrightarrow{e_2} y$, where $x \in \mathscr{D}_1$, $y \in \mathscr{D}_2$, e_1 is an edge of an application-dependent type and e_2 is an edge of type `annotatesWith`. We work on type graphs TG resulting from the disjoint union of the type graphs defining $\mathscr{D}_1$ (TG_1) and $\mathscr{D}_2$ (TG_2), together with the relevant annotation node and edge types, and we consider two types of constraints related to annotations, assuming that all the constraints on the application domain are preserved by the domain rules.

[1] G_1 is also called P, from *premise*, and G_2 is also called C, from *conclusion*.

Constraints of the first type have the form $\mu : \boxed{A} \rightarrow \boxed{X \xleftarrow{e_1} A \xrightarrow{e_2} Y}$, for X some type in TG_1, Y some node type in TG_2, A an annotation node type and e_1 and e_2 as described before, with possible restrictions on the association of X and Y, and they are derived from π_a. Well-formedness results from conformance with the disjunction of all such constraints. Given a type graph TG, $\mathscr{L}_{\pi_a}(TG)$ is the language consisting of all the graphs typed on TG which are well-formed with respect to the individual domains and the annotation pattern. Note that, according to the metamodel presented in [BP13b], X can be an edge type, since associations result to be annotatable entities. To be precise, we should enrich our model of annotated graphs so that the codomains of the s and t functions include the set E for the subgraphs in the TG_1 component.

The second type of constraints expresses specific policies via morphisms of the form $\mu : G_1 \rightharpoonup G_2$ where G_1 is typed on the TG_1 and TG_2 components of TG, G_2 is typed on TG with well-formed annotations, and its projection on the disjoint union of TG_1 and TG_2 is isomorphic to G_1. In other words, there exists two subgraphs, $G_1^1 \in \mathscr{D}_1$ and $G_1^2 \in \mathscr{D}_2$, with jointly surjective immersions in G_1, such that G_1^1 includes all the elements which receive an annotation in G_2, in the context in which the annotation is required. The discrete graph G_1^2 contains all and only the nodes reached by an `annotatesWith` edge in G_2. We call constraints of the latter form *annotation constraints*. All the constraints considered in the paper are annotation constraints, while all the rules in the paper are typed only on TG_1.

In Section 5 we will introduce *contracts* based on morphisms involving graphs typed on TG.

3 Annotations and collections

Definition 3 from [BP13b] extends graphs and morphisms to include the notion of *box*.

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

A *type B-graph* includes a set of box types B^T which are sources or targets for edge types. Moreover, a function $cnt^T : B^T \rightarrow \wp(V^T \cup B^T)$ associates each type of box with the set of types of elements it can contain. Similarly, a (total) morphism on B-graphs was defined in [BP13b] by adding a component μ_B, preserving the content function, i.e. for all $x \in V \cup B, b \in B$, one has $x \in cnt_1(b) \implies \mu_{V \cup B}(x) \in cnt_2(\mu_B(b))$, with $\mu_{V \cup B}$ the (disjoint) union of μ_V and μ_B. In [BP13b] total morphisms allowed DPO transformations of B-graphs.

Based on these notions, annotation nodes can be used not just with reference to individual nodes in TG_1 and values in TG_2, but also to collections in either domain, i.e. a collection of annotation values can be used in an atomic way to annotate some resource or collection thereof. However, in the original formulation of rewriting with boxes in [BP13b], removing an annotated element from a model $G' \in \mathscr{L}_{\pi_a}(TG)$ would create dangling annotation edges, forbidding rule application in the DPO form. The situation is no better if the SPO form is employed: in this case, if the annotation edges were removed, there would be *orphan annotations*, i.e. annotation nodes

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

to which no annotation edge is attached, resulting in a graph not in $\mathscr{L}_{\pi_a}(TG)$. Hence, we devise an original mechanism, based on Construction 1, to complement a transformation performed according to the DPO appproach in TG_1 so that well-formedness is preserved.

Construction 1. With reference to Figure 2, let its top two rows depict the usual DPO application of a rule $L \leftarrow K \rightarrow R$ to a graph G obtained by applying to G' the morphism $\mathbf{f_1}$ induced by the forgetful functor given by the restriction of TG to TG_1. The unique morphism induced by the immersion of the image of $\mathbf{f_1}(G')$ into G' is also derived. Then D' is the unique (up to isomorphisms) graph in $\mathscr{L}_{\pi_a}(TG)$, i.e. with well-formed annotations, which is maximal w.r.t. the occurrences of the annotation pattern and for which the left square at the bottom of the diagram in Figure 2 is a pullback (hence its restriction to the elements typed on TG_1 is isomorphic to D). Finally, H' and morphisms h' and h_π are obtained by calculating the pushout of H and D' along D.

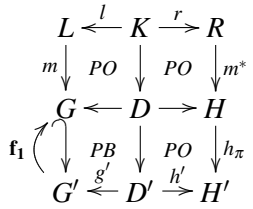


Figure 2: Avoiding orphan annotations.

Figure 3 illustrates Construction 1 with a model example where teams in an organisation temporarily gather members for specific tasks [BP13b]. G' describes a configuration where `frank` - a member, together with `paul`, of the `softEng` team - is also the only member of the `security` team. For both `frank` and `security`, time annotations indicate that they can operate within the organisation only at daytime. The DPO rule at the top, removing a team with only one member, is applied to G, the projection according to f_1 of G', by identifying `security` and `frank` with `1:Team` and `2:Member` in L, respectively. As a consequence, the annotation on `security` and the edges touching it are removed, updating the *cnt* function accordingly, while the one for `paul` is preserved. Note that Construction 1 applies to removal of the application domain elements, not of contextual ones. Actually, we assume here that contextual domains are fixed. The correctness of Construction 1 is expressed by Proposition 1.

Proposition 1 *For $G' \in \mathscr{L}_{\pi_a}(TG)$ and a rule $L \leftarrow K \rightarrow R$, the graph H' generated following Construction 1 is in $\mathscr{L}_{\pi_a}(TG)$. Moreover, $Ann(H') \subseteq Ann(G')$, where $Ann(X)$ denotes the set of annotation nodes in a graph X.*

Proof. We first observe that $Ann(H') = Ann(D')$ since H, which is typed on TG_1, does not present any new annotation. . Hence $Ann(H') \subseteq Ann(G')$. Since D' is well-formed by definition, also X', the pushout object of D' and H, does not have orphan annotations. □

4 Case study:linguistic services and licenses

A *license* specifies admissible forms of access, usage and redistribution of resources and of the results of manipulating them. While individual resources can be associated with specific licenses, the licenses for a resource usually depend on some policy proper to the repository from which it is extracted. Repositories often publish resources under a number of licenses, to be all simultaneously respected, and which are normally transferred to the extracted resources. As this im-

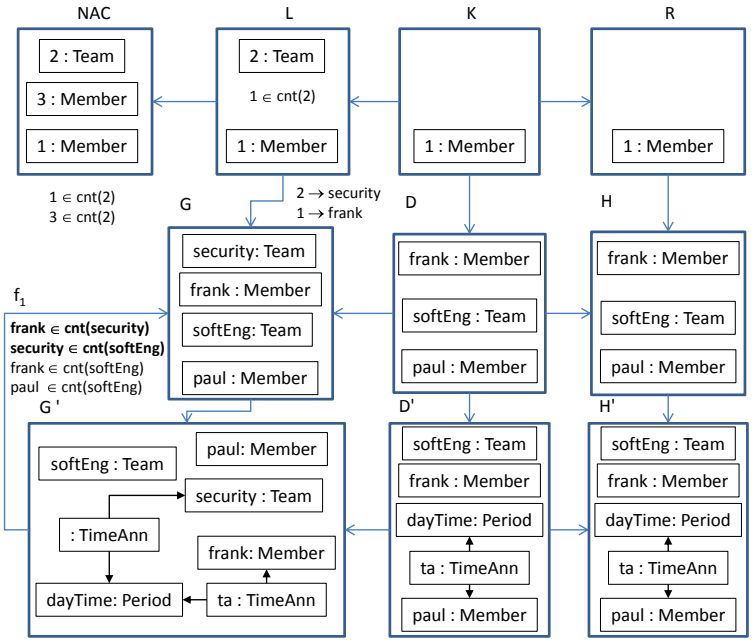


Figure 3: An example of Construction 1 for the removal of an annotated team.

poses some form of compatibility among licenses, deciding whether a certain usage is admitted may become complex. As an example, elements published under a `Creative Commons`, `CC`, scheme (hence of `PublicDomain` (`PD`)) can be associated with combinations of the following licenses: `NC` for `NonCommercial` (the resource cannot be used for commercial purposes), `BY` for `Attribution` (credit to the author is acknowledged), `SA` for `ShareAlike` (derived resources must be redistributed preserving the original licenses), and `ND` for `NoDerivatives` (remixing, transforming, or building upon the resource may not grant redistribution).

BabelNet[3] [NP12] is a multilingual semantic network of semantically related millions of concepts (e.g. the **apple fruit** concept) and named entities (e.g. the **Apple Inc.** entity). A single node in the network is called a Babel *synset* and contains a set of terms which express a given concept or named entity in different languages. For instance, the **apple fruit** concept is represented by the synset $\{apple\_EN, pomme\_FR, mela\_IT, \ldots, manzana\_ES\}$. A term in a synset is called *sense* (e.g., *apple_EN* in the above synset is the fruit sense of the ambiguous word *apple*).

BabelNet itself is obtained as the result of the automatic mapping (considered in turn a kind of resource unique to BabelNet) and integration of several, publicly available, knowledge repositories, each providing resources (e.g. synsets and senses) under different licenses. For example, WordNet [Fel98], Wikidata[4] and parts of the OMWN project [BP12a] are released under a permissive license, allowing any use, whether commercial or non-commercial, of the data; Wikipedia and Wiktionary are released with a `CC-BY-SA` license; OmegaWiki and other wordnets in OMWN are released under `CC-BY`; the Basque Wordnet in OMWN is released under

[3] http://babelnet.org
[4] http://wikidata.org

CC-BY-NC-SA and so is the BabelNet mapping Unfortunately, not all of the licenses are compatible with one another, as shown in the compability chart for CC licenses in Table 1, where ✓ indicates compatibility, × incompatibility, and ! that usage is not recommended. For instance, Wikipedia, whose license is CC-BY-SA, cannot be merged with data from the Basque Wordnet or the BabelNet mapping. Interestingly, some mergings can be done in one direction only, e.g. from a resource in a CC-BY repository, such as OmegaWiki, one can derive a new resource with a more restrictive CC-BY-SA license, thereby making it compatible with, e.g., Wikipedia.

Table 1: The compatibility chart for Creative Commons licenses.

Compatibility chart		Terms that may be used for a derivative work or adaptation						
		BY	BY-NC	BY-NC-ND	BY-NC-SA	BY-ND	BY-SA	PD
Status of original work	PD	✓	✓	✓	✓	✓	✓	✓
	BY	✓	✓	✓	✓	✓	✓	!
	BY-NC	!	✓	✓	✓	!	!	!
	BY-NC-ND	×	×	×	×	×	×	×
	BY-NC-SA	×	×	×	✓	×	×	×
	BY-ND	×	×	×	×	×	×	×
	BY-SA	×	×	×	×	×	✓	×

However, the opposite is not possible, as a SA license is not modifiable. Hence, some licenses, such as CC-BY-NC-SA and CC-BY-SA, are inherently and mutually incompatible. To solve this problem, BabelNet is viewed as a collection of knowledge resources with heterogeneous licenses. For instance, it is possible to consider a subset of resources which can be used commercially, e.g. Wikipedia, WordNet and others. However, resources with NC license (e.g. the Basque Wordnet and the BabelNet mapping) cannot be used commercially. As the mapping is the enabling technology for interconnecting the various resources into a whole unified, multilingual network, any commercial use requires a suitable license from the BabelNet's authors.

To represent the management of licenses and languages in BabelNet, we model synsets, senses, etc. as nodes (or boxes) of specific types from a domain, $\mathscr{R}$, of *Resources*. Similarly, licenses are modeled as nodes of type `License`, from the domain of *Licenses*, $\mathscr{L}$, with an attribute `name` ranging over strings identifying the different kinds of license, and languages are modeled as nodes of type `Language`, from the domain of *Languages*, $\mathscr{K}$, with `name` ranging over the available languages. We consider $\mathscr{R}$ as the application domain, and $\mathscr{L}$ and $\mathscr{K}$ as contextual domains, typing the annotation edges accordingly. Thus, for `xxx` a type in $\mathscr{R}$, edges of types **xxx**`LicAnnotation` and **xxx**`LangAnnotation` allow its annotation with elements of $\mathscr{L}$ and $\mathscr{K}$, through edges of type `annotatesWith`.

Figure 4 presents the type graph TG for BabelNet. Stereotypes indicate whether an element is of the *Node* or *Box* sort, and if it comes from the domain $\mathscr{R}$, $\mathscr{L}$ or $\mathscr{K}$, or is an `AnnotationTypeNode`. In $\mathscr{R}$, a `Sense` has an attribute `content`, with values in the sort of strings. Moreover, the three types of box, `Synset`, `Collection` and `LicenseBundle`, can contain only elements of suitable types, namely `Sense`, `Synset` and `License`, respectively. A `Synset` has a `concept` represented by a collection of senses. `LicenseBundleAnn` is of the *Node* sort and is derived from `AnnotationTypeNode`; it can be used to relate resources

with `LicenseBundle`. However, as it inherits from `LicenseAnn`, one can annotate any element in the resource domain with single licenses or with license bundles. A `Request` to *obtain* a `Collection` can be annotated with both license and language information (not indicated to avoid cluttering) and *activates* a `Service` *producing* the collection.

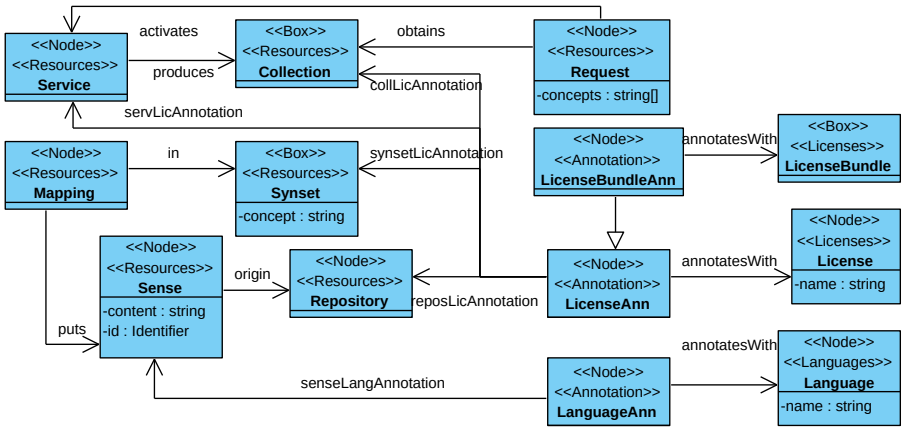


Figure 4: The type graph for modeling the running example.

The basic working of BabelNet services is modeled via increasing rules in the *Resources* domain. Rule `createCollection` in Figure 5 (top) creates an initially empty collection to be served in response to a query to define the concepts in a set Z. Two rules allow the inclusion in the collection of a synset for a concept in the request. Rule `addSynset` in Figure 5 (middle) is used to add an already available synset to the collection if it is not already there, as indicated by the NAC. The other one, not shown, creates and adds a synset for a concept, if it does not exist already. Rule `createMapping` in Figure 5 (bottom) populates synsets with senses representing the concept, according to a mapping, with a NAC to avoid including a sense twice.

In all these cases, the bundle of licenses associated with the invoked service must be associated with the produced collection, as will be discussed in Section 5.2. Moreover, requests can be further characterised, for example by annotating them with particular licenses or languages, so that only senses annotated with those licenses or languages are included in the obtained collection, as per suitable application conditions, following the constructions in [BP13a].

5 Maintaining consistency with constraints and contracts

While the result of applying a rule is guaranteed to produce a correctly typed graph, it might be the case that such a graph does not conform to further conditions imposed on the model at hand. We identify two dimensions along which conditions can be distinguished: one pertaining to the identification of the domain for which the condition is defined (whether the application domain or the domain resulting from the annotation process) and one relative to the scope of the condition (whether global to the domain or local to specific transformations).

Concerning the latter dimension, we use constraints to impose well-formedness conditions for a domain, and identify mechanisms to ensure that the transformation process preserves them.

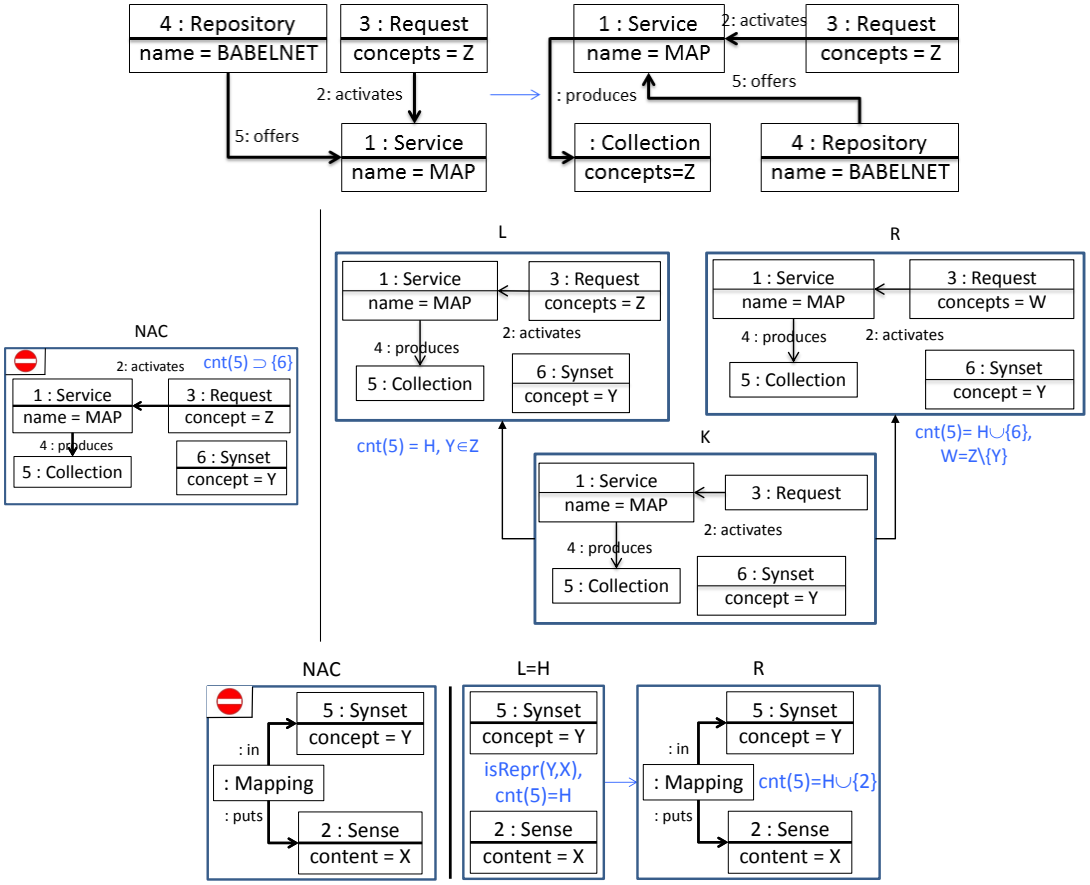


Figure 5: Rule `createCollection` prepares a container for serving a request (top). Rule `addSynset` sets up a synset for a requested concept (middle). Rule `createMapping` relates a sense to a synset (bottom). Vertical lines separate NACs from rule morphisms.

In [BP13b], we have presented some constructions to derive application conditions for rules from annotation constraints. By leaving the rule morphism alone, we maintain a form of separation of concerns, making rule reuse simpler when different forms of annotation are involved. Intuitively, those constructions work when a rule adds elements related to elements matched by the left-hand side, but the annotation constraint imposes these relations to exist only between elements annotated in specific ways. An application condition ensures that such an annotation context already exists in the host graph when the rule is applied.

In this section we focus on situations in which adding application conditions is not sufficient, since the required context cannot already exist, in particular if a newly created element must be annotated in specific ways. This would require the right-hand side of a rule to be enriched with the appropriate annotation, but then the rule would no longer be defined only on the application domain. To approach this problem, we consider separately situations violating global constraints, and situations violating conditions on specific rules, that we model as contracts. Since we are interested in rules which add new elements, i.e. $L = K$ for a DPO rule p, we present them as

simple morphisms (called $\mu_1$or μ_2 in the following diagrams), and denote by α the morphism induced by the application of such a p between two graphs G_7 and G_8 with $(G_7, G_8) \in \Rightarrow_p$.

We consider that graphs can be associated with information on values from some algebraic structure D, as is typical for attributed graphs, and assume that types enforce constraints on the set of attributes which can be associated with their instances together with possible constraints on their admissible values. The morphisms defining constraints and rules from Section 2 can therefore be enriched to express constraints on the values of these attributes. In order to treat with these values, we follow the approach of symbolic attributed graphs [OL10b], which are pairs $\langle G, D \rangle$ where G is a graph labelled with values from the Σ-algebra D, for Σ some signature, using constraint satisfaction to evaluate conditions and attributes. To be precise, attributes are here represented as variables and constraints refer to the values of these labels.

In particular, we consider the use of delayed constraint solving for symbolic graphs [OL10a], which allows the evaluation of constraints when all the required context is available. DPO rules therefore assume the form $\langle p, \Gamma \rangle$, where $p = L \leftarrow K \rightarrow R$ is a DPO rule on graphs labelled with variables with values in D and Γ is a collection of constraints on all the variables in p.

We also define a version of symbolic graphs for B-graphs, by including constraints on values of the *cnt* function for boxes. To this end, we supplement the algebra D with the Boolean algebra on the set $V \cup E$. We can then use additional constraints using the algebra operators $\cup$, $\cap$, and $\neg$. The shortcut $a \in X$ for some element a and some set X is used to denote the constraint $\{a\} \subset X$. The pushout construction is then complemented by taking the conjunction of such constraints, while the pullback construction by taking their disjunctions (as in classical symbolic graphs).

In the graphical representation of rules and constraints, we will use the traditional UML-like representation of types and attributes, and present separately the constraints on the variables.

5.1 Management of constraints

As shown in Table 1, licenses can require or forbid the presence of one another. While the first case can be modelled by a positive constraint[5], we model the second via forbidden graphs. Figure 6 (left) shows the forbidden graph expressing that no resource can be annotated with both licenses `SA` and `ND`. An analogous graph will forbid the presence of both licenses in the same bundle. The constraint on the right requires that each resource be `PD`, where we use the generic type name `Resource` to refer to any of the types from the *Resource* domain.

The application of a rule may disrupt a constraint, typically by not creating the proper annotations. Hence, given a constraint μ, *constraint repair actions* are automatically inferred and applied, which modify the derivation relation so that the result satisfies μ.

Definition 4 (Constraint repair action) Let $\mu_1 \colon G_1 \rightarrow G_2$ be a rule and $\mu_2 \colon G_3 \rightarrow G_4$ an annotation constraint. We define the relation $\Longrightarrow_{\mu_1,\mu_2}$ with reference to Figure 7. For any two graphs G_5 and G_6 such that $(G_5, G_6) \in \Longrightarrow_{\mu_1}$ as witnessed by the leftmost square, and $G_6 \not\models \mu_2$ (i.e. there exists a morphism $\mu_5^i \colon G_3 \rightarrow G_6$ but no morphism $\mu_6 \colon G_4 \rightarrow G_6$ for which the triangle formed by μ_2, μ_5^i and μ_6 commutes), we have $(G_5, G_7) \in \Longrightarrow_{\mu_1,\mu_2}$, where G_7 is constructed as the colimit of all the diagrams constructed by taking the pushout of μ_5^i and μ_2 along G_3 for each

[5] This would be a constraint with annotations both in P and C, not considered here.

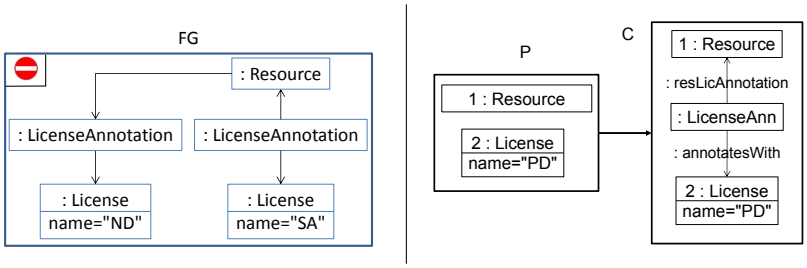


Figure 6: A graph forbidding the simultaneous presence of licenses `SA` and `ND` (left) and a positive constraint assessing that each resource is `PD`.

μ_5^i via the morphisms $\mu_7^i : G_4 \to G_7^i$ and $\mu_4^i : G_4 \to G_7^i$. The set of all the μ_4^i is called a *constraint repair action*.

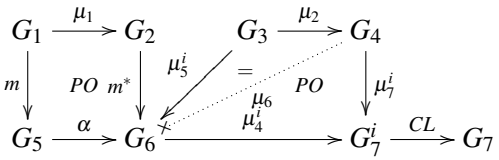


Figure 7: Direct Derivation Diagram for a rule with constraint repair action.

Figure 8 illustrates the construction needed to repair the violation of the constraint of Figure 6 (right) as a consequence of the application of rule `createCollection` in Figure 5.

Following Proposition 2 a repair action produces a graph compliant with μ_2.

Proposition 2 *Given G_7 and μ_2 as in Definition 4 we have $G_7 \models \mu_2$.*

Proof. We know that G_3 has a match in G_5, so that the constraint is violated by the presence of some element without proper annotation, which is added in each G_7^i as an effect of the pushout. Since we only deal with annotation constraints, each G_7^i does not present violations of μ_2 which were not in G_4, but actually presents one less violation. By taking the colimit, no violation appears in G_7. □

Proposition 3 allows the cumulative application of repair actions.

Proposition 3 *Let G_6 be as in Definition 4 and $M_2 = \{\mu_2^j : G_3^j \to G_4^j \mid G_6 \not\models \mu_2^j\}$. Then, let G_7' the colimit of all the graphs G_7 constructed as in Definition 4 for each $\mu_2^j \in M_2$. Then $G_7' \models \mu_2^j$ for each $\mu_2^j \in M_2$.*

Proof. Since the premise of each μ_2^j does not contain annotation elements, no G_7 constructed for one constraint can add new violations of any other constraint. The result then follows by the associativity of colimits. □

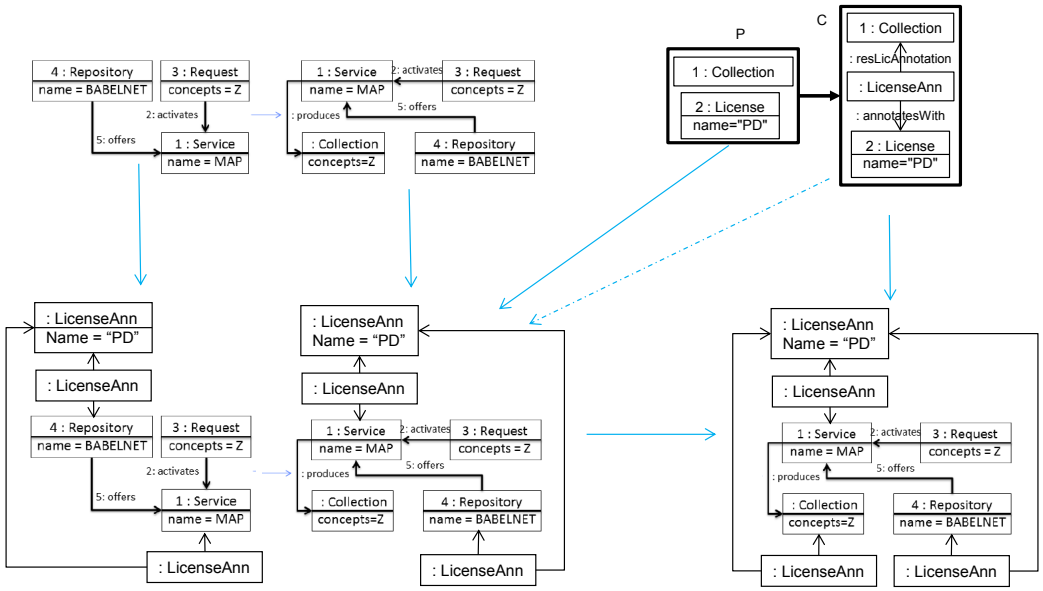


Figure 8: Applying the repair construction to ensure that each new collection is annotated with the PD license.

5.2 Contracts

Contracts define situations in which the application of a rule requires the production of some annotation, but only in situations in which elements in its left-hand side are associated with specific annotations.

Definition 5 (Contract) Given a rule $\mu_2 \colon G_3 \to G_4$, a *contract on* μ_2, γ, is given by a morphism $\mu_1 \colon G_1 \to G_2$ (G_1 and G_2 typed on TG) together with spans $G_1 \xleftarrow{\mu_3} G_5 \xrightarrow{\mu_4} G_3$, $G_4 \xleftarrow{\mu_5} G_6 \xrightarrow{\mu_6} G_2$ (formed by total injective morphisms) and a morphism $\mu_7 \colon G_5 \to G_6$, (G_5 and G_6 typed on TG_1), such that all the closed paths in the upper part of Figure 9 commute.

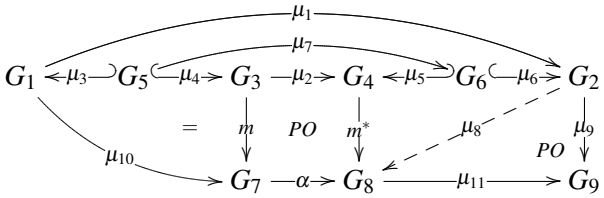


Figure 9: Direct Derivation Diagram for a rule with contract enforcement action.

As an example, the top contract in Figure 10 describes the overall policy for license assignment: each collection is generated with the same collection of licenses of the generating service. Here and in the following we collapse the left-hand sides of a rule and a contract together, and do the same for right-hand sides: non-primed numbers are used to represent elements in G_1, G_5,

G_6 and G_2 identified via the morphisms μ_3, μ_4, and μ_2, as well as the induced identifications through morphisms μ_1, μ_5, and μ_6; primed numbers are the residual elements identified in G_6, G_4 and G_2 via the morphisms μ_5 and μ_6; letters denote residual elements in G_1 and G_2 identified via the contract morphism μ_1; finally, anonymous elements are those only present in G_2, i.e. additional elements required by the contract, or additional elements introduced by the rule but not considered in the contract.

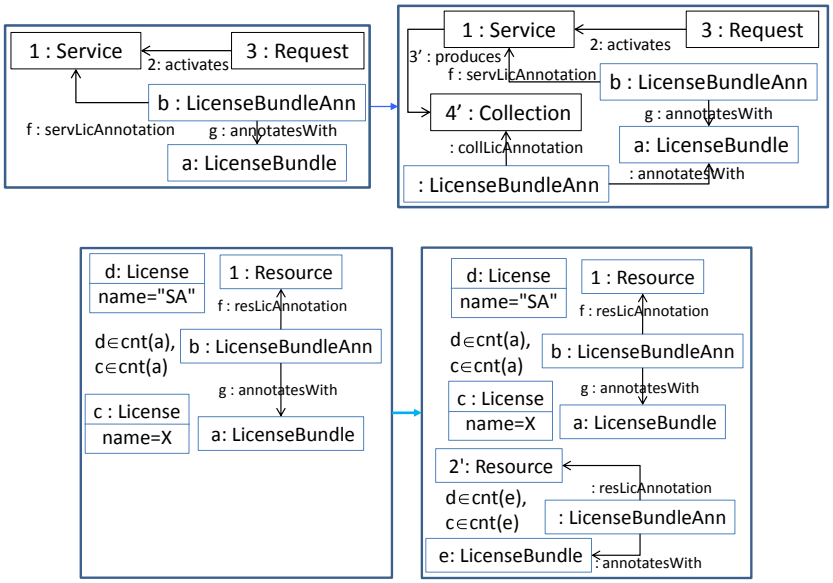


Figure 10: A contract stating that each collection comes with the license of the service which generated it (top) and a contract specifically modeling the SA licence (bottom).

We can now model the asymmetry in license extension described in Section 4 with reference to the contract in Figure 10 (bottom): if a copy of a resource[6] annotated with a bundle including the SA license is generated, the bundle annotating the new resource must preserve all the original licenses. This forbids the possibility of contracts which remove some license when SA is present, while it allows adding more licenses to the bundle, if not in contrast with others already present. In this case, we have used the e identifier in order to write the constraint on the presence of licenses in the new bundle. Analogous contracts can be devised for multiple annotations with single licenses, rather than with a bundle of licenses. It is important to note that the directionality inherent to this contract would not be expressible via constraints, which would either impose or forbid the presence of annotations in the bundles both before and after rule application.

The application of a domain rule creating new elements typically violates contracts requiring that they be annotated in certain ways. Hence, actions must be taken, as specified in Definition 6.

Definition 6 (Contract enforcement action) Let $\mu_2 : G_3 \to G_4$ and γ be as in Definition 5. A derivation $(G_7, G_8) \in \Longrightarrow_{\mu_1}$ *fulfills* the contract γ *iff* for each morphism $\mu_{10} : G_1 \to G_7$ such that the leftmost square in the diagram of Figure 9 commutes, and for each morphism $\mu_{5_i} : G_6 \to G_4$

[6] We use here the type Resource, and a generic type of annotation edge, to refer to any element from the $\mathscr{R}$ domain.

there exists at least one morphism $\mu_{8_i}\colon G_2 \to G_8$ so that the triangle $m^* \circ \mu_{5_i} \circ \mu_7\colon G_5 \to G_8$, $\mu_1 \circ \mu_3\colon G_5 \to G_2$ and $\mu_{8_i}\colon G_2 \to G_8$ commutes. If the above does not hold, the pair (G_7, G_8) is said to *breach* the contract μ_1. A breach can be repaired by a *contract enforcement action* μ_{11} for μ_1, μ_2 on G_7 by constructing $G_8 \overset{\mu_{11}}{\to} G_9 \overset{\mu_9}{\leftarrow} G_2$ as the pushout of the span $G_8 \overset{\alpha\circ\mu_{10}}{\leftarrow} G_1 \overset{\mu_1}{\to} G_2$, under the assumptions of Definition 5 on the commutative properties of the closed paths in the upper row of Figure 9. We denote the derivation thus obtained by $(G_7, G_9) \in \Longrightarrow_{\mu_2,\mu_1}$. Note that if no μ_{10} exists, then (G_7, G_8) also fulfills the contract.

Whenever multiple contracts apply to μ_2, the final graph to be obtained for its application is represented by the colimit of all the diagrams thus formed, noting that in all such diagrams the pushout formed by $G_7 \leftarrow G_3 \to G_4$ and $G_7 \to G_8 \leftarrow G_4$ remains the same. Notice also that by a straightforward application of the associativity and commutativity of colimits, the same result can be obtained by successively applying the above construction to individual contracts, regardless of the order. The proof of Proposition 4 is then straightforward.

Proposition 4 *For a rule μ_2 and a contract μ_1 on it, each derivation in $\Longrightarrow_{\mu_2,\mu_1}$ fulfills μ_1.*

Theorem 1 states that applying the enforcement action is equivalent to applying a rule resulting from the composition of μ_2 and μ_1.

Theorem 1 *Given a rule $\mu_2\colon G_3 \to G_4$ and a contract $\mu_1\colon G_1 \to G_2$ on μ_2, there exists a rule μ_2' such that for each pair $(G_7, G_9) \in \Longrightarrow_{\mu_2,\mu_1}$, we have $(G_7, G_9) \in \Longrightarrow_{\mu_2'}$.*

Sketch. Referring back to the diagram in Figure 9, the new left-hand side is constructed as the pushout of the span $G_1 \overset{\mu_3}{\leftarrow} G_5 \overset{\mu_4}{\to} G_3$, while the right-hand side as the pushout of the span $G_4 \overset{\mu_5}{\leftarrow} G_6 \overset{\mu_6}{\to} G_2$. Shown in Figure 11 are the induced matching morphism $G_3' \to G_7$, the new rule $\mu_2'\colon G_3' \to G_4'$, and the result G_9 of applying μ_2' to G_7' via the induced matching. Since G_4' already "contains" G_2, there is no need for a subsequent enforcement action. □

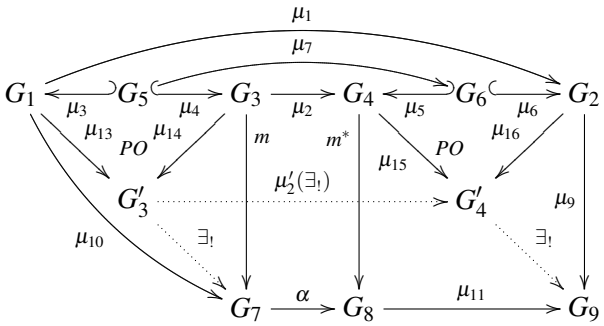


Figure 11: Direct Derivation Diagram for composition of rule and contract.

One word of caution is in order. Symbolic graphs define families of (B-)graphs, collapsing to normal (B-)graphs when all constraints are equalities. It is typically the case that, as in the case of the contract in Figure 10, the constraints on *cnt* only specify minimal content for a box. In this

case, a further transformation step may map G_9 to a canonical B-graph G_{10} in the family defined by G_9, where each box contains exactly its minimal required content.

We can also define *negative contracts*, indicated as $\mu_1 : G_1 \overset{\neg}{\to} G_2$ such that $(G_5, G_6) \in \Longrightarrow_{\mu_1,\mu_2}$ only if $G_6 \not\models \mu_2$. This prevents the application, to the same μ_1, of other contracts of the form $\mu_1' : G_1' \to G_2'$ with $G_1' \hookrightarrow G_1$ and $G_2 \hookrightarrow G_2'$.

The constructions above can be adapted to general, i.e. not only increasing, DPO rules $G_3 \leftarrow G_{10} \rightarrow G_4$ by considering contracts in the form of spans $G_1 \leftarrow G_{11} \rightarrow G_2$ and $G_5 \leftarrow G_{12} \rightarrow G_6$, together with an additional span $G_{10} \leftarrow G_{12} \rightarrow G_{11}$, such that all closed paths in the upper rows of Figures 9 and 11 commute. In all these cases, we consider morphisms which are injective in the restrictions to $\mathscr{D}_1$ of the involved graphs, as well as in the immersion of $\mathscr{D}_1$ into $\mathscr{D}$, while they can be non-injective in the restriction to $\mathscr{D}_2$.

6 Extending contracts to contract schemes

Contracts can be used as *contract schemes* to automatically derive specialised contracts for rules which *extend* the original *kernel* rule to which the scheme applied (see Definition 7). In the rest of the section we will use the term contract scheme for the contract associated with the kernel rule and derived contract for the one associated with the extended rule.

Definition 7 (Derived contract) With reference to Figure 12, let $\mu_2 \colon G_3 \to G_4$ be a kernel rule and let γ a contract scheme on μ_2. Let $\mu_2' \colon G_3' \to G_4'$ be a rule with two collections of total injective morphisms $M_3 = \{\mu_j : G_3 \to G_3'\}$ and $M_4 = \{\mu_i : G_4 \to G_4'\}$. Then $\gamma(\mu_2')$ is the *contract* on μ_2' *derived* from γ defined as follows:
1. define $\widehat{G_5} \to G_5$ as the equalizer of the morphisms $\mu_j' \circ \mu_4 : G_5 \to G_3 \to G_3'$ ($\widehat{G_5}$ represents the "context" not affected by the addition of the different μ_j' to G_3);
2. define $G_5^j = \mu_j'(\mu_4(G_3))$ for each $\mu_j' \in M_3$;
3. let G_5' be the colimit object of the diagram $\widehat{G_5} \to G_5 \to G_5^j$ (the "union" of the different copies of $\mu_j'(\mu_4(G_3))$ in G_3' identifying only the "common context" $\widehat{G_5}$);
4. let $\mu_4' : G_5' \to G_3'$ be the unique morphism induced by the universal property of G_5' with respect to the obvious inclusions $G_5^j \to G_3'$;
5. define $G_1 \to G_1^j \leftarrow G_5^j$ as the pushout of $G_1 \leftarrow G_5 \to G_5^j$ and G_1' as the colimit of the G_1^j with respect to $\widehat{G_5}$. By the universal property of the construction of G_5', there is a unique $\mu_3' : G_5' \to G_1'$ (a mono since $G_5 \to G_1$ is);
6. the same procedure on the right hand side of the diagram produces $G_4' \leftarrow G_6' \to G_2'$ and the morphism $\widehat{G_5} \to \widehat{G_6}$ by the universal property of equalizers;
7. finally μ_1' and μ_2' are the unique morphisms induced by the universal property of the construction of G_1' and G_5' (and the use of μ_1 and μ_2 for the required commutativity).

A contract scheme γ produces immediately a contract for the rule μ_2 for each isomorphism of G_3 into itself, with the property illustrated by Theorem 2.

Theorem 2 *Let μ_2 be a kernel rule, μ_2' an extended rule, γ a contract on μ_2 and γ' the contract derived from γ on μ_2' as in Definition 7, with M_3 the set of morphisms from G_3 into G_3'. Let G_7*

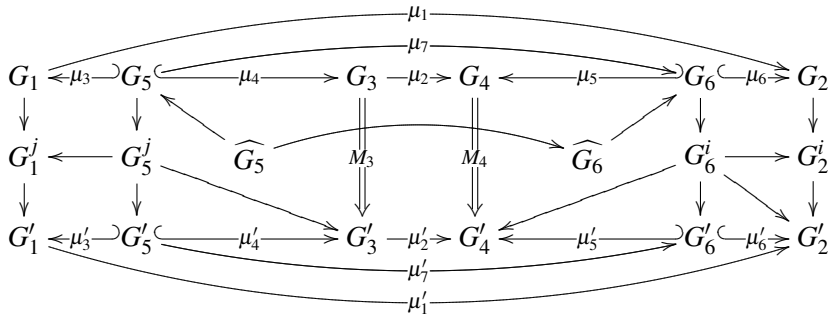


Figure 12: Construction of a contract from a contract scheme.

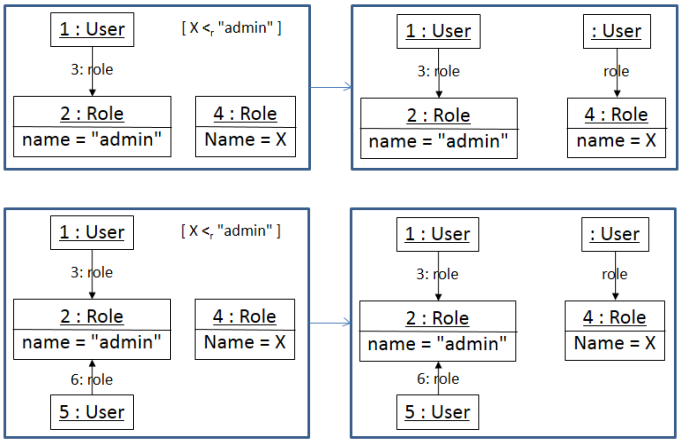


Figure 13: A kernel rule and an extended rule for creating users.

be a graph typed on TG s.t. G_3' and G_1' have matches m and μ_{10}' in G_7 and let G_9 be the graph obtained by the contract enforcement action for γ'. Then there exists a set, M^k, of matches of G_3 into G_7 with $|M_3^k| = |M_3|$ and a bijection $b : M_3^k \to M_3$ s.t. $m_i^k = m \circ b(m_i^k)$ for $m_i^k \in M^k$. For each match $m_i^k \in M^k$ let G_8^i be the result of the application of μ_2 via m_i^k and G_9^i the graph obtained by the contract enforcement action for γ. Then G_9 is isomorphic to the colimit of the G_9^i.

Sketch. The proof derives from the universal properties of the equalisers, pushouts and colimits used in the construction, for the existence and uniqueness of the needed morphisms, the commutativity of colimits, and the independence in the applications of μ_2. □

We illustrate the notion of schemes with an example taken from the organisational domain, as licenses do not seem to require them. Figure 13 presents a kernel rule `createUser`, allowing an administrator to create users with inferior, with respect to a partial order $<_r$ roles, and an extended rule `twoAdmins` which requires two administrators to create a user.

Figure 14 presents a contract scheme on `createUser`: if the role of administrator is restricted to specific periods, then the role of the created user is restricted to the same period. Figure 15 presents the derived contract for `twoAdmins`. The user role is valid for both the pe-

riods of validity of the administrator roles. Here we have used annotations on edges, as allowed by the metamodel for annotations. By using boxes, we could also achieve the same effect by annotating a box containing only the edge to be annotated.

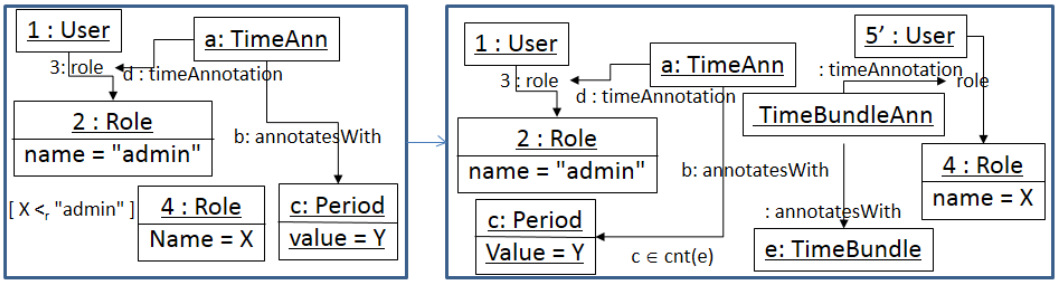


Figure 14: A contract scheme on rule `createUser` for time annotations on administrators.

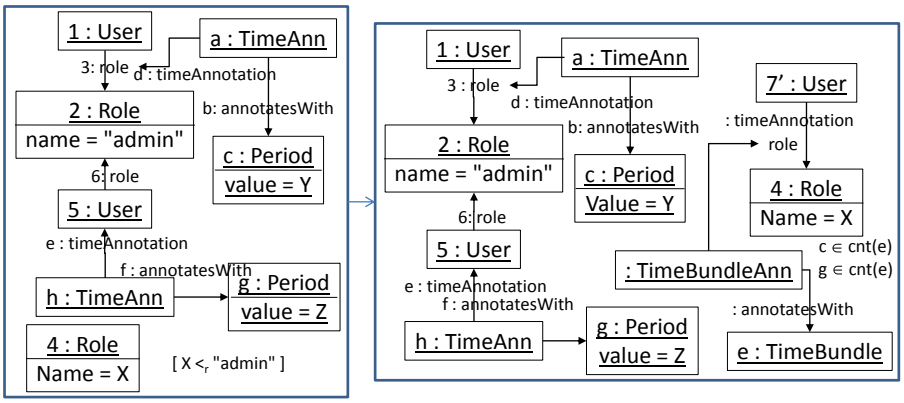


Figure 15: A contract on rule `twoAdmins` derived from the scheme of Figure 14.

7 Related Work

A number of approaches have considered the management of inconsistencies in the field of graph transformations. In particular, mechanisms for ensuring that invariants are maintained throughout transformations have lead to the identification of mechanisms for the generation of pre- or post-application conditions to be associated with rules, or for manipulation of the left-hand or right-hand side of a rule. This topic was started in [HHT96] and extensively explored in [HP09]. In [BP13a, BP13b], we have shown how the separation of the application and contextual domains allows some simplified constructions for such a generation.

Another line of research refers to inconsistencies of a model with respect to some (un)desired property established at the metamodel level. For example, conformance to a pattern can be imposed by completing the missing required parts of the pattern by a co-limit construction [BGL10], required ordering for refactorings can be established by analysis of their conflicts [MTR07],

explicit transformations can be associated with guidelines to repair violations [ALSS08]. We consider here only inconsistencies involving annotations, again leveraging domain separation.

Contracts were introduced in the form of pairs of graphs indicating pre- and post-conditions of operations [HHL05], defining rule schemes which are typically instantiated by setting some parameters [ELSH06]. One can then consider satisfaction of conditions for service composition [NHOH10] or generate tests against these schemes to check correctness of model evolutions [RKH13]. We allow the definition of multiple contracts for a single rule and enforce correctness on a local basis, as opposed to corrections required by violations of global constraints.

Annotations are a relatively new concept in the graph transformation field, although they are largely used in modeling, human computer interaction, information retrieval and the Semantic Web. 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 our approach the structure of annotations is embedded in the construction of a typing system relating different domains, and annotations become first class elements, associated with domain elements through specific types of edges, so that, rather than using morphisms for single annotation, we model annotation and de-annotation processes via functors, thus allowing a flexible usage of annotations [BP12b].

This kind of flexibility may be related to the notion of partial attribution, where it is not required that all nodes are equipped with attribute edges connecting them with labels on which computations can be performed [FKTV99, FKB00]. However, to the best of our knowledge, the notion of partial attribution was developed in the framework of attributed graphs, although its lifting to typed attributed graphs would be quite straightforward. Moreover, our approach supports the annotation of annotation nodes, while partial attribution relies on attribution operations, for which a specific notion of annotation should be defined.

A formalisation of licenses has been advocated for services, whose composition has to take into account that different services may run under different licenses, as part of service level agreement, focusing mostly on service behaviour [GD11]. As for data, work on the definition of ontologies for the management of digital rights have been conducted by Garcia *et al.* [GGD07] and Rodríguez-Doncel et al. [RVG14]. Graph transformations have been employed in the closely related field of access control, using similar techniques to those proposed here (see e.g. [KP06]).

8 Conclusions and Future Work

We have extended the framework of annotation from [BP13a, BP13b] by considering the problem of orphan annotations, obtained when annotated elements are deleted without deleting the corresponding annotation edge, and by introducing contracts, relating pre- and post-conditions on the usage of annotated elements. Here, annotations model the role of licenses in the open data environment defining the approved usages for resources. In this context, license bundles are seen as a technique to manage sets of resources homogeneous with respect to the applicable licenses.

With respect to the formal framework, future research needs to address constraints with annotations also in the premise, and to study dependencies and conflicts [RET12] within sets of contracts and within compositions of rules and contracts. More work is also planned on contract schemes and their use in allowing the customised generation of contracts for different rules.

For the application domain of licenses, tools from graph transformation theory could be used to define a generic framework for managing them possibly from different licensing schemes, by which declarative specifications of resource usages could be checked for verification of conformance to licenses. Analysis tools could also verify the internal consistency of license bundles.

Bibliography

[ABK+07] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web*. LNCS 4825, pp. 722–735. Springer, 2007.

[ALSS08] C. Amelunxen, E. Legros, A. Schürr, I. Stürmer. Checking and Enforcement of Modeling Guidelines with Graph Transformations. In *Proc. AGTIVE 2007*. LNCS 5088, pp. 313–328. Springer, 2008.

[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.

[BP12a] F. Bond, K. Paik. A survey of wordnets and their licenses. In *Proc. GWC 2012*. Pp. 64–71. 2012.

[BP12b] P. Bottoni, F. Parisi Presicce. Modeling context with graph annotations. *ECEASST* 47, 2012.

[BP13a] P. Bottoni, F. Parisi Presicce. Annotation processes for flexible management of contextual information. *JVLC* 24(6):421–440, 2013.

[BP13b] P. Bottoni, F. Parisi-Presicce. Annotations on Complex Patterns. *ECEASST* 58, 2013.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.

[ELSH06] G. Engels, M. Lohmann, S. Sauer, R. Heckel. Model-Driven Monitoring: An Application of Graph Transformation for Design by Contract. In *Proc. ICGT 2006*. LNCS 4178, pp. 336–350. Springer, 2006.

[Fel98] C. Fellbaum (ed.). *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[FKB00] I. Fischer, M. Koch, M. Berthold. Attributed Graph Transformation with Partial Attribution. In *Proc. GraGra 2000*. Pp. 171–178. Berlin: Technical University, 2000.

[FKTV99] I. Fischer, M. Koch, G. Taentzer, V. Volle. Distributed Graph Transformation with Application to Visual Design of Distributed Systems. In *Handbook of Graph Grammars and Computing by Graph Transformation*. Pp. 269–340. World Scientific Publishing Co., Inc., 1999.

[GD11] G. Gangadharan, V. DAndrea. Service licensing: conceptualization, formalization, and expression. *Serv. Orient. Comp. and Appl.* 5(1):37–59, 2011.

[GGD07] R. Garca, R. Gil, J. Delgado. A web ontologies framework for digital rights management. *Artificial Intelligence and Law* 15(2):137–154, 2007.

[HHL05] J. H. Hausmann, R. Heckel, M. Lohmann. Model-Based Development of Web Services Descriptions Enabling a Precise Matching Concept. *Int. J. Web Service Res.* 2(2):67–84, 2005.

[HHT96] A. Habel, R. Heckel, G. Taentzer. Graph Grammars with Negative Application Conditions. *Fundam. Inform.* 26(3/4):287–313, 1996.

[HP09] A. Habel, K.-H. Pennemann. Correctness of high-level transformation systems relative to nested conditions. *MSCS* 19(2):245–296, 2009.

[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.

[MTR07] T. Mens, G. Taentzer, O. Runge. Analysing refactoring dependencies using graph transformation. *Software and System Modeling* 6(3):269–285, 2007.

[NHOH10] M. Naeem, R. Heckel, F. Orejas, F. Hermann. Incremental Service Composition Based on Partial Matching of Visual Contracts. In *Proc. FASE 2010*. LNCS 6013, pp. 123–138. Springer, 2010.

[NP12] R. Navigli, S. P. Ponzetto. BabelNet: The Automatic Construction, Evaluation and Application of a Wide-Coverage Multilingual Semantic Network. *Artificial Intelligence* 193:217–250, 2012.

[OL10a] F. Orejas, L. Lambers. Delaying Constraint Solving in Symbolic Graph Transformation. In *Proc. ICGT 2010*. Pp. 43–58. Springer, 2010.

[OL10b] F. Orejas, L. Lambers. Symbolic Attributed Graphs for Attributed Graph Transformation. *ECEASST* 30, 2010.

[RET12] O. Runge, C. Ermel, G. Taentzer. AGG 2.0 - New Features for Specifying and Analyzing Algebraic Graph Transformations. In *Proc. AGTIVE 2011*. LNCS 7233, pp. 81–88. Springer, 2012.

[RKH13] O. Runge, T. A. Khan, R. Heckel. Test Case Generation Using Visual Contracts. *ECEASST* 58, 2013.

[RVG14] V. Rodríguez-Doncel, S. Villata, A. Gómez-Pérez. A Dataset of RDF Licenses. In *Proc. (JURIX) 2014*. Pp. 187–189. IOS Press, 2014.