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

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

Università di Roma "Sapienza" (Italy)
(bottoni,navigli,parisi)@di.uniroma1.it

Abstract. Organisational policies express constraints on generation and processing of resources. However, application domains rely on transformation processes, which are in principle orthogonal to policy specifications and domain rules and policies may evolve in a non-synchronised way. In previous papers, we have proposed annotations as a flexible way to model aspects of some policy, and showed how they could be used to impose *constraints* on domain configurations, how to derive application *conditions* on transformations, and how to annotate complex patterns. We extend the approach by: allowing domain model elements to be annotated with collections of elements, which can be collectively applied to individual resources or collections thereof; proposing an original construction to solve the problem of annotations remaining *orphan*, when annotated resources are consumed; introducing a notion of *contract*, by which a policy imposes additional pre-conditions and post-conditions on rules for deriving new resources. 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.

1 Introduction

Organisational policies express constraints on generation and processing of resources which are accepted by agents subject to, or anyway acknowledging, the authority of such organisations. However, agents 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 [2] 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, it is usually the case that the usage of certain resources in the construction of the service prevents

the possibility of imposing such restrictions, or forces specific forms of access compatible with those for the used resource.

In [5, 6] 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 could 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 policies of access change, or different policies have to be simultaneously applied. Annotations are indeed a way of flexibly associating elements of different domains, and a number of techniques have been developed to express the constraints which are 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.

In this paper 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*, which remain dangling 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. Again, the use of contracts 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 discussing related work in Section 2 and recalling fundamental notions in Section 3, we provide a construction for avoiding orphan annotations in Section 4 and introduce a motivational case study concerning linguistic resources annotated with licenses in Section 5. The model for rewriting under annotation constraints and contracts, illustrated via the case study, is presented in Section 6. Section 7 concludes the paper.

2 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 [12] and extensively explored in [13]. In [5, 6], we have shown how the separation of the application and contextual domains allows some simplified constructions for such a generation.

Another line of research involves the identification of inconsistencies of a model with respect to some desired (or undesired) 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 [4], required ordering for refactorings can be established by analysis of their conflicts [16], explicit transformations can be associated with guidelines to repair violations [1]. In this paper we consider 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 [14]. In this way they define rule schemes which are typically instantiated by setting some parameters [8]. Based on this, one can consider satisfaction of conditions for service composition [17] or generate tests against these schemes to check correctness of model evolutions [22]. Our usage of contracts allows the definition of multiple contracts for a single rule and provides a basis for enforcing correctness on a local basis, as opposed to corrections required by violations of global constraints.

The need for a formalisation of licenses has been addressed for services, where service 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 [10]. As for data, work on the definition of ontologies for the management of digital rights have been conducted by Garcia *et al.* [11] and Rodríguez-Doncel et al. [20]. Graph transformations have been employed in the closely related field of access control, where techniques similar to the ones employed here were used (see e.g. [15]).

3 Preliminaries

Graphs and morphisms. We introduce classical notions from graph transformation theory (see [7]).

Definition 1. *A* graph *is a tuple* $G = (V_G, E_G, s, t)$*, 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, i.e. graphs are considered to be* directed*.*

Definition 2. *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*, $\mu\colon G_1 \to G^T$ is a *typing morphism* if μ is total. Typed graphs are 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))

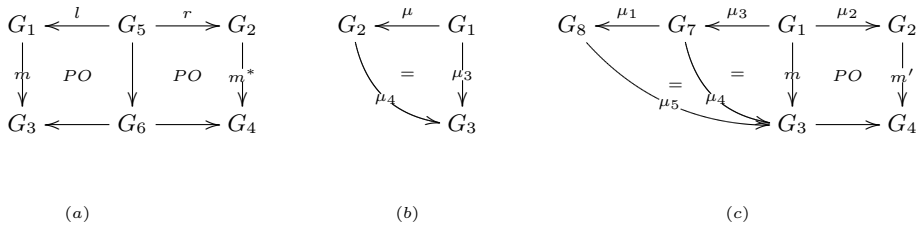

Fig. 1. (a) DPO Derivation, (b) satisfaction of constraint, (c) SPO derivation with AC.

type-preserving morphisms are used to define rules. Regardless of the form, 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 exist 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 $(G3, G_4)$ 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 exist 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 constraints and application conditions can refer to prescribed 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. In all these cases,we follow the approach of symbolic attributed graphs [19], using constraint satisfaction to evaluate conditions and attributes.

Annotations. Annotations of nodes of a domain $\mathcal{D}_1$ with nodes of a domain $\mathcal{D}_2$ are defined via nodes of types derived from `AnnotationNodeType`. We call $\mathcal{A}$ the domain of such annotation nodes. Each annotation node $a \in \mathcal{A}$ participates in exactly one instance of the pattern $\pi_a = x \xleftarrow{e_1} a \xrightarrow{e_2} y$, where $x \in \mathcal{D}_1$, $y \in \mathcal{D}_2$, 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 $\mathcal{D}_1$ (TG_1) and $\mathcal{D}_2$ (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

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

preserved by the domain rules. Constraints of the first type are derived from π_a and are of the form $\mu : \boxed{A} \rightarrow \boxed{X \xleftarrow{e_1} A \xrightarrow{e_2} Y}$, for X some node 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. Well-formedness results from conformance with the disjunction of all such constraints. Given a type graph TG we call $\mathcal{L}_{\pi_a}(TG)$ the language of all the graphs in TG which are well-formed with respect to the individual domains and the annotation pattern. 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 component of TG, G_2 is typed on TG with well-formed annotations, and its projection on TG_1 is isomorphic to G_1. We call these *annotation constraints*. In Section 6 we will also introduce *contracts* based on morphisms involving two graphs typed on TG. All the rules in the paper are typed on TG_1, and all the constraints are annotation constraints.

4 Annotations and collections

Definition 3 from [6] extend graphs and morphisms to include the notion of *box*.

Definition 3. *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 [6] 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 [6] 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 [6], removing an annotated element from a model $G' \in \mathcal{L}_{\pi_a}(TG)$ would create dangling annotation edges, forbidding rule application in the DPO form. Nor can deletion in the SPO form be employed. Even if in this case the annotation edges would be removed, we would be left with *orphan annotations*, i.e. annotation nodes to which no annotation edge is attached, resulting in a graph not in $\mathcal{L}_{\pi_a}(TG)$. 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'

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

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 $\mathcal{L}_{\pi_a}(TG)$, i.e. with well-formed annotations, which is maximal with respect to 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' is given by the pushout of H and D' along D.

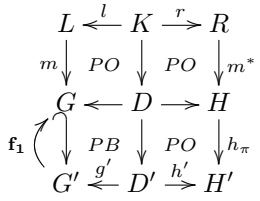

Fig. 2. Avoiding orphan annotations.

Figure 3 illustrates Construction 1 with a model example where teams in an organisation temporarily gather members for specific tasks [6]. 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.

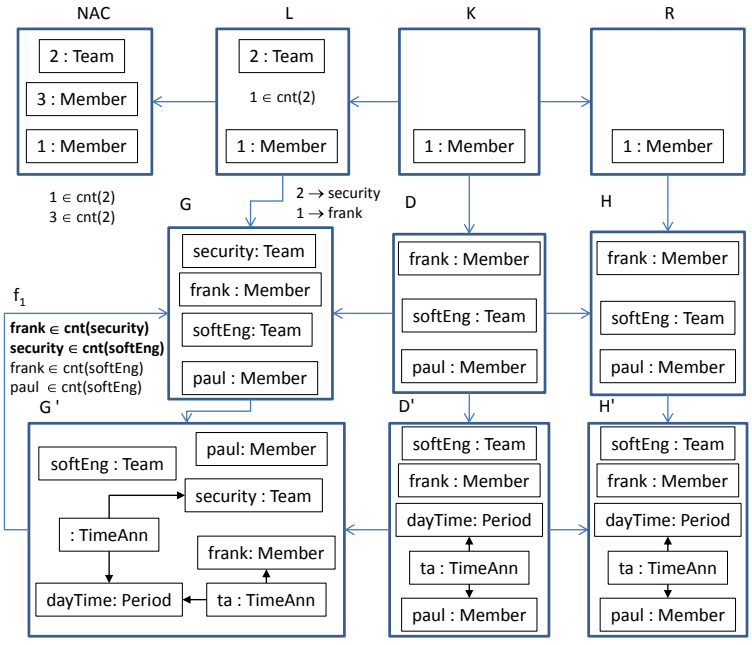

Fig. 3. An example of Construction 1 for the removal of an annotated team.

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

Proof. We first observe that $Ann(H') = Ann(D')$, so that $Ann(H') \subseteq Ann(G')$, since H, which is typed on TG_1, does not present any new annotation. Since D' is well-formed, none of its annotation nodes is an orphan.

5 Case study: 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 connected to a resource usually depend on some policy associated with the repository from which it is extracted. Moreover, such repositories often publish resources under a number of licenses, which must be all simultaneously respected, and which are normally transferred to the extracted resources. As this imposes 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 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). Each element is considered to be of `PublicDomain` (`PD`).

BabelNet[3] [18] is a multilingual semantic network containing millions of concepts (e.g. the **apple fruit** concept) and named entities (e.g. the **Apple Inc.** entity), interlinked via semantic relations. A single node in the network is called a Babel *synset* and contains a set of synonyms 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*, ..., *manzana_ES*}. A word occurring 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 (which is in turn considered a kind of resource unique to BabelNet) and integration of several, publicly available, knowledge repositories. However, each repository provides resources (e.g. synsets and senses) under different licenses. For example, WordNet [9], Wikidata[4] and parts of the OMWN project [3] are released under a permissive license, which allows any use, either commercial or non-commercial, of the data;

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

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 `CC-BY-NC-SA` and so is the BabelNet mapping between all these resources. 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 no-one can modify a `SA` license. As a result, 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, the 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 is subject to obtaining a commercial license from the BabelNet's authors.

5.1 A model of linguistic services and licenses

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, $\mathcal{R}$, of *Resource*s. Similarly, licenses are modeled as nodes of type `License`, from the domain of *Licenses*, $\mathcal{L}$, 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*, $\mathcal{K}$, with `name` ranging over the available languages. We consider $\mathcal{R}$ as the application domain, and $\mathcal{L}$ and $\mathcal{K}$ as

contextual domains, typing the annotation edges accordingly. Thus, for `xxx` a type in $\mathcal{R}$, edges of types `xxxLicAnnotation` and `xxxLangAnnotation` allow its annotation with elements of $\mathcal{L}$ and $\mathcal{K}$, 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 $\mathcal{R}$, $\mathcal{L}$ or $\mathcal{K}$, or is an `AnnotationTypeNode`. In the domain $\mathcal{R}$, a `Sense` is characterised by 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. The attribute `concept` for `Synset` indicates the concept represented by that collection of senses. The type `LicenseBundleAnn`, of the *Node* sort and derived from `AnnotationTypeNode`, 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.

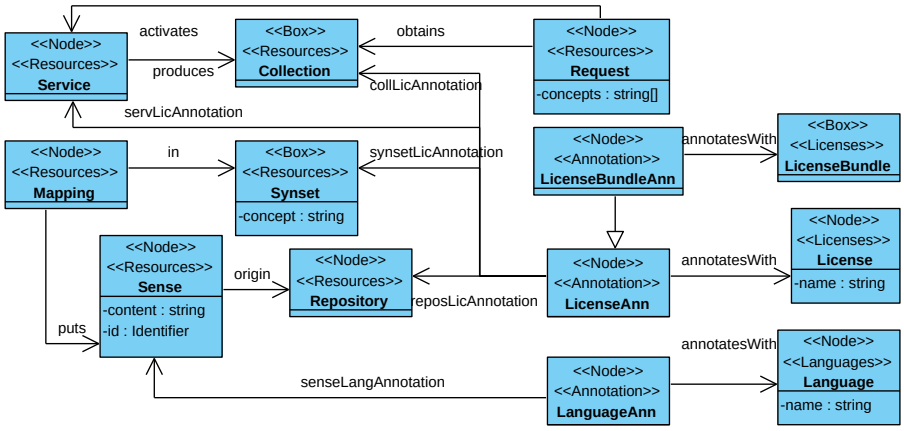

Fig. 4. The type graph for modeling the running example

We first model the basic working of BabelNet services via increasing rules in the *Resources* domain. Rule `createCollection` in Figure 5 (top) creates an initially empty collection for the synsets 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 in 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) is used to populate synsets with senses representing the concept, according to a mapping, with a NAC to avoid including a sense twice.

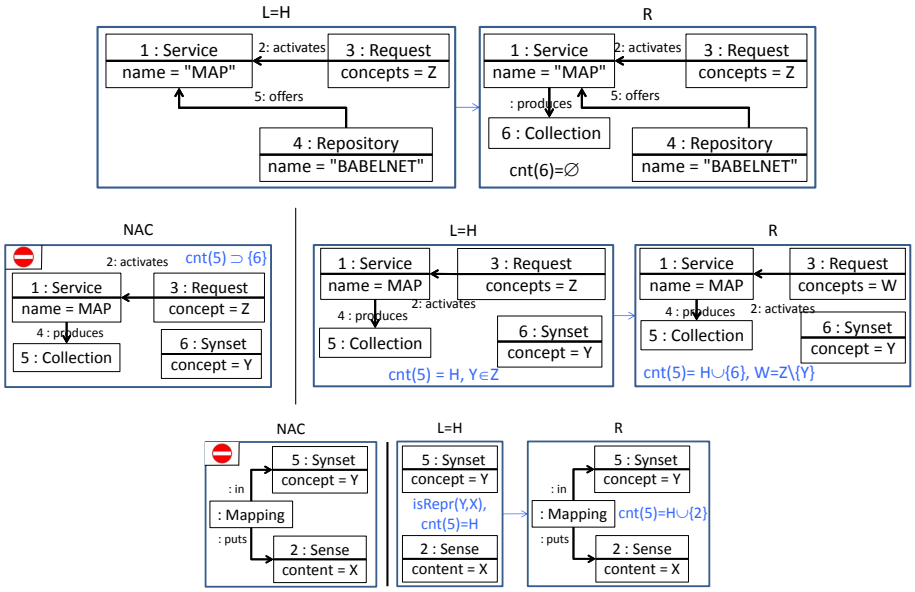

Fig. 5. Rule `createCollection` prepares a container for serving a request (top). Rule `createSynset` sets up a synset for a requested concept (middle). Rule `createMapping` relates a sense to a synset (bottom). Vertical lines separate NACs from rule morphisms.

In 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 6.2. 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 [5].

6 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, as introduced in Section 3, to impose well-formedness conditions for a domain. We assume that the host graph to which a rule is applied is well-formed, and we identify mechanisms to ensure that the transformation process produces another well-formed graph.

In [6], 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 the addition of 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, we present them as simple morphisms.

6.1 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.

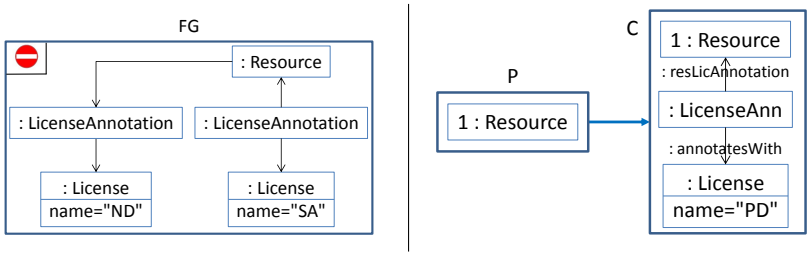

Fig. 6. A graph forbidding the simultaneous presence of licenses `SA` and `ND` (left) and a positive constraint assessing that each resource is `PD`.

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.

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

Definition 4 (Constraint repair action). *Let $\mu_1 \colon G_1 \to G_2$ be a rule and $\mu_2 \colon G_3 \to G_4$ 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 $\mu_5^i \colon G_3 \to G_6$ but no morphism $\mu_6 \colon G_4 \to G_6$ 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 μ_5^i via the morphisms $\mu_7^i \colon G_4 \to G_7^i$ and $\mu_4^i \colon G_4 \to G_7^i$. The set of all the μ_4^i is called a* constraint repair action.

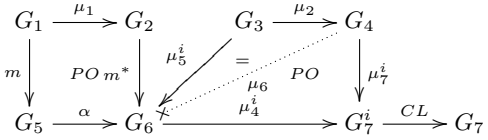

Fig. 7. Direct Derivation Diagram for a rule with constraint repair action.

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 $M_2 = \{\mu_2^j \colon G_3^j \to G_4^j \mid G_6 \not\models \mu_2^j\}$. 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.

6.2 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 $G_1 \overset{\mu_3}{\leftarrow} G_5 \overset{\mu_4}{\rightarrow} G_3$, $G_4 \overset{\mu_5}{\leftarrow} G_6 \overset{\mu_6}{\rightarrow} G_2$ (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 8 commute.*

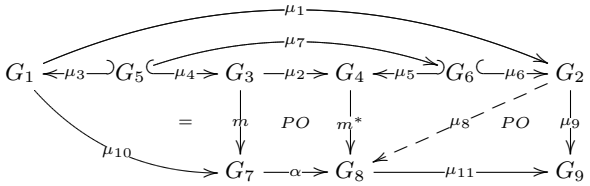

Fig. 8. Direct Derivation Diagram for a rule with contract enforcement action.

As an example, the top contract in Figure 9 describes the overall policy for license assignment: each collection is generated with the same collection of licenses of the generating service. Here, numbers are used to identify the nodes common to the rule and the contract, and letters for identifying elements related by the contract morphism.

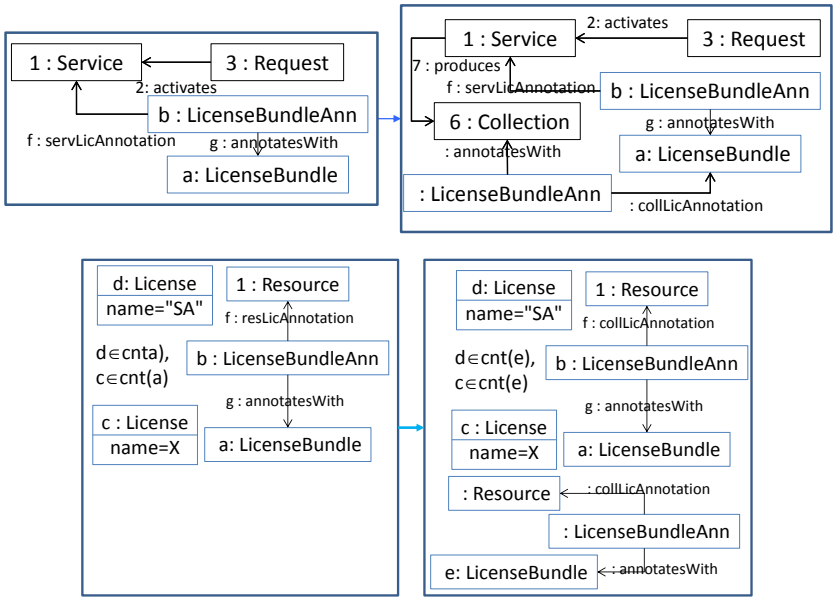

Fig. 9. A contract stating that each collection comes with the license of the service which generated it (top) and a contract specifically modeling the `SA` licence (bottom).

We can now model the asymmetry in license extension described in Section 5 with reference to the contract in Figure 9 (bottom): if a copy of a resource 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. 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* $\mu_2 \colon G_3 \to G_4$ *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* $\mu_{10} \colon G_1 \to G_7$ *such that the leftmost square in the diagram of Figure 8 commutes, and for each morphism* $\mu_{5_i} \colon G_6 \to G_4$ *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* $G_8 \overset{m^* \circ \mu_{5_i}}{\leftarrow} G_6 \overset{\mu_{11}}{\to} G_2$. *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* $(G_7, G_9) \in \Longrightarrow_{\mu_2'}$.

Proof (Sketch.). Referring back to the diagram in Figure 8, 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 colimit of the collection of spans $G_4 \overset{\mu_{5i}}{\leftarrow} G_6 \overset{\mu_6}{\to} G_2$. Shown in Figure 10 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.

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 $G_2 \hookrightarrow G_2'$.

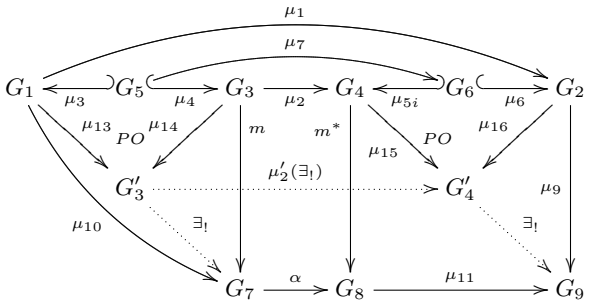

Fig. 10. Direct Derivation Diagram for composition of rule and contract.

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 identifying the spans $G_5 \leftarrow G_{12} \rightarrow G_6$ and $G_{10} \leftarrow G_{13} \rightarrow G_{11}$.

In all these cases, we consider morphisms which are injective in the restrictions to $\mathcal{D}_1$ of the involved graphs, as well as in the immersion of $\mathcal{D}_1$ into $\mathcal{D}$, while they can be non-injective in the restriction to $\mathcal{D}_2$.

7 Conclusions and Future Work

We have extended the theory of annotation from [5, 6] by considering the problem of orphan annotations, left when annotated elements are deleted, and introducing contracts, relating pre- and post-conditions on the usage of annotated elements. We have used annotations to model the role of licenses in the open data environment defining the approved usages for resources. In this context, license bundles have been seen as a technique to manage sets of resources homogeneous with respect to the applicable licenses.

With respect to the theory, future research will involve considering constraints with annotations also in the premise, and to study dependencies and conflicts [21] within sets of contracts and within compositions of rules and contracts. Also, we want to generalise the notion of contract to that of contract schemes, allowing the customised generation of contracts for different rules.

As concerns the application domain considered in the paper, we plan to extend this work towards a generic framework for managing licenses, taking into consideration also other licensing schemes, by which declarative specifications of resource usages could be checked for verification of conformance to licenses. Moreover, the mentioned analysis tools in the field of graph transformations can be applied to verify the internal consistency of license bundles.

References

1. C. Amelunxen, E. Legros, A. Schürr, and I. Stürmer. Checking and enforcement of modeling guidelines with graph transformations. In *Proc. AGTIVE 2007*, volume 5088 of *LNCS*, pages 313–328. Springer, 2008.

2. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A nucleus for a web of open data. In *The Semantic Web*, volume 4825 of *LNCS*, pages 722–735. Springer, 2007.
3. F. Bond and K. Paik. A survey of wordnets and their licenses. In *Proc. GWC 2012*, pages 64–71, 2012.
4. P. Bottoni, E. Guerra, and 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.
5. P. Bottoni and F. Parisi Presicce. Annotation processes for flexible management of contextual information. *JVLC*, 24(6):421–440, 2013.
6. P. Bottoni and F. Parisi-Presicce. Annotations on complex patterns. *ECEASST*, 58, 2013.
7. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.
8. G. Engels, M. Lohmann, S. Sauer, and R. Heckel. Model-driven monitoring: An application of graph transformation for design by contract. In *Proc. ICGT 2006*, volume 4178 of *LNCS*, pages 336–350. Springer, 2006.
9. C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.
10. G. Gangadharan and V. DAndrea. Service licensing: conceptualization, formalization, and expression. *Serv. Orient. Comp. and Appl.*, 5(1):37–59, 2011.
11. R. Garca, R. Gil, and J. Delgado. A web ontologies framework for digital rights management. *Artificial Intelligence and Law*, 15(2):137–154, 2007.
12. A. Habel, R. Heckel, and G. Taentzer. Graph grammars with negative application conditions. *Fundam. Inform.*, 26(3/4):287–313, 1996.
13. A. Habel and K.-H. Pennemann. Correctness of high-level transformation systems relative to nested conditions. *MSCS*, 19(2):245–296, 2009.
14. J. H. Hausmann, R. Heckel, and M. Lohmann. Model-based development of web services descriptions enabling a precise matching concept. *Int. J. Web Service Res.*, 2(2):67–84, 2005.
15. M. Koch and F. Parisi Presicce. UML specification of access control policies and their formal verification. *Software and System Modeling*, 5(4):429–447, 2006.
16. T. Mens, G. Taentzer, and O. Runge. Analysing refactoring dependencies using graph transformation. *Software and System Modeling*, 6(3):269–285, 2007.
17. M. Naeem, R. Heckel, F. Orejas, and F. Hermann. Incremental service composition based on partial matching of visual contracts. In *Proc. FASE 2010*, volume 6013 of *LNCS*, pages 123–138. Springer, 2010.
18. R. Navigli and S. P. Ponzetto. BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250, 2012.
19. F. Orejas and L. Lambers. Symbolic attributed graphs for attributed graph transformation. *ECEASST*, 30, 2010.
20. V. Rodríguez-Doncel, S. Villata, and A. Gómez-Pérez. A dataset of RDF licenses. In *Proc. (JURIX) 2014*, pages 187–189. IOS Press, 2014.
21. O. Runge, C. Ermel, and G. Taentzer. AGG 2.0 - new features for specifying and analyzing algebraic graph transformations. In *Proc. AGTIVE 2011*, volume 7233 of *LNCS*, pages 81–88. Springer, 2012.
22. O. Runge, T. A. Khan, and R. Heckel. Test case generation using visual contracts. *ECEASST*, 58, 2013.