Propagation of Policies in Rich Data Flows

Enrico Daga[†], Mathieu d'Aquin[†], Aldo Gangemi[‡] and Enrico Motta[†]
[†] Knowledge Media Institute, The Open University, United Kingdom
{enrico.daga,mathieu.daquin,enrico.motta}@open.ac.uk
[‡] Université Paris13, France and Institute of Cognitive Sciences and Technologies - CNR, Italy
aldo.gangemi@{univ-paris13.fr,cnr.it}

ABSTRACT

Governing the life cycle of data on the web is a challenging issue for organisations and users. Data is distributed under certain policies that determine what actions are allowed and in which circumstances. Assessing what policies propagate to the output of a process is one crucial problem. Having a description of policies and data flow steps implies a huge number of propagation rules to be specified and computed (number of policies times number of actions). In this paper we provide a method to obtain an abstraction that allows to reduce the number of rules significantly. We use the Datanode ontology, a hierarchical organisation of the possible relations between data objects, to compact the knowledge base to a set of more abstract rules. After giving a definition of *Policy Propagation Rule*, we show (1) a methodology to abstract policy propagation rules based on an ontology, (2) how effective this methodology is when using the Datanode ontology, (3) how this ontology can evolve in order to better represent the behaviour of policy propagation rules.

CCS Concepts

•**Information systems** → *Semantic web description languages;* •**Social and professional topics** → *Computing / technology policy;* •**Computing methodologies** → *Semantic networks;* •**Applied computing** → *Law;*

Keywords

Formal Concept Analysis, RDF Licenses, Datanode

1. INTRODUCTION

Governing the life cycle of data on the web is a challenging issue for organisations and users. Data is distributed under certain policies that determine what actions are allowed and in which circumstances. In a smart cities data hub [6] information is gathered in several different ways, spanning from the loading of static files to very dynamic data like the ones pushed by sensors. Store retailers load the frequency of customers entering and exiting shops, places in car parks are monitored by sensors that register whether a space is busy or not, etc. Consequently, information is owned by different users, that fix the constraints on the usage of the data they provide under specific terms or licences. In addition, a smart city data hub plays the role of a mediator between data providers and data consumers [6], where consumers are companies establishing processes that integrate, manipulate, analyse the data in order to build services for end-users. The way terms and licences propagate through these operations therefore needs to be managed.

For example, the Chief Technology Officer of Busy Times wants to decide which terms of use to apply to the data delivered by their service, performing an estimatation of the busyness (i.e. crowd density) of an area in a given future time frame. They hope to sell this data to companies in the transport or advertisement domain. In order to do that, they access and reuse a set of data sources among the ones that are suitable to be used with a commercial purpose, according to the licences assigned by the data providers. The company wants to limit its customers in the reuse of the data (for example, prohibiting re-sell to third parties). While doing this, they need to be sure not to invalidate policies that are inherited from the original data and that may be still valid for the output of the process. In this context, assessing what policies propagate to the output of a process is an important, and difficult problem.

Policies and processes can be represented as ontologies within the Semantic Web framework. The Open Digital Rights Language (ODRL)[1] is an emerging information model to support the exchange of policies on the World Wide Web. The Datanode [5] ontology[2] has been designed to enable formally describing how applications use data, and consists in a taxonomy of ontological relations between data objects. With Datanode, processes can be represented as networks of data objects connected by semantic relations. With these descriptions, propagation rules can be established in order to capture how policies, represented in ODRL, propagate in rich data flows. These rules would then tell us what are, in given circumstances, the set of policies to apply to the output of a complex data processing task. However, having a description of policies and data flow steps implies a huge number of propagation rules to be specified and computed (number of policies times number of possible actions). We aim to provide an abstraction that allows to reduce the

K-CAP 2015 October 07 - 10, 2015, Palisades, NY, USA
ACM ISBN [POSTPRINT].
DOI: [POSTPRINT]

[1]https://www.w3.org/community/odrl/
[2]http://purl.org/datanode/ns/

number of rules significantly.

In this paper we show

1. a methodology to abstract policy propagation rules based on an ontology,

2. how effective this methodology is when using the Datanode ontology,

3. how this ontology can evolve in order to better represent the behaviour of policy propagation rules.

The next Section describes the background and related work for our research. In Section 3 a definition of Policy Propagation Rule is given, before introducing the approach we propose and provide an overview on the methodology. We describe how the methodology has been applied using the Datanode ontology in Section 4. Finally, we derive some conclusions in the last section.

2. BACKGROUND AND RELATED WORK

Data governance is at the centre of the current effort to master the information flows in modern smart cities [6]. In the world of (linked) open data, licences are important in order to enable a conscious exploitation of the resources and to develop technologies that allow policies to be negotiated and enforced [2]. The Open Digital Rights Language (ODRL)[3] is an emerging information model to support the exchange of policies on the World Wide Web. The W3C ODRL Community Group work at the development of a set of specifications to enable interoperability and transparent communication of policies associated with software, services and data. A policy expressed with the ODRL model includes a deontic aspect - `odrl:duty`, `odrl:permission` or `odrl:prohibition`, associated to a set of `odrl:Action`s. The RDF Licenses Database [12] is a notable effort towards the establishment of a common database of licence descriptions based on RDF and the ontology provided by ODRL (among others).

Rule based representation and reasoning on policies [7, 9] is required in order to enable secure data access and usage in distributed environments, particularly the Semantic Web [3]. Processes can be described in the Semantic Web using the Provenance Ontology (PROV-O) [10]. PROV-O describes workflow executions in terms of agents, actions and assets involved. The Datanode ontology [5] has been designed to describe Semantic Web applications by the means of the relations between the data involved in their processes. The ontology is a taxonomy of possible relations that may occur between data object, which might be part of a process execution, such as one represented using PROV-O. It can therefore be used to further qualify the implications of the actions performed in such a process. Datanode can describe process implications in a data-oriented way: as network of data objects. The RDF Licenses Database and Datanode are the primary resources for the present work.

Policies Propagation Rules are Horn rules. Reasoning on Horn rules is one way of dealing with policies, particularly because they allow a treatable defeasable reasoning [1].

Formal Concept Analysis (FCA) has the capability of classifying collections of objects depending on their *features*. FCA has been used for ontology alignments (for instance in [14]). In our case we use it to detect common behavior of relations in terms of policy propagation, and then to test and refine the Datanode ontology. In [4] FCA has been applied as part of a method for classifying licences based on common features, to reduce the cost of licence selection. We refer the reader to [4] for a description of the Contento tool, that implements FCA as well as other functionalities for evolving concept lattices in Semantic Web ontologies, also part of the approach we present here.

The approach described in this paper clearly relates to principles and methods of knowledge engineering [13]. In [11], knowledge acquisition is considered as an iterative process of model refinement, and this is exactly how we decided to tackle our problem here. More recently, problem solving methods have been studied in relation to the task of understanding process executions [8].

While policies and process executions can be represented, what we aim to do here is to investigate the propagation of policies across data flows.

3. APPROACH

The intent of the present work is to verify to what extend it is possible to compact a knowledge base of propagation rules using an ontology that organizes data flow steps in a hierarchy. In this section we outline the methodology designed for this task, while in the next section we will apply it concretely.

Before going into the details of the methodology it is worth introducing the concept of *Policy Propagation Rule*, as we define it. A Policy Propagation Rule is a Horn Clause of the following form:

$$has(X,P) \wedge propagates(P,R) \wedge relation(R,X,Y) \\ \rightarrow has(Y,P)$$

Where X and Y are data objects, P is a policy and R a relation between two data objects. For example a Policy Propagation Rule (PPR) could be used to represent the fact that downloading a file F distributed with an attribution requirement will result in a local copy D, that also needs to be used according to the attribution requirement. Therefore, the above abstract rule could be instantiated as follows:

$$has(F, attribution) \wedge \\ propagates(attribution, isCopyOf) \wedge \\ relation(isCopyOf, F, D) \rightarrow has(D, attribution)$$

In fact, we can reduce a PPR to a more compact form, i.e. a binary association between a policy and a relation:

$$propagates(policy, relation)$$

as the other parts can be generated automatically from the above representation.

From this definition, we obtain a method to generate a knowledge base of rules from a binary matrix of relations and policies. This is the first assumption of our methodology.

The second assumption is an ontology is available to organise data flow steps in a hierarchy, by the means of its semantics. For example, this ontology would tell us that the relation *isCopyOf* is a kind of *isDerivationOf*.

The methodology is composed of the following phases:

A1 **Acquisition.** The initial task is to setup a knowledge base of PPRs.

[3]https://www.w3.org/community/odrl/

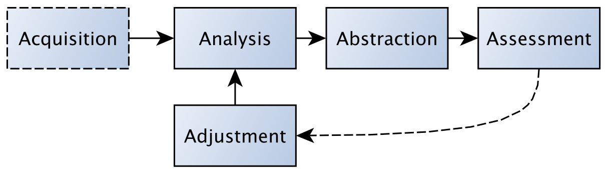

Figure 1: (A)AAAA Methodology.

A2 **Analysis.** The FCA algorithm is performed on the knowledge base of PPRs. The output of the process is a set of concepts: clusters of policies that propagate with the same set of relations, ordered in a lattice.

A3 **Abstraction.** In this phase we search for matches between the ontology and the FCA lattice. When a match occurs, we subtract the rules that can be abstracted through the ontology's taxonomy.

A4 **Assessment.** We check to what extent a hierarchical organization of the relations matches the clusters produced by FCA (developing measures). This step evaluates how much the ontology compresses the knowledge base (i.e. the compression factor), and detects partial matches;

A5 **Adjustment.** Observing the measures produced in the previous phase, particularly about partial matches, in this phase we perform operations to fix inaccuracies, evolve the ontology and improve the compression factor.

The above methodology consitutes an iterative process, as shown in Figure 1.

3.1 Acquisition

The initial task is to setup a knowledge base of Policy Propagation Rules (PPRs). As illustrated at the beginning of this Section, a PPR can be conceived as an association between a policy and a relation between two data objects. The knowledge base is then a binary matrix, where each row represents a possible relation between two data objects, and each column a policy. The cells in the matrix can be 1 or 0, depending on whether the policy propagates or not with the given relation. For example, the matrix might contain the cells

```
hasCopy,duty_shareAlike,1
hasDerivation,duty_shareAlike,1
hasDerivation,permission_reproduce,0
...
```

From each positive cell in the matrix, we can generate a PPR, and populate the set of rules R.

3.2 Analysis

The objective of the second phase is to detect common behaviors of relations with respect to policy propagation. We achieve this by applying FCA, implemented in the Contento tool [4]. The output of the FCA algorithm is an ordered set of *concepts* C. In FCA terms, each concept groups a set of objects (the concept's *extent*) and maps it to a set of attributes (the concept's *intent*). An FCA Concept groups a set of objects all having a given set of attributes (and vice-versa). In our case, each concept maps a group of relations propagating a group of policies. These concepts are organized hierarchically by FCA (in a lattice), from the top concept T, that includes all the objects and potentially no attributes, to the bottom concept B, including all the attributes with potentially an empty extent (set of objects). All other concepts are ordered from the top to the bottom. For example, a first layer of concepts right below T would include large groups of objects all having few attributes in common. Layers below would have more attributes and less objects, until the bottom B is reached. In our setting, T includes all relations and no policies, while the bottom concept B includes all the policies but no relations.

The concepts identified by FCA collect relations that have a common behavior in our knowledge base R (they propagate the same policies). However, we don't have an explanation for these clusters. In other words we don't know *why* they do it. A suitable abstraction should not only have operational effectiveness, but also be meaningful. We make the hypothesis that an ontology of relations, organized in a hierarchy by the means of their semantics, should contribute to enlighten the meaning of these concepts.

3.3 Abstraction

The abstraction process is based on applying an ontology that organizes the relations in a hierarchy. For instance, the relation *hasCopy* is a sub-relation of *hasDerivation*. A number of policies propagated by *hasDerivation* should be also propagated by *hasCopy* and all the others sub-relations in that branch of the hierarchy. By grouping all the relations below *hasDerivation* in a transitive closure, we obtain a cluster of relations similar to the ones in the FCA concepts that we call the *hasDerivation branch*, for example. We expect the branches of the ontology to be reflected in the clusters of relations obtained by FCA, thus we search for matches between the ontology and the FCA lattice. When a match occurs, we subtract the rules that can be abstracted.

Listing 1: Abstraction algorithm.

```
R = Rules()
C = FCAConcepts()
H = ComputeBranches()
ForEach (c,h) in (C,H)
        (p,r) = Match(c,h)
        when  p == 1.0
            Subtract(R, Policies(c), Branch(h))
```

The process is summarized in Listing 1, and described as follows:

i **Concepts.** From the result of the FCA algorithm we obtain a set of concepts C including relations r (extent of the concept) all propagating a set of common policies p (intent of the concept):

$$C = ([r_1, p_1], [r_2, p_2], \ldots, [r_n, p_n])$$

ii **Branches.** For each relation in the ontology, we compute the transitive closure of its sub-relations, obtaining a set of mappings:

$$H = ([t_1, b_1], [t_2, b_2], \ldots, [t_n, b_n])$$

where t is a relation in the ontology and b the related branch, i.e. all the sub-relations of t.

iii **Matching.** We search for (partial) overlaps between branches in H and clusters in C. The measure of the possible match of each branch with each cluster is evaluated with *precision* and *recall*. In the case a branch is fully in the cluster, we say the match has the highest precision (1.0). Inversely, recall indicates how much a branch covers the cluster. A cluster including only relations in the branch would result in a match with maximum recall 1.0.

iv **Compression.** When a match has full precision, we can use the top relation as abstraction for its sub-relations, for all the policies in the intent of the concept. In other words, we remove all the rules referring to a subsumed relation, as they are implied by a more abstract one.

3.4 Assessment

The result of the abstraction phase can be represented as a set of measures between concepts and portions of the ontology. The measures we are interested for the assessment phase are:

- Extent size (ES). Number of relations in the concept (the size of the cluster).
- Intersection size (IS). Number of relations of the branch that are also present in the extent of the concept.
- Branch size (BS). Number of relations in the branch.
- Precision (Pre). It is calculated as $Pre = IS/BS$, meaning the degree of matching of a branch with the extent of the concept.
- Recall (Rec). Recall, calculated as $Rec = IS/ES$, i.e. how much the extent of the concept is covered by the branch.
- F-Measure (F1). F-Measure, the well-known fitness score, calculated from precision and recall as $F1 = 2 * ((Pre * Rec)/(Pre + Rec))$.

These measures are now considered to quantify and qualify the way the ontology aligns with the propagation rules: precision and recall indicate how much a relation is close to being a suitable abstraction for policy propagation. Table 1 shows an example taken from the experiment (see Section 4). Here it is worth mentioning some general considerations that can be made by inspecting these measures.

Table 1: Excerpt from the table of measures computed by the abstraction algorithm in Listing 1.

c	ES	IS	BS	Pre	Rec	F1	Branch
79	52	52	115	0.45	1	0.62	relatedWith
77	46	19	21	0.9	0.41	0.56	hasDerivation
75	44	8	11	0.73	0.18	0.29	samePopulationAs
67	35	7	7	1	0.2	0.33	hasPart
67	35	6	7	0.86	0.17	0.28	isPartOf
36	16	3	3	1	0.19	0.32	hasCopy
36	16	3	3	1	0.19	0.32	isCopyOf
24	12	6	6	1	0.5	0.67	hasVocabulary
9	8	1	1	1	0.12	0.21	hasReification
0	4	4	115	0.03	1	0.06	relatedWith

c=Concept ID, ES=Extent Size, IS=Intersection Size, BS=Branch size, Pre=Precision, Rec=Recall, F1=F-Measure.

When $Rec = 1$, the whole extent of the concept is in the branch. The branch might also include other relations, that do not propagate the policies included in the concept. When $Pre = 1$, we can perform the subtraction of rules, as in Listing 1. A low recall indicates that a high number of exceptions still need to be kept in the rule set. It also reflects a high ES, from which we can deduce a low number of policies in the concept. As a consequence of that, inspecting a partial match with high precision and low recall highlights a problem that might be easy to fix, as the number of relations and policies to compare will be low. For example, row 2 of Table 1 relates to a relation with $BS - IS = 2$, so we need only to check whether 2 relations in the `hasDerivation` branch might also propagate the policies in concept 77. The perfect match between a concept and a branch of the ontology would be $F1 = 1$. However, when this does not happen we can try to improve the approximation.

A general estimation of the effectiveness of the approach is given by the *compression factor* (CF). We calculate the CF as the number of abstracted rules divided by the total number of rules:

$$CF = \frac{|A|}{|R|}$$

with R the set of rules, and A the set of rules that can be subtracted. Depending on the compression factor, we can choose whether to perform any adjustments to the knowledge base or any refinement of the ontology. At the same time, the compression factor indicates how much the ontology is coherent with the common behavior of relations in the knowledge base of Policy Propagation Rules.

3.5 Adjustment

In this phase, we try to make adjustments in order to improve the compression factor. We defined a set of operations, targeted to fix errors in the initial knowledge base (R) as well as to evolve the ontology.

3.5.1 Fill

Sometimes a branch is not completely in the cluster. We can inspect the missing relations, and realize that they should be (diagnosis). This operation is illustrated by Figure 2. The ticked circles represents relations in the ontology that are in the cluster of a given concept. Unticked circles are missing, meaning they do not propagate the policies in the concept. The Fill operation makes a branch b be fully in a cluster of concept c, attempting to push Pre up to 1. This is achieved by adding to R all the rules generated from the association between the policies in concept c and the relations in branch b. This change affects the PPR knowledge base R, increasing the number of rules.

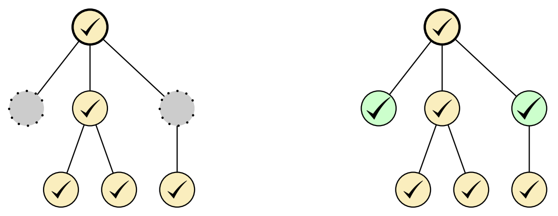

Figure 2: The *Fill* operation. The diagram on the left shows the diagnosis of the issue, while the diagram on the right shows the way it was repaired.

3.5.2 Wedge

Sometimes a branch is abstracted by a relation that is too general, so that all its sub-relations actually propagate the policies in c but the top relation cannot. Figure 3 illustrates the operation. As a result, a new relation is wedged between a top relation and all its direct subproperties. The new branch will allow to perform a *Fill* operation, in the next iteration.

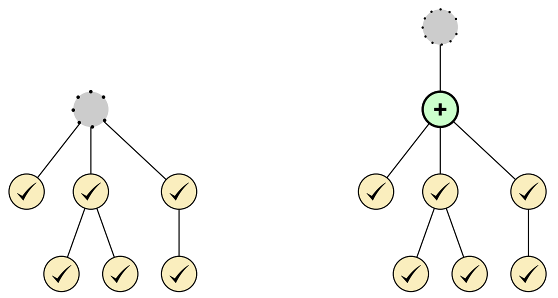

Figure 3: The Wedge operation.

3.5.3 Merge

We observe a concept matched by two branches with max precision. It is possible that the two top relations can be abstracted by a new common relation. If this is the case we perform this operation and create a new relation, as shown in Figure 4. The new branch will allow to perform a *Fill* operation, in the next iteration.

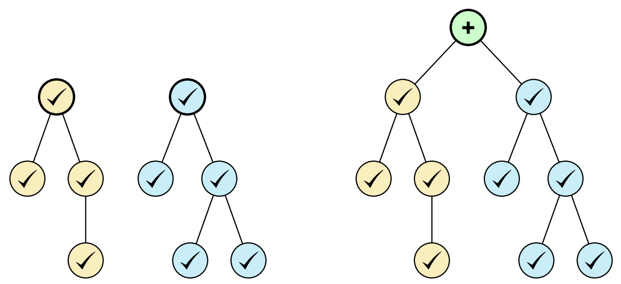

Figure 4: The Merge operation.

3.5.4 Group

Figure 5 illustrates the *Group* operation. A set of relations are all together in the extent of a concept, but belong to different branches. We want to create a common parent relation for them. Again, the new branch enables a *Fill* operation, to be executed in the next iteration.

3.5.5 Remove and Add

We can *Remove* a relation as a subproperty of another (and possibly cut a sub-branch). This operation removes a single subsumption relation in the ontology. A relation is not in the cluster of a concept, the branch is almost all there, and we realize that the relation is not really subsumed by the other. Basically we detail the semantic of the relation and we remove it from the branch. After that, we might relocate it elsewhere with a *Add* operation.

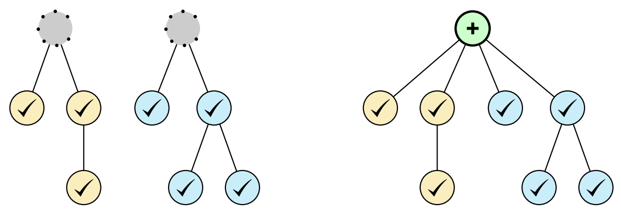

Figure 5: The Group operation.

Except from the *Fill* operation, all the others are performed on the ontology. After the change in the ontology, normally the *Fill* operation is performed on the newly created branch, in order to populate the rule base with the new rules. As shown in Figure 1, after the Adjustment phase we restart a new iteration. The process is repeated until a reasonably good compression factor is reached, or no more meaningful changes are possible.

4. APPLICATION

In complex environments such as a city data hub, data have a very diverse life cycle, and understanding how policies propagate between the data objects is a crucial aspect of data governance. In this Section we describe how we applied the methodology to compress the knowledge base of policy propagation rules that can be used for this task. As we described in the previous section, the knowledge base can be considered as a binary matrix associating a set of policies and a set of relations between data objects, resulting from a workflow execution or a de-facto situation in the datahub.

4.1 Acquisition

In order to setup the knowledge base of Policy Propagation Rules we relied on the RDF Licenses Database [12], and extracted 113 possible policies. Each policy is an association of one deontic element (permission, prohibition or duty) and one action (see Listing 2 for examples).

Listing 2: Examples of policies.

```
permission odrl:derive
prohibition ldr:extraction
duty odrl:attachPolicy
```

We could have generated the policies by combining any ODRL action with any deontic component. However, this would have led to a large number of meaningless policies (eg: `duty odrl:use`). The adoption of the RDF License Database permitted to obtain a list of meaningful policies only. We used the Datanode Ontology [5] to extract a list of 115 possible relations between data objects.

This phase required a manual supervision of all associations between policies and relations in order to enstablish the initial set of propagation rules. The two collections have been used together in the Contento tool [4], that generated the FCA formal context, i.e. a binary matrix of 12995 cells. At this stage, manual supervision was required to inspect all the cells and decide whether to enstablish a rule or not. The Contento tool allows to incrementally inspect different portions of the context relying on filtering capabilities and bulk operations. We were able to check and uncheck collections of cells with similar status, and to keep track on the cells that were still to be supervised. Thanks to Contento, it was possible to edit manually the formal context with a reasonable effort. Listing 3 displays a sample of binary relations that can be true or false in the formal context.

Listing 3: Example of cells of the binary matrix associating policies and rules (FCA formal context)

```
dn:hasPortion,permission odrl:copy,1
dn:identifiersOf,prohibition cc:DerivativeWorks,1
dn:isDependencyOf,permission odrl:derive,0
dn:usesSchema,permission cc:Reproduction,1
dn:processedInto,duty odrl:shareAlike,1
dn:isVocabularyOf,prohibition odrl:use,1
dn:metadata,prohibition odrl:transform,0
```

At the end of this process, the matrix had 3363 cells marked as true. The initial knowledge base was then composed of 3363 Policy Propagation Rules (Listing 4). The reader can deduce that a large part of Datanode included relations that do not propagate any policy, for example the top relation `dn:relatedWith`, but also `dn:overlappingCapabilityWith`, `dn:about`, among others.

Listing 4: Examples of Policy Propagation Rules.

```
propagates(dn:hasPortion,permission odrl:copy)
propagates(dn:identifiersOf,prohibition cc:DerivativeWorks)
propagates(dn:usesSchema,permission cc:Reproduction)
propagates(dn:processedInto,duty odrl:shareAlike)
propagates(dn:isVocabularyOf,prohibition odrl:use)
propagates(dn:metadata,prohibition odrl:transform)
```

4.2 Analysis

Following the method described in Section 3.2, we applied the FCA algorithm and obtained 80 concepts, each representing a cluster of relations propagating the same set of policies. Listing 5 shows one exemple. All the relations in the extent of this concept propagates the policies in the intent.

Listing 5: Example of a Concept.

```
Concept 74
Extent                     Intent
dn:attributesOf            duty cc:Attribution
dn:cleanedInto             duty cc:Copyleft
dn:combinedIn              duty cc:Notice
dn:datatypesOf             duty cc:SourceCode
dn:descriptorsOf           duty odrl:attachPolicy
dn:duplicate               duty odrl:attachSource
dn:hasAnonymized           duty odrl:attribute
dn:hasAttributes
dn:hasCache
dn:hasChange
dn:hasComputation
dn:hasCopy
dn:hasExample
dn:hasExtraction
dn:hasIdentifiers
dn:hasInterpretation
dn:hasPart
dn:hasPortion
dn:hasReification
dn:hasSample
dn:hasSection
dn:hasSelection
dn:hasSnapshot
dn:hasStandIn
dn:hasTypes
dn:hasVocabulary
dn:identifiersOf
dn:isCacheOf
dn:isChangeOf
dn:isCopyOf
dn:isExampleOf
dn:isPartOf
dn:isPortionOf
dn:isSampleOf
dn:isSectionOf
dn:isSnapshotOf
dn:isVocabularyOf
dn:optimizedInto
dn:processedInto
dn:refactoredInto
dn:relationsOf
dn:remodelledTo
dn:typesOf
```

4.3 Abstraction

In the abstraction phase we search for matches between clusters of relations obtained by FCA (the concepts' extents) and branches in the Datanode hierarchy. Listing 6 shows the output of the abstraction algorithm, returning all the branches that are (partially) matched by a given concept. For example, the extent of Concept 74 is a cluster of 43 relations, matching several branches in Datanode in different ways. At the begninning there is the Top Property of Datanode: `dn:relatedWith`. Its branch size (bs) is 115 relations, constituting the whole hierarchy in the ontology. Clearly, the size of the intersection is the same as the size of the extent, as all the 43 relations of the cluster are in the branch. However, the precision (pre) of the matching is pretty low: 0.37. This branch obviously matches the whole extent of Concept 74 with full recall (rec) 1.0 - like with any other concept.

A more interesting case is the branch associated with `dn:hasPart`, which intersection with the concept is made of all 7 sub-relations. In fact, looking at the intent of Concept 74 in Listing 5, it sounds reasonable that all the parts of a given data objects inherit all the cited duties[4]. The full precision enables the rules reduction process. A similar case is with the relation `dn:isVocabularyOf`.

Listing 6: Example of the matches between a concept and the branches in the Datanode hierarchy.

```
c    es   is   bs    pre    rec   f1    branch
74   43   43   115   0.37   1     0.54  dn:relatedWith
[...]
74   43   7    8     0.87   0.16  0.27  dn:sameCapabilityAs
74   43   7    7     1      0.16  0.28  dn:hasPart
74   43   6    7     0.86   0.14  0.24  dn:isPartOf
74   43   3    6     0.5    0.07  0.12  dn:hasVocabulary
74   43   6    6     1      0.14  0.25  dn:isVocabularyOf
[...]
```

4.4 Assessment

By only considering the branches matched with full precision by each concept, we can substract 1925 rules, for a compression factor of 0.572. At this stage we can make the following considerations:

- The size of the matrix that was manually supervised is large, and it is possible that errors have been made at that stage of the process.
- The Datanode ontology has not been designed for the purpose of representing a common behavior of relations in terms of propagation of policies. It is possible to refine the ontology in order to make it cover this use case better (and reduce the number of rules even more).

4.5 Adjustment

In Section 3.4 we described a method to catch possible errors in the matrix, based on the identification of partial matches with high precision and low recall. Such cases highlight a branch that is close to be fully included in the extent of a concept. As example, we can pick branch `dn:isPartOf` from Listing 6. Listing 7 shows the details about how the concept matches this branch. It happens that all relations except `dn:isSelectionOf` are part of this concept. In other

[4]They inherit many other policies as well, and those are considered by other concepts in a lower layer of the FCA lattice.

words, they propagate the policies listed in the intent of Concept 74 (Listing 5). However, this is a mistake that happened in the acqusition phase, as `dn:isSelectionOf` should behave in a similar way to `dn:isExampleOf`.

Listing 7: Example of the matches between a concept and the branches in the Datanode hierarchy.

```
c    es   is   bs   pre   rec   f1    branch
74   43   6    7    0.86  0.14  0.24  dn:isPartOf
                          + dn:isPartOf
                          ! dn:isSelectionOf
                          + dn:isExampleOf
                          + dn:isSectionOf
                          + dn:identifiersOf
                          + dn:isPortionOf
                          + dn:isSampleOf
```

We decide then to perform a *Fill* operation, adding all the necessary rules to make the branch `dn:isPartOf` fully covering the intent of Concept 74.

A branch with similar scores is `dn:sameCapabilityAs`. Listing 8 shows that the only missing relation is the top one. In Datanode, `dn:sameCapabilityAs` is defined as the relation between two objects having the same vocabulary and the same population (containing actually the same data). However, it is possible that two objects have the same "data" without having the same policies. For example, datasets like lists of cities or postcodes might be imported from different sources, and having different policies while containing the same data! In this case we opted for adding a new relation to Datanode that can abstract all the branches with a more focused semantic: `dn:sameIdentityAs`. `dn:sameIdentityAs` tries to capture exactly the fact that two data objects share the same origin, the same population and the same vocabulary. The operation performed to add this relation is *Wedge*, as the new property is injected between `dn:sameCapabilityAs` and its direct sub-relations.

Listing 8: Example of the matches between a concept and a branch in the Datanode hierarchy.

```
c    es   is   bs   pre   rec   f1    branch
74   43   7    8    0.87  0.16  0.27  dn:sameCapabilityAs
                          ! dn:sameCapabilityAs
                          + dn:hasCopy
                          + dn:hasSnapshot
                          + dn:hasCache
                          + dn:isCopyOf
                          + dn:isSnapshotOf
                          + dn:isCacheOf
                          + dn:duplicate
```

After each operation we run our process again from the Analysis phase to the Assessment, in order to evaluate how much the change affected the compression factor.

Table 2 lists the changes performed, with statistics about the impact on number of rules, number of concepts generated by FCA, number of rules abstracted, rules to compute and compression factor.

Figure 6 shows how the compression rate increases with the number of adjustment operations done.

As a result we obtained: 3865 rules in total, 78 concepts, 2817 rules abstracted and 1048 rules remaining - for a compression factor of 0.729.

Thanks to this methodology we have been able to fix many errors in the initial data, to refine Datanode by clarifying the semantics of many properties and adding new ones. The version of the ontology at the beginning of this work can be found at http://purl.org/datanode/0.3/ns/. The current version of the ontology is at http://purl.org/datanode/0.4/ns/.

Table 2: List of changes performed.

+	C	R	A	R_+	CF
0	80	3363	1925	1438	0.572
1	80	3370	1953	1417	0.58
2	80	3370	1953	1417	0.58
3	80	3480	2283	1197	0.656
4	80	3482	2299	1183	0.66
5	78	3500	2376	1124	0.679
6	78	3608	2484	1124	0.688
7	78	3716	2592	1124	0.698
8	96	3822	2698	1124	0.706
9	93	3824	2706	1118	0.708
10	93	3824	2706	1118	0.708
11	93	3824	2706	1118	0.708
12	93	3824	2706	1118	0.708
13	76	3837	2765	1072	0.721
14	76	3844	2778	1066	0.723
15	78	3865	2817	1048	**0.729**

The first column identifies the change performed (starting from the initial state).
C=Number of concepts in the FCA lattice
R=Number of PPRs before the process
A=Number of rules abstracted (subtracted)
R_+=Size of the rule set without the abstracted rules
CF=Compression Factor

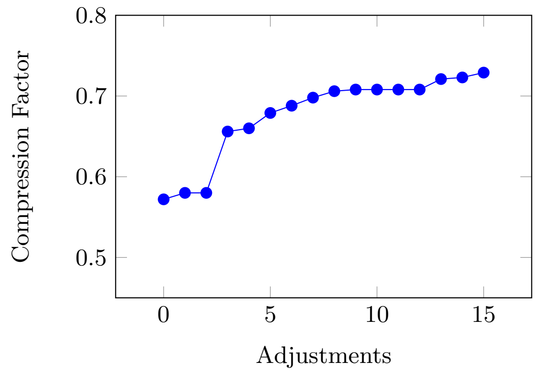

Figure 6: Progresss of the CF.

5. CONCLUSIONS

In this paper we presented a method to abstract Policy Propagation Rules. The approach relies on the Datanode ontology [5], a hierarchical organization of the possible relations between data objects. We demonstrated that applying Datanode to this task allows us to reduce the number of rules to a factor of 0.5. Moreover we have been able to analyse the ontology in relation to this task and enhance it having as result not only a further reduction of rules, but also a better ontology. In addition, applying the ontology and the method we were able to find and correct errors in the rules.

A similar approach could be applied to the set of policies, by relying on the ODRL ontology, that contains subsumption relations between actions. However, the hierarchy of actions in ODRL is quite small compared to Datanode.

An interesting problem is related to the changes occurring in the initial knowledge base. The representation of licences in RDF/ODRL is an ongoing effort, and evolutions in the initial knowledge base can contribute to further improve Datanode. We expect that new policies could contribute to expand and improve Datanode even more.

As future work we will extend the Assessment phase of the methodology to also include coherency check between the hierarchy of the FCA lattice and the one of the ontology, and build additional measures to support the Adjustment phase. Currently, while the Aquisition and Analysis phases have been performed using Contento, the Assessment and Adjustment phases have been implemented in dedicated scripts. We are investigating if it would make sense to integrate the measures and operations of the methodology in the Contento tool, and apply the approach to other use cases.

6. REFERENCES

[1] G. Antoniou and G. Wagner. Rules and defeasible reasoning on the semantic web. In Schröder, Michael and Wagner, Gerd, editor, *Rules and Rule Markup Languages for the Semantic Web*, volume 2876 of *Lecture Notes in Computer Science*, pages 111–120. Springer Berlin Heidelberg, 2003.

[2] O. Boissier, M. Colombetti, M. Luck, J.-J. Meyer, and A. Polleres. Norms, organizations, and semantics. *The Knowledge Engineering Review*, 28(01):107–116, 2013.

[3] P. A. Bonatti and D. Olmedilla. Rule-based policy representation and reasoning for the semantic web. In *Proceedings of the Third International Summer School Conference on Reasoning Web*, RW'07, pages 240–268, Berlin, Heidelberg, 2007. Springer-Verlag.

[4] E. Daga, M. d'Aquin, A. Gangemi, and E. Motta. A bottom-up approach for licences classification and selection. In S. Villata and S. Peroni, editors, *Proc. of the International Workshop on Legal Domain And Semantic Web Applications (LeDA-SWAn) held during the 12th Extended Semantic Web Conference (ESWC 2015)*, pages 33–40. ACM, 2012.

[5] E. Daga, M. d'Aquin, A. Gangemi, and E. Motta. Describing semantic web applications through relations between data nodes. Technical Report kmi-14-05, Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, 2014.

[6] M. d'Aquin, A. Adamou, E. Daga, S. Liu, K. Thomas, and E. Motta. Dealing with diversity in a smart-city datahub. In T. Omitola, J. Breslin, and P. Barnaghi, editors, *Proceedings of the Fifth Workshop on Semantics for Smarter Cities, a Workshop at the 13th International Semantic Web Conference (ISWC 2014)*, Riva del Garda, Italy, 19 October 2014. CEUR-WS.org.

[7] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. E. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *The Semantic Web: Research and Applications*, pages 342–356. Springer, 2004.

[8] J. M. Gómez-Pérez and O. Corcho. Problem-solving methods for understanding process executions. *Computing in Science & Engineering*, 10(3):47–52, 2008.

[9] H. Li, X. Zhang, H. Wu, and Y. Qu. Design and application of rule based access control policies. In *Proc of the Semantic Web and Policy Workshop*, pages 34–41, 2005.

[10] D. McGuinness, T. Lebo, and S. Sahoo. PROV-o: The PROV ontology. W3C recommendation, W3C, Apr. 2013. http://www.w3.org/TR/2013/REC-prov-o-20130430/.

[11] E. Motta, T. Rajan, and M. Eisenstadt. Knowledge acquisition as a process of model refinement. *Knowledge acquisition*, 2(1):21–49, 1990.

[12] V. Rodríguez-Doncel, S. Villata, and A. Gómez-Pérez. A dataset of RDF licenses. In R. Hoekstra, editor, *Legal Knowledge and Information Systems. JURIX 2014: The Twenty-Seventh Annual Conference.* IOS Press, 2014.

[13] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: principles and methods. *Data & knowledge engineering*, 25(1):161–197, 1998.

[14] G. Stumme and A. Maedche. Fca-merge: Bottom-up merging of ontologies. In *IJCAI*, volume 1, pages 225–230, 2001.