Semantic Web 1 (2009) 1–5
IOS Press

Reasoning with Data Flows and Policy Propagation Rules

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

Enrico Daga [a], Aldo Gangemi [b], Enrico Motta [a]
[a] *Knowledge Media Institute, The Open University*
Walton Hall, Milton Keynes, United Kingdom
E-mail: {enrico.daga,enrico.motta}@open.ac.uk
[b] *Université Paris13, France and Institute of Cognitive Sciences and Technologies - CNR, Italy*
E-mail: aldo.gangemi@{univ-paris13.fr,cnr.it}

Abstract.

Data-oriented systems and applications are at the centre of current developments of the World Wide Web. In these scenarios, assessing what policies propagate from the licenses of data sources to the output of a given data-intensive system is an important problem. Both policies and data flows can be described with Semantic Web languages. Although it is possible to define Policy Propagation Rules (PPR) by associating policies to data flow steps, this activity results in a huge number of rules to be stored and managed. In a recent paper, we introduced strategies for reducing the size of a PPR database by using an ontology of the possible relations between data objects, the Datanode ontology, and applying the (A)AAAA methodology, a knowledge engineering approach that exploits Formal Concept Analysis (FCA). In this article, we investigate whether this reasoning is feasible and how it can be performed. For this purpose, we study the impact of compressing a rule base associated with an inference mechanism on the performance of the reasoning process. Moreover, we report on an extension of the (A)AAAA methodology that includes a *coherency check* algorithm, that makes this reasoning possible. We show how this compression, in addition to being beneficial to the management of the rule base, also has a positive impact on the performance and resource requirements of the reasoning process for policy propagation.

Keywords: Data Hub, Data Flows, Policies, Rules, Formal Concept Analysis, RDF Licenses

1. Introduction

Data-oriented systems and applications are at the centre of current developments of the World Wide Web (WWW). Emerging enterprises focus their business model on providing value from data collection, integration, processing, and redistribution. These kind of systems are not new, as the Web has enabled for a long time tools such as news aggregators, which collect articles from various providers, and republish them as collections of short readings, often focusing on specific topics (politics, sport, etc.)[1]. Nowadays, the extraction, publication, and reuse of data on the Web is an established practice, and a large number of APIs provide access to JSON documents, data tables, or Linked Data for a variety of use cases, spanning from content and media linkage [20] to science and education [18].

The key aspect on which we are focusing here is the publication of Licenses and Terms and Condition

[1] Wikipedia: `https://en.wikipedia.org/wiki/News_aggregator`.

documents associated with those APIs and data artefacts, that declare the associated rights and policies that should guide their use. Data Hubs collect a large variety of data sources and process them in order to implement the workflow that *connects data in their original sources to applications that might want to exploit these data* [10]. Theses systems create new challenges in terms of the volume of data to be stored and require novel processing techniques (for example stream-based analysis [23]), but more importantly they demand for more sophisticated approaches to data governance [6]. In the Web of (open) data, developers can access a large variety of information, and often publish the results of their processing. Therefore, they need to know what are the usage constraints attached to a data source they want to exploit, and they need support in order to publish the appropriate policies alongside the data they distribute.

In this complex scenario, assessing what policies propagate from the licenses associated with the data sources to the output of a given data-intensive process is an important problem. Both policies and data flows can be described within the Semantic Web, relying on standards like the W3C PROV model[2] to describe process executions and the Open Digital Rights Language[3], which can be exploited for policies formalisation and validation. Particularly, it is possible to specify Policy Propagation Rules (PPR) [9] by associating policies with data flow steps, although this activity results in a large number of rules to be stored and managed. In [9], we studied how it is possible to compress a PPRs database by using an ontology of the possible relations between data objects, the Datanode ontology[4], and by applying the (A)AAAA methodology, a knowledge engineering approach that exploits Formal Concept Analysis (FCA).

In this article we illustrate how reasoning on policy propagation can be practically performed. Building upon [9], we report on an extension of the (A)AAAA methodology that includes a *coherency check* between the hierarchy of the FCA lattice and the Datanode ontology. This extension was necessary in order to exploit the compressed rule base during reasoning and avoid incorrect results. While the compression of the rule base reduces the number of rules to be managed, it requires the reasoner to compute more inferences. Therefore, we study the impact of rule base compression on the performance of the reasoning process. In other words, this article focuses on two contributions that relate to the aspect of reasoning with (compressed) PPR bases, which was missing [9]: 1- the extension of the (A)AAAA methodology with *coherency check* and 2- the evaluation of the effect of compression on reasoning performance.

The article is structured as follows. Section 2 traverses the relevant literature. Section 3 presents an exemplary use case, and introduces the elements for reasoning on policies propagation, going through the description of the data flow, the representation of policies, and the notion of Policy Propagation Rule (PPR) [9]. Section 4 provides a summary of the (A)AAAA methodology previously introduced in [9], integrated with a novel *Assessment* phase that includes a *coherency check* algorithm that allows effective reasoning with a compressed rule base. We also evaluate the impact of this evolved methodology on the compression factor of the rule base. In Section 5, we report on experimental results about the impact of a compressed rule base on reasoning. To this aim, we compare the performance of reasoning with an uncompressed rule base against reasoning with a compressed one. We perform this comparison using two different reasoners, the first computing the inferences at query time, the second materializing them at load time. Finally, we discuss our observations before closing the article with some conclusions and perspectives on future work.

2. Related Work

In recent years, data repositories and registries have been growing, spanning from data cataloguing services (Datahub[5]), data collections (Wikidata[6], Europeana[7]), to platforms that manage the collection and redistribution of data (Socrata[8]). An emerging category of such systems are city Data Hubs, which need to support developers not only in obtaining data, but also in assessing the policies associated with data resulting from complex pipelines [10,3,2]. It is therefore for these

[2] W3C PROV, https://www.w3.org/TR/prov-overview/.

[3] ODRL W3C Community Group: https://www.w3.org/community/odrl/.

[4] Datanode, http://purl.org/datanode/ns/ and http://purl.org/datanode/docs.

[5] Datahub. https://datahub.io/

[6] Wikidata. https://www.wikidata.org

[7] Europeana. http://labs.europeana.eu/

[8] Socrata. https://www.socrata.com/

systems to implement technologies that allow policies to be negotiated between distributed agents. In this article we concentrate on the problem of reasoning with propagating policies.

Policies can be represented on the Web in a machine-readable format. The W3C ODRL Community Group[9] works on the development of a set of specifications to enable interoperability and transparent communication of policies associated with software, services, and data. The Open Digital Rights Language (ODRL)[10] is an emerging language to support the definition, exchange and validation of policies [16]. Although ODRL is also available as an ontology, it only defines the semantics of policies in terms of natural language descriptions. An extension of the ODRL semantics have been proposed in [29] by considering dependencies between actions, and discussing the impact of explicit and implicit dependencies on the evaluation of policy expressions. The idea of establishing dependencies between ODRL actions in order to enhance the evaluation of ODRL expressions is comparable to our work, where we abstract relations in data flows. The Datanode ontology [8] which we use here is however designed to express a wide range of relations between data artefacts, and not only the ones derivable from actions. Nevertheless, a PPR reasoner can surely benefit from a well-defined semantics of ODRL actions. Recently, the W3C Permissions & Obligations Expression Working Group followed up on ODRL to develop an official W3C standard for defining permissions and obligations.

The RDF Licenses Database [26] is an attempt to establish a database of license descriptions based on RDF and the ontology provided by ODRL. It also exploits other vocabularies aimed to extend the list of possible actions, for instance the Linked Data Rights[11] vocabulary.

Process executions can be described in the Semantic Web using the Provenance Ontology (PROV-O) [24]. PROV-O describes workflow executions in terms of *agents*, *actions* and *assets* involved. The Datanode ontology has been designed to describe Semantic Web applications by means of the relations between the data involved in their processes [8]. The ontology is a taxonomy of possible relations that may occur between data objects, which might be part of a process execution, such as the ones described with 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, namely as network of data objects. While policies and process executions can be represented, in the present paper we aim at studying the process of reasoning upon the propagation of policies across a data flow.

Rule-based representation and reasoning on policies is required in order to enable secure data access and usage in distributed environments, particularly in the Semantic Web [22,11,4]. Defeasible logic is used to reason with deontic statements, for example to check compatibility of licenses or to validate constraints attached to components on multi-agent systems [27]. The problem of licenses' compatibility has been extensively studied in the literature [13,14] and tools that can perform such assessment do exist [21]. Our previous work introduces a form of policy reasoning, namely *policy propagation* [9]. A Policy Propagation Rule (PPR) is a Horn clause defined by associating a Datanode relation with a ODRL policy. Reasoning with Horn rules is an effective way of dealing with policies, particularly because Horn rules allow tractable defeasible reasoning [1]. While in this article we only focus on policies propagation, PPRs can in principle be integrated with rule-based reasoners for policy validation.

Formal Concept Analysis (FCA) [31] has the capability of classifying collections of objects depending on their *features*. We apply FCA to detect a common behaviour of relations in terms of policy propagation in order to compress the rule base, in conjunction with the Datanode ontology. We refer the reader to [7] 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 [30]. In [25], knowledge acquisition is considered as an iterative process of model refinement. More recently, problem solving methods have been studied in relation to the task of understanding process executions [12]. These contributions form the background of the approach we are following in the present work. The problem of compression of propositional knowledge bases has been deeply studied, focusing on the opti-

[9] W3C ODRL Community Group https://www.w3.org/community/odrl/

[10] ODRL Vocabulary & Expression, https://www.w3.org/TR/2016/WD-vocab-odrl-20160721/.

[11] Linked Data Rights (LDR): http://purl.oclc.org/NET/ldr/ns#.

mization of a Horn minimization process to boost rule execution [15,5]. Differently, we deal with compression as a mean to reduce the number of minimal rules to be managed (each PPR being already an atomic rule), by means of an additional knowledge base (the Datanode ontology).

It is worth noting that our problem is not one of policy enforcement, but of providing the right information about policies that might affect the terms of use of a given asset produced by a complex data flow. This problem is also different from the one of minimising access control policies (example, the abstraction by subsumption proposed in [17]), as the abstraction required is on the propagation of the policy, not the policy itself. Reasoning on policy propagation does not require the policies to be validated per se. On the contrary, we claim that validating the policies of a data artefact, which is the result of some manipulation, should consider the policies inherited from the input data, according to the particular actions performed.

To our knowledge, the problem of propagation of usage policies in data flows has not been tackled before the contribution in [9]. However, in [9] as well as in the present work, we do not focus on the quality of the data flow representations, and assume a machine-readable description of the policies of the input asset, as well as the existence of an accurate data flow.

3. Reasoning on policy propagation

In this section, we describe the approach for reasoning on policy propagation, and we present a use case as an example.

3.1. Approach

We define the problem of policy propagation as identifying the set of policies associated with the output of a process, implied by the policies associated with the input data source. In order to perform reasoning on policy propagation, we need:

a) descriptions of policies attached to data sources;
b) a description of the data flow (the actions performed on the data), and
c) policy propagation rules (which actions do propagate a given policy).

Next, a discussion of each one of the above elements follows.

Description of policies. We assume the policies of data sources are described as licenses or "terms and conditions" documents, and that they are expressed in RDF according to the ODRL ontology[12]. An ODRL `odrl:Policy` is an *entity to capture the statements of the policy*, specifying a set of `odrl:Rules`, each including a deontic aspect (`odrl:permission` or `odrl:prohibition`), which are defined for a set of `odrl:Actions` and a `odrl:target` `odrl:Asset`. Permissions, in turn, can comprise a `odrl:duty` (or more). For example, the RDF Licenses Database [26] is a source of such descriptions. In our work, we also developed ad-hoc RDF documents to satisfy this requirement, when necessary.

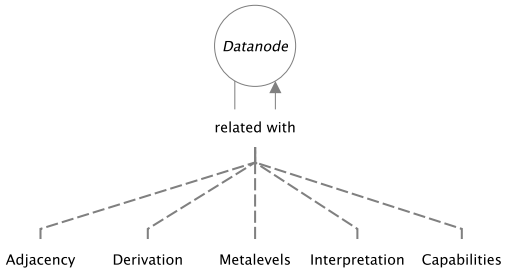


Fig. 1. Top hierarchy of the Datanode ontology.

Description of the data flow. Data flows are represented with the Datanode ontology [8]. The terms are defined under the `http://purl.org/datanode/ns/` namespace (we use the prefix `dn:`, for readability). The ontology defines a unique type - `dn:Datanode` - and 115 relations, starting from a single top property: `dn:relatedWith`, having `dn:Datanode` as `rdfs:domain` and `rdfs:range`. An instance of `dn:Datanode` is any data object that can be the input or output of a process. Figure 1 provides an overview of the top of the property hierarchy. The ontology groups the relations under five fundamental dimensions[13]:

- **Adjacency.** `dn:adjacentTo` represents proximity between two datanodes in a data

[12] ODRL 2.1: `https://www.w3.org/ns/odrl/2/ODRL21`.

[13] In this section we only summarize the basic features of the ontology, and we omit to specify inverse relations (for example `dn:isDerivationOf`), for clarity. The interested reader is referred to [8] and to the online documentation: `http://purl.org/datanode/docs`

container. For example, proximity may result from being parts of the same dataset - `dn:disjointPartWith`.

- **Derivation.** This branch specialises `dn:hasDerivation` in a number of different forms. Examples cover activities like mining - `dn:hasExtraction`, selection - `dn:isSelectionOf` reasoning - `dn:hasInference`, remodelling - `dn:remodelledFrom` or the activity of making snapshots of data or caches - `dn:hasSnapshot`, `dn:hasCache`, to mention a few.
- **Metalevels.** This dimension covers the relations between something and its metadata. The property `dn:metadata` is used to designate a relation with information that applies to the data node as a whole, possibly including it. This relation specialises as `dn:describes`, `dn:hasAnnotation` and `dn:hasStatistics`.
- **Interpretation.** This is designed to capture the possibility that a datanode might contribute to infererences that can be made in another one. Two datanodes *might* be "understood" together, i.e. their content can be compared, or the interpretation (inferences) of one may affect the interpretation (inferences) of another. The more intuitive examples are `dn:consistentWith` and `dn:inconsistentWith`. However, this is also the area of the ontology that covers partitive relations: `dn:isPartOf` and the two specialisations `dn:isPortionOf` and `dn:isSectionOf`. In Datanode, *portion* refers to a part of the population of a dataset (such as the rows of a spreasheet), while *section* refers to a set of values for a certain dimension in a dataset (for example, a column of a spreadsheet).
- **Capabilities.** Capability is intended as *the power or ability to generate an outcome*[14]. Capability is covered with two separate branches starting from `dn:overlappingCapabilityWith` and `dn:differentCapabilityFrom`, respectively. Two data nodes may have similar (or different) potential. For example, `dn:overlappingVocabularyWith` and `dn:overlappingPopulationWith` express the similarity between two data objects in terms of vocabulary or population of a dataset. Under this scope, we also positioned `dn:optimizedInto`, to state the empowerment of an existing capability.

It is worth noting that Datanode relations have often multiple ancestors. For example, `dn:hasStatistics` is both a `dn:hasComputation` and a `dn:describedBy` kind of relation, which in turn are subsumed by `dn:hasDerivation` and `dn:metadata` respectively. We refer to [8] for a discussion on the genesis of Datanode.

In this work, we use the representatons of data flows extracted from the descriptions of several Semantic Web applications prepared in [8].

Policy Propagation Rules. A Policy Propagation Rule (PPR) establishes a binding between a Datanode relation r and a policy p. A PPR 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 Datanode relation between the two. When the policy p holds for a data object X, related to another data object Y by the relation r, then the policy p will also hold for the data object Y. For example a 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, which 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 derived from the above representation.

With these elements established, we can trace the policies propagated within the data flow connecting input and output.

[14] Definition from `http://en.wiktionary.org/wiki/capability`.

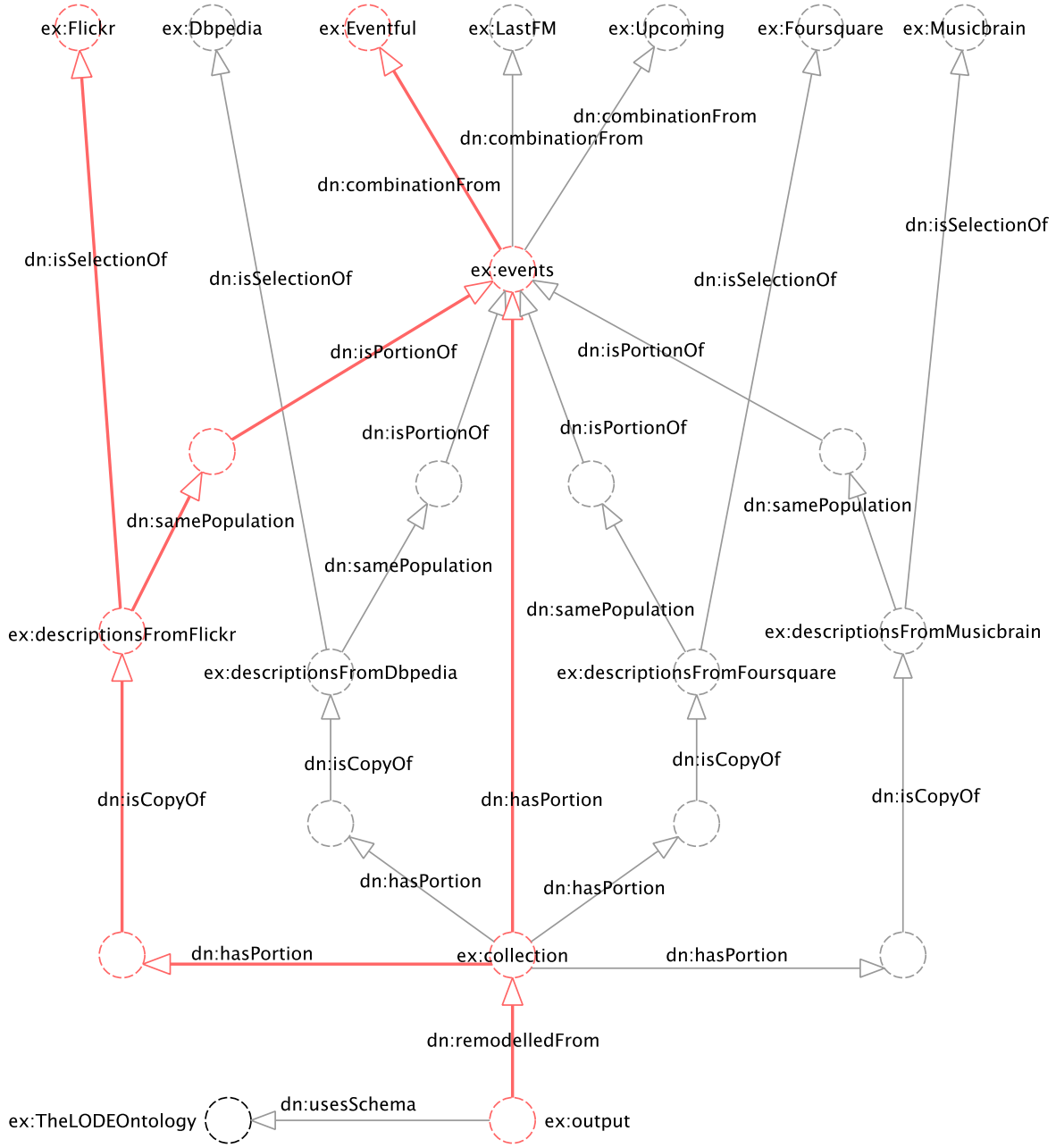


Fig. 2. The data flow of EventMedia. Input sources are the top nodes. The node at the bottom depicts the output data, which is a remodelling of the data collected from various sources according to a specific schema.

3.2. Example use case

We introduced the elements required to reason upon policy propagation in data flows. We now introduce a guide use case. The following namespace prefixes will be used in the description:

```
rdfs: <http://www.w3.org/2000/01/rdf-schema#>
odrl: < http://www.w3.org/ns/odrl/2/>
cc: <http://creativecommons.org/ns#>
dn: <http://purl.com/datanode/ns/>
ppr: <http://purl.com/datanode/ppr/ns/>
ex: <http://purl.org/datanode/ex/>
```

We selected EventMedia [19] as an exemplary data-oriented system. EventMedia exploits real-time connections to enrich content describing events and associate it with media objects[15]. The application reuses data exposed by third parties, aggregating data about events and exposing them alongside multimedia objects retrieved on the Web. Aggregated data are internally represented using the LODE ontology [28]. In order to associate the right policies to these data, we need all the elements described above, particularly a description of the policies of the input data, a description of the data flow, and a PPRs database.

Table 1 lists the licenses or terms of use documents associated with the input data objects. In the present case we were able to collect information from five out of six sources, since the *Upcoming* service not available at the time of writing[16]. Listing 1 lists the set of policies extracted from the Flickr APIs Terms of Use[17].

Listing 1: Policies representation extracted from the Flickr APIs Terms of Use.

```
ex:FlickrTC a odrl:Policy;
 a odrl:Agreement;
 rdfs:label "Flickr APIs Terms of Use";
 rdfs:seeAlso
  <https://www.flickr.com/services/api/tos/>;
 odrl:prohibition [
    odrl:target ex:EventMedia ;
    odrl:action odrl:sell ];
 odrl:prohibition [
    odrl:target ex:EventMedia ;
    odrl:actionodrl:license ];
 odrl:prohibition [
    odrl:target ex:EventMedia ;
    odrl:actioncc:CommercialUse ];
 odrl:permission [
    a odrl:Permission;
    odrl:target ex:EventMedia ;
    odrl:action odrl:use;
    odrl:duty odrl:attribute ]
  .
```

Figure 2 illustrates the EventMedia data flow and Listing 2 the equivalent RDF description. Data are processed from event directories and enriched with additional information and media from sources like DBpedia[18], Flickr[19] or Foursquare[20]. In the figure, circles are data objects and arcs are Datanode relations. We will follow the path that connects the `ex:output` data object to two of the input data objects, namely `ex:Flickr` - that represents the Flickr API[21] (this path is highlighted in the figure), and Eventful[22] - a portal to search for upcoming events and related tickets. Apart from using the LODE ontology, `ex:output` is remodelled from an aggregation of various sources, named as `ex:collection`. The population (entities) of `ex:collection` includes `ex:events`, a `dn:combinationFrom ex:Eventful` with other sources (central path in the figure). Moreover, `ex:collection` includes descriptions of media from `ex:Flickr`, expressed by the path `dn:hasPortion / dn:isCopyOf / dn:isSelectionOf`. The data selected from `ex:Flickr` also refer to (some of) the entities aggregated in `ex:events`. This is expressed by the path `ex:descriptionsFromFlickr dn:samePopulation / dn:isPortionOf ex:events`. Therefore, the data flow is a backtrace of the abstract process of the EventMedia system, from the `ex:output` data object towards the input data sources.

Listing 2: The EventMedia data flow in RDF.

```
ex:events
    dn:combinationFrom :Eventful ,
```

[15] See `http://eventmedia.eurecom.fr/`.

[16] The Technical Report was produced in 2014, when the EventMedia dataset description article was firstly submitted to the Semantic Web Journal. The description produced refers to the submitted version, which could be changed in the published version.

[17] Flickr API Terms of Use: `https://www.flickr.com/services/api/tos/`.

[18] DBpedia: `http://dbpedia.org`.

[19] Flickr: `http://www.flickr.com`.

[20] This description has been initially elaborated in [8].

[21] Flickr API: `https://www.flickr.com/services/api/`.

[22] Eventful: `http://eventful.com/`

```
      :LastFM , :Upcoming .

ex:collection
    dn:hasPortion [
        dn:isCopyOf ex:descriptionsFromFlickr
    ], [
        dn:isCopyOf ex:descriptionsFromDbpedia
    ], [
        dn:isCopyOf ex:descriptionsFromMusicbrain
    ], [
        dn:isCopyOf ex:descriptionsFromFoursquare
    ], ex:events .

ex:descriptionsFromDbpedia
    dn:isSelectionOf ex:Dbpedia ;
    dn:samePopulation [
        dn:isPortionOf ex:events
    ] .

ex:descriptionsFromFlickr
    dn:isSelectionOf ex:Flickr ;
    dn:samePopulation [
        dn:isPortionOf ex:events
    ] .

ex:descriptionsFromFoursquare
    dn:isSelectionOf ex:Foursquare ;
    dn:samePopulation [
        dn:isPortionOf ex:events
    ] .

ex:descriptionsFromMusicbrain
    dn:isSelectionOf ex:Musicbrain ;
    dn:samePopulation [
        dn:isPortionOf ex:events
    ] .

:output
    dn:isRemodelledFrom ex:collection ;
    dn:usesSchema ex:TheLODEOntology .
```

Table 1

Sources of Terms and conditions associated with the data sources exploited by EventMedia.

Source	T&C
Flickr	Flickr APIs Terms of Use[23]
Dbpedia	Creative Commons CC-BY-SA 3.0
Eventful	Eventful API Terms of Use[24]
LastFM	LastFM Terms of Service[25]
Upcoming	?
Musicbrain	Creative Commons CC0
Foursquare	Foursquare Developers Policies[26]

[23]Flickr APIs Terms of Use. `https://www.flickr.com/services/api/tos/`

The data flow described so far can be exploited by a reasoner in conjunction with the ODRL policies of the inputs and the PPRs, to infer the policies associated with `ex:output`, as the ones in Listing 3.

Listing 3: Example of policies associated with the output of EventMedia.

```
ex:outputPset a odrl:Policy ;
  odrl:prohibition [
    odrl:target ex:output ;
    odrl:action odrl:modify ];
  odrl:prohibition [
    odrl:target ex:output ;
    odrl:action cc:commercialUse ];
  odrl:permission [
    a odrl:Permission;
    odrl:target ex:output ;
    odrl:action odrl:use;
    odrl:duty odrl:attribute ];
  odrl:prohibition [
    odrl:target ex:output ;
    odrl:action odrl:sell ]
  .
```

In [9] we considered the set of relations defined by Datanode and the policies defined in the RDF License Database to generate a database of 3865 propagation rules. With the goal of improving the management of the rules, we studied to what extent it is possible to reduce the number of rules without loss of information. Such an abstraction requires to be complemented by inferences produced by a reasoner. In the present work, we study whether this reasoning is practically feasible, and make the hypothesis that compressing the size of the rule base will not negatively impact the efficiency of the reasoner in computing the propagated policies.

4. (A)AAAA Methodology: overview and *coherency check*

Firstly introduced in [9], the (A)AAAA methodology covers all the phases necessary to set up a compact knowledge base of PPRs[27]. With respect to the

[24]Eventful API Terms of Use. `http://api.eventful.com/terms`

[25]LastFM Terms of Service. `http://www.last.fm/api/tos`

[26]Foursquare Developers Policies. `https://developer.foursquare.com/overview/community`

[27]In out work, it has been applied with the support og the command line tool PPR-A-FIVE: `https://github.com/enridaga/ppr-a-five`.

methodology already presented, the novel contribution of this article is the introduction of a *coherency check* method in the *Assessment* phase. We summarize the methodology, focusing on the *coherency check* element, and refer the reader to [9] for a general overview.

The methodology is based on two assumptions: 1) Policy Propagation Rules are associations between policies and data flow steps, and 2) an ontology is available to organise data flow steps in a semantic hierarchy, e.g., for expressing the fact that the relation *is a copy of* is a sub-relation of *is a derivation of*[28].

The methodology is composed of the following phases:

A1 - Acquisition.

The initial task is to set up a knowledge base of PPRs. We used the Datanode ontology to extract a list of 115 possible relations between data objects, and combined with 113 policies derived from the ones defined in the RDF License Database. The combination of relations and policies lead to a matrix of 12995 cells. This phase required a manual supervision of all associations between policies and relations in order to establish the initial set of propagation rules. This was performed with the support of the Contento tool [7].

A2 - 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, providing as input the binary matrix representation of the rule base R consisting of PPRs. 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*). In our case, each concept maps a group of relations propagating a group of policies. These concepts are organized hierarchically in a *lattice*, ordered from the top concept T, which includes all the objects and potentially no attribute, 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, usually 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 case, the top concept T would include all relations and no policy, while the bottom concept B includes all the policies but no relation. The concepts identified by FCA group relations that have a common behavior in our rule base R, as they propagate the same policies. The output of the process is an ordered *lattice* of concepts: clusters of policies that propagate with the same set of relations.

A3 - Abstraction.

In this phase, we apply a method for subtracting rules in order to reduce the size of the rule base. The abstraction process is based on applying an ontology that organizes the relations in a hierarchy (the Datanode ontology). For instance, the relation $hasCopy$ is a sub-relation of $hasDerivation$. Intuitively, a number of policies propagated by $hasDerivation$ should be also propagated by $hasCopy$ and by all the other sub-relations in that branch of the hierarchy. By grouping all the relations below $hasDerivation$ in a transitive closure, we obtain a group of relations similar to the ones in the FCA concepts, that we call the $hasDerivation$ *branch.*

Since we expect branches of the ontology to be reflected in the clusters of relations obtained by FCA, we therefore search for matches between the ontology and the FCA lattice. When a match occurs, we subtract the rules that can be abstracted.

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. Concrete examples of the application of this phase can be found in [9].

A4 - Assessment.

The objective of this phase is to assess to what extent the ontology and the FCA lattice are coherent. In particular, we want to:

1. detect mismatches (*coherency check*) to be resolved before using the compressed rule base with a reasoner, and
2. identify *quasi matches* that could be boosted to become a full match performing changes in the rule base or the ontology

[28] In our work we rely on Datanode as reference ontology, even if this is not required by the methodology itself.

Coherency check. The abstraction process is based on the assumption that it is possible to replace asserted rules with inferences implied by subsumed relations in the ontology. This implies that all policies propagated by a given relation must be propagated by all sub-relations in the original (uncompressed) rule base. A coherency check process is necessary to identify whether this assumption does hold for all the relations in the extent. In case it does not, we want to collect and report all the mismatches in order to be able to fix them at a later stage in the methodology. Listing 4 shows the algorithm used to detect such problems on a given concept in the lattice.

Listing 4: Coherency check algorithm.

```
c = // a concept
M = []
S=SuperConcepts(c)
ForEach s in S
  E=Extent(c)
  E1=Extent(s)
  ForEach(e1 in E1)
    if not(Contains(E,e1))
      ForEach e in E
        if Contains(Branch(e),e1)
          M = [e,e1]|P // Mismatch detected
return M
```

We know from the definition of a FCA lattice that superconcepts will include a larger set of relations propagating a smaller number of policies. Given a concept c, the algorithm extracts the relations (extent) of each of any superconcept (S denotes the set of all super concepts s of c). In case these relations are not present in (the extent of) c, it is mandatory for them not to be sub-relations of any relation in the extent of c. In case they are, this means that a sub-relation is not inheriting all the policies of the parent one, thus invalidating our assumption. Mismatches M are identified and reported. Listing 5 shows the result of the algorithm for a concept. In this example, a number of sub-relations of `dn:isVocabularyOf` do not propagate some of the policies of Concept 71.

Listing 5: Coherency check result for Concept 71: mismatches.

Concept	Branch	Relation
71	dn:isVocabularyOf	dn:attributesOf
71	dn:isVocabularyOf	dn:datatypesOf
71	dn:isVocabularyOf	dn:descriptorsOf
71	dn:hasVocabulary	dn:hasDatatypes
71	dn:hasVocabulary	dn:hasDescriptors
71	dn:hasVocabulary	dn:hasRelations
71	dn:isChangeOf	dn:isAdditionOf
71	dn:isChangeOf	dn:isDeletionOf
71	dn:isVocabularyOf	dn:relationsOf
71	dn:isVocabularyOf	dn:typesOf

Quasi matches. The result of the *Abstraction* phase includes a set of measures between concepts and portions of the ontology. Table 2 shows an example of the measures obtained by applying the approach in the context of the Datanode ontology. The measures defined in the *Abstraction* phase are now considered to quantify and qualify the way the ontology aligns with the propagation rules: precision and recall indicate how close a relation is to being a suitable abstraction for policy propagation.

Table 2

Excerpt from the table of measures computed during the abstraction phase.

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

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

Some general considerations can be made by inspecting these measures. When $Rec = 1$, the whole extent of the concept is in the branch. The branch might also include other relations, which do not propagate the policies included in the concept. When $Pre = 1$, we can perform the subtraction of rules. 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 2 relates to a relation with $BS - IS = 2$, so we only need 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.

At this stage we can make the following considerations:

- The presence of mismatches between the lattice and the ontology will cause the reasoner to return wrong results. They must, therefore, be eliminated.
- The size of the matrix that was manually prepared in the *Acquisition* phase is large (13k cells), and even with the support of the Contento tool it is still possible that errors or misjudgments are made at that stage of the process.
- The Datanode ontology was not designed for the purpose of representing a common behavior of relations in terms of propagation of policies. It should be possible to refine the ontology in order to make it cover the current use case in a better way (and to further reduce the number of rules).

A5 - Adjustment

In this phase we perform operations that change the rule base or the ontology in order to repair mismatches, correct inaccuracies, evolve the ontology, and improve the compression factor as a consequence. Six operations can be performed: *Fill*, *Wedge*, *Merge*, *Group*, *Remove*, *Add* (details about each operation can be found in [9]).

The *Assessment* phase of the methodology reported possible mismatches between the FCA output and the ontology hierarchy. These errors must be repaired if we want the compressed rule base to be used by a reasoner. For example, Listing 5 shows the set of mismatches detected for concept 71. In this list, the `dn:isVocabularyOf` branch contains a number of relations that do not propagate the related policies, breaking the assumption that all the policies of `dn:isVocabularyOf` are also propagated by all the other relations in his branch. With a *Fill* operation, we can add all the necessary rules to remove this mismatch.

After each operation, we run our process again from the *Analysis* phase to the *Assessment*, in order to evaluate whether the change fixed the mismatch and/or how much the change affected the compression factor. The process is repeated until all mismatches have been fixed, and there are no other quasi matches that can be adjusted to become full matches. Moreover, when new policies are defined in the database of licenses (the RDF License Database, in our work), the process has to be repeated in order to insert the new propagation rules. However, this is only required after changes in the licenses, as changes in the associations between policies and data objects do not affect the PPRs, e.g., changing the license of a data source or adding new data flows.

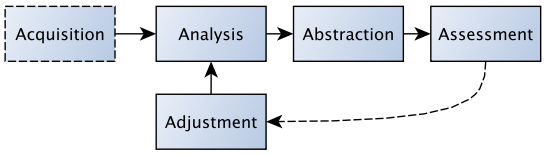


Fig. 3. (A)AAAA Methodology.

5. Experiments

The methodology described in the previous section allows to reduce the number of rules that need to be stored and managed. The results of applying this methodology on the PPR base derived from the RDF Licenses Database have already been described in [9], showing how the compression factor can be dramatically increased after several iterations. Our assumption in this work is not only that a reduced set of PPRs will be more manageable, but also that it might positively affect our ability to apply reasoning for policy propagation. Here, we therefore assess through realistic cases the performance of reasoners when dealing with a compressed set of PPRs, as compared to when dealing with the full set.

We took 15 data flow descriptions from previous work [8], referring to 5 applications that rely on data obtained from the Web. Each data flow represents a data manipulation process, consuming a data source (sometimes multiple sources), and returning an output data object. Given a set of policies P associated with the input data, the objective of a reasoner is to find the policies P^i associated with the output of the data flow.

The experiments have the objective to compare the performance of a reasoner when using an *Uncompressed* or a *Compressed* rule base respectively. Therefore, each reasoning task is performed twice: at first time, to provide the full set of propagation rules; the second time, to provide the compressed rule base in conjunction with the hierarchy of relations (required to produce the inferences).

Table 3
Data flows used in the experiments.

Data flow	has policy	has relation	relations	data objects	policies	sources	output policies
AEMOO-1	6	18	13	10	6	1	6
DBREC-1	6	2	2	2	6	1	3
DBREC-2	6	8	7	5	6	1	6
DBREC-3	6	12	7	8	6	1	6
DBREC-4	6	14	8	9	6	1	3
DBREC-5	6	14	10	10	6	1	6
DBREC-6	6	10	10	6	6	1	3
DBREC-7	6	9	6	10	6	1	6
DBREC-8	6	5	4	5	6	1	6
DISCOU-1	7	22	10	14	7	1	5
DISCOU-11	5	13	9	12	5	1	0
EventMedia-1	37	25	8	24	25	6	4
REXPLORE-1	16	14	8	14	8	3	3
REXPLORE-2	32	23	4	18	14	6	6
REXPLORE-4	32	18	8	14	15	6	3

Highlighted are the maximum and minimum values for each of the data flow inputs. In one case (DISCOU-11), none of the policies attached to the source are propagated to the output.

Reasoners infer logical consequences from a set of asserted facts and inference rules (knowledge base). A reasoner can compute the possible inferences from the rules and the facts any time it is queried, thus exploring the inferences required to provide the complete answer. Alternatively, a reasoner can compute all possible inferences at the time the knowledge base is loaded, and only explore the materialized facts at query time. To take into account these two alternative reasoning strategies, we run the experiments with two different reasoners. The first reasoner performs the inference at query time using a backward chaining approach; is implemented as Prolog program and we will refer to it as the *Prolog* reasoner. The second reasoner computes all the inferences at loading time (materialisation); is implemented as an RDFS reasoner in conjunction with SPIN rules, and we will refer to it as the *SPIN* reasoner. Both reasoners are implemented in Java within the PPR Reasoner project[29]. Both reasoners have the capability of executing PPRs and expand the results according to the ontology hierarchy.

The *Prolog* implementation is a program relying on JLog, a Prolog interpreter written in Java[30]. The program incorporates a *meta rule* that traverses the set of PPRs, encoded as facts. At the same time, it supports the subsumption between relations. Listing 6 shows an excerpt of the program.

Listing 6: Excerpt of the Prolog reasoner program.

```
i_rdfs_sub_property_of(X,X) .
i_rdfs_sub_property_of(X,Y) :-
        rdfs_sub_property_of(X,Z),
        i_rdfs_sub_property_of(Z,Y) .
i_propagates(X,Y) :- propagates(X,Y) .
i_propagates(X,Y) :-
        i_rdfs_sub_property_of(X,Z),
        propagates(Z,Y) .
i_has_policy(T,P,_) :- has_policy(T,P) .
i_has_policy(T,P,L) :-
        i_has_relation(S,T,R),
        not(visited(S,L)),
        i_propagates(R,P),
        i_has_policy(S,P,[S|L]) .
i_has_policy(T,P) :- i_has_policy(T,P,[]) .
```

The *SPIN* reasoner is built upon the RDFS reasoner of Apache Jena[31] in combination with SPIN[32], a rule engine that allows to define rules using SPARQL. The core part of the reasoner executes PPRs as a SPARQL meta query (Listing 7).

[29] PPR Reasoner: https://github.com/enridaga/pprreasoner. The experiments were performed within the *ppr-evaluation* module, that includes instructions about how to reproduce them.

[30] http://jlogic.sourceforge.net/

[31] http://jena.apache.org/

[32] http://spinrdf.org/

Listing 7: Construct meta-query of the *Optimied* SPIN based reasoner.

```
CONSTRUCT {
  ?this ppr:policy ?policy
} WHERE {
  ?int ?relatedWith ?this .
  ?int ppr:policy ?policy .
  ?relatedWith ppr:propagates ?policy
}
```

We performed the experiments with the data flows listed in Table 3. Each data flow describes a process executed within one of the 5 systems selected as exemplary data-oriented applications. These data flows were formalised before the present work (in [8]), and were reused for the experiments without changes. However, the information about the policies of the input was added. The table illustrates the properties of these data flows, and compares them along several dimensions. The *has policy* column reports the number of statements about policies, from a minimum of 5 to 37 policies. The size of the data flow is reported in the *has relation* column of the table, as it is measured in number of Datanode relations used, spanning from 2 to the maximum of 25. The *relations* column reports the number of distinct relations, the same applying to *data objects*, *policies*, *sources* and the propagated *output policies*. Highlighted are the maximum and minimum values for each of the dimensions. In one case (DISCOU-11), none of the policies attached to the source are propagated to the output.

Each experiment takes the following arguments:

- *Input*: a data flow description
- *Compression*: $True/False$
- *Output*: the output resource to be queried for policies

In case *compression* is $False$, we provide the complete set of PPRs as input of the reasoning process without including information on subsumption between the relations described in the dataflow. Conversely, when *compression* is set to $True$, the compressed set of PPRs is used in conjunction with the Datanode ontology. It is worth noting that the (A)AAAA methodology is also an ontology evolution method, as most of the operations targeted to improve the compression of the rule base are performed on the ontology by adding, removing and replacing relations in the hierarchy. In these experiments, we are considering the evolved rule base (and ontology), that has been harmonised by fixing mismatches between the rule set and the ontology.

The experiments were executed on a MacBook Pro with an Intel Core i7/3 GHz Dual Core processor and 16 GB of RAM. In case a process was not completed within five minutes, it was interrupted. Each process was monitored and information about CPU usage and RAM (RSS memory) was registered at intervals of half a second. We compared the results of the experiments with and without compression, and we always obtained the same result set. When terminating, the experiment output would include: total time (t), resources load time (l), setup time (s), and query time (q). The size of the input for each experiment is reported in the diagrams in Figure 4.

We consider performance on two main dimensions: time and space.

Time performance is measured under the following dimensions:

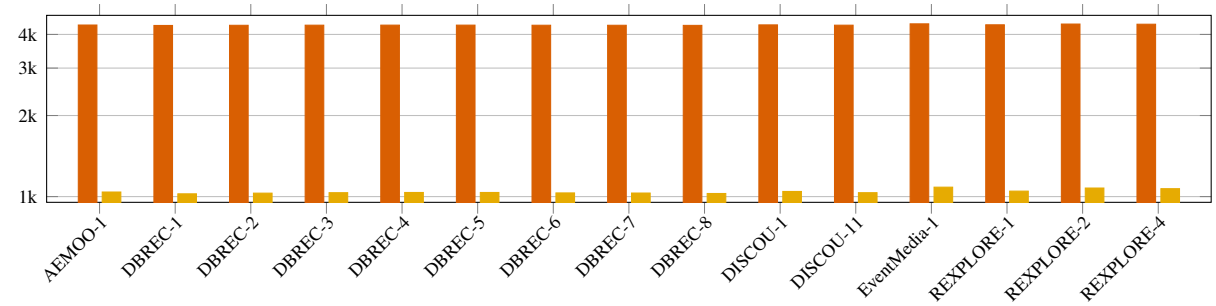
- L Resources load time.
- S Setup time. It includes L, in addition to any other operation performed before being ready to receive queries (e.g., materialisation).
- Q Query time.
- T Total duration: $T = S + Q$.

Space is measured as follows:

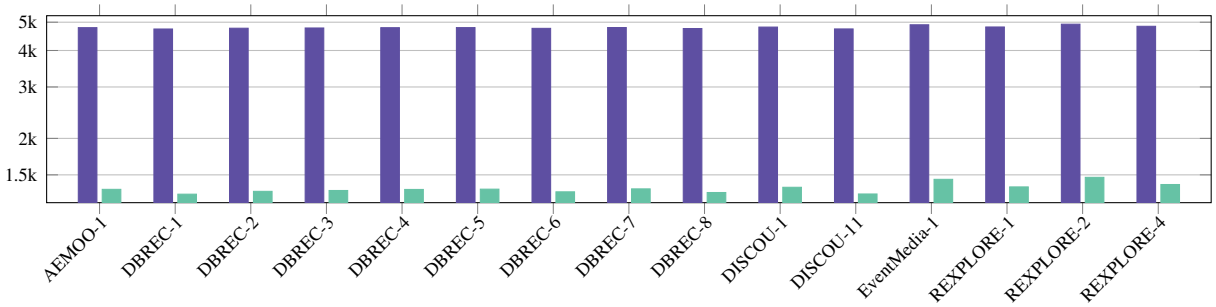
- Pa Average *CPU* usage.
- M Maximum memory required by the process

Each experiment was executed 20 times. In the present report, we show the average of the measures obtained in the different executions. In order to evaluate the accuracy of the computed average measure from the twenty executions of the same experiment, we calculated the related *Coefficient of Variation* (CV)[33]. CV is a measure of spread that indicates the amount of variability relative to the mean. A high CV indicates a large difference in the observed measures, thus reducing the significance of the computed mean. Diagrams 5a and 5b display the CV of all the measures for the *Prolog* and *SPIN* reasoner, respectively. In almost all the cases the CV for the *Prolog* reasoner was below 0.1, with the exception of memory usage M, that in many cases showed a fluctuation between 0.2 and 0.4. Experiments with the *SPIN* reasoner reported a much more stable behaviour in terms of consumed resources, the CV being assessed below 0.1 in almost all the cases, except the Query time of some experiments

[33] Coefficient of Variation, also known as Relative Standard Deviation (RSD). https://en.wikipedia.org/wiki/Coefficient_of_variation



(a) *Prolog* reasoner: input size computed as number of Prolog facts with the original (dark orange) and compressed (light yellow) input for each data flow.



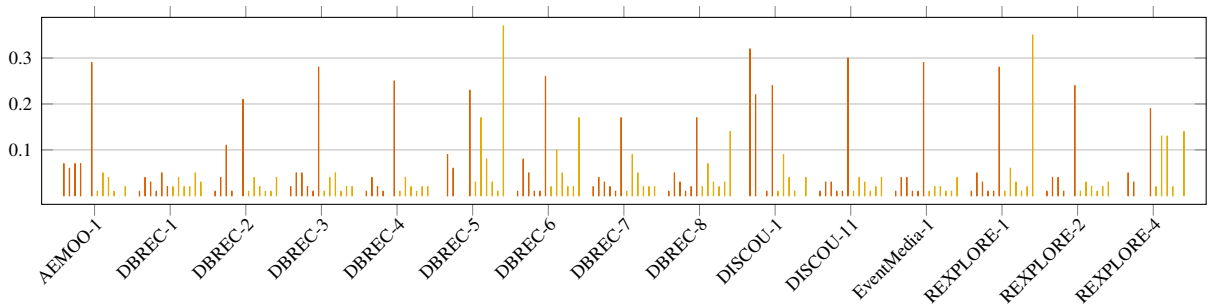
(b) *SPIN* reasoner: input size computed as number of RDF triples with the original (dark purple) and compressed (light green) input for each data flow.

Fig. 4. Input size for the *Prolog* (4a) and *SPIN* (4b) reasoners. It can be deduced that the size of the data flow has a small impact on the general size of the input.

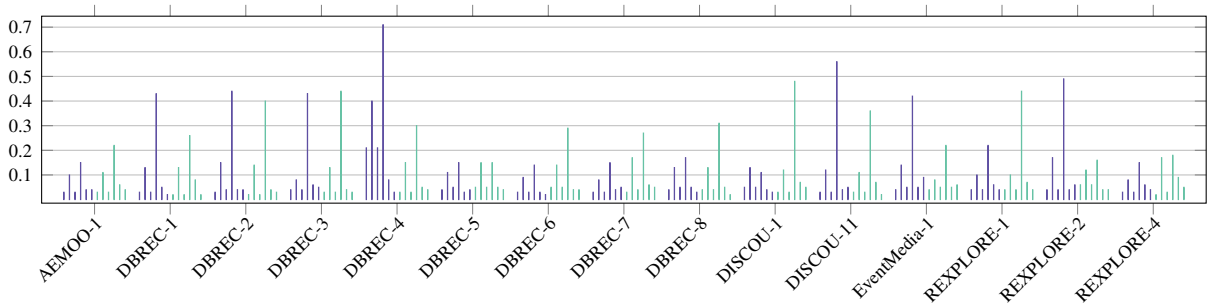
(the peak is on DBREC-4). However, Q with the *SPIN* reasoner were fluctuating around an average of 10ms, making the observed variation irrelevant. Finally, we consider the computed mean of the observed measures in these experiments to be significant.

Before discussing the results, it is worth reminding the reader that this evaluation is not targeted to compare the two implementations of a PPR reasoner, but to observe the impact of our compression strategy on the *Prolog* and the *SPIN* implementations, assuming that any other implementation is likely to make use of a combination of the two reasoning strategies they respectively implement.

Figures 6 and 7 illustrate the results of the experiments performed with the *Prolog* and the *SPIN* reasoner, respectively. For each data flow, the bar on the left displays the time with an *Uncompressed* input, and the one on the right the time with a *Compressed* input. We will follow this convention in all the other diagrams as well. Figure 6c displays a comparison of the total time between an *Uncompressed* and *Compressed* input with the *Prolog* reasoner. In all cases, there has been a significant increase in performance: in three cases (DBREC-5, DISCOU-1, REXPLORE-4) the *Uncompressed* version of the experiment could not complete within the five minutes, while the *Compressed* version returned results in less then a minute. The total time of the experiments with the *SPIN* reasoner (Figure 7c) is much smaller (fractions of a second), having the maximum total time of approximately 2 seconds (EventMedia-1). However, in this case too, we report an increase in every case in performance for all the data flows, with some cases performing much better than others (DBREC-3, DBREC-4). The total time T of the experiment can be broken up into setup time S (including load time L) and query time Q. This observation is depicted in Figures 6a and 7a, and in both cases the impact of the rule reduction process is evident. An interesting difference between the two implementations can be seen by comparing Figures 6b and 7b. The cost of the query time in the *Prolog* reasoner is very large compared to the related setup time S. The *SPIN* reasoner, conversely, showed a larger setup time S with a very low cost on query time Q. The reason is that the second materialises all the inferences at setup time, before query execution. This accounts for the lack of difference in query time between



(a) *Prolog* reasoner.



(b) *SPIN* reasoner.

Fig. 5. Coefficient of Variation (CV) of the observed measures. Each experiment was run 20 times. In the diagram we see the CV of measures observed. For each data flow, the diagram shows twelve bars. The first six bars refer to the uncompressed rule base, the second six to the compressed rule base. Each group shows the CV for, in order: T, L, S, Q, Pa and M.

the *Uncompressed* and *Compressed* version of the experiments.

We did not observe changes in Pa for the *Prolog* reasoner (Figure 6d), while the differences in memory consumption M is significant (Figure 6e), demonstrating a performance improvement caused by the compressed input. An increase in space consumption has been also observed using the *SPIN* reasoner (Figures 6d and 6e), even if smaller, and negative in only 2 cases with regard to memory consumption M (DBREC-1 and DBREC-6).

A summary of the impact of the compression on the different measures is depicted in Figures 8 and 9. The diagram shows how much a given measure is reduced by compressing the input. The first bar on the left of both diagrams illustrates the reduction of the size of the *Input*, while the others how much each measure is reduced. A serious improvement has been achieved in the case of the *Prolog* reasoner, implementing a backward chaining algorithm executed at query time. A PPR reasoner could also be implemented to perform inferencing at loading time (materialisation). The experiments with the *SPIN* implementation is therefore used to show that the effect on reasoning performance exists in both cases, even if in different ways depending on the approach to inferencing. The main conclusion from our experiments is therefore that the methodology presented in [9] and extended with *coherency check* leads to compressed PPR bases that are not only more manageable for the knowledge engineers maintaining them, but also improve our ability to apply reasoning for the purpose of policy propagation. In addition, it appears clearly that, when dealing with a compressed PPR base, an approach based on materialisation of inferences at load time is preferable to one based on computing the inferences at query time.

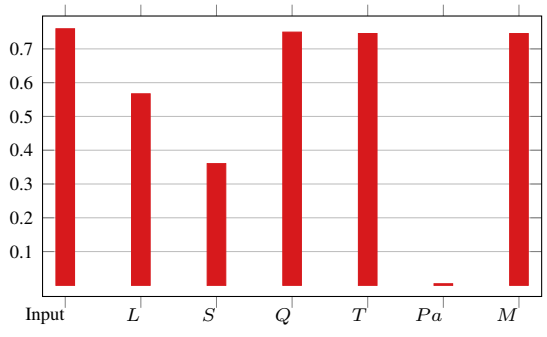
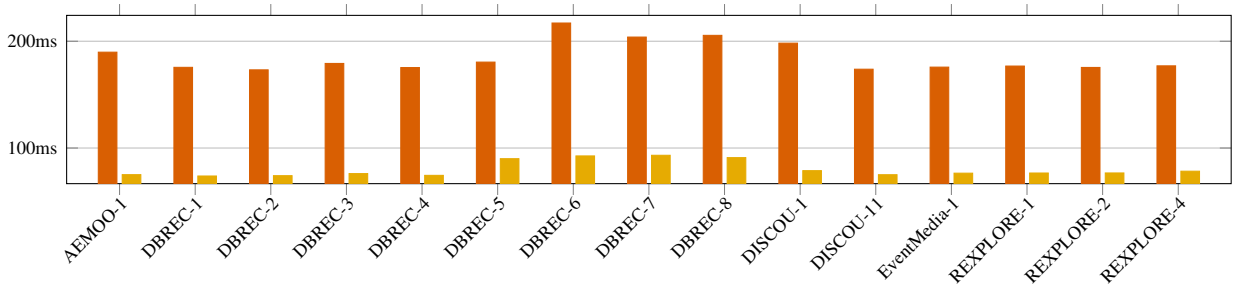
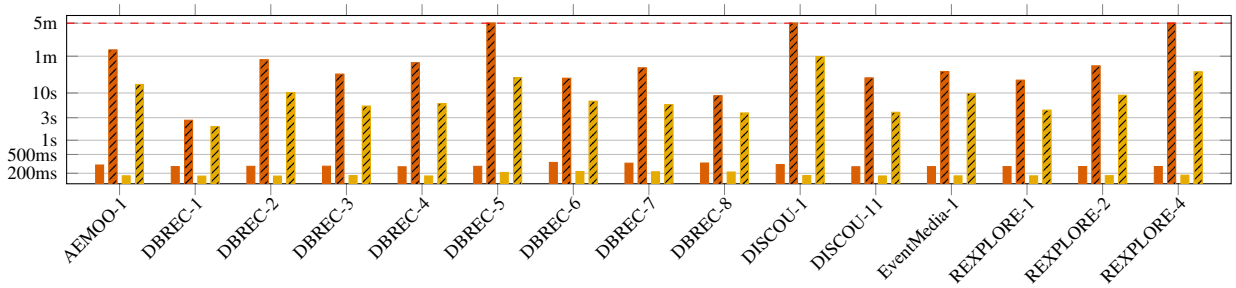


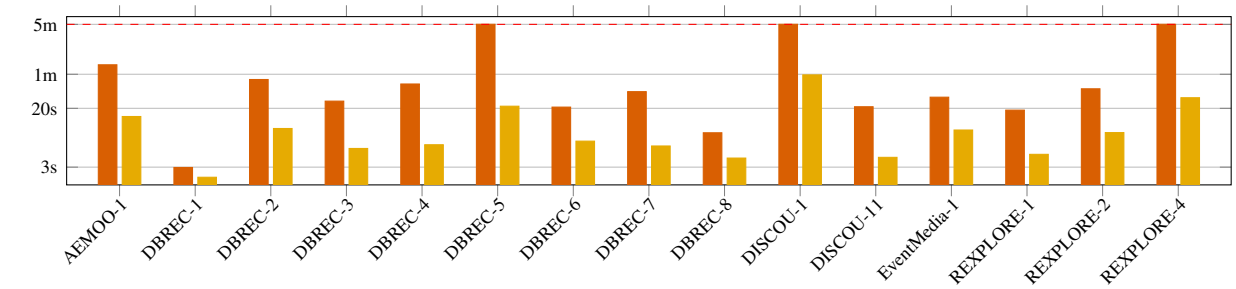
Fig. 8. *Prolog* reasoner: boost analysis.



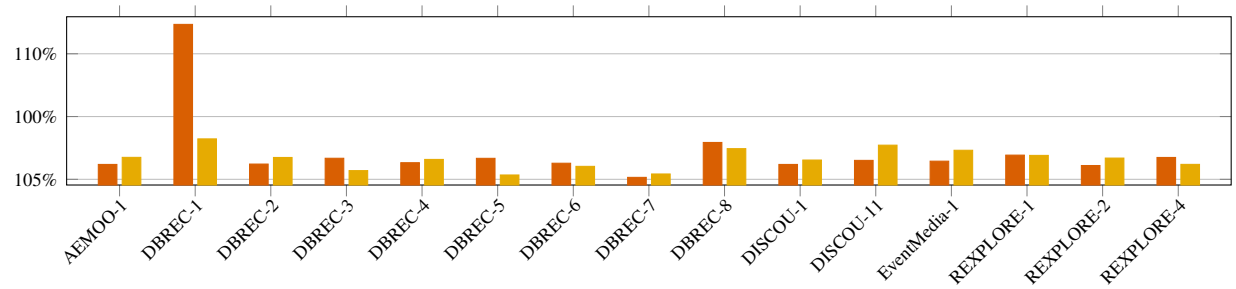
(a) *Prolog* reasoner: load time L.



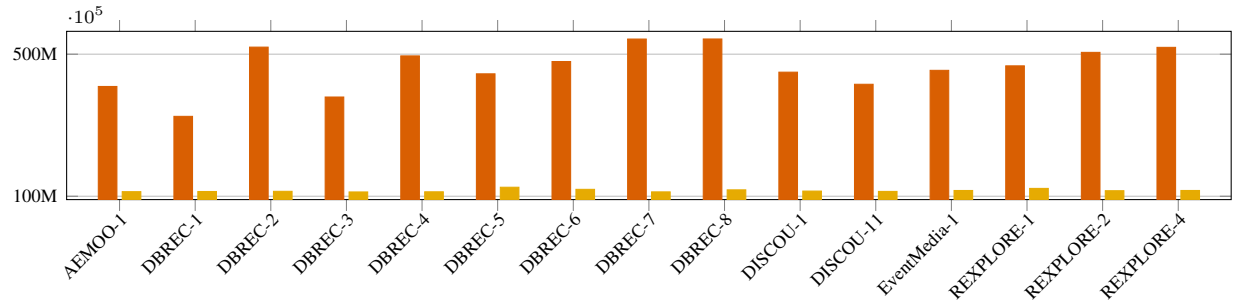
(b) *Prolog* reasoner: comparison of setup time S and query time Q (bars with pattern).



(c) *Prolog* reasoner: total time T.

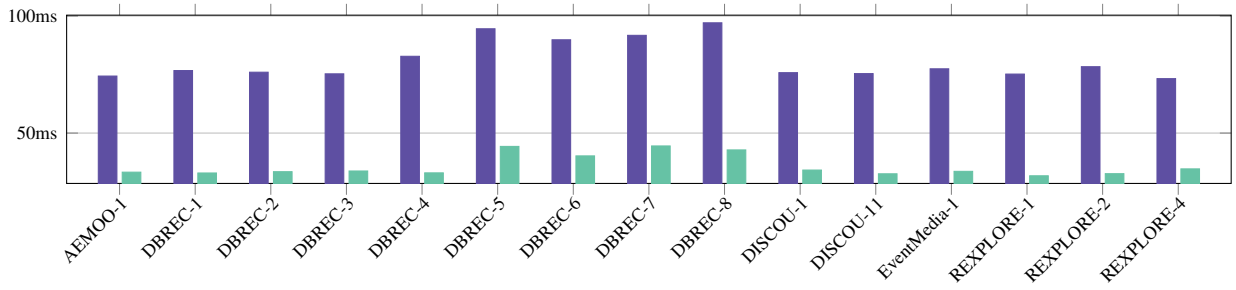


(d) *Prolog* reasoner: average CPU Pa.

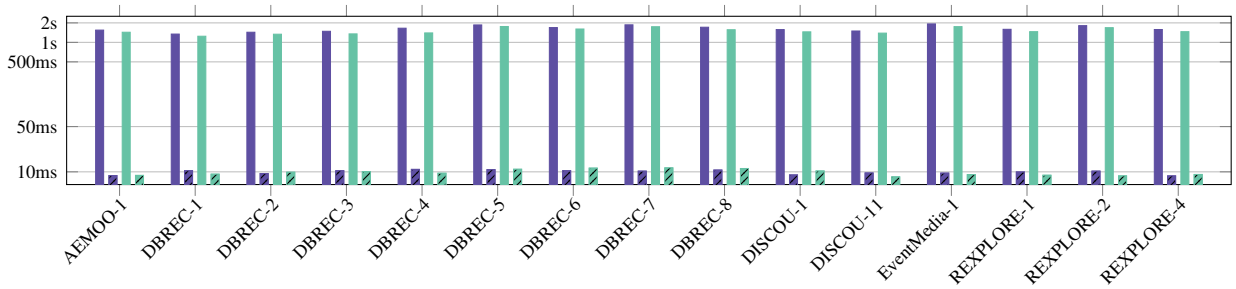


(e) *Prolog* reasoner: max memory consumption M.

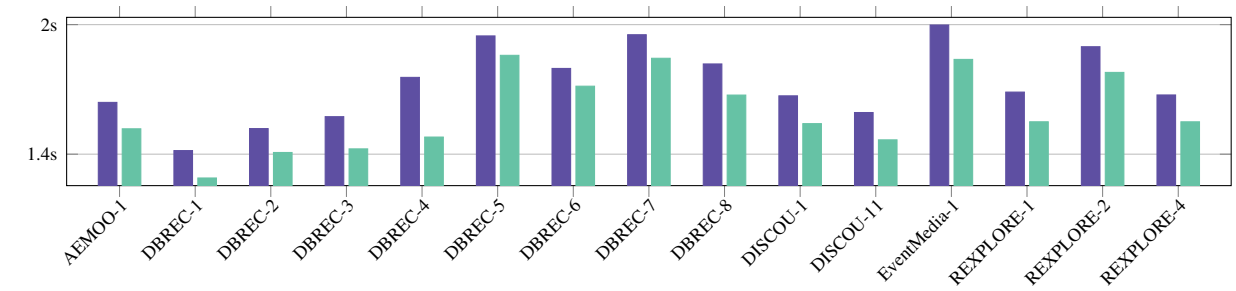
Fig. 6. *Prolog* reasoner: performance measures. Each diagram reports on the performance of this reasoner with an *Uncompressed* or *Compressed* rule base with respect to a given measure. The bars on the left (in dark orange) refer to the *Uncompressed* rule base, while the bars on the right (in light yellow) the *Compressed* one.



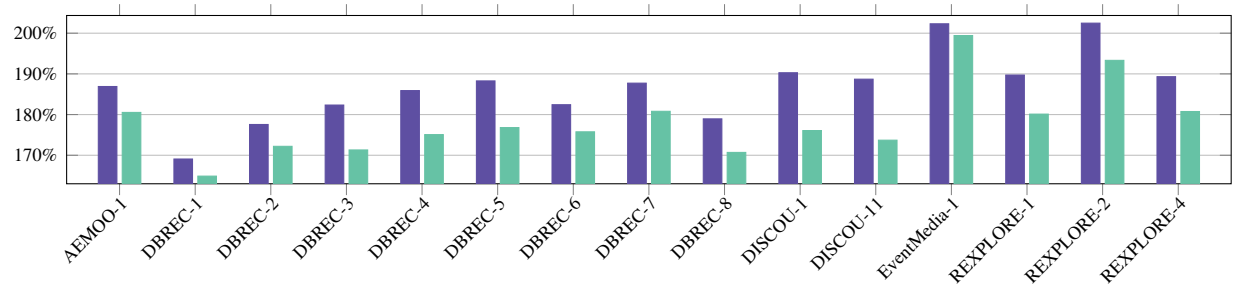
(a) *SPIN* reasoner: load time L.



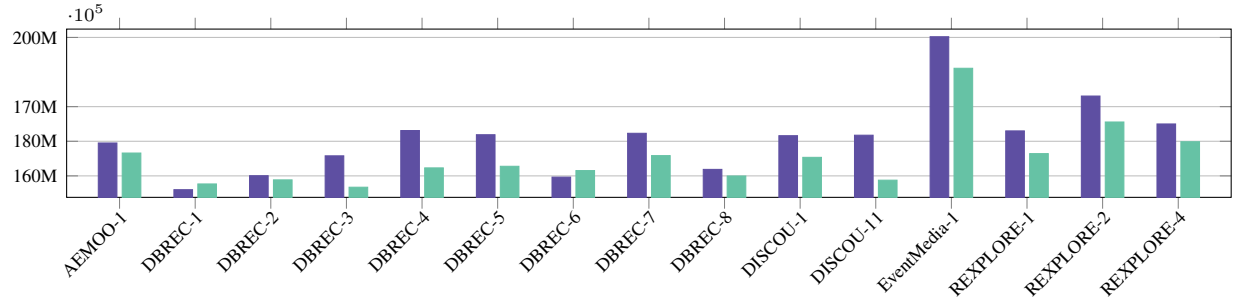
(b) *SPIN* reasoner: comparison of setup time S and query time Q (bars with pattern).



(c) *SPIN* reasoner: total time T.



(d) *SPIN* reasoner: average CPU Pa.



(e) *SPIN* reasoner: max memory consumption M.

Fig. 7. *SPIN* reasoner: performance measures. Each diagram reports on the performance of this reasoner with an *Uncompressed* or *Compressed* rule base with respect to a given measure. The bars on the left (in dark purple) refer to the *Uncompressed* rule base, while the bars on the right (in light green) the *Compressed* one.

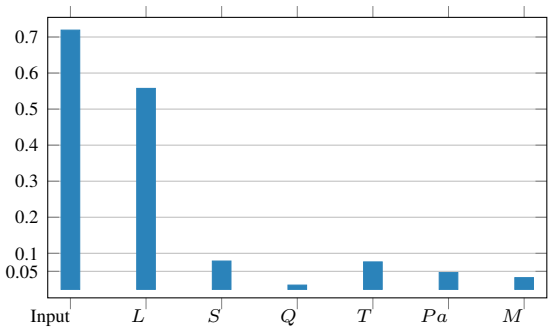


Fig. 9. *SPIN* reasoner: boost analysis.

6. Conclusions

In this article, we presented an approach for reasoning on the propagation of policies in a data flow. This method is grounded on a rule base of Policy Propagation Rules (PPRs). Rules can easily grow in number, depending on the size of the possible policies and the one of the possible operations performed in a data flow. The (A)AAAA methodology can be used to reduce this size significantly, as demonstrated in [9], by relying on the inference properties of the Datanode ontology, applied to describe the possible relations between data objects. We presented an evolved version of the methodology, which was required to be sure the inferred policies were correct when using the compressed rule base. However, while this activity reduces the size of the input of the reasoner, it requires more inferences to be computed. Therefore, we performed experiments to assess the impact of the compression on reasoning performance. The present article provides two major comntributions:

- the (A)AAAA methodology has been extended by including a *coherency check* algorithm, and
- experimental results demonstrating that a compressed knowledge base makes the reasoning on policy propagation more efficient.

This is a preliminary step on studying compression in knowledge management and its impact on reasoning in a more general point of view. Reasoning on policy propagation requires a formalisation of the data flow, and producing such representation can be time consuming. It would be of interest to explore methods for supporting and automating the generation of such data flows from pre-existing artefacts (e.g., code bases and their documentation). Future work includes defining new measures to describe the complexity of a data flow and how it affects reasoning on propagating policies, as well as studying the validation of data flows with respect to policies, particularly when multiple sources are used. Finally, we are currently setting up an experimental evaluation in the environment of the MK:Smart Data Hub [10].

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] S. Bischof, C. Martin, A. Polleres, and P. Schneider. Collecting, integrating, enriching and republishing open city data as linked data. In *International Semantic Web Conference*, pages 57–75. Springer, 2015.

[3] J.-M. Bohli, A. Skarmeta, M. Victoria Moreno, D. Garcia, and P. Langendorfer. Smartie project: Secure iot data management for smart cities. In *Recent Advances in Internet of Things (RIoT), 2015 International Conference on*, pages 1–6. IEEE, 2015.

[4] 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.

[5] E. Boros, O. Čepek, and P. Kučera. A decomposition method for cnf minimality proofs. *Theoretical Computer Science*, 510:111–126, 2013.

[6] E. Daga, M. d'Aquin, A. Adamou, and E. Motta. Addressing exploitability of Smart City data. In *Smart Cities Conference (ISC2), 2016 IEEE Second International*, page (Accepted). IEEE, 2016.

[7] 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.

[8] 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.

[9] E. Daga, M. d'Aquin, A. Gangemi, and E. Motta. Propagation of policies in rich data flows. In *Proceedings of the 8th International Conference on Knowledge Capture*, K-CAP 2015, pages 5:1–5:8, New York, NY, USA, 2015. ACM.

[10] 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.

[11] 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.

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

[13] G. Governatori, H.-P. Lam, A. Rotolo, S. Villata, G. Atemezing, and F. Gandon. Checking licenses compatibility between vocabularies and data. In *Proceedings of the Fifth International Workshop on Consuming Linked Data (COLD2014)*, 2014.

[14] G. Governatori, A. Rotolo, S. Villata, and F. Gandon. One license to compose them all. In *The Semantic Web–ISWC 2013*, pages 151–166. Springer, 2013.

[15] P. L. Hammer and A. Kogan. Optimal compression of propositional horn knowledge bases: complexity and approximation. *Artificial Intelligence*, 64(1):131–145, 1993.

[16] R. Iannella, S. Guth, D. Pähler, and A. Kasten. ODRL: Open Digital Rights Language 2.1. Technical report, W3C Community Group, 2015.

[17] S. Javanmardi, M. Amini, R. Jalili, and Y. GanjiSaffar. Sbac: A semantic based access control model. In *11th Nordic Workshop on Secure IT-systems (NordSec'06), Linkping, Sweden*, volume 22, 2006.

[18] R. Kawase, M. Fisichella, K. Niemann, V. Pitsilis, A. Vidalis, P. Holtkamp, and B. Nunes. Openscout: harvesting business and management learning objects from the web of data. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 445–450. ACM, 2013.

[19] H. Khrouf and R. Troncy. EventMedia: A LOD dataset of events illustrated with media. *Semantic Web journal, Special Issue on Linked Dataset descriptions*, 7(2):193–199, 2016.

[20] G. Kobilarov, T. Scott, Y. Raimond, S. Oliver, C. Sizemore, M. Smethurst, C. Bizer, and R. Lee. Media meets semantic web–how the bbc uses dbpedia and linked data to make connections. In *European Semantic Web Conference*, pages 723–737. Springer, 2009.

[21] H.-P. Lam and G. Governatori. The making of spindle. In *Rule Interchange and Applications*, pages 315–322. Springer, 2009.

[22] 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.

[23] A. Margara, J. Urbani, F. van Harmelen, and H. Bal. Streaming the web: Reasoning over dynamic data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 25:24–44, 2014.

[24] 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/.

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

[26] 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.

[27] M. Sensoy, T. J. Norman, W. W. Vasconcelos, and K. Sycara. Owl-polar: A framework for semantic policy representation and reasoning. *Web Semantics: Science, Services and Agents on the World Wide Web*, 12:148–160, 2012.

[28] R. Shaw, R. Troncy, and L. Hardman. Lode: Linking open descriptions of events. In *The Semantic Web*, pages 153–167. Springer, 2009.

[29] S. Steyskal and A. Polleres. Towards formal semantics for odrl policies. In *International Symposium on Rules and Rule Markup Languages for the Semantic Web*, pages 360–375. Springer, 2015.

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

[31] R. Wille. Formal concept analysis as mathematical theory of concepts and concept hierarchies. In *Formal Concept Analysis*, pages 1–33. Springer, 2005.