Ontological Definition of Governance Framework for Security Policies in Cloud Environments

Simeon Veloudis
South East European Research Centre (SEERC), The University of Sheffield International Faculty CITY College
Thessaloniki, Greece
sveloudis@seerc.org

Iraklis Paraskakis
South East European Research Centre (SEERC), The University of Sheffield International Faculty CITY College
Thessaloniki, Greece
iparaskakis@seerc.org

Christos Petsos
South East European Research Centre (SEERC), The University of Sheffield International Faculty CITY College
Thessaloniki, Greece
chpetsos@seerc.org

ABSTRACT

The cloud computing paradigm enables enterprises to realise significant cost savings whilst boosting their agility and productivity. However, security and privacy concerns generally deter enterprises from migrating their critical data to the cloud. One way to alleviate these concerns, hence bolster the adoption of cloud computing, is to devise adequate *security policies* that control the manner in which these data are stored and accessed in the cloud. Nevertheless, for enterprises to entrust these policies, a framework capable of providing assurances about their *correctness* is required. This work proposes such a framework. In particular, it proposes an approach that enables enterprises to define their own view of what constitutes a correct policy through the formulation of an appropriate set of *well-formedness constraints*. These constraints are expressed *ontologically* thus enabling—by virtue of semantic inferencing—automated reasoning about their satisfaction by the policies.

CCS CONCEPTS

• **Information systems → Ontologies**; *Secure online transactions*; • **Security and privacy → Software and application security**; • **Applied computing →** *IT governance*; • **Computer systems organization → Cloud computing**;

KEYWORDS

Policies, security, privacy, ontologies, cloud computing, governance, OWL 2

ACM Reference format:
Simeon Veloudis, Iraklis Paraskakis, and Christos Petsos. 2017. Ontological Definition of Governance Framework for Security Policies in Cloud Environments. In *Proceedings of PCI 2017, Larissa, Greece, September 28–30, 2017,* 6 pages.
DOI: 10.1145/3139367.3139401

1 INTRODUCTION

By enabling access to shared pools of virtualised resources, cloud computing signifies a shift towards service-based architectures that offer, theoretically, a boundless scalability and a flexible pay-per-use model [7]. Such a shift brings about significant advantages for users in terms of cost, flexibility and business agility. In particular, it enables inherently heterogeneous stakeholders, ranging from small and medium enterprises (SMEs) to large retailers and health care providers, to realise significant cost savings by delegating the storage and processing of their data to servers that are under the control of third-party cloud providers. However, relinquishing control of—oftentimes critical—corporate data naturally raises significant security and privacy concerns that may deter stakeholders from embracing the cloud paradigm [4].

One way to alleviate these concerns, hence bolster the adoption of cloud computing, is to infuse adequate *security policies* into the applications through which sensitive data are stored and accessed in the cloud [14]. For example, policies may be required that articulate adequate *data fragmentation* and *distribution* schemes that control the manner in which critical data are partitioned and distributed over distinct cloud servers for privacy reasons. Consider, for instance, a relational database table `t` holding sensitive customer information. A policy may be required whereby `t` is fragmented such that customer credit card numbers and customer names are always stored on separate servers that are administered by distinct cloud providers.

We argue that, for stakeholders to entrust such policies with the protection of their critical data, a framework that provides assurances about the *correctness* of the policies is required. The work conducted as part of the PaaSword project [2] offers such a framework. In particular, PaaSword provides a security-by-design solution—essentially a PaaS offering—that facilitates the *governance* of security policies. To this end, it draws upon a *semantic representation* of policies, one that ontologically captures the various *knowledge artefacts* that are encoded in the policies. Such a representation unravels the expression of policies from the code of the applications in which they are infused; in this respect, it lends itself to *automated reasoning* about the *correctness* of the policies.

This paper proposes an approach to such reasoning, one that is based upon a class of ontologically-expressed constraints, the so-called *well-formedness* constraints. Well-formedness constraints articulate all those knowledge artefacts, and the values thereof, that *must*, *may* or *must not* be embodied in a security policy. In this respect, they give rise to a set of *ontological templates* against which the correctness of security policies can be judged. For instance, going back to the example above, a well-formedness constraint may insist that any data fragmentation and distribution policy concerning the relational table `t` must demand that `t`'s fragments are invariably persisted across servers that reside in the EU only; any data fragmentation and distribution policy not embodying such a location of distribution is assumed to be not well-formed and thus incorrect.

Evidently, well-formedness constraints empower stakeholders to harness the knowledge artefacts embodied in the security policies that protect their sensitive data. In other words, they empower stakeholders to instill into these policies their business logic and overall stance towards security. In this respect, well-formedness constraints compel the developers of the applications through which these data are stored and accessed to devise, and subsequently infuse into these applications, security policies that are adequate for the stakeholders' needs.

One of the main strengths of modelling well-formedness constraints *ontologically*, using OWL [16], is that it enables *new* knowledge artefacts to be *semantically inferred* from existing ones. This potentially allows the application of well-formedness constraints in situations in which the knowledge artefacts encoded in a policy do not necessarily match, at the syntactic level, the corresponding artefacts encoded in the constraints. For instance, returning to the example above, a security policy that states that the table `t` must be fragmented and distributed over servers that reside in, say, the Dublin area, is deemed to abide by the aforementioned well-formedness constraint as semantic inferencing allows us to determine that this area is indeed located in the EU. This clearly absolves stakeholders from having to specify fine-grained well-formedness constraints—i.e. constraints that cover each permissible location of data distribution that may be specified in a policy.

Although well-formedness constraints are applicable to a wide range of policies, here we confine ourselves to two particular kinds of policy encountered in the PaaSword project, namely: (i) *Data Encryption* (*DE*) policies that articulate the kind of cryptographic protection that a sensitive data object must enjoy in the cloud; (ii) *Data Fragmentation* and *Distribution* (*DFD*) policies that specify the manner in which sensitive data objects are fragmented and distributed across cloud servers.

The rest of this paper is structured as follows. Section 2 outlines an ontological framework for the representation of the knowledge artefacts that are encoded in DE and DFD policies. Section 3 draws upon this framework and presents a set of well-formedness constrains for DE and DFD policies. Section 4 describes how DE and DFD policies are checked for abidance by the well-formedness constraints. Finally, Section 5 presents related work and Section 6 outlines conclusions and future work.

2 REPRESENTING KNOWLEDGE ARTEFACTS

The Context Model (CM) proposed in [15] provides an ontological framework for the representation of the *knowledge artefacts* that lurk behind DE and DFD policies. Fig. 1 depicts a portion of the CM that includes only the concepts that are of interest to the work reported in this paper. At the core of this portion is the class[1] `DEFDElement` (stands for Data Encryption, Fragmentation and Distribution Element) which encompasses the concepts `Object`, `DEScheme`, `DFScheme` and `DDScheme`.[2] Instances of the first concept represent the actual sensitive data assets under protection; instances of the latter three concepts represent, respectively, particular data *encryption*, *fragmentation* and *distribution* schemes; brief accounts of these schemes are provided below—for fuller accounts the interested reader is referred to [15].

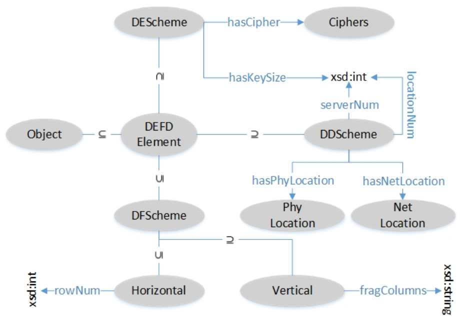

Figure 1: Fragment of the Context Model (namespaces omitted to reduce clutter)

A *data encryption scheme* specifies a particular symmetric or asymmetric cryptographic cipher (e.g. AES, RSA, etc.), one that is represented as an instance of the class `Cipher`. Ontologically this is captured through the object property `hasCipher` which interrelates data encryption schemes with cipher instances (see Fig. 1); in addition, a cipher instance is linked to the key length that characterises its strength through the data property `hasKeySize`.

Data fragmentation schemes are categorised into *horizontal* and *vertical* ones. Horizontal schemes shard database tables at the *row* level and are represented as instances of the concept `Horizontal` (see Fig. 1); vertical schemes fragment database tables at the *column* level and are represented as instances of the concept `Vertical`. A vertical fragmentation scheme is associated, through the data property `fragColumns` (see Fig. 1), with the identifiers of those columns that must *not* form part of the same fragment. A horizontal fragmentation scheme is associated, through the property `rowNum`, with the row number(s) at which the fragmentation takes place.

Finally, a *data distribution scheme* may specify, through the properties `serverNum` and `locationNum` respectively, the number of distinct (physical or virtual) machines, and the number of distinct (physical or network) locations, over which sensitive data fragments are distributed for privacy reasons. It may also specify, through the properties `hasPhyLocation` and `hasNetLocation` respectively, the particular physical or network locations at which sensitive data fragments must be stored; physical and network locations are represented as instances of the concepts `PhyLocation` and `NetLocation` respectively.

3 CONSTRAINING DE AND DFD POLICIES

This section elaborates on well-formedness constraints and the restrictions that they impose on the allowable forms that a DE or

[1]The terms 'class' and 'concept' are used interchangeably in this work.

[2]All concepts and properties depicted in Fig. 1 are included in the `pdefd` namespace which is omitted to reduce clutter and increase readability.

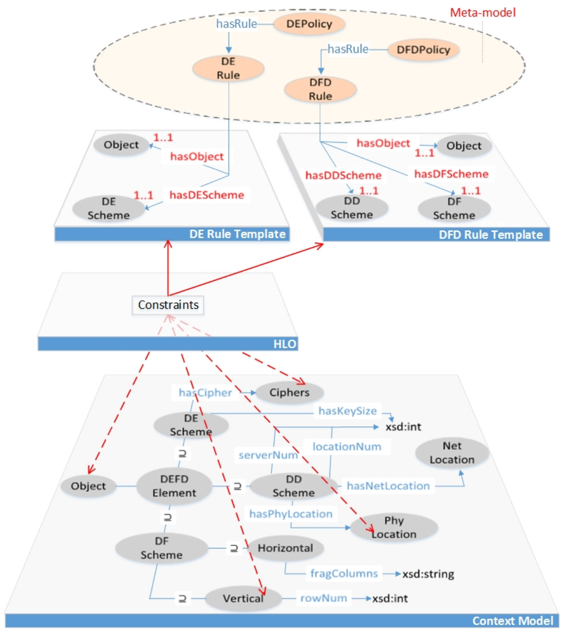

Figure 2: HLO Constraints

a DFD policy may assume. More specifically, it outlines how well-formedness constraints give rise to *ontological templates* that are used as touchstones for judging the correctness of a DE or DFD policy. A brief account of the ontological meta-model for DE and DFD policies is first in order.

3.1 Ontological Meta-Model for DE and DFD Policies

Inspired by the XACML standard [10], our work views a DE (DFD) policy as a container of one or more DE (DFD) *rules*. These rules are the actual carriers of the core logic conveyed by a policy. In this respect, they are associated with the knowledge artefacts in terms of which this logic is expressed. In particular, they are associated with a framework of concepts and properties drawn from the underlying CM that captures these knowledge artefacts and their interrelations. Ontologically, a DE (DFD) policy takes the form of an instance of the concept `DEPolicy` (`DFDPolicy`), and a DE (DFD) rule takes the form of an instance of the concept `DERule` (`DFDRule`)—see upper part of Fig. 2. A policy is associated with its constituent rules through the object property `pwd:hasRule`.

3.2 Well-Formedness Constraints for DE and DFD Rules

Well-formedness constraints are classified into *primary* and *secondary* ones. The former specify all those knowledge artefacts from the underlying CM that must, may or must not be embodied in a DE (DFD) rule; they also determine the allowable cardinalities with which these artefacts may appear in a rule. In this respect, they define the *exoteric* form of a DE (DFD) rule. On the other hand, secondary well-formedness constraints go a step further and restrict

Table 1: Axioms for primary well-formedness constraints

Axiom 1	`DERule`	$\sqsubseteq$	(=1 `hasObject.Object`)
Axiom 2	`DERule`	$\sqsubseteq$	(=1 `hasDEScheme.DEScheme`)
Axiom 3	`DFDRule`	$\sqsubseteq$	(=1 `hasObject.Object`)
Axiom 4	`DFDRule`	$\sqsubseteq$	(=1 `hasDFScheme.DFScheme`)
Axiom 5	`DFDRule`	$\sqsubseteq$	(=1 `hasDDScheme.DDScheme`)

the allowable values, or ranges of values, that these knowledge artefacts may assume.

Primary and secondary well-formedness constraints collectively define a *higher-level ontology* (*HLO*), one that essentially articulates all those conditions that must hold in order for a DE (DFD) rule to be considered *correct*. In this respect, the HLO gives rise to an *ontological template* by which a DE (DFD) rule must abide (see Fig. 2). Below we elaborate on a particular set of HLO constraints[3] for DE (DFD) rules that has been devised in the frame of the PaaSword project. These constraints are formally expressed in terms of $\mathcal{SROIQ}$ TBox axioms[4] [5] that restrict the classes `DERule` and `DFDRule`. It is to be emphasised here that the proposed constraint set is *malleable* in the sense that its contents can be altered to express alternate ontological templates for DE (DFD) rules, i.e. templates that potentially reflect more accurately the application-specific needs of an organisation adopting the PaaSword framework. This malleability is of utmost significance for it empowers stakeholders to instill into DE and DFD policies their own business logic and overall stance towards security.

3.3 Primary Well-Formedness Constraints

We start off with DE rules. The first constraint states that each rule must embody *exactly one* protected asset. Ontologically, this is captured through a TBox axiom that demands that a DE rule, i.e. each instance of the concept `DERule`, is associated with exactly one individual[5] from the class `Object` of the CM, and that this association should be realised through the object property `hasObject`. Axiom 1 of Table 1 provides a formal expression of this constraint[6]. The second constraint states that each DE rule must be associated with *exactly one* encryption scheme from the class `DEScheme`, and that this association should be realised through the property `hasDEScheme`. Axiom 2 of Table 1 provides a formal expression of this constraint.

A similar set of constraints applies to DFD rules: the first states that each DFD rule must embody *exactly one* protected asset through the object property `hasObject`; the second states that each DFD rule must embody *exactly one* data fragmentation scheme (i.e. exactly one instance of the CM class `DFScheme`) through the property

[3]The term 'HLO constraints' will henceforth be used to collectively refer to primary and secondary well-formedness constraints.
[4]$\mathcal{SROIQ}$ is the Description Language (DL) underpinning OWL 2: any $\mathcal{SROIQ}$ axiom is expressible as an OWL 2 assertion or an OWL 2 expression axiom and vice-versa. In this paper we resort to $\mathcal{SROIQ}$ due to the conciseness and rigorousness of its notation.
[5]The terms 'individual' and 'instance' are used interchangeably in this work.
[6]The axioms of Table 1 use the following notational convention: for any property R and concept C, the class $(= 1\ R.C)$ is an abbreviation for the class $(\leq 1\ R.C) \sqcap (\geq 1\ R.C)$ which denotes the class of all individuals that enjoy *exactly one* association through R with some individual from C.

Table 2: Axioms for secondary constraints

$\texttt{DFDRule} \sqsubseteq \neg(\leq 1\ \texttt{hasObject}.\{\texttt{t}\}) \sqcup ((\leq 1\ \texttt{hasDFScheme}.C_1) \sqcap (\leq 1\ \texttt{hasDDScheme}.C_2))$ where $C_1 \equiv (\leq 1\ \texttt{fragColumns}.\{\texttt{CCN}\}) \sqcap (\leq 1 \texttt{fragColumns}.\{\texttt{CId}\})$, $C_2 \equiv (\leq 1\ \texttt{hasPhyLocation}.\{\texttt{EU}\}) \sqcap \neg(\leq 1\ \texttt{hasPhyLocation}.\neg\{\texttt{EU}\})$
$\texttt{DERule} \sqsubseteq \neg(\leq 1\ \texttt{hasObject}.\{\texttt{t}\}) \sqcup (\leq 1\ \texttt{hasDEScheme}.C_3)$ where $C_3 \equiv (\leq 1\ \texttt{hasCipher}.\{\texttt{AES}\}) \sqcap \neg(\leq 1\ \texttt{hasCipher}.\neg\{\texttt{AES}\}) \sqcap (\leq 1\ \texttt{hasKeySize}.\{\texttt{256\^{}\^{}xsd:integer}\})$

hasDFScheme; the third states that each DFD must embody *exactly one* data distribution scheme (i.e. exactly one instance of the class DDScheme) through the property hasDDScheme. These constraints are expressed in terms of Axioms 3-5 of Table 1.

3.4 Secondary Well-Formedness Constraints

Secondary well-formedness constraints enable stakeholders to further instill into DE and DFD rules their security requirements by designating the *allowable values* that the knowledge artefacts embodied in there rules may assume. As an example, consider a situation whereby an organisation demands that a particular sensitive relational database table t is always fragmented vertically such that the columns identified by CCN (stands for "Credit Card Number") and CId (stands for "Customer Id") are never included in the same fragment; moreover, suppose that the organisation demands that all fragments of t invariably land on servers located in, say, the EU.

Ontologically, this constraint is captured through the $\mathcal{SROIQ}$ axiom of the 1st row of Table 2. This axiom comprises two disjuncts. The first disjunct, namely $\neg(\leq 1\ \texttt{hasObject}.\{\texttt{t}\})$, demands that a DFD rule has *no* association with t through the property hasObject. The second disjunct, namely $(\leq 1\ \texttt{hasDFScheme}.\ C_1)) \sqcap (\leq 1\ \texttt{hasDDScheme}.\ C_2))$, imposes that any DFD rule is associated, through the properties hasDFScheme and hasDDScheme, with instances of the (abstract) classes C_1 and C_2 respectively. C_1 is the class of all individuals that exhibit some association with the column identifiers CCN and CId through the property fragColumns; in other words, it is the class of all data fragmentation schemes that demand that the columns CCN and CId are stored in separate fragments. $C2$ is the class of all individuals that exhibit some association with the individual EU through the property hasPhyLocation and do not exhibit any other associations, through the same property, with locations other than the EU; in other words, it is the class of all data distribution schemes that are confined in the EU. Clearly, if a DFD rule is indeed associated with t (and therefore violates the first disjunct above), then for the axiom of Table 2 to be satisfied its second disjunct, along with the associations that it imposes, must hold.

Secondary constraints naturally apply to DE rules as well. As an example, consider a situation whereby an organisation wishes to impose that the table t is invariably encrypted with the AES cipher and a key length of 256 bits. Such a constraint is formally expressed in terms of the $\mathcal{SROIQ}$ axiom of the 2nd row of Table 2.

4 REASONING ABOUT THE CORRECTNESS OF DE AND DFD POLICIES: ENFORCING HLO CONSTRAINTS

Reasoning about the correctness of DE (DFD) rules, hence about the correctness of the DE (DFD) policies encompassing these rules, involves reasoning about the abidance of the rules by the HLO constraints. Below, we outline how such reasoning is performed by a mechanism that we have developed as part of the PaaSword project.

As an example, suppose the following set $\mathcal{R}$ of $\mathcal{SROIQ}$ axioms which articulate all those knowledge attributes associated with a DFD rule r; we shall term such an axiom-set a *knowledge base* (KB) [12].

$$\begin{aligned}\mathcal{R} \equiv \{&\texttt{DFDRule(r)}, \texttt{Object(t)}, \texttt{DFScheme(fs)}, \texttt{DDScheme(ds)},\\ &\texttt{Vertical(fs)}, \texttt{PhyLocation(EU)}, \texttt{hasObject(r,t)},\\ &\texttt{hasDDScheme(r,ds)}, \texttt{hasPhyLocation(ds,EU)}\}\end{aligned} \tag{1}$$

According to $\mathcal{R}$, r is associated, through the properties hasObject and hasDDScheme respectively, with the object t and the distribution scheme identified by the instance ds; ds is additionally associated, through hasPhyLocation, with the location EU.

4.1 Open and Closed-World Reasoning

Two seminal assumptions underpinning OWL are the Open-World Assumption (OWA) and the non-Unique Name Assumption (non-UNA). The former posits that lack of knowledge of a fact does not necessarily imply knowledge of the negation of the fact; the latter states that resources with distinct identifiers are not necessarily distinct. These assumptions stem from the fact that OWL is designed to describe facts about distributed KBs that are often incomplete and, to this end, it is geared towards inferring new facts from the ones already existing in the KBs [12].

Nevertheless, the OWA and non-UNA render the use of OWL (hence of $\mathcal{SROIQ}$) cumbersome when reasoning about constraint satisfaction. Consider, for example, the KB $\mathcal{R}$ above. $\mathcal{R}$ fails to attach to the rule r a fragmentation scheme for the object t. However, according to the OWA, this does not mean that r does not have such a scheme associated with it: it merely means that this association is not specified in $\mathcal{R}$. Thus, we cannot assert with certainty that Axiom 4 of Table 1 that demands that a fragmentation scheme must always be present in a DFD rule is violated. In order to overcome this obstacle, we adopt the approach proposed in [12] and dispense with the OWA and the non-UNA, effectively enabling closed-world reasoning when checking the abidance of DE (DFD) rules by HLO constraints. This reasoning is based on an extended semantics of OWL, namely the Integrity Constraint semantics [12]; an outline of how such reasoning is performed is in order.

4.2 Reasoning about Abidance by HLO Constraints

Each HLO constraint is translated into a *query*, one that is posed to the KB under validation with the aim of discovering any individuals that *violate* the constraint: if the query returns an empty set of

Table 3: SPARQL Query

```
SELECT DISTINCT * WHERE
{?x0 pre1:type pre2:DFDRule
 FILTER NOT EXISTS {?x0 pre2:hasDFScheme ?x1 .
                    ?x1 pre1:type pre2:DFScheme}
 UNION {  ?x0 pre2:hasDFScheme ?x2 .
          ?x0 pre2:hasDFScheme ?x2 .
          ?x2 pre1:type pre2:DFScheme .
          ?x0 pre2:hasDFScheme ?x3 .
          ?x3 pre1:type pre2:DFScheme
          FILTER (?x2 != ?x3) }
}
pre1 ::= http://www.w3.org/1999/02/22-rdf-syntax-ns
pre2 ::= http://seerc.org/pci/spv
```

individuals, the constraint is considered to hold[7]; otherwise, it is considered to be violated. The query is, in fact, a $\mathcal{SROIQ}$ assertion axiom that uses variables in place of individuals and expresses the *negation* of the HLO constraint that it translates. As an example, suppose the KB $\mathcal{R}$ of equation 1 and consider the HLO constraint whereby any DFD rule must embody *exactly one* data fragmentation scheme. Axiom 4 of Table 1 that expresses this constraint is translated into a query that attempts to discover in $\mathcal{R}$ any individuals that belong to the class DFDRule and which either enjoy *no* associations (through the property hasDFDScheme) with instances of the class DFScheme, or enjoy *two or more* such associations with *distinct* instances of DFScheme. Formally:

$$\begin{aligned}\texttt{DFDRule}(x) \wedge (&\textbf{not}(\texttt{hasDFScheme}(x,y) \wedge \texttt{DFScheme}(y)) \vee \\ &(\texttt{hasDFScheme}(x,y) \wedge \texttt{hasDFScheme}(x,z) \wedge \\ &\texttt{DFScheme}(y) \wedge \texttt{DFScheme}(z) \wedge \textbf{not}(y = z))\end{aligned} \tag{2}$$

Such queries are termed in [12] Distinguished Conjunctive Queries with Negation as Failure (DCQ$^{\textbf{not}}$). DCQ$^{\textbf{not}}$ are posed to the KB under validation in the form of SPARQL queries [1]. Table 3 shows the SPARQL query for the assertion axiom 2 above. SPARQL queries are executed in the Pellet reasoner [11] (however, any other OWL reasoner could have been used instead). In [12], a set of translation rules for turning any $\mathcal{SROIQ}$ axiom into a DCQ$^{\textbf{not}}$, and subsequently into a SPARQL query, is presented[8].

Our mechanism has been widely used in the frame of the PaaSword project. Although we have not performed any extensive performance analysis, the mechanism has been able to check a moderate size KB (comprising 1210 ABox axioms from the CM used in PaaSword) against constraint axioms in the order of milliseconds. It is to be noted here that our mechanism is intended to be used during application design time for assisting developers in devising correct security policies for their applications. In this respect, performance is not a critical feature and moderate variations in latencies are tolerable. Of course, we intend to conduct further performance tests to determine the scalability of our mechanism to larger KBs.

[7]This is an instance of *Negation as Failure* (*NAF*) *inferencing*: an axiom is derived on the premise that its negation is not derivable. NAF is widely used in logic programming.
[8]The translation rules are not reproduced here for reasons of space.

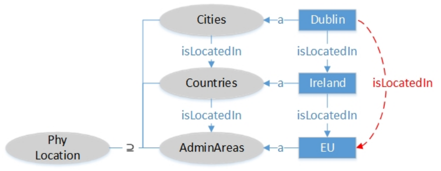

Figure 3: CM semantic inferencing

4.3 Semantic Inferencing During Constraint Checking

When reasoning about the abidance of a DE (DFD) rule by the HLO constraints, the concepts and properties of the CM may be exploited in order to semantically infer *new knowledge artefacts* from the ones already attached to a rule. This facilitates the expression of constraints that potentially cover a wide range of rules. As an example, suppose the KB

$$\begin{aligned}\mathcal{R}' \equiv \mathcal{R} \cup \{&\texttt{DFScheme(fs)}, \texttt{hasDFScheme(r,fs)}, \\ &\texttt{fragColumns(fs,CCN)}, \texttt{fragColumns(fs,CId)}\}\end{aligned} \tag{3}$$

where $\mathcal{R}$ is the KB of equation 1. Let us examine whether the DFD rule r in $\mathcal{R}'$ abides by the axiom of the 1st row of Table 2. Firstly r is associated, through the property hasObject, with the object t and hence the first disjunct of the axiom is violated (see Section 3.4). This means that for the axiom to be satisfied its second disjunct must hold. The first conjunct of this disjunct, namely (≤ 1 hasDFScheme.C_1), demands that the fragmentation scheme associated with r is associated through the property fragColumns with the columns identified by CCN and CId. We observe that these associations are indeed present in $\mathcal{R}'$ for the fragmentation scheme fs.

Turning now to the second conjunct, namely (≤ 1 hasDDScheme.C_2)), we observe that r's distribution scheme ds is associated, through the property hasPhyLocation with the location Dublin (rather than with the location EU as the axiom demands). $\mathcal{R}'$ therefore fails to satisfy (at least at the syntactic level) the axiom. Nevertheless, semantic inferencing at the level of the CM allows us to conclude that Dublin is in fact located in the EU and that therefore $\mathcal{R}'$ satisfies the axiom. Such inferencing is performed automatically by a DL reasoner and, besides unveiling any inconsistencies in the CM, it also greatly facilitates the definition of HLO constraints as it absolves stakeholders from having to specify fine-grained constraints. For instance, in the example above, it absolves stakeholders from having to specify constraints that cover each permissible location of distribution.

Of course, for such semantic inferencing to be possible certain knowledge artefacts must be included in the CM. For instance, in the example above, the concept PhysicalLocation needs to be extended with such concepts as AdminArea, Country and City, as well as with the transitive property isLocatedIn (see Fig. 4). Note that these concepts are introduced during the process of priming the CM—a process undertaken by the stakeholders in order to customise the CM for their particular purposes.

5 RELATED WORK

Various approaches to the semantic representation of policies have been proposed in the literature [6, 8, 13]. In [13] KaoS is presented—an OWL-based framework for the specification and enforcement of policies that offers: (i) a user interface for formulating policies; (ii) a policy management layer that identifies and resolves any conflicting policies; (iii) an engine for encoding policies in a programmatic format suitable for their efficient enforcement. A drawback of the KaoS approach is that the programmatic translation of the policies precludes any policy updates during runtime, as such updates require that the policies are re-compiled to the programmatic format. In addition, KaoS does not provide any means for reasoning about the correctness of the policies.

The work in [6] proposes Rei: a framework based on OWL-Lite [17] for specifying policies concerning the desirable behaviours of autonomous entities and reasoning about their compliance. A policy in Rei embodies a list of rules that take the form of OWL properties, as well as a malleable framework of concepts and their interrelations that captures the underlying context of application. Rei adopts the use of placeholders, as in rule-based programming languages, for the definition of *variables* that are purpotedly required for expressing the rules. This, however, prevents Rei from exploiting the full inferencing potential of OWL when reasoning about compliance as policy rules are expressed in a formalism that is alien to OWL. In addition, Rei does not provide any means for reasoning about the correctness of the policies.

In [8] the POLICYTAB approach is presented for enabling a trust negotiation process that aims at controlling accesses to sensitive Web resources. More specifically, POLICYTAB adopts OWL for representing policies that guide this negotiation by allowing the specification of the credentials that an entity must possess for performing an action on a resource owned by another entity. Nevertheless, no attempt is made to model the underlying context of application or determine the correctness, hence the effectiveness, of these policies.

On a different note, the works in [3, 9, 10] provide declarative formalisms for the expression of policies. These formalisms, however, lack the means of providing any semantic representation of the policies, which precludes from the outset any form of generic reasoning about their correctness. In addition, it restricts the portability and reusability of the policies as any interoperability crucially depends on the use of shared vocabularies that need to be adopted by the parties involved in an interaction. This clearly rules out any form of semantic agreement beyond the boundaries of the organisations that adopt these vocabularies.

6 CONCLUSIONS

We have presented an approach to defining, and reasoning about, the correctness of DE and DFD policies in cloud environments. The approach is based on a class of ontologically-expressed constraints, the so-called HLO constraints, that enable stakeholders to instill into DE and DFD policies their business logic and overall stance towards security. Reasoning about the correctness of the policies thus amounts to determining their abidance by these constraints. Such reasoning is based on an extended semantics of OWL, one that dispenses with the OWA and the non-UNA, allowing the transformation of the HLO constraints into queries that are posed against the KBs that represent the policies.

As part of future work we intend to construct an editor that will provide two main functionalities: firstly, it will facilitate stakeholders in expressing HLO constraints and, secondly, it will facilitate developers in formulating correct security policies through the application of these constraints. More specifically, regarding the latter functionality, each time a developer attempts to embody a knowledge artefact into a security policy, the constraints will intervene in order to determine whether this artefact is allowable and, if it is, to provide all permissible values that it may assume.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644814.

REFERENCES

[1] 2013. SPARQL 1.1 Query Language W3C Recommendation. https://www.w3.org/TR/sparql11-query/. (March 2013).
[2] 2015. PaaSword - A Holistic Data Privacy and Security by Design Platform-as-a Service Framework. https://www.paasword.eu. (2015).
[3] Harold Boley, Tara Athan, Adrian Paschke, Adrian Giurca, Nick Bassiliades, Guido Governatori, Monica Palmirani, Adam Wyner, and Gen Zou. 2016. Specification of Deliberation RuleML 1.01. (June 2016). http://wiki.ruleml.org/index.php/Specification_of_Deliberation_RuleML_1.01.
[4] Cloud Security Alliance 2015. *What's Hindering the Adoption of Cloud Computing in Europe?* Cloud Security Alliance. https://blog.cloudsecurityalliance.org/2015/09/15/whats-hindering-the-adoption-of-cloud-computing-in-europe/.
[5] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. 2006. The Even More Irresistible SROIQ. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning*, Patrick Doherty, John Mylopoulos, and Christopher A. Welty (Eds.). AAAI Press, 57–67.
[6] L. Kagal, T. Finin, and A. Joshi. 2003. A policy language for a pervasive computing environment. In *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks.* 63–74. DOI: https://doi.org/10.1109/POLICY.2003.1206958
[7] F Liu, J Tong, J Mao, R Bohn, J Messina, L Badger, and D Leaf. 2011. *NIST Cloud Computing Reference Architecture.*
[8] Wolfgang Nejdl, Daniel Olmedilla, Marianne Winslett, and Charles C. Zhang. 2005. Ontology-Based Policy Specification and Management, Asunción Gómez-Pérez and Jérôme Euzenat (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 290–302. DOI: https://doi.org/10.1007/11431053_20
[9] OASIS 2008. *Security Assertions Markup Language (SAML) Version 2.0. Technical Overview.* OASIS. https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf.
[10] OASIS 2013. *eXtensible Access Control Markup Language (XACML) Version 3.0.* OASIS. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.
[11] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. 2007. Pellet: A Practical OWL-DL Reasoner. *Web Semant.* 5, 2 (June 2007), 51–53. DOI: https://doi.org/10.1016/j.websem.2007.03.004
[12] Jiao Tao, Evren Sirin, Jie Bao, and Deborah L. McGuinness. 2010. Integrity Constraints in OWL. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI'10).* AAAI Press, 1443–1448. http://dl.acm.org/citation.cfm?id=2898607.2898837
[13] A Uszok, J Bradshaw, R Jeffers, M Johnson, A Tate, J Dalton, and S Aitken. 2004. KAoS Policy Management for Semantic Web Services. *IEEE Intel. Sys.* 19, 4 (2004), 32–41.
[14] Simeon Veloudis and Iraklis Paraskakis. 2016. Defining an Ontological Framework for Modelling Policies in Cloud Environments. In *8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom'16).*
[15] Yiannis Verginadis, Ioannis Patiniotakis, and Gregoris Mentzas. 2015. *Context-aware Security Model, PaaSword Project Deliverable D2.1.* https://www.paasword.eu/wp-content/uploads/2016/09/D2-1_Context-awareSecurityModel.pdf.
[16] W3C 2004. *W3C Recommendation. 2004. OWL Web Ontology Language Reference.* W3C. https://www.w3.org/TR/owl-ref/.
[17] W3C 2004. *W3C Recommendation. 2004. OWL Web Ontology Language Semantics and Abstract Syntax.* W3C. https://www.w3.org/TR/2004/REC-owl-semantics-20040210/.