Ontological Framework for Ensuring Correctness of 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

By embracing the cloud computing paradigm enterprises are able to boost their agility and productivity whilst realising significant cost savings. However, many enterprises are reluctant to adopt cloud services for supporting their critical operations due to security and privacy concerns. One way to alleviate these concerns is to devise *policies* that infuse suitable security controls in cloud services. This work proposes a class of ontologically-expressed rules, namely the so-called *axiomatic rules*, that aim at ensuring the *correctness* of these policies by harnessing the various *knowledge artefacts* that they embody. It also articulates an adequate framework for the expression of policies, one which provides *ontological templates* for modelling the knowledge artefacts encoded in the policies and which form the basis for the proposed axiomatic rules.

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 Framework for Ensuring Correctness of Security Policies in Cloud Environments. In *Proceedings of BCI '17, Skopje, Macedonia, September 20–23, 2017,* 8 pages.
DOI: 10.1145/3136273.3136289

1 INTRODUCTION

By enabling ubiquitous access to shared pools of distributed and elastic resources, cloud computing represents a significant shift towards service-based architectures that offer a theoretically boundless scalability and a flexible pay-per-use model [8]. Such a shift brings about significant advantages for users in terms of cost, flexibility and business agility. In particular, it enables a multitude of inherently heterogeneous stakeholders, ranging from small and medium enterprises (SMEs) to 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—data naturally raises significant security and privacy concerns that deter, in general, stakeholders from embracing the cloud paradigm [6].

We argue that one way to alleviate these concerns, hence bolster the adoption of cloud computing, is to devise suitable *policies* that infuse adequate *security controls* into the applications through which critical 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 holding sensitive customer information. A policy may be required whereby this table is fragmented such that customer credit card numbers and customer names are always stored on separate physical servers that are administered by distinct cloud providers.

The work conducted as part of the PaaSword project [2] offers an adequate framework for the expression of such policies. PaaSword aspires to provide a security-by-design solution—essentially a PaaS offering—that facilitates developers in formulating suitable security policies for dynamic cloud environments. To this end, it proposes a novel approach to modelling such policies, one which draws upon *ontological templates* that capture the various *knowledge artefacts* that are encoded in the policies [15]. In this respect, it advocates a clear separation of concerns by disentangling the representation of policies from the actual code that is employed for enforcing them. This brings about the following seminal advantages with regard to the *governance* of policies.

Firstly, it enables automated reasoning about potential *inter-policy relations* such as *subsumption* and *contradiction*. This is particularly important as it increases the effectiveness of the policies. Secondly, it lends itself to *automated checks* regarding the *correctness* of the policies by *harnessing* the various knowledge artefacts that are encoded in the policies. More specifically, it enables the articulation of a set of *axiomatic rules* that restrict the allowable values that one or more knowledge artefacts encoded in a policy

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
BCI '17, Skopje, Macedonia
© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM.
978-1-4503-5285-7/17/09…$15.00
DOI: 10.1145/3136273.3136289

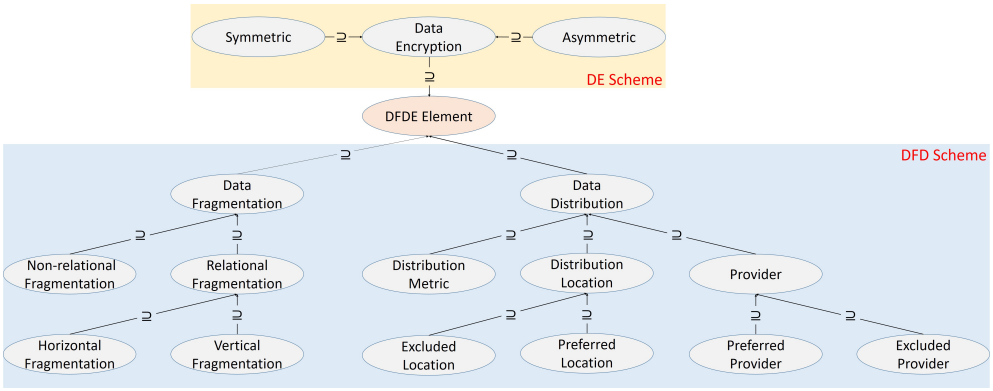

Figure 1: Fragment of the Context Model (namespaces omitted to reduce clutter)

may assume on the basis of the values assigned to certain other knowledge artefacts. For example, an axiomatic rule may insist that, if a sensitive data object—e.g. the aforementioned relational database table—is encrypted with a cipher other than AES-256, then any fragments of that object should *not* be landing on servers that are outside the EU area.

This paper proposes an approach for the representation of axiomatic rules as reifications of abstract *ontological templates* that draw upon the ontological templates proposed in [14, 15] for the expression of policies. One of the main strengths of our approach is the fact that it enables—by virtue of *semantic inferencing*—the generation of *new knowledge artefacts* that potentially allow the application of axiomatic rules in situations in which the knowledge artefacts encoded in a security policy do not match, at the syntactic level, the corresponding artefacts encoded in the axiomatic rules. For instance, returning to the example above, a security policy that states that the relational database table must be fragmented and distributed over servers that reside in, say, the Frankfurt and Dublin areas, is deemed to abide by the aforementioned axiomatic rule as semantic inferencing allows us to determine that these areas are indeed located in the EU. This clearly absolves developers from the burden of having to specify fine-grained axiomatic rules that cover each permissible location of data distribution articulated by a security policy.

Although our ontological templates are applicable to axiomatic rules that constrain a wide range of policies, here we confine ourselves to two particular kinds of policy, 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 different 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 ontological templates for the representation of DE and DFD policies. Section 4 proposes ontological templates for the representation of axiomatic rules that constrain DE and DFD policies. Section 5 describes how DE and DFD policies are checked for conformance with axiomatic rules. Finally, Section 6 presents related work and Section 7 outlines conclusions and future work.

2 REPRESENTING KNOWLEDGE ARTEFACTS

The Context Model (CM) proposed in [16] 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 (classes) that are of interest to the work reported in this paper. At the core of this portion is the class `pdm:DFDEElement` which is partitioned by the classes `pdm:DataEncryption`, `pdm:DataFragmentation` and `pdm:DataDistribution`;[1] instances of these classes represent, respectively, particular data *encryption*, *distribution* and *fragmentation* schemes. Sections 2.1, 2.2 and 2.3 below provide brief accounts of these classes; for fuller accounts the interested reader is referred to [16].

In addition, the CM comprises the concept `pcm:Object` (not shown in Fig. 1 to avoid clutter) which encompasses instances that represent the actual sensitive data objects that are amenable to the aforementioned schemes.

2.1 Data Encryption Scheme

The `pdm:DataEncryption` class encompasses two subclasses, namely `pdm:Symmetric` and `pdm:Asymmetric` (see Fig. 1). These subclasses comprise instances that represent, respectively, concrete *symmetric* and *asymmetric* cryptographic schemes, with any instances lying at the intersection of the two subclasses representing *hybrid* cryptographic schemes—i.e. schemes that combine both symmetric and asymmetric ciphers (e.g. OpenPGP [4]).

Furthermore, the two subclasses feature properties that associate the cryptographic schemes that they encompass with the actual ciphers that implement them (e.g. AES, RSA, etc.), as well as with the corresponding key sizes used. In particular, for symmetric cryptographic schemes, the data properties `pdm:hasSymmetricCipher` and `pdm:hasSymmetricKeySize` are defined; similar properties are defined for asymmetric cryptographic schemes.

[1] The namespace `pdm` includes all concepts depicted in Fig. 1.

2.2 Data Fragmentation Scheme

The pdm:DataFragmentation class encompasses two subclasses, namely pdm:RelationalFragmentation and pdm:Non-relationalFragmentation (see Fig. 1). These subclasses comprise instances that represent, respectively, fragmentation schemes suitable for relational and non-relational database tables. The latter subclass will not further concern us here for, due to space limitations, we confine ourselves to relational databases.

Relational fragmentation schemes are further subdivided into *horizontal* and *vertical* ones. Horizontal schemes shard database tables at the *row* level and are represented by instances of the class pdm:HorizontalFragmentation (see Fig. 1); vertical schemes fragment database tables at the *column* level and are represented by instances of the class pdm:VerticalFragmentation (see Fig. 1); fragmentation schemes that partition database tables both horizontally and vertically are represented by instances that lie at the intersection of the two classes. The class pdm:VerticalFragmentation features the data property pdm:hasPrivacyConstraint which determines one or more column pairs that should *not* appear as part of the same fragment. A similar data property is defined for the class pdm:HorizontalFragmentation for determining the row number(s) at which the fragmentation actually takes place.

2.3 Data Distribution Scheme

The pdm:DataDistribution class is partitioned by the classes pdm:DistributionMetric, pdm:DistributionLocation and pdm:Provider (see Fig. 1). The pdm:DistributionMetric class comprises instances that represent different metrics that quantify the distribution of a sensitive data object over physical and virtual servers, as well as over geographical locations. To this end, it features the data properties pdm:NumberOfServers, pdm:NumberOfVMs and pdm:numberOfLocations that associate, respectively, these instances with counts of physical and/or virtual machines, as well as with counts of distinct geographical locations.

The pdm:DistributionLocation class is further partitioned into the classes pdm:PreferredLocation and pdm:ExcludedLocation. The former encompasses instances that represent distinct physical locations that should be considered as 'appropriate' when distributing sensitive data; the latter encompasses instances that represent locations that should be avoided when distributing sensitive data.

Finally, the pdm:Provider class is further partitioned into the classes pdm:PreferredProvider and pdm:ExcludedProvider that encompasses instances that represent, respectively, trusted and untrusted IaaS providers that should be opted for, or avoided, when distributing sensitive data.

3 ONTOLOGICAL TEMPLATES FOR POLICY RULES

The ontological meta-model depicted in Fig. 2 forms the basis for the representation of security policies in the PaaSword project. This meta-model is suitably *reified* in order to accommodate the *ontological templates* in terms of which the security policies are expressed. These ontological templates bundle together all those concepts and properties from the underlying CM that are required for capturing the various *knowledge artefacts* that lurk behind the policies.

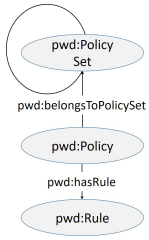

Figure 2: Ontological meta-model for policies

The reification process for the meta-model, as well as the ontological templates for the expression of DE and DFD policies, are elaborated in Section 3.2 below; an outline of the ontological meta-model is first in order—for more details the interested reader is referred to [13].

3.1 Ontological Meta-model

Following an approach inspired by the XACML standard [11], the meta-model discerns three levels of structural elements: *Rules*, *Policies* and *Policy sets* (see Fig. 2). Rules are the most elementary structural elements and the basic building blocks of policies: they are, in fact, the carriers of the core logic conveyed by the policies. 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. For DE and DFD rules this framework is represented by the *ontological templates* depicted in Figures 3 and 4 and elaborated in Section 3.2.

Rules are represented ontologically as instances of the class pwd:Rule[2] (see Fig. 2), whereas policies take the form of instances of the class pwd:Policy. A policy is associated with its constituent rules—hence with the ontological templates associated with these rules—through the object property pwd:hasRule (see Fig. 2).

Policies are also grouped into *policy sets*. A policy set takes the form of an instance of the class pwd:PolicySet and is associated with its constituent policies through the object property pwd:belongsToPolicySet. A policy set may exhibit a hierarchical structure and comprise one or more other policy sets; such recursive inclusions are captured by rendering the pwd:belongsToPolicySet property applicable to policy sets as well (i.e. in addition to policies—see Fig. 2).

3.2 Ontological Templates for DE and DFD Rules

We first outline the process whereby the meta-model of Fig. 2 is reified in order to accommodate the ontological templates that are associated with DE and DFD rules and hence DE and DFD policies; we then proceed to present these ontological templates.

[2]The namespace pwd includes all concepts and properties related to the ontological meta-model; it also includes the concepts and properties related to the ontological meta-model for axiomatic rules outlined in Section 4 (see, for example, Fig. 5).

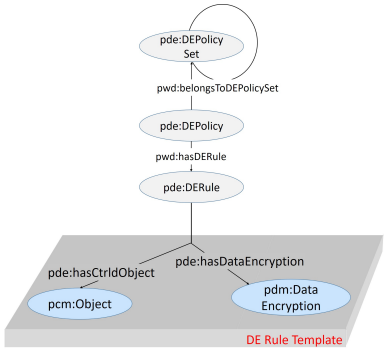

Figure 3: Ontological template for DE rules

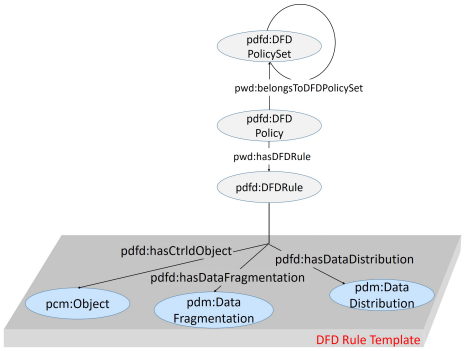

Figure 4: Ontological template for DFD rules

3.2.1 Meta-model Reification. DE (DFD) rules are represented as instances of the class `pde:DERule` (`pdfd:DFDRule`)[3] depicted in Fig. 3 (Fig. 4) which is a subclass—hence a reification—of the class `pwd:Rule`. Similarly, DE (DFD) policies take the form of instances of the class `pde:DEPolicy` (`pdfd:DFDPolicy`) which is a subclass of the class `pwd:Policy`. A DE (DFD) policy is tied to the DE (DFD) rules that it comprises through the object property `pde:hasDERule` (`pdfd:hasDFDRule`) which is a sub-property of the property `pwd:hasRule`. In a similar vein, DE (DFD) policy sets take the form of instances of the class `pde:DEPolicySet` (`pdfd:DFDPolicySet`) which is a subclass—hence a reification—of the class `pwd:PolicySet`. A DE (DFD) policy set is tied to the DE (DFD) policies that it comprises through the property `pde:belongsToDEPolicySet` (`pdfd:belongsToDFDPolicySet`) which is a sub-property—hence a reification—of the `pwd:belongsToPolicySet` property.

3.2.2 Ontological Templates for DE Rules. A DE rule is associated with the ontological template depicted in Fig. 3. This template specifies a generic framework of relevant knowledge artefacts in terms of which the core logic conveyed by DE policies is expressed. More specifically, it comprises the knowledge artefacts represented by the concepts `pcm:Object` and `pdm:DataEncryption`. The former concept identifies the sensitive data object which is to be encrypted according to the encryption scheme specified by the latter concept; both concepts were outlined in Section 2. Concrete DE rules are derived as reifications of this ontological template, by substituting specific instances for these concepts. For example, Table 1 depicts a concrete DE rule whereby the sensitive data object identified by the `pcm:Object` instance, say, `:o` must be encrypted with the AES symmetric cipher and with a key length of 128 bits[4].

3.2.3 Ontological Templates for DFD Rules. A DFD rule is associated with the ontological template depicted in Fig. 4. This template specifies a generic framework of relevant knowledge artefacts in terms of which the core logic conveyed by DFD policies is expressed. More specifically, it comprises the knowledge artefacts represented by the concepts `pcm:Object`, `pdm:DataFragmentation` and `pdm:DataDistribution`. The first concept represents the sensitive data object which is to be fragmented and distributed, whereas the second and third concepts represent, respectively, the data fragmentation and distribution schemes that are to be applied to the sensitive data object; all three concepts were outlined in Section 2. As in the case of DE rules, concrete DFD rules are derived as reifications of this ontological template, by substituting specific instances for these concepts. For example, Table 1 depicts a concrete DFD rule whereby the sensitive data object identified by the `pcm:Object` instance `:o` (say a relational database table) must be vertically fragmented such that the columns identified by `CCN` (stands for "Credit Card Number") and `CId` (stands for "CustomerId") should never be included in the same fragment and all fragments should land in servers located in the Frankfurt and Dublin areas (these areas take the form of instances of the class `pdm:PreferredLocation` depicted in Fig. 1).

Table 1: Example DE and DFD rules (in RDF Turtle [1])

DE Rule

```
:rule1 a pdm:DERule;
       pde:hasCtrldObject :o;
       pde:hasDataEncryption :crypto1.
:o a pcm:Object.
:crypto1 a pdm:Symmetric;
         pdm:hasCipher "AES"^^xsd:string;
         pdm:hasSymmetricKeySize
               "128"^^xsd:nonNegativeInt;
```

DFD Rule

```
:rule2 a pdm:DFDRule;
       pdfd:hasCtrldObject :o;
       pdfd:hasDataFragmentation :fScheme;
       pdfd:hasDataDistribution :Frankfurt,
                                :Dublin.
:o a pcm:Object.
:fDcheme a pdm:VerticalFragmentation;
         pdm:hasPrivacyConstraint
               "{CCN,CId}"^^xsd:string.
:Frankfurt,:Dublin a pdm:PreferredLocation.
```

[3] All concepts and properties of the DE and DFD models are defined in the `pde` and `pdfd` namespaces respectively.

[4] No particular namespace is provided for concepts, instances and properties that are introduced as part of examples, e.g. the instance `:o`.

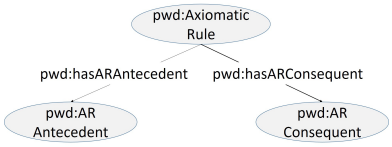

Figure 5: Ontological meta-model for ARs

4 ONTOLOGICAL TEMPLATES FOR AXIOMATIC RULES

As mentioned in Section 1, axiomatic rules (ARs) restrict the allowable values that one or more knowledge artefacts encoded in a policy rule may assume on the basis of the values assigned to certain other knowledge artefacts through other policy rules. ARs essentially express *high-level policies* that reflect an organisation's business logic, its high-level directions of operation and its overall stance towards security[5].

The ontological meta-model depicted in Fig. 5 forms the basis for the representation of ARs. This meta-model is suitably *reified* in order to accommodate the ontological templates in terms of which concrete ARs are expressed. These ontological templates bundle together all those concepts and properties from the underlying CM that are required for capturing the various knowledge artefacts that are encoded in the ARs. The reification process for the meta-model, as well as the ontological templates for the expression of ARs suitable for DE and DFD rules, are elaborated in Section 4.2 below; an outline of the ontological meta-model of Fig. 5 is first in order.

4.1 Ontological Meta-Model

Any AR comprises an *antecedent* and a *consequent*. The former articulates a (*pre*)condition that must be satisfied in order for the (*post*)condition articulated by the latter to be *enforceable*. More specifically, the former articulates the particular values that certain knowledge artefacts attached to a policy rule must possess, in order for the values articulated by the latter—and which refer to certain other knowledge artefacts attached to one or more other policy rules—to be enforceable. Suppose, for instance, the AR of Example 4.1.

Example 4.1. The fragments of all relational database tables that are characterised as 'critical' and have been encrypted with the AES cipher and a key size of 128 bits must be distributed over servers that are located in the EU.

Such an AR comprises: (i) An antecedent that essentially describes all those DE rules that are associated with a 'critical' database table and an encryption scheme that employs the AES cipher and a key size of 128 bits. (ii) A consequent that essentially describes all those DFD rules that are associated with a 'critical' database table and a data distribution scheme whose locations are confined in the EU.

Ontologically, an AR takes the form of an instance of the concept `pwd:AxiomaticRule` (see Fig. 5). The antecedent and consequent of an AR are captured, respectively, in terms of the concepts `pwd:ARAntecedent` and `pwd:ARConsequent`; these concepts are attached to an AR through the object properties `pwd:hasARAntecedent` and `pwd:hasARConsequent` respectively (see Fig. 5).

4.2 Ontological Templates for DE and DFD Axiomatic Rules

We first outline the process whereby the meta-model of Fig. 5 is reified in order to accommodate the ontological templates that are associated with DE and DFD ARs; we then proceed to present these ontological templates.

4.2.1 Meta-model Reification. DFDE ARs take the form of instances of the class `pdfde:AxiomaticDFDERule` (see Fig. 6)[6] which is a subclass—hence a reification—of the class `pwd:AxiomaticRule`. The antecedent and consequent of a DFDE AR are captured, respectively, in terms of two subclasses of the classes `pwd:ARAntecedent` and `pwd:ARConsequent`, namely the classes `pdfde:DFDEARAntecedent` and `pdfde:DFDEARConsequent` (see Fig. 6). Similarly, a DFDE AR is associated with its antecedent and consequent through two sub-properties of the object properties `pwd:hasARAntecedent` and `pwd:hasARConsequent`, namely the properties `pdfde:hasDFDEARAntecedent` and `pdfde:hasDFDEARConsequent` respectively.

4.2.2 Ontological Templates for DFDE ARs. The antecedent of a DFDE AR is formally described in terms of the ontological template shown in Fig. 6. This template encompasses all those knowledge artefacts whose instances may be specified during the formation of a DE or DFD rule, namely the concepts `pcm:Object`, `pdm:DataEncryption`, `pdm:DataFragmentation` and `pdm:DataDistribution` (see Figures 3 and 4). These instances are associated with an instance of the class `pdfde:DFDEARAntecedent` through the very same properties used for associating them with a DE or a DFD rule, namely `pde:hasCtrldObject`, `pdfd:hasCtrldObject`, `pde:hasDataEncryption`, `pdfd:hasDataFragmentation` and `pdfd:hasDataDistribution`. The intention here is to specify through these instances a particular data encryption, fragmentation or distribution scheme that must be imposed by a DE or DFD rule on a sensitive data object in order for that rule to be *amenable* to the AR, hence for the AR to be *enforceable*. The *consequent* of a DFDE AR is formally described in terms of an analogous ontological template (see Fig. 6). Its purpose is to formulate a particular data encryption, fragmentation or distribution scheme that must be *abided* by the DE and DFD rules.

5 ENFORCING DFDE ARS

We now elaborate on the approach that we have implemented for enforcing the ontological template for the expression of DFDE ARs outlined in Section 4 upon DE and DFD rules. In particular, Section 5.1 outlines the actual enforcement process and Section 5.2 briefly elaborates on how *semantic inferencing* at the level of the CM can complement, and therefore reinforce, this process.

5.1 Enforcement Process

The enforcement of DFDE ARs proceeds as follows. Each policy rule R is transformed programmatically into an OWL 2 *abstract*

[5] Of course, they may also be in line with relevant governmental rules and regulations (e.g. the EU's General Data Protection Regulation (EU) 2016/679).

[6] The namespace `pdfde` includes all concepts and properties related to DE and DFD ARs.

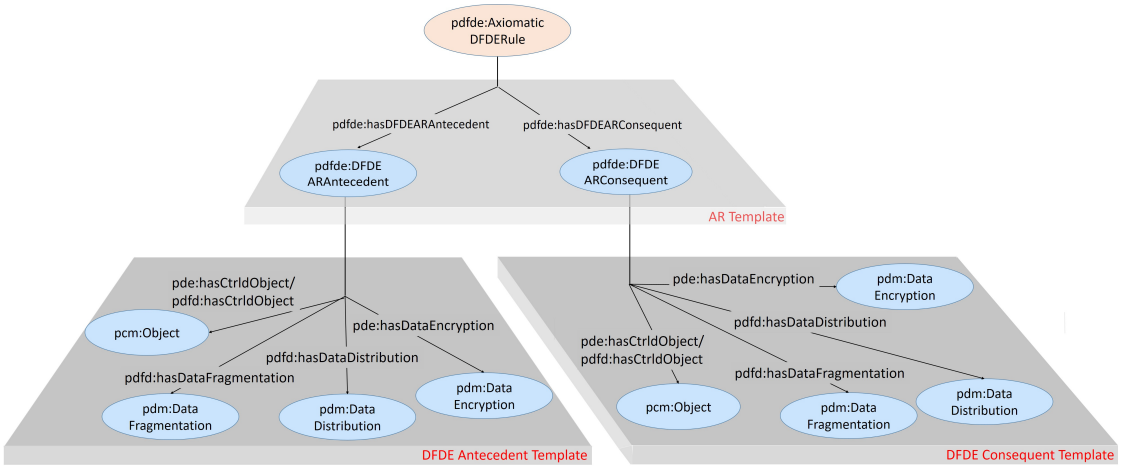

Figure 6: Ontological template for DFDE ARs

class [19], one that comprises all those individuals that are associated, through the appropriate object properties, with the knowledge artefacts attached to *R*. Consider, for example, the DE rule of Table 1. This rule is associated with the following knowledge artefacts: the controlled object instance `:o` (through the property `pde:hasCtrldObject`) and the data encryption scheme `:crypto1` (through the property `pde:hasDataEncryption`). This rule is thus represented by an abstract class that comprises all those individuals `:x` such that: (i) `:x` has some association with `:o` through the property `pde:hasCtrldObject`; and (ii) `:x` has some association with `:crypto1` through the property `pde:hasDataEncryption`. Such an abstract class is expressed in OWL 2 as shown in the first row of Table 2. Similarly, the DFD rule of Table 1 is expressed in terms of the abstract class shown in the second row of Table 2.

In an analogous manner, the antecedent and consequent of a DFDE AR are too expressed as OWL 2 abstract classes. For instance, Table 3 depicts the abstract classes corresponding to the antecedent and consequent of the DFDE AR of Example 4.1. Note that the concept `:CriticalObject` is assumed to be a subclass of the class `pcm:Object` that encompasses all those database tables that are deemed 'critical' (including the database table represented by the individual `:o`).

We have employed the Pellet DL reasoner [5] for automatically determining whether a DFDE AR is enforceable and whether a DE or DFD rule abides by it. In particular, let *R* be a DE (DFD) rule associated with a sensitive data object *d* and let *AR* be a DFDE AR whose antecedent specifies a particular data encryption, fragmentation or distribution scheme for *d*. The reasoner checks whether any of the already defined DE and DFD rules are *amenable* to *AR*, i.e. whether their corresponding abstract classes are included in the abstract class corresponding to the antecedent of *AR*. If such an inclusion is found, then the data encryption, fragmentation or distribution scheme specified by the antecedent of *AR* has already been imposed on *d* by one or more of these rules. In such a case, the *AR* is deemed *enforceable* and the reasoner proceeds to check whether *R* abides by the *AR*, i.e. whether the abstract class corresponding to *R* is a subclass of the abstract class corresponding to the consequent of *AR*.

Table 2: Abstract classes for DE and DFD rules

DE rule	`[ a owl:Class; owl:intersectionOf ( [ a owl:Restriction; owl:onProperty pde:hasCtrldObject; owl:onClass owl:oneOf (:o) ] [ a owl:Restriction; owl:onProperty pde:hasCryptoType; owl:onClass owl:oneOf (:crypto1) ] ) ]`
DFD Rule	`[a owl:Class; owl:intersectionOf ([a owl:Restriction; owl:onProperty pdfd:hasCtrldObject; owl:onClass owl:oneOf (:o)] [a owl:Restriction; owl:onProperty pdfd:hasDataFragmentation; owl:onClass owl:oneOf (:fScheme)] [a owl:Restriction; owl:onProperty pdfd:hasDataDistribution; owl:onClass owl:oneOf (:Frankfurt, :Dublin)])]`

Suppose, for instance, the DE and DFD rules of Table 1; we want to check whether the DFD rule abides by the DFDE AR of Example

Table 3: Abstract classes for the AR of Example 4.1

antecedent	[a owl:Class; owl:intersectionOf ([a owl:Restriction; owl:onProperty pde:hasCtrldObject; owl:onClass :CriticalObject] [a owl:Restriction; owl:onProperty pde:hasCryptoType; owl:onClass owl:oneOf (:crypto1)])]
consequent	[a owl:Class; owl:intersectionOf ([a owl:Restriction; owl:onProperty pdfd:hasCtrldObject; owl:onClass :CriticalObject] [a owl:Restriction; owl:onProperty pdfd:hasDataDistribution; owl:onClass owl:oneOf (:EU)])]

4.1. The reasoner first checks whether there exists a DE or DFD rule that is amenable to the DFDE AR, i.e. whether there exists a DE or DFD rule that refers to `:o` (i.e. the sensitive object associated with the DFD rule of Table 1) and whose corresponding abstract class is a subclass of the antecedent of the DFDE AR. This is true as the abstract class of the first row of Table 2 is indeed a subclass of the abstract class that corresponds to the antecedent of the DFDE AR (see Table 3). The reason for this is that the former abstract class demands an association, through the property `pde:hasCtrldObject`, with the class that comprises just the individual `:o`, whereas the latter abstract class demands an association, through the same property, with the entire `:CriticalObject` class (which encompasses `:o`). The DFDE AR is thus *enforceable* as the data encryption scheme specified by its antecedent has already been imposed upon `:o` by the DE rule of Table 1. The reasoner thus proceeds to determine whether the abstract class of the second row of Table 2 (which corresponds to the DFD rule of Table 1) is a subclass of the abstract class that corresponds to the consequent of the DFDE AR (see Table 3). If it is, we may conclude that the DFD rule of Table 2 indeed abides by the DFDE AR of Example 4.1; otherwise, a DFDE AR violation is detected.

5.2 Semantic Inferencing During DFDE AR Enforcement

One of the benefits of representing DE and DFD rules ontologically is the fact that we can draw upon the knowledge artefacts that are associated with these rules in order to *semantically* infer *new knowledge*. This potentially facilitates the enforcement of DFDE ARs upon DE and DFD rules as now the enforcement may take place in situations in which the knowledge artefact values that are associated with the antecedent and consequent of a DFDE AR do not (syntactically) match the values of the corresponding knowledge artefacts encoded in the DE and DFD rules.

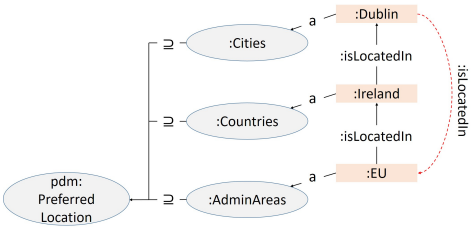

Figure 7: CM semantic inferencing

Suppose, for instance, the DFDE AR of Example 4.1 and let the DFD rule of Table 1. The knowledge artefacts associated with the consequent of the DFDE AR do not match, at the syntactic level, the knowledge artefacts that are encoded in the DFD rule: the former specifies as a location of distribution an administrative area (the EU area), whereas the latter specifies two cities: Dublin and Frankfurt. However, these cities do belong to EU countries and therefore to the EU area; this information is reflected in the CM by extending the `pdm:PreferredLocation` concept with such concepts as `:AdminAreas`, `:Country` and `:City`, as well as with the transitive object property `:isLocatedIn` (see Fig. 7). Note that these concepts are included in `pdm:PreferredLocation` during the process of *priming* the CM—a process that aims at customising the CM for the needs of a particular domain of application. Thus, when the DFD rule of Table 1 is specified, a number of facts regarding `:o`'s distribution location can be semantically inferred automatically, through the use of Pellet. In particular, as depicted in Fig. 7, from the premise that a fragment of `:o` is associated with, and therefore located in, Dublin we can infer that the same fragment is also associated with, and therefore located in, Ireland and also is associated with, and therefore located in, the EU. An entirely symmetrical reasoning naturally applies to the Frankfurt location. We can thereby infer the new knowledge that the DFD rule of Table 1 distributes fragments of `:o` in the EU area only and therefore abibes by the DFDE AR of Example 4.1. Clearly, such semantic inferencing absolves developers from having to specify fine-grained axiomatic rules that cover each permissible location of data distribution articulated in a DFD policy. This generally facilitates the process of formulating suitable security policies for a particular domain of application.

6 RELATED WORK

A number of approaches have been proposed for the semantic representation of policies [7, 9, 12]. These generally rely on OWL [17] for capturing the various knowledge artefacts that underpin the definition of a policy. In [12] KaoS is presented—a generic framework offering: (i) a human interface layer for the expression of policies; (ii) a policy management layer that is capable of identifying and resolving conflicting policies; (iii) a monitoring and enforcement layer that encodes policies in a programmatic format

suitable for enforcing them. A limitation of KaoS approach is that the programmatic translation of policies precludes, from the outset, the performance of any updates to the policies dynamically, i.e. during system execution, as such updates would naturally require that policies are re-compiled to the programmatic format.

In [7] Rei is proposed, a framework for specifying, analyzing and reasoning about policies. Rei adopts OWL-Lite [18] for the semantic specification of policies. A policy comprises a list of rules that take the form of OWL properties, as well as a context that defines the underlying policy domain. Rei provides a suitable ontological abstraction for the representation of desirable behaviours that are exhibited by autonomous entities. Rei resorts to the use of placeholders as in rule-based programming languages for the definition of *variables*. This, however, essentially prevents Rei from exploiting the full inferencing potential of OWL as policy rules are expressed in a formalism that is alien to OWL. In contrast, variables could have instead been modelled in terms of OWL's anonymous individuals.

In [9] the authors propose POLICYTAB for facilitating trust negotiation in Semantic Web environments. POLICYTAB adopts ontologies for the representation of policies that guide a trust negotiation process ultimately aiming at granting, or denying, access to sensitive Web resources. These policies essentially specify the credentials that an entity must possess in order to carry out an action on a sensitive resource that is under the ownership of another entity. Nevertheless, no attempt is made to semantically model the context associated with access requests, rendering POLICYTAB inadequate for dynamic and heterogeneous cloud environments.

On a different note, the markup languages [3, 10, 11] provide declarative formalisms for the specification of policies. Nevertheless, they do not provide any means of capturing the knowledge that dwells in policies: they are simple data models that do not provide any semantic agreement beyond the boundaries of the organisations that adopted them. Any interoperability therefore hinges necessarily upon the use of vocabularies that are shared among the parties involved in an interaction. This leads to ad-hoc reasoning about the abidance of policies by rules concerning their correctness, it restricts the portability and reusability of policies, it prohibits the identification of relations among policies (e.g. contradiction, subsumption, etc.) and, finally, it restricts the ability to generically perform rule-based policy lifecycle governance.

7 CONCLUSIONS

Axiomatic rules represent an important class or rules that restrict the allowable values that one or more knowledge artefacts encoded in a security policy may assume on the basis of the values assigned to certain other knowledge artefacts. This work has proposed an approach for the representation of axiomatic rules for DE and DFD policies as reifications of abstract *ontological templates*. These templates are underpinned by the Context Model—an ontological representation of the knowledge artefacts encoded in the security policies. This enables—by virtue of *semantic inferencing*—the generation of *new knowledge artefacts* that potentially allow the application of axiomatic rules in situations in which the knowledge artefacts encoded in a security policy do not syntactically match the corresponding knowledge artefacts encoded in the axiomatic rules.

As part of future work we intend to construct an editor that will provide two main functionalities: firstly, it will facilitate the expression of axiomatic rules and, secondly, it will facilitate the formulation of security policies through the application of these axiomatic rules. More specifically, regarding the latter functionality, each time a knowledge artefact is embodied in a security policy, the axiomatic rules will be applied in order to provide all allowable values, or range of values, that this knowledge artefact 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] 2014. RDF 1.1 Turtle. (February 2014). https://www.w3.org/TR/turtle/.
[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] J. Callas, L. Donnerhacke, and D. Shaw. 2007. *OpenPGP Message Format.* https://tools.ietf.org/pdf/rfc4880.pdf.
[5] Clark & Parsia, LLC 2011. *Pellet.* Clark & Parsia, LLC. https://www.w3.org/2001/sw/wiki/Pellet.
[6] 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/.
[7] L. Kagal, T. Finin, and Anupam 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
[8] F Liu, J Tong, J Mao, R Bohn, J Messina, L Badger, and D Leaf. 2011. *NIST Cloud Computing Reference Architecture.*
[9] 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
[10] 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.
[11] 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.
[12] 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.
[13] Simeon Veloudis and Iraklis Paraskakis. 2015. *Access Policies Model.* PaaSword Project Deliverable D2.2.
[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] Simeon Veloudis and Iraklis Paraskakis. 2016. Ontological Templates for Modelling Security Policies in Cloud Environments. In *Proceedings of the 20th Pan-Hellenic Conference on Informatics (PCI '16).* ACM, New York, NY, USA, Article 65, 6 pages. DOI: https://doi.org/10.1145/3003733.3003796
[16] 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.
[17] W3C 2004. *W3C Recommendation. 2004. OWL Web Ontology Language Reference.* W3C. https://www.w3.org/TR/owl-ref/.
[18] W3C 2004. *W3C Recommendation. 2004. OWL Web Ontology Language Semantics and Abstract Syntax.* W3C. https://www.w3.org/TR/2004/REC-owl-semantics-20040210/.
[19] W3C 2012. *W3C Recommendation. 2012. OWL 2 Web Ontology Language Document Overview (Second Edition).* W3C. https://www.w3.org/TR/owl2-overview/.