2017 IEEE 30th International Symposium on Computer-Based Medical Systems

# Ontological Templates for Regulating Access to Sensitive Medical Data in the Cloud

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

Yiannis Verginadis, Ioannis Patiniotakis, Gregoris Mentzas
Institute of Communications and Computer Systems
National Technical University of Athens
Athens, Greece
{jverg,ipatini,gmentzas}@mail.ntua.gr

***Abstract*—By embracing the cloud computing paradigm for storing and processing electronic medical records (EMRs), modern healthcare providers are able to realise significant cost savings. However, relinquishing control of sensitive medical data by delegating their storage and processing to third-party cloud providers naturally raises significant security concerns. One way to alleviate these concerns is to devise appropriate *policies* that infuse adequate access controls in cloud services. Nevertheless, the heterogeneous nature of these services, coupled with the dynamicity inherent in cloud environments, hinder the formulation of effective and interoperable policies that are appropriate for the underlying domain of application. To this end, this work adopts the *ontological templates* proposed in [5] for the representation of access control policies in the medical sector. By capturing the *knowledge* that must be infused into an access control policy, these templates sufficiently address the needs of the underlying domain of application in which such a policy is to be enforced; at the same time, they facilitate developers in infusing adequate access controls to their cloud applications.**

***Keywords- Context-aware access control; Electronic medical records; Security-by-design; Ontologies***

## I. INTRODUCTION

Modern healthcare providers use electronic medical records (EMRs) [1] in order to conveniently create, access and govern medical data and share them with relevant stakeholders (e.g. patients, insurance companies, medical researchers, etc.). In current practice, EMRs are typically stored and processed in dedicated data centres that are established and maintained by the healthcare providers themselves. Nevertheless, as the amount of medical data proliferates, this practice incurs significant costs [2]. A viable solution for reducing these costs is to migrate the storage and processing of EMRs to the *cloud*. The reasons for this are as follows. Cloud computing enables ubiquitous access to shared pools of distributed, configurable and diverse computing resources that range from infrastructural ones—e.g. storage space for persisting EMRs, computational power for processing EMRs—to software resources that offer a multitude of operations on EMRs. These resources are abstracted as services and delivered to the healthcare providers remotely, over the Internet, with a theoretically boundless scalability and in a flexible and cost-effective pay-per-use manner [3]. Moreover, these resources are under the control of their third-party providers, hence absolving healthcare providers from the costs of managing and maintaining them.

However, relinquishing control of sensitive medical data by delegating their storage and processing to third-party cloud providers naturally raises significant security concerns [4]. Clearly, if healthcare providers are to embrace the cloud paradigm and benefit from the cost reductions that it brings about, these concerns must be alleviated. To this end, appropriate *security policies* must be infused into the applications through which EMRs are stored and accessed in the cloud. For example, access control policies are required that take into account the inherently dynamic and unpredictable nature of cloud environments by enabling the articulation of all those *contextual attributes* that need to be satisfied (e.g. the location from which an access request originates, the time of access, the identity or role of the subject that issues the request, etc.) in order to grant, or deny, access to sensitive data.

The work conducted as part of the PaaSword project [5] offers a suitable 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. It proposes a novel approach to policy modelling, one which formulates policies as reifications of abstract *ontological templates* that semantically capture the knowledge that lurks behind policies. It therefore advocates a clear separation of concerns by unravelling the representation of policies from the code that is employed for enforcing them. This brings about the following seminal advantages. Firstly, it enables—by virtue of *semantic inferencing*—the generation of *new knowledge* on the basis of the knowledge already encoded in the policies, and can therefore successfully tackle situations in which the contextual information included in an access request does not necessarily match, at the syntactic level, the corresponding information encoded in the policies. For example, if a policy states that a sensitive data object (say o) is only readable by requests that originate from within a location A, then a request that originates from a location B that is contained within A, will be permitted to read o, as semantic inferencing allows us to determine that the request indeed originates from location A. This potentially absolves developers from the burden of having to specify fine-grained

2372-9198/17 $31.00 © 2017 IEEE
DOI 10.1109/CBMS.2017.154

policies that cover each possible location within A that a request may originate from. Secondly, it enables automated reasoning about potential *inter-policy relations* such as *subsumption* and *contradiction*, as well as about *the well-formedness* of policies, i.e. whether policies incorporate all required knowledge artefacts for protecting sensitive data in the cloud.

This paper demonstrates, with the aid of a short case study, how PaaSword's ontological templates may facilitate the process of expressing access control policies for protecting EMRs in the cloud. The rest of this paper is structured as follows. Section II outlines the Context Model—an ontological representation of the various knowledge artefacts associated with an access control policy, and Section III outlines our ontological model for access control policies. Section IV demonstrates, with the aid of a simple case study, how our ontological models can be used for: (i) regulating access to EMRs; (ii) performing semantic inferencing during policy enforcement; (iii) reasoning about inter-policy relations and about the well-formedness of policies. Finally, Section V summarises related work and Section VI presents conclusions and future work.

## II. Modelling Context

The Context Model (CM) proposed in [6] provides an ontological model for the representation of the various knowledge artefacts that lurk behind an access control policy. Fig. 1 depicts a simplified view of the CM that includes only the concepts (classes) considered in this work. At the core of the CM is the class `pcm:SecurityContextElement`. This class encompasses concepts that represent the various *contextual attributes* that may be associated with an *access request*; these concepts are depicted in the shaded area of Fig. 1 and are further outlined in Table II.

Now, an access request takes the form of an instance of the class `pcm:Request`; it is associated with the contextual attributes that characterise it through the object property `pcm:hasAttribute` (see Fig. 1). The same object property is used for associating an access request with its *subjects* and *objects*. A subject of a request is any entity (human or machine) whose contextual attributes must be taken into account for deciding whether to permit, or deny, the request;

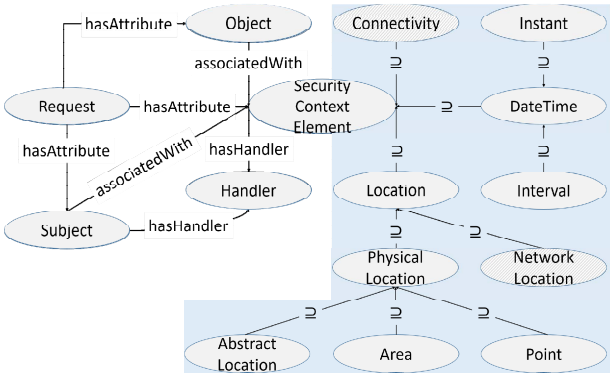


Figure 1: Context Model (namespaces omitted to reduce clutter)

TABLE I: Security context element Concepts

| | |
|---|---|
| `pcm: Location` | An instance of this class describes a physical or a network location that characterises any entity whose whereabouts must be considered for deciding whether to permit, or deny, an access request. It includes the classes `pcm:PhysicalLocation` and `pcm:NetworkLocation`. The former comprises the concepts `pcm:Address`, `pcm:Point`, `pcm:Area` and `pcm:AbstractLocation`. Instances of `pcm:Point` and `pcm:Area` are further associated, through suitable data properties, with their corresponding geographical coordinates (not shown in Fig. 1). The class `pcm:AbstractLocation` bundles together such locations as particular buildings, offices, medical laboratories, etc. Network locations are not considered in this work and hence the class `pcm:NetworkLocation` is not further analysed here—the interested reader is referred to [6] for more details. |
| `pcm: DateTime` | An instance of this class describes the specific chronological point or time interval at which an access request is received. |
| `pcm: Connectivity` | Captures information pertaining to the connection or the type of device (e.g. desktop, smart phone, tablet, etc.) used by an entity for accessing sensitive data. Such information is not considered in this work and hence this class is not further analysed here—the interested reader is referred to [6] for more details. |

an object of a request is a resource (e.g. relational database table, non-relational data store, file, etc.) targeted by the request. Subjects are represented as instances of the class `pcm:Subject`; they are associated with the contextual attributes that characterise them through the object property `pcm:associatedWith` (see Fig. 1). Objects are represented as instances of the class `pcm:Object`; they too are associated with the contextual attributes that characterise them through the object property `pcm:associatedWith`.

Fig. 1 also includes the concept `pcm:Handler` whose instances represent dedicated software components—the so-called *handlers*—that interface with hardware sensors and provide the current values of the contextual attributes attached to a request (or to the subjects or objects of a request). Handlers may include, for example, authentication handlers, physical location resolution handlers, IP address to physical location handlers, etc. A contextual attribute is associated with its corresponding handler(s) through the property `pcm:hasHandler`; the same property is used for associating subjects with handlers (for example, authentication handlers that provide the identity of subjects).

## III. Modelling Access Control Policies

We consider Attribute-based Access Control (ABAC) policies which, due to their inherent generality stemming from their reliance on the generic concept of an *attribute*, are deemed suitable for dynamic and heterogeneous cloud environments [7]. Following the XACML standard [8], each ABAC policy comprises one or more *ABAC rules*. An ABAC rule is associated with a set of relevant knowledge artefacts, or attributes, whose values need to be taken into

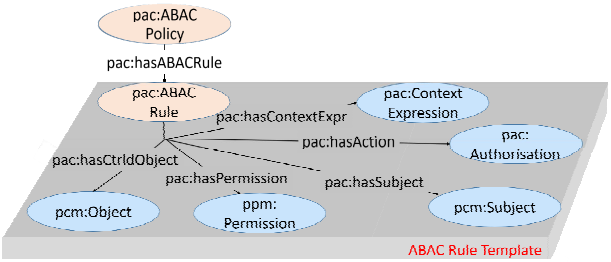


Figure 2: ABAC Policy Model

account when deciding whether to permit, or deny, an access request. These attributes are described abstractly in terms of the concepts introduced by the CM.

In the ontological representation proposed in [7] and depicted in Fig. 2, an ABAC rule takes the form of an instance of the class `pac:ABACRule`; the knowledge artefacts attached to the rule are described generically in terms of the depicted *ABAC rule template*. More specifically, each class of this template identifies a particular knowledge artefact, whilst each object property attaches such a knowledge artefact to the rule; Table III briefly elaborates on the concepts and properties of this template; in the remaining of this section we focus on a particular concept, namely that of a *context expression*.

A context expression is represented as an instance of the class `pac:ContextExpression` (see Fig. 3). The various attributes that it binds, i.e. its *parameters*, are represented as instances of the CM—in particular, as instances of the subclasses of the `pcm:SecurityContextElement` class (see Table I). These parameters are associated with the context expression through the object property `pac:hasParameter`; they may be combined with each other through the usual logical connectives; to this end, the classes `pac:XContextExpression` (where `X` stands for one of `AND`, `OR`, `XOR`, `NOT`) are introduced as subclasses of the `pac:ContextExpression` class (see Fig. 3). Their intended meaning is as follows: if a context expression is represented by an instance of the class say `pac:ANDContextExpression`, its parameters, i.e. the contextual attributes associated with it through the `pac:hasParameter` property, are interpreted as being pairwise conjuncted; analogous interpretations apply to the rest of the subclasses of `pac:ContextExpression`.

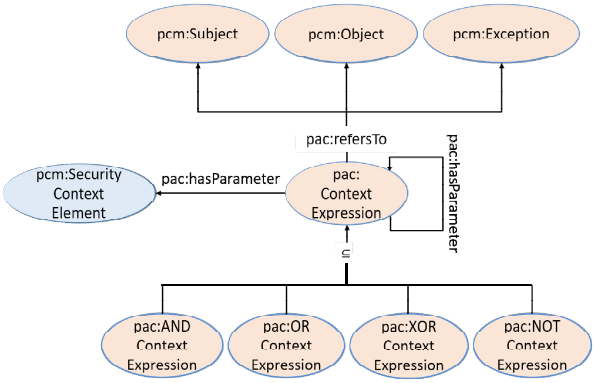


Figure 3: Context Expressions Ontological Template

A context expression may be defined recursively, in terms of one or more other context expressions. Ontologically, this is captured by including the class `pac:ContextExpression` in both the domain and the range of the object property `pac:hasParameter` (see Fig. 3). The `pac:refersTo` property depicted in Fig. 3 attaches a context expression to the entity (either subject, object or access request) that it refers to.

## IV. A Simple Case Study Scenario

We next demonstrate how access control policies devised to protect EMRs can be expressed as reifications of the ontological representation of ABAC rules outlined above. We then demonstrate how this ontological representation can form the basis for performing *semantic inferencing*—hence generating new knowledge—during policy enforcement, as well as for identifying *inter-policy relations* and for checking the *well-formedness* of ABAC rules.

Suppose a fictitious healthcare provider, call it $HCP_x$, that is interested in migrating to the cloud the EMRs that it currently hosts on proprietary data servers. We assume that the EMRs are persisted in the cloud in a relational database table identified as HCPx_EMR. A number of access control policies for safeguarding HCPx_EMR are required. These policies generally reflect $HCP_x$'s stance towards security and are also assumed to be in line with relevant governmental rules and regulations (e.g. the EU's directive 2016/680 regarding the protection of natural persons with regard to the processing of personal data [9]). Let us initially

TABLE II: Generic knowledge artefacts associated with the ABAC Rule Template

| **Knowledge artefact** | **Descriptions** | **Ontological representation** |
|---|---|---|
| Controlled object | The resource on which access is requested. | Instance of the class `pcm:Object` (see Fig. 1). |
| Authorisation | The kind of authorisation granted (either 'permit' or 'deny') | Instance of the class `pac:Authorisation` (see Fig. 2) which comprises the individuals `pac:permit` and `pac:deny`. |
| Action | The action (either 'read', 'write' or 'read/write) to be performed on the controlled object | Instance of the class `ppm:DataPermission` (see Fig. 2) which comprises the individuals `ppm:read` and `pac:write`. |
| Actor | The subject issuing an access request | Instance of the class `pcm:Subject` (see Fig. 1). |
| Context expression | A propositional logic expression that must be satisfied in order to permit (or deny) an access request; it binds together a number of contextual attributes expressed as instances of the CM | Instance of the class `pac:ContextExpression` (see Fig. 3). |

TABLE III: $HCP_x$ POLICY RULE

| | |
|---|---|
| Rule 1 | HCPx_EMR can be written by doctors during working hours and only from within 'building1'[a]. |

a. 'building1' is assumed to belong to the premises of $HCP_x$

assume a simple policy comprising the rule shown in TABLE III and let us demonstrate how this rule is expressed as a reification of the ontological representation outlined in Section III.

### A. *Modelling $HCP_x$ Policies*

Table IV specifies (using the RDF Turtle notation [10]) the rule of Table III. The rule itself is represented by the instance `:Rule1` of the class `pac:ABACRule`; the subject of the rule is represented by the instance `:s` which is further associated through the property `pac:hasRole` with the role 'doctor'. `pac:hasRole` is introduced by the CM in order to associate subjects (i.e. instances of the class `pcm:subject`) with roles (i.e. instances of the class `usdl-core:Role`—a class adopted from the usdl-core ontological framework [11] for capturing the various roles that a subject may assume). The authorisation that the rule grants is represented by the instance `pac:permit` (see Table II), whereas the action that the rule permits is represented by the instance `pac:write`. Finally, the context expression of the rule is represented by the instance `:expr1` and is associated with two parameters: the instance `:bldg1` which belongs to the class `pcm:AbstractLocation` and which represents 'building1' (see Table III), and the instance `:workingHours` which belongs to the class `pcm:DateTimeInterval` and which is delimited, through the data properties `pcm:hasBeginning` and `pcm:hasEnd`.

### B. *Semantic Inferencing During Policy Enforcement*

The concepts and properties introduced by the CM (see Fig. 1) may be exploited during the evaluation of an access request in order to *semantically infer* the *context* that is associated with the request (or with the subject and/or object of the request). Suppose, for example, that a subject entity represented by the instance `:s` and acting in the capacity of the role 'doctor' issues an access request to the HCPx_EMR table; the request is assumed to be issued from 'room1001' which is located on the 1st floor of 'building1'. Note that this $HCP_x$-specific location information must be reflected in the CM: we assume that during the process of priming the CM—a process that aims at rendering the CM suitable for the needs of $HCP_x$—the following concepts are introduced as subclasses of the class `pcm:AbstractLocation` (see Fig. 4): `:HCPx_Room`, `:HCPx_Floor`, `:HCPx_Building`; moreover, the individuals `:Room_1001`, `:Floor_01` and `:Bldg_01` are also defined as instances of these classes and interconnected through the object property `pcm:associatedWith` (see Fig. 4 and Table V). In addition, we assume that, based on the available handlers, the system is capable of collecting location information only at the level of rooms (and not at the level of buildings or floors).

TABLE IV: ABAC POLICY RULE 1

```
:Rule1 a pac:ABACRule;
   pac:hasCtrldObject :HCPx_EMR;
   pac:hasActor :s;
   pac:hasAuthorisation pac:permit;
   pac:hasAction pac:write;
   pac:hasContextExpression :expr1.
:HCPx_EMR a pcm:Object.
:s a pcm:Subject;
   pac:hasRole :doctor.
:doctor a usdl-core:Role.
:expr1 a pac:ANDContextExpression;
   pac:hasParameter :bldg1;
   pac:hasParameter :workingHours
   pac:refersTo :s;
:bldg1 a pcm:AbstractLocation;
:workingHours a pcm:DateTimeInterval;
   pcm:hasBeginning "08:00"^^xsd:dateTime;
   pcm:hasEnd "16:00"^^xsd:dateTime;
```

Once the request issued by the subject `:s` is intercepted with the resolved location for `:s` being reported as 'room1001', a number of facts regarding `:s`'s location can be *semantically inferred* automatically, through the use of an OWL 2 DL reasoner such as Pellet [12]. This inferencing is based on the *transitivity* of the object property `pcm:associatedWith`, as well as of the subclass relation. In particular, from the premise that `:s` is associated with, and therefore located in, 'room1001' we can infer that `:s` is also associated with, and therefore located on, 'floor01' (since 'room1001' is associated with 'floor01') and similarly that `:s` is also associated with, and therefore located in, 'building1' (since 'floor01' is associated with 'building1'). These inferred facts essentially render the evaluation, hence the application, of the access control policy feasible, as the system is able to determine that the requestor is actually located in 'building1', even though the intercepted contextual information reports that the requestor is located in a room.

### C. *Determining Inter-Policy Relations*

One of the main advantages brought about by expressing access control policies declaratively, in terms of the ontological model outlined in Section III, is the ability to automatically reason, through the use of a DL reasoner, about inter-policy relations such as *subsumption* and

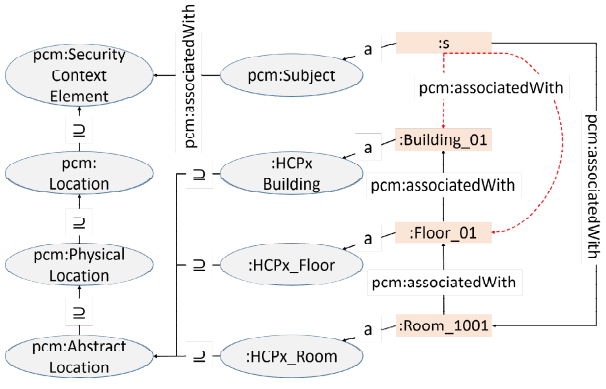


Figure 4: Semantic Inferencing at the level of the CM

TABLE V: ABAC POLICY

```
:Room_1001 a :HCPx_Room;
pcm:associatedWith :Floor_01.
:Floor_01 a :HCPx_Floor;
pcm:associatedWith :Building_01.
:Building_01 a :HCPx_Building.
:HCPx_Room rdfs:subClassOf
           pcm:AbstractLocation.
:HCPx_Floor rdfs:subClassOf
           pcm:AbstractLocation.
HCPx_Building rdfs:subClassOf
           pcm:AbstractLocation.
```

*contradiction*.

With regard to policy subsumption, this reasoning is based on semantic inferencing that takes place either at the level of the CM, such as the inferencing outlined in Section IV.B, or at the level of the actual ABAC rule model. Regarding inferencing at the level of the CM, suppose Rule 2 of Table VI (it is assumed that all doctor offices are located on 'floor01' of 'building1'). An inferencing process analogous to the one outlined in Section IV.B allows us to conclude that Rule 2 is, in fact, a special case of Rule 1 of Table III: any request that is permitted by Rule 2 is also permitted by Rule 1. It is therefore concluded that Rule 2 is subsumed by Rule 1. In this respect, it would be meaningless to retain both rules in the rule repository: either Rule 2 is specified by mistake and should be dropped from the system, or Rule 2 is intended as a replacement of Rule 1 (in an attempt, for example, to introduce a more restrictive regime) in which case Rule 1 should be retired from the system.

Regarding inferencing at the level of the ABAC rule model, suppose Rule 3 of Table VI. This rule is represented ontologically as shown in Table VII. We observe that, with the exception of the context expression, all knowledge artefacts associated with the representation of Rule 3 (i.e. with the instance `:Rule3`) are identical to the ones associated with Rule 1 (see Table IV). We also observe that the context expression of Rule 3 is associated with the same parameters as the ones of the context expression of Rule 1; in addition, these parameters refer in both context expressions to the same entity—the subject instance `:s`. Nevertheless, the parameters of `:expr3` are logically *disjuncted* as opposed to the ones of `:expr1` which are logically conjuncted. This clearly renders `:expr3` a more general expression than `:expr1` and hence *inferable* from `:expr1`. In other words, any request that is permitted by Rule 1 is inevitably also permitted by Rule 3. It is therefore concluded that Rule 1 is subsumed by Rule 3.

With regard to policy contradiction, two ABAC rules are considered contradicting when they are associated with identical knowledge artefacts but the one yields a 'permit' decision whereas the other one yields a 'deny' decision. Clearly, a DL reasoner can be employed to detect whether two ABAC rules are contradicting.

TABLE VI: $HCP_x$ ADDITIONAL POLICY RULES

| | |
|---|---|
| Rule 2 | HCPx_EMR can be written by doctors during working hours and only from 'floor01'. |
| Rule 3 | HCPx_EMR can be written by doctors during working hours or from within 'building1'. |

TABLE VII: ABAC POLICY RULE 3

```
:Rule3 a pac:ABACRule;
   pac:hasCtrldObject :HCPx_EMR;
   pac:hasActor :s;
   pac:hasAuthorisation pac:permit;
   pac:hasAction pac:write
pac:hasContextExpression :expr3.
:expr3 a pac:ORContextExpression;
   pac:hasParameter :bldg1;
   pac:hasParameter :workingHours
   pac:refersTo :s;
```

### D. Determining the Well-Formedness of Rules

Another crucial advantage brought about by expressing access control policies declaratively, in terms of the ontological model outlined in Section III, is the ability to automatically reason about the *well-formedness* of the policy rules. This reasoning is performed by a DL reasoner through a series of automated checks that aim at assessing the well-formedness, hence the validity, of a policy with respect to *a higher-level ontology* (HLO) that captures a set of meta-policies that essentially articulate all those ingredients that a policy rule may, or may not, comprise. These correctness checks are clearly of utmost importance for they increase assurance on the effectiveness of the policies. For example, in the case of $HCP_x$ policies, we might be interested in specifying constraints such as the ones outlined in Table VIII. These constraints are ontologically expressed in the HLO and enforced each time a new policy rule is created or an existing policy rule is updated.

## V. RELATED WORK

A number of approaches have been proposed for the representation of policies [13—15]. These generally rely on the expressivity of OWL [16] for capturing the various knowledge artefacts that underpin the definition of a policy. In [13] 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 suitable programmatic format for enforcing them. Contextual conditions that must be taken into account in access control decisions are expressed as OWL property restrictions. A main drawback of the KAoS approach is the fact that its reliance on OWL raises concerns about the efficiency with

TABLE VIII: $HCP_x$ ADDITIONAL POLICY RULES

| |
|---|
| Each policy rule must be associated with *exactly one* subject, *exactly one* controlled object, *exactly one* kind of authorisation (either permit or deny), *exactly one* type of action (either read or write) and *at most one* context expression. |
| Any location attribute that forms a parameter of a context expression must be an instance of the class `HCPx_Building` (see Fig. 4). |

which semantic inferencing can be performed dynamically, when policies are evaluated against incoming access requests. In order to alleviate these concerns, KAoS encodes policies in a programmatic format. Nevertheless, this precludes the performance of any updates to the policies dynamically, during system execution, as such updates would naturally require the (updated) policies to be re-compiled to the programmatic format.

In [14] Rei is proposed – a framework for specifying, analysing and reasoning about policies. Rei adopts OWL-Lite [17] 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 [15] 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 model the context associated with access requests.

On a different note, the markup languages [18,8,19] provide declarative formalisms for the specification of policies. Nevertheless, they do not provide any means of capturing the knowledge that dwells in policies.

## VI. Conclusions

This paper has proposed a novel approach for modelling access control policy rules. We argue that our approach facilitates developers in expressing effective policies which give rise to security controls appropriate for dynamic and heterogeneous cloud environments. The approach is founded on the basis of an ontological template that captures a wide range of contextual attributes that must be taken into account during the evaluation of a policy. One of the virtues of the proposed ontological template is that it enables the evaluation of a request against an access control policy to be performed, and reasoned about, at the *semantic level*; furthermore, our ontological template paves the way for the performance of automated reasoning about potential *inter-policy relations*, such as the identification of subsuming or contradicting policies, as well as about the well-formedness, hence the effectiveness, of the policies.

Currently, we are in the process of finalising the mechanism that is able to reason about inter-policy relations. A policy validator that assesses the well-formedness of policies against the constraints expressed in the HLO is also being finalised.


## Acknowledgements

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] C. DesRoches et al., "Electronic health records in ambulatory care: a national survey of physicians," *New England Journal of Medicine*, 359(1):50–60, 2008.

[2] R. Wu, G. J. Ahn and H. Hu, "Secure sharing of electronic health records in clouds," *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, Pittsburgh, PA, 2012, pp. 711-718.

[3] Cloud Computing Reference Architecture, National Institute of Standards and Technology (NIST), 2011.

[4] R. Zhang and L. Liu. Security models and requirements for healthcare application clouds. In *Proceedings of 3rd IEEE International Conference on Cloud Computing*, pages 268–275. IEEE, 2010.

[5] PaaSword project, http://www.paasword.eu/.

[6] PaaSword Deliverable 2.1. https://www.paasword.eu/deliverables/.

[7] Simeon Veloudis and Iraklis Paraskakis, "Defining an Ontological Framework for Modelling Policies in Cloud Environments," In *Proceedings of 8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom'16*, 2016, in press.

[8] eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. OASIS Standard. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[9] EU 2016/680. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016L0680

[10] RDF 1.1 Turtle, 2014. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[11] Linked USDL, http://www.linked-usdl.org/.

[12] Pellet, http://clarkparsia.com/pellet.

[13] Uszok, A., Bradshaw, J., Jeffers, R., Johnson, M., Tate, A., Dalton, J., and Aitken, S.: KAoS Policy Management for Semantic Web Services. IEEE Intel. Sys. 19, 4, 32—41, 2004.

[14] Kagal, L., Finin, T., Joshi, A.: A Policy Language for a Pervasive Computing Environment. In 4th IEEE Int. Workshop on Policies for Distributed Systems and Networks (POLICY '03), pp. 63--74, IEEE Computer Society, Washington, DC, 2003

[15] Nejdl, W., Olmedilla, D., Winslett, M, Zhang. C.C.: Ontology-Based policy specification and management. In Gómez-Pérez, A. and Euzenat, J. (eds.) ESWC'05, pp. 290-302, Springer-Verlag, Berlin, Heidelberg, 2005.

[16] OWL 2 Web Ontology Language Primer (2nd Edition), https://www.w3.org/TR/owl2-primer/.

[17] OWL Web Ontology Language Overview, https://www.w3.org/TR/2004/REC-owl-features-20040210/#s2.1.

[18] Specification of Deliberation RuleML 1.01, http://wiki.ruleml.org/index.php/Specification_of_Deliberation_RuleML_1.01.

[19] Security Assertions Markup Language (SAML) Version 2.0. Technical Overview, https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf.