Achieving Security-By-Design Through Ontology-Driven Attribute-Based Access Control in Cloud Environments

Simeon Veloudis[1], Iraklis Paraskakis[1], Christos Petsos[1], Yiannis Verginadis[2], Ioannis Patiniotakis[2], Panagiotis Gouvas[3] and Gregoris Mentzas[2]

[1] *South East European Research Centre (SEERC), The University of Sheffield, International Faculty CITY College, Thessaloniki, Greece*

[2] *Institute of Communications and Computer Systems, National Technical University of Athens, Athens, Greece*

[3] *Ubitech, Athens, Greece*

{sveloudis, iparaskakis, chpetsos}@seerc.org, {jverg, ipatini, gmentzas}@mail.ntua.gr, pgouvas@ubitech.eu

Abstract: The constantly increasing number of cyberattacks worldwide raise significant security concerns that generally deter small, medium and large enterprises from adopting the cloud paradigm and benefitting from the numerous advantages that it offers. One way to alleviate these concerns is to devise suitable *policies* that infuse adequate access controls into cloud services. However, the dynamicity inherent in cloud environments, coupled with the heterogeneous nature of cloud services, hinders the formulation of effective and interoperable access control policies that are suitable for the underlying domain of application. To this end, this work proposes an approach to the *semantic representation* of access control policies and, in particular, to the semantic representation of the *context expressions* incorporated in such policies. More specifically, the proposed approach enables stakeholders to accurately define the *structure* of their policies, in terms of relevant knowledge artefacts, and thus infuse into these policies their particular security and business requirements. This clearly leads to more effective policies, whilst it enables *semantic reasoning* about the abidance of policies by the prescribed structure. In order to alleviate the scalability concerns associated with semantic reasoning, the proposed approach introduces a reference implementation that extends XACML 3.0 with an *expert system* fused with reasoning capabilities through the incorporation of suitable meta-rules.

Keywords: Context-aware security, Ontologies, Access Control Policies, Data privacy, Security-by-design, Semantic reasoning.

1 INTRODUCTION

Enterprises increasingly embrace the cloud computing paradigm in order to gain access to a wide range of infrastructure, platform, and application resources that are abstracted as services and delivered remotely, over the Internet, by diverse providers. The main force that fuels this trend is the significant cost savings that these services instigate, as well as the acceleration in the development and deployment of new applications that boosts innovation and productivity.

However, due to security concerns, many enterprises are reluctant to migrate their critical operations and sensitive data to the cloud [1]. A promising approach to alleviating these security concerns is to assist application developers in infusing adequate *access control policies* in cloud applications for safeguarding their data against unauthorised accesses [2]. In this respect, we envisage a generic security-by-design framework, essentially a PaaS offering, which facilitates developers in devising, and ultimately implementing, such policies. Nevertheless, in order for the policies to be effective, they must take into account the dynamically-evolving nature of cloud environments. In particular, they must take into account the *contextual information* that needs to be associated with an access request in order for it to be permitted or denied.

To this end, the work reported in [2] outlined the construction of an ontological model for access control policies, one that bears the following characteristics: it is underlain by a suitable Context-Aware Security Model – an extensible framework of interrelated concepts that capture a wide range of relevant *contextual attributes*, thus embracing the attribute-based access control (ABAC) scheme [3]; it uses a generic and extensible formalism that is able to capture the *knowledge* that lurks behind access control policies and thus unravel the definition of a policy from the code employed for enforcing it. These characteristics bring about the following seminal advantages: (i) they allow the policy-related knowledge to be extended and instantiated to suit the needs of *any*

particular cloud application, independently of the code employed by that application; (ii) they allow *new knowledge artefacts* to be *inferred* from existing ones thus enabling the enforcement of access control policies in situations in which the knowledge artefacts encoded in a policy do not necessarily match, at the syntactic level, the corresponding artefacts encoded in access requests. As an example, consider a policy whereby a particular subject (say `s`) is allowed to read a particular sensitive data object only when `s` issues the request from within the geographical area identified as South Europe. Suppose now that an access request is reported to be issued from within the city of Athens. Although the contextual information incorporated in the policy differs syntactically from the contextual information reported in the request, semantic inferencing allows us to conclude[1] that Athens is indeed located in Greece and thus `s` satisfies the contextual condition set by the policy; this clearly absolves application developers from having to specify fine-grained access control policies – i.e. policies that cover every permissible location of access.

Nevertheless, the ontological model devised in [2] suffers a number of limitations. Firstly, it assumes that the contextual attributes articulated in an access control policy are invariably associated with the subject of a request, ignoring the fact that contextual attributes may need to be associated with other entities such as the *object* of a request, the request itself, or any other entity that is deemed relevant for determining whether the request should be granted or denied. As an example, consider a policy whereby a particular subject (say `s`) is allowed to read a sensitive data object (say `o`) only when: `o` resides in a data centre in the EU; `s` issues the request from within a particular subnet (say `subnet1`); the request takes place during a specific time interval; another entity (say `s'`) resides in a particular geographical area – say `bldg1`. Evidently, in addition to the subject `s` of a request, this policy needs to attach context to the object `o` of a request (namely, the location of the object), to the request itself (namely, the time a request is issued) and to the entity `s'` (namely, the location in which `s'` resides).

Secondly, although the ontological model in [2] prescribes a set of *knowledge artefacts* on which access control policies, hence access control decisions, are based –one that invariably comprises the *subject* and *object* of a request, the *kind of access* sought (e.g. read or write access), and a body of *contextual attributes* pertaining to the subject of a request– it fails to specify the *cardinalities* with which these knowledge artefacts may be incorporated in a policy, or the allowable manners which they may be combined (e.g. conjunctively, disjunctively, etc.). For instance, it fails to specify whether an access control policy may be based on two or more subjects or objects, whether it may allow more than one kind of access, or whether it may incorporate one or more contextual attributes that are conjunctively –or for that matter disjunctively– combined. This essentially precludes the articulation of any *constraints* regarding the exact *structure*, or *form*, that an access control policy may assume; in other words, it precludes stakeholders[2] from infusing into access control policies their particular security and business requirements. For example, a stakeholder may wish to articulate that any access control policy that refers to a particular class of sensitive objects must be based on, hence incorporate, contextual attributes that provide information about the geographical location of the subject of a request, as well as about the time `t` at which the request is issued; any policy that does abide by this structure is considered not well-formed and thus invalid. The model in [2] does not support the articulation of such constraints.

In order to overcome these limitations, this paper extends the work presented in [4] by proposing a generalisation to the ontological model outlined in [2] that bears the following seminal characteristics. Firstly, it is able to attach context to *any* entity that is deemed relevant to a request at two distinct levels: (i) at the level of the access control policy, indicating the *contextual conditions* that must be satisfied by an entity in order for an access request to be permitted (or denied); (ii) at the level of the request itself, indicating the *actual* context attached to an entity at the time of the request. Secondly, it enables stakeholders to *harness* the knowledge artefacts embodied in an access control policy through the articulation of a set of *well-formedness constraints* that prescribe the knowledge artefacts, and their corresponding cardinalities, that may be incorporated in a policy. In other words, it empowers stakeholders to accurately define the *structure* by which their access control policies must abide and thus infuse into these policies their particular security and business requirements. Clearly, this leads to more effective access control policies. Moreover, a seminal benefit of the proposed generalisation is that it enables *reasoning* –performed automatically by a *policy validator*– about the well-formedness of a particular access control policy, i.e. about its abidance by the constraints stipulated by the stakeholder. Similarly, the

[1] This of course presupposes that the knowledge that Athens is a city in Greece is encoded in the underlying Security Context Element – see Section 3 for more details.

[2] By "stakeholders" we henceforth refer to people that responsible for determining the security policy of an organisation – e.g. security officers, CTOs, Data Protection Officers (DPOs), etc.

proposed generalisation paves the way for automated reasoning regarding the identification of potential *inter-policy relations*, such as contradicting or subsuming polices, that may affect the overall effectiveness of the policies. For instance, the policy of the example above subsumes a policy that permits `s` to read `o` from within `subnet1` between 09:00 and 17:00 and when `s'` resides in a location *within* `bldg1` (say the location identified as `room123`).

Last but not least, our generalised approach implements a context-aware access control engine, one that infuses semantic inferencing capabilities into a widely-adopted expert system for enabling the *efficient* generation of new knowledge, hence allowing access control policies to be enforced at the semantic, rather than at the syntactic, level.

The rest of this paper is structured as follows. Section 2 presents the ontologically-expressed Context-Aware Security Model that underpins our access control policies. Section 3 outlines how the object properties of this model can be utilised in order to perform semantic inferencing at the level of access requests. Section 4 presents our generalisation to the ontological model in [2] and Section 5 outlines how context-based inferencing can be performed in order to identify inter-policy relations. Section 6 presents an ABAC reference implementation that enables policy enforcement to be performed and reasoned about at the semantic, rather than at the syntactic, level. Section 7 presents related work and, finally, Section 8 presents conclusions.

2 MODELLING CONTEXT

In [2], a meta-model for capturing the primary facets of the Context-aware Security Model was presented. An updated version of this meta-model is depicted in Figure 1; the main change with respect to [2] is the extraction of the classes `pcm:Object`, `pcm:Subject`, `pcm:Request` and `pcm:Handler` from the class `pcm:SecurityContextElement`[3] – a change which, as we shall see in Sections 3, 4 and 5, simplifies semantic inferencing at the level of requests, as well as the incorporation of context in access control policies.

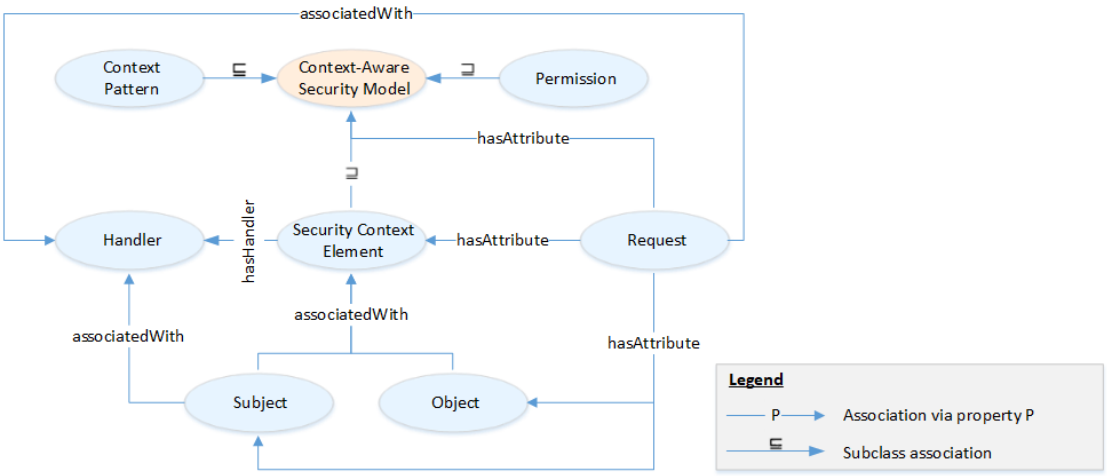

Figure 1: Context-aware security meta-model (namespace prefixes are omitted in figures to reduce clutter).

Ontologically, the facets of the Context-aware Security Model are represented in terms of the following classes:

- `pcm:Request` – Captures the characteristics that should be considered for evaluating an intercepted request.
- `pcm:Subject` – An instance of this class represents either the entity seeking access to a particular object (i.e. the 'requestor'), or the entity whose state should be considered for allowing a certain requestor to access sensitive data. Such an entity can be an organisation, a person, a group or a service.

[3] Definitions for all namespace prefixes encountered in this section can be found in [5].

- `pcm:Object` – Describes the protected resources – e.g. relational or non-relational database tables, files, software artefacts that manage sensitive data, etc.
- `pcm:Handler` – This class refers to the characteristics of dedicated software components that are used for federating and processing raw data relevant to an access control decision and semantically uplifting them as instances of the `SecurityContextElement`. Handlers are responsible for fusing a context-aware policy enforcement mechanism with contextual information in a usable format that will allow for the evaluation of access control policies. Different kinds of handlers include, for example, authentication handlers, request handlers, (reverse) location geocoding handlers, IP-address-to-city handlers, etc.
- `ppm:Permission` – This class refers to the allowed actions that an individual[4] of the class `pcm:Subject` is able to perform upon an individual of the class `pcm:Object`, including data permissions (e.g. Datastore, File, Web endpoint, Volume permissions) and data definition language (DDL) permissions (e.g. Datastore, File system structure permissions).
- `pcpm:ContextPattern` – This class refers to recurring motives of data accesses. Future access requests on sensitive data can be permitted, or denied, on the basis of such information which may include, for example, the typical date/time interval during which requests take place, or the most frequently-used device type for issuing incoming access requests.

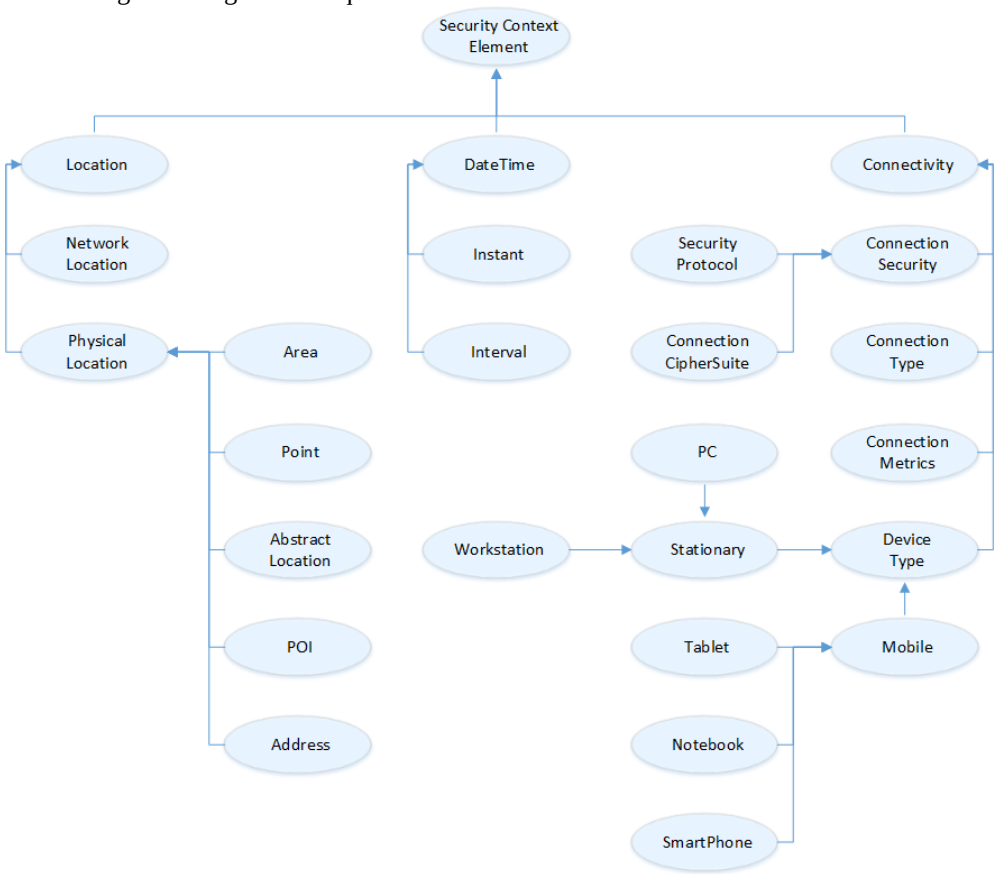

Figure 2: Security Context Element overview (arrows indicate subclass relations)

The `pcm:SecurityContextElement` class describes the various contextual attributes that may be associated with the subjects and/or the objects of a request, as well as with the request itself. As depicted in Figure 2, it encompasses the following top-level concepts.

- `pcm:Location` – Describes a physical or a network location where data are stored or from where a particular entity is requesting access to data, as well as the location of an entity that must be taken into account in order to permit or deny an access request. Its main subclasses are `pcm:PhysicalLocation`

[4] The terms "individual" and "instance" are used interchangeably.

and pcm:NetworkLocation. A physical location may involve: an address, a geographical position, an area, an abstract location and/or a point of interest (POI) defined in terms of geographical coordinates. A network location corresponds to an identifier for a node or network telecommunications interface from which a particular entity is requesting access to data.

- pcm:DateTime – Describes the specific chronological point expressed as an instant or interval that characterises an access request. Its main sub-classes are: pcm:Instant, pcm:DateTimeInterval
- pcm:Connectivity – Captures information related to the connection used by an entity for accessing sensitive data. Its main subclasses are: pcm:DeviceType, pcm:ConnectionType, pcm:ConnectionMetrics and pcm:ConnectionSecurity. The pcm:DeviceType class describes the device used for requesting access to sensitive data. The pcm:ConnectionType class refers to the different ways of transmitting an access request (e.g. LTE, 3G, WiFi, Cable, Satellite). The class pcm:ConnectionMetrics provides quantitative characteristics of the connection type used for accessing sensitive data (e.g. the download rate). Finally, the pcm:ConnectionSecurity class provides details on the level of security in the established connection for accessing sensitive data (e.g. TLS_ECDHE_RSA_WITH_AES_128_GCMSHA256 as a connection cipher suite).

3 CONTEXT-BASED INFERENCING AT THE LEVEL OF REQUESTS

The meta-model of Figure 1 provides a suite of object properties that aim at: (i) interrelating a request with its subject(s) and object(s); (ii) interrelating the subjects and objects of a request –as well as the request itself– with relevant contextual attributes from the class pcm:SecurityContextElement. The former interrelation is achieved by associating the class pcm:Request with the classes pcm:Object and pcm:Subject through the property pcm:hasAttribute. The latter interrelation is achieved by associating the classes pcm:Object and pcm:Subject with the class pcm:SecurityContextElement through the property pcm:associatedWith. Contextual attributes that are relevant to a request itself, and not to the subject(s) or object(s) of a request –e.g. the date/time at which a request takes place– are piggy-backed to a request through the property pcm:hasAttribute which interrelates the classes pcm:Request and pcm:SecurityContextElement.

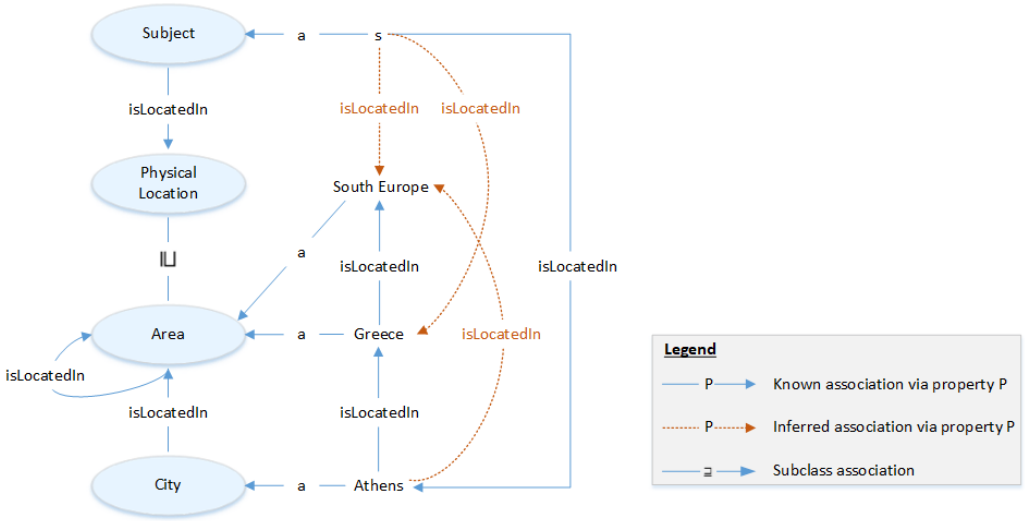

Figure 3: Inferencing based on property transitivity

Finally, the subjects and objects associated with a request, as well as the request itself, are interrelated through the object property pcm:associatedWith with the *handlers* that are responsible for providing the actual *measured* contextual values that these entities possess. The pcm:associatedWith property interrelates the classes pcm:Object, pcm:Subject and pcm:Request with the class pcm:Handler.

The aforementioned interrelations are exploited during the evaluation of a request in order to *semantically infer* the context that is attached to the subjects and objects of a request, or to the request itself. Suppose, for example, an access control policy that demands that a subject is allowed to access a sensitive data object as long as the subject is located in South Europe (SE). Let us assume that, based on the available handlers, the system is capable of only collecting location information at the level of cities. Once a request is intercepted with the resolved location for the requestor being, say the city of Athens, a number of facts can be semantically inferred based on the *transitivity* of the `pcm:associatedWith` property and of the subclass relation. These inferred facts (see Table 1 and Figure 3) 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 SE, even though the intercepted contextual information is specified at a different level of abstraction (i.e. at the city level as opposed to the European region level). Note that in Table 1 and Figure 3, the property `pcm:isLocatedIn` is used instead of the property `pcm:associatedWith`. The former is a sub-property of the latter that interconnects a subject *directly* with the `pcm:Location` subclass of the `pcm:SecurityContextElement` class (Figure 2). The use of this sub-property makes the inferencing process more efficient as now the system can infer from the outset that the individual `:Athens` is in fact an instance of the class `pcm:Location` and not of any of the other top-level concepts of the `pcm:SecurityContextElement` class. This renders the process of determining which handler to invoke for evaluating the request more efficient.

Facts	`:s a pcm:Subject;`
	`  pcm:isLocatedIn :Athens.`
	`:Athens a pcm:City;`
	`  pcm:isLocatedIn :Greece.`
	`:Greece a pcm:Area;`
	`  pcm:isLocatedIn :SE.`

Inferred facts	`Athens pcm:isLocatedIn :SE.`
	`:s pcm:isLocatedIn Greece.`
	`:s pcm:isLocatedIn :SE.`

Table 1: Inferred facts expressed as RDF triples (Turtle notation [6])

4 INCORPORATING CONTEXT IN ACCESS CONTROL POLICIES

The ontological model for access control policies proposed in [2] provides a suitable abstraction that enables the contextual knowledge pertaining to access requests to be incorporated into policies, hence considered in access control decisions. Nevertheless, as indicated in Section 1, this model suffers several limitations; to overcome these limitations, we propose in this section a generalisation of the model in [2]; in particular, in Section 4.1 we provide a brief outline of the model and, in Section 4.2, we present the proposed generalisation.

4.1 An Ontological Model for Access Control Policies

The model in [2] is based on the *Attribute-based Access Control* (*ABAC*) scheme [3] which, due to its inherent generality –stemming from its reliance on the generic concept of an *attribute*– is deemed particularly suitable for capturing the contextual knowledge pertaining to access requests [7]. Following the XACML standard [8], the model treats an ABAC policy as a nonempty set of *ABAC rules*; an ABAC rule is associated with a set of relevant *knowledge artefacts*, or *attributes*, that need to be taken into account for deciding whether an access request must be permitted or denied. In this respect, ABAC rules are regarded as *knowledge containers* for their encompassing policies. Ontologically, an ABAC rule takes the form of an instance of the class `pac:ABACRule` of Figure 4, whereas the various knowledge artefacts attached to an ABAC rule are described by the ontological template depicted in Figure 4.[5] In particular, each concept included in this template identifies a particular knowledge artefact, whilst each object property associates a knowledge artefact with an ABAC rule; a brief outline of these concepts and properties is provided in Table 2.

[5] The `pac` namespace prefix is defined as part of the ontological model for ABAC policies [9].

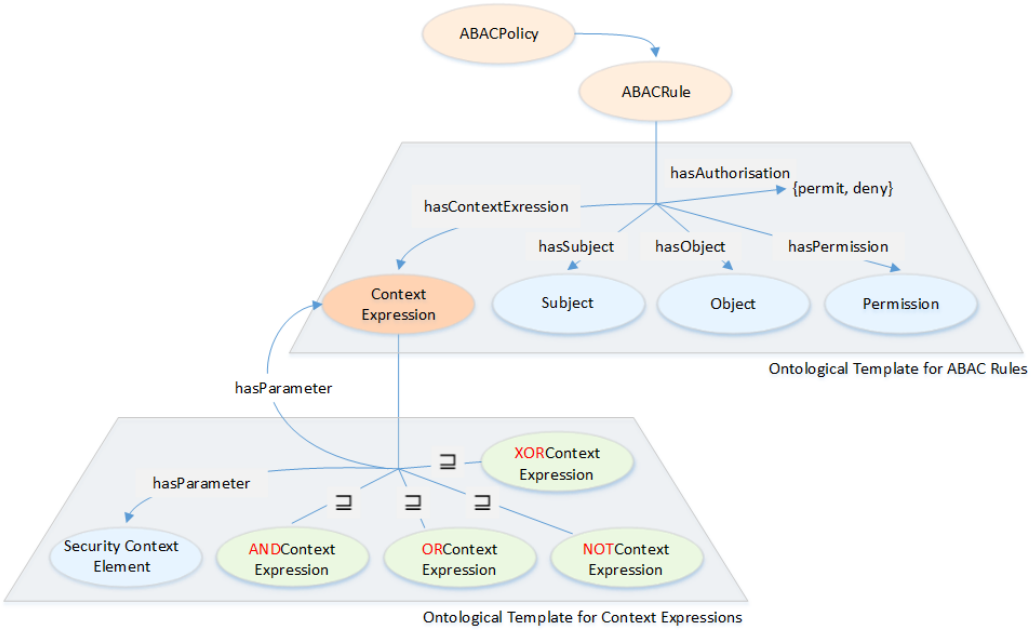

Figure 4: Ontological model for ABAC policies [2] (namespace prefixes are omitted to reduce clutter).

Knowledge artefact	Description	Associating object property
`pcm:Object`	Identifies the sensitive object on which access is requested.	`pac:hasObject` Domain: `pac:ABACRule` Range: `pcm:Object`
`{pac:permit, pac:deny}`	Determines the type of authorisation granted; in our work we discern between two kinds of authorisation, either 'permit', or 'deny'.	`pac:hasAuthorisation` Domain: `pac:ABACRule` Range: `pac:Authorisation`
`ppm:Permission`	Identifies the operation (e.g. read, write) to be performed on the protected sensitive object.	`pac:hasPermission` Domain: `pac:ABACRule` Range: `ppm:Permission`
`pcm:Subject`	Identifies the entity requesting access to the protected object.	`pac:hasSubject` Domain: `pac:ABACRule` Range: `pcm:Subject`
`pac:Context Expression`	Identifies the contextual conditions that must be satisfied in order to permit (or deny) a request.	`pac:hasContextExpression` Domain: `pac:ABACRule` Range: `pac:ContextExpression`

Table 2: Generic knowledge artefacts associated with the ABAC rule template

The model in [2] takes special care of context expressions; in particular, it treats context expressions as reified versions of the ontological template depicted in Figure 4, hence as individuals of the class `pac:ContextExpression`. The various knowledge artefacts –essentially contextual attributes– that are bound by a context expression take the form of *parameters* of the expression and are represented ontologically as instances of the classes that comprise the Security Context Element (see Figure 2); they are associated with the individual that represents a context expression through the object property `pac:hasParameter`. The parameters of an expression may be combined through the usual logical connectives. Such logical combinations, however, tend to be overly verbose when serialised in an ontology language such as OWL [10] and thus, to increase readability, we introduce the classes `pac:XContextExpression` (where `X` stands for one of `AND`, `OR`, `XOR`, `NOT` – see Figure 4) with the following intended meanings: 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*; likewise, if a context expression

is represented by an instance of the class say `pac:NOTContextExpression`, its (single) parameter is interpreted as being negated; analogous interpretations apply to the rest of the classes `pac:ORContextExpression` and `pac:XORContextExpression`. Formal definitions –expressed in OWL 2– of these interpretations, hence of the meanings intended for each of the classes `pac:XContextExpression`, can be found in the appendix. Table 3 presents an example context expression represented by the individual `:expr` that conjunctively combines two parameters represented by the individuals `:para1` and `:para2`. The former is an instance of the class `pcm:AbstractLocation` of the Security Context Element depicted in Figure 2 and specifies the location 'Athens'. The latter is an instance of the class `pcm:NetworkLocation` and specifies a network endpoint. The data properties `pcm:hasName` and `pcm:hasIPAddress` form part of the Security Context Element with the obvious meanings.

```
:expr a pac:ANDContextExpression;
   pac:hasParameter :para1;
   pac:hasParameter :para2.
:para1 a pcm:AbstractLocation;
   pcm:hasName :Athens.
:para2 a pcm:NetworkLocation;
   pcm:hasIPAddress 123.123.123.123"^^xsd:string.
```

Table 3: Context expression example

A context expression may be defined recursively, in terms of one or more other context expressions. Ontologically, this is represented by including the class `pac:ContextExpression` in both the domain and range of the object property `pac:hasParameter` (see Figure 4). The example of Table 4 shows a recursively-defined context expression that includes the context expression represented by the individual `:expr1` as a parameter.

```
:expr a pac:ANDContextExpression;
   pac:hasParameter :para1;
   pac:hasParameter :expr1.
:expr1 a pac:NOTContextExpression;
   pac:hasParameter para2.
```

Table 4: Recursive context expression example (`:para1` and `:para2` are defined as in Table 3)

4.2 A Generalised Ontological Template for ABAC Rules

As discussed in Section 1, the model in [2] suffers several limitations: (i) it invariably associates context expressions –hence contextual knowledge– with the subject of a request, ignoring the fact that such expressions may also need to be associated with the object of a request, the request itself, or any other entity that is deemed relevant; (ii) although it prescribes a set of *knowledge artefacts* that are taken into account by an access control policy –namely the *subject* and *object* of a request, the *kind of access* sought (e.g. read or write access), and a body of *contextual attributes* pertaining to the subject of a request– it fails to specify the *cardinalities* with which these knowledge artefacts may be encountered in a policy, or the allowable manners in which they may be logically combined; this essentially precludes the articulation of any *constraints* regarding the exact *structure*, or *form*, that an access control policy may assume.

In order to overcome these limitations, we propose a generalisation to the ontological model in [2]. More specifically, with regard to the 1st limitation, we propose an approach that enables the association of context expressions, hence of contextual knowledge, with any entity that is deemed relevant and not just with the subject of a request; this approach is elaborated in Section 4.2.1. With regard to the 2nd limitation, we propose an approach that enables stakeholders to *harness* the knowledge artefacts embodied in an ABAC rule through the articulation of a set of *well-formedness constraints* that form part of a *higher-level ontology* (HLO): a generic ontological model that enables stakeholders to select the knowledge artefacts that are to be incorporated into an ABAC rule and, crucially, to prescribe the allowable *cardinalities*, as well as the allowable logical ways (e.g. conjunctively, disjunctively, etc.), with which these artefacts may be combined. In other words, the HLO empowers stakeholders to accurately define the structure –in terms of relevant knowledge artefacts– by which their access control policies must abide, thus infuse into these policies their particular security and business requirements. Clearly, this leads

to more effective access control policies. Moreover, a seminal benefit of our approach is that it enables *reasoning* –performed automatically by a *policy validator*– about the well-formedness of a particular ABAC rule, i.e. about its abidance with the constraints of the HLO, thus increasing assurance on the *effectiveness* of ABAC rules. The HLO as well as our approach to reasoning about the correctness of an ABAC rule are further elaborated in Section 4.2.2; more details can be found in [11].

4.2.1 Associating Contextual Knowledge

A straightforward approach to allowing a context expression to be attached to any entity that is associated with a request, and not just with the subject of a request, is to render the `pac:hasContextExpression` property applicable to: (i) *any* individual of the class `pcm:Subject` of the Security Context Element that may participate in a request without necessarily this individual being the actual subject of the request; (ii) the object associated with a request. This can be readily achieved by extending the domain of `pac:hasContextExpression` to include, in addition to the class `pac:ABACRule` (see Table 2), the classes `pcm:Subject` and `pcm:Object`. Nevertheless, associating a context expression *solely* with a subject, or *solely* with a controlled object, is problematic as demonstrated by the example of Table 5.

```
:rule1 a pac:ABACRule;
   pac:hasPermission :read;
   pac:hasSubject :s;
   pac:hasAuthorisation pac:permit;
   pac:hasObject :o.
:s pac:hasContextExpression :expr1.
   :expr1 a pac:ContextExpression;
   pac:hasParameter :para2.
:rule2 a pac:ABACRule;
   pac:hasPermission :read;
   pac:hasSubject :s;
   pac:hasAuthorisation pac:permit;
   pac:hasObject :o.
:s pac:hasContextExpression :expr2.
:expr2 a pac:ContextExpression;
   pac:hasParameter :para3.
:para3 a pcm:NetworkLocation;
   pcm:hasIPAddress "120.120.120.120"^^xsd:string.
```

Table 5: Associating context solely with a subject or an object (`:para2` is defined as in Table 3)

In this example, two ABAC rules, `:rule1` and `:rule2`, are defined. The intended meaning behind the second rule is that the subject `:s` can read the object `:o` only when the context expression `:expr2` is satisfied; `:expr2` states that the IP address associated with `:s` must be equal to `120.120.120.120`. However, from the triples of the example of Table 5 there is no way of discerning which context expression, `:expr1` or `:expr2`, refers to which rule. This ambiguity stems from the fact that the approach outlined above neglects that the context expression that is associated with an entity inside a rule is not the actual, or per se, context of the entity but the context that the *rule expects* to be associated with the entity. In other words, the mere association of a context expression with an entity is insufficient by itself to discern the context that a rule requires from an entity to possess.

```
:rule1 a pac:ABACRule;
   pac:hasPermission :read;
   pac:hasSubject :s;
   pac:hasAuthorisation pac:permit;
   pac:hasObject :o.
   pac:hasContextExpression :expr1.
:s pac:hasContextExpression :expr1.
:expr1 a pac:ContextExpression;
   pac:hasParameter :para2.
:rule2 a pac:ABACRule;
   pac:hasPermission :read;
   pac:hasSubject :s;
```

```
   pac:hasAuthorisation pac:permit;
   pac:hasObject :o;
   pac:hasContextExpression :expr2.
:s pac:hasContextExpression :expr2.
:expr2 a pac:ContextExpression;
   pac:hasParameter :para3.
:para3 a pcm:NetworkLocation;
   pcm:hasIPAddress 120.120.120.120"^^xsd:string.
```

Table 6: Using pac:hasContextExpression twice (`:para2` is defined as in Table 3)

One solution that circumvents this problem is to require that, each time the property `pac:hasContextExpression` is used to associate a context expression with an entity, the same context expression is also associated with the underlying rule that requires the particular context expression to be associated with that entity. This solution, however, requires that the `pac:hasContextExpression` property is used twice each time a context expression is associated with an entity. This is demonstrated by the example of Table 6.

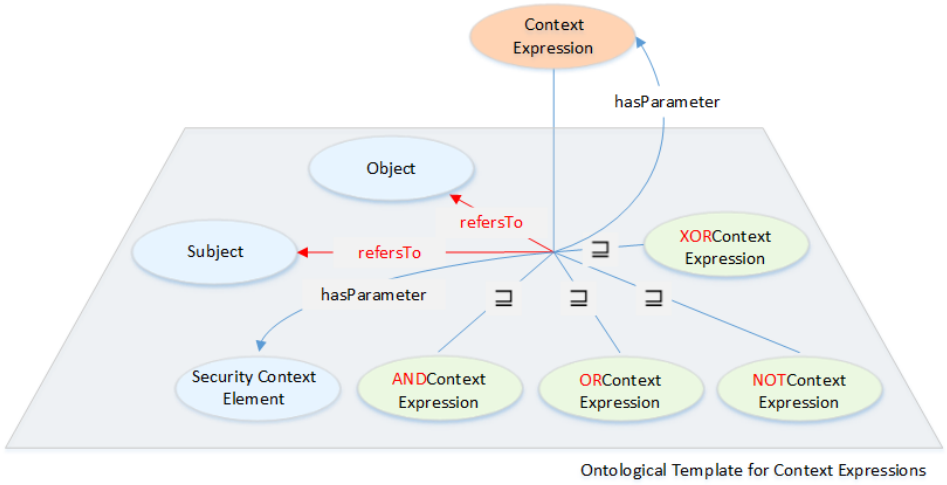

Figure 5: Extended ontological template

A more elegant solution is to extend the ontological template of Figure 4 through the introduction of a new object property, namely `pac:refersTo`. As depicted in Figure 5, this property has as domain the class `pac:ContextExpression` and as range the union of the classes `pcm:Subject` and `pcm:Object`. As its name suggests, its purpose is to attach a context expression to the entity that it refers to. This way, when a context expression is associated with an ABAC rule (through the `pac:hasContextExpression` property of Figure 4), it is already attached to the actual entity that it refers to. Adhering to this solution, the example of Table 6 now takes the form shown in Table 7.

```
:rule1 a pac:ABACRule;
   pac:hasPermission :read;
   pac:hasSubject :s;
   pac:hasAuthorisation pac:permit;
   pac:hasObject :o;
   pac:hasContextExpression :expr1.
:expr1 a pac:ContextExpression;
   pac:hasParameter :para2;
   pac:refersTo :s.
:rule2 a pac:ABACRule;
   pac:hasPermission :read;
   pac:hasSubject :s;
   pac:hasAuthorisation pac:permit;
   pac:hasObject :o;
   pac:hasContextExpression :expr2.
:expr2 a pac:ContextExpression;
   pac:hasParameter :para3;
```

```
   pac:refersTo :s.
:para3 a pcm:NetworkLocation;
   pcm:hasIPAddress "120.120.120.120"^^xsd:string.
```

Table 7: Associating context using the extended model of Figure 5 (`:para2` is defined as in Table 3)

It is to be noted here that the `pac:refersTo` property is *not* obligatory in the sense that not all context expressions need be associated with an entity from the classes `pcm:Subject` or `pcm:Object`. This might be the case under the following circumstances: (i) When a context expression refers to the *request itself* rather than an entity that is associated with the request (e.g. when the context expression constrains the time at which a request is issued). (ii) When a context expression forms a constituent part of another (recursively-defined) context expression which is associated with an entity from the classes `pcm:Subject` or `pcm:Object` and the constituent expression refers to that same entity. This is depicted, for example, in Table 8 where the context expressions `:expr1` and `:expr2` form constituent parts of the expression `:expr` and neither of `:expr1` and `:expr2` is attached to an entity from the classes `pcm:Subject` or `pcm:Object` as they both refer to the same entity (`:s`) that is referred to by the encompassing expression `:expr`. Note that, in Table 8, the identifier `:para4` specifies the type of device (e.g. stationary as opposed to mobile) through which a request must take place as well as the OS type of that device.

```
:expr a pac:XORContextExpression;
   pac:hasParameter :expr1;
   pac:hasParameter :expr2;
   pac:refersTo :s.
:expr1 a pac:ORContextExpression;
   pac:hasParameter :para1;
   pac:hasParameter :para2.
:expr2 a pac:ANDContextExpression;
   pac:hasParameter :para3;
   pac:hasParameter :para4.
:para4 a pcm:Stationary;
   pcm:hasStationaryOS "Windows10"^^xsd:string.
```

Table 8: Associating context using the model of Figure 5 (`:para1`, `:para2` and `:para3` are defined as in Table 3)

It is also to be noted that the `pac:refersTo` property is *not* functional: the same context expression instance may be associated with two or more distinct entities, i.e. two or more distinct individuals from the classes `pcm:Subject` or `pcm:Object`.

4.2.2 Higher-Level Ontology

The HLO comprises a set of well-formedness constraints that specify all those knowledge artefacts –i.e. all those concepts from the underlying Context-Aware Security Model– that *must*, *may* or *must not* be embodied in an ABAC rule; they also specify the allowable *cardinalities* with which these knowledge artefacts may appear, as well as the allowable *values*, or *ranges of values*, that they may assume. In this respect, the HLO defines the allowable *form*, or *structure*, of an ABAC rule. For example, the ontological template for ABAC rules presented in [2] (see Figure 4) may be specified in terms of the well-formedness constraints of Table 9[6]; these constraints are expressed as $\mathcal{SROIQ}$ terminological (TBox) axioms[7] [12] that restrict the individuals of the class `pac:ABACRule`, i.e. the individuals that represent ABAC rules.

Axiom 1	`pac:ABACRule` $\sqsubseteq$	$(\leq 1$ `pac:hasObject.pcm:Object`$)\ \sqcap$
		$(\geq 1$ `pac:hasObject.pcm:Object`$)$
Axiom 2	`pac:ABACRule` $\sqsubseteq$	$(\leq 1$ `pac:hasPermission.ppm:Permission`$)\ \sqcap$
		$(\geq 1$ `pac:hasPermission.ppm:Permission`$)$
Axiom 3	`pac:ABACRule` $\sqsubseteq$	$(\leq 1$ `pac:hasAuthorisation.{pac:permit,pac:deny}`$)\ \sqcap$

[6] The well-formedness constraints of Table 9 provide, in fact, a more detailed specification of the ontological template of Figure 4 for it includes cardinality constraints.

[7] $\mathcal{SROIQ}$ is the DL underlying OWL 2; we resort to $\mathcal{SROIQ}$ due to the conciseness and rigorousness of its notation.

		(≥ 1 pac:hasAuthorisation.{pac:permit,pac:deny})
Axiom 4	pac:ABACRule ⊑	(≥ 1 pac:hasSubject.pcm:Subject)
Axiom 5	pac:ABACRule ⊑	(≤ 1 pac:hasContextExpression.pac:ContextExpression)

Table 9: Well-formedness constraints for the ontological template of Section 4.1

The 1st constraint stipulates that each ABAC rule *must* embody *exactly one* protected resource; ontologically, this is expressed by requiring that each instance of the concept `pac:ABACRule` is associated with *exactly one* individual from the class `pcm:Object` of the Context-Aware Security Model, and that this association should be realised through the object property `pac:hasObject`. Similarly, the 2nd and 3rd constraints stipulate that each ABAC rule must be associated –through the appropriate properties– with *exactly one* action from the class `ppm:Permission` (i.e. with exactly one action that is to be performed on the protected resource), and with exactly one kind of authorisation (i.e. with exactly one of the instances `pac:permit` or `pac:deny`). The 4th and 5th axioms demand, respectively, that each ABAC rule must be associated –through the appropriate properties– with *at least one* subject from the class `pcm:Subject` (i.e. with at least one entity requesting access to the protected asset), and with *at most one* context expression.

Constraining Context Expressions

The HLO may encompass constraints that specify the allowable forms of a context expression. As an example, consider a constraint whereby *any* context expression associated with an ABAC rule should invariably incorporate at least one location parameter that confines the whereabouts of the subject (say `s`) of a request to the physical location identified as, say, `Athens`, or to the network location identified by the subnet, say, 123.0.0.0/8. Ontologically, this constraint takes the form of a TBox axiom whereby each individual of the class `pac:ContextExpression` is associated with either the individual `Athens`, or the individual 123.0.0.0/8, and that these associations are realised via the property `pac:hasParameter`; it also demands that the context expression refers to the subject `s` (see Table 10 for a formal definition of this axiom).

pac:ContextExprssion ⊑	((≤ 1 pac:hasParameter.{Athens})) ⊔ (≤ 1 pac:hasParameter.{123.0.0.0/8}))

Table 10: Example well-formedness constraint for a context expression

Reasoning About Well-Formedness

As already indicated, one of the main virtues of the proposed generalisation to the ontological model in [2] is that it enables *reasoning* about the correctness of an ABAC rule. This involves reasoning about the abidance of the rule by the HLO constraints; an outline of how such reasoning may be performed is in order.

Firstly, the ABAC rule under validation, say `r`, is expressed in terms of a set of TBox axioms that specify all those knowledge artefact *values* that are associated with `r`. As an example, suppose the axiom set of Table 11 whereby `r` permits access to the subject `s` when the action requested is `read`.

$R \equiv$ {pac:ABACRule(r), pcm:Subject(s), ppm:Permission(read), pac:Authorisation(permit), pac:haSuject(r,s), pas:hasPermission(r,read), pac:hasAuthorisation(r,permit)}

Table 11: Axiom set for rule `r`

Secondly, each $\mathcal{SROIQ}$ TBox axiom corresponding to an HLO constraint is translated into a *Distinguished Conjunctive Query* with *Negation as Failure* (DCQ$^{\mathbf{not}}$) [13] that is posed to the axiom set corresponding to r. Such a query aims at discovering any individuals of the class *ABACRule* that *violate* the axiom: if the query returns no such individuals, the axiom –hence the corresponding constraint– is considered to be satisfied; otherwise, it is considered to be violated[8]. Consider, for example, the 1st axiom of Table 9. This axiom is translated

[8] This is, in fact, an instance of *Negation as Failure* (*NAF*) *inferencing*: an axiom is considered to hold iff its negation is not derivable.

into a DCQ^{not} that attempts to discover whether the class `pac:ABACRule` contains any individuals that either enjoy no associations through the property `pac:hasControlledObject` with instances of the class `pcm:Object`, or enjoy two or more distinct such associations (see Table 12). This query is posed to the axiom set under validation in the form of a SPARQL query that is executed in an OWL reasoner.

$\texttt{pac:ABACRule}(x) \wedge$
$((\textbf{not}(\texttt{pac:hasObject}(x, y) \wedge \texttt{pcm:Object}(y)) \vee$
$(\texttt{pac:hasObject}(x, y) \wedge \texttt{pac:hasObject}(x, z) \wedge \texttt{pcm:Object}(y) \wedge \texttt{pcm:Object}(z) \wedge \textbf{not}(x = y)))$

Table 12: Example query

It is to be noted here that two seminal assumptions underpinning OWL are the Open World Assumption (OWA) and non-Unique Name Assumption (non-UNA), which generally prevent reasoning about *constraint satisfaction* [13]. Consider, for instance, the axiom set of Table 9 which fails to specify the resource that is protected by r. According to the OWA, this does not necessarily mean that `r` does not have such a resource associated with it: it merely means that this association is not specified in the particular axiom set. Thus, we cannot assert with certainty that the 1st axiom of Table 9 is violated. In order to overcome this obstacle, the approach proposed in [13] is adopted whereby the axiom sets that are validated through DCQ^{not} are based on the Integrity Constraints (IC) semantics for OWL 2 proposed in [13]; this effectively, enables closed-world reasoning when checking the correctness of an ABAC rule again the HLO constraints.

5 OPTIMISING CONTEXT-BASED INFERENCING AT THE POLICY LEVEL

One of the main virtues of ontologically specifying the contextual conditions that must be satisfied in order for a request to be permitted (or denied), is the fact that it enables the identification of *inter-policy relations*. In particular, it enables us to identify whether one ABAC rule is *subsumed* by another.

```
:expr1 a pac:ContextExpression;
   pac:hasParameter :Athens.
:expr2 a pac:ContextExpression;
   pac:hasParameter :iPadPro9.7.
:Athens a pcm:AbstractLocation.
:iPadPro9.7 a pcm:Tablet.
```

Table 13: Checking property subsumption

Suppose two ABAC rules represented by the instances `:rule1` and `:rule2` of the class `pac:ABACRule`. Naturally, a prerequisite for `:rule2` to be subsumed by `:rule1` is that the context expression associated with the former logically subsumes the context expression associated with the latter. In other words, the context expression associated with `:rule2` must be *logically inferable* from the context expression associated with `:rule1`. Let the context expressions associated with the two rules be represented, respectively, by the instances `:expr1` and `:expr2` of the class `pac:ContextExpression`. In order to determine whether `:expr2` is logically inferable from `:expr1`, the following conditions must hold: (i) If `:expr1` is attached, via the property `pac:refersTo`, to an entity (say `:e1`), then `:expr2` must also be attached, via the same property, to an entity (say `:e2`) such that either `:e1` and `:e2` represent the same entity, or `:e1` represents an entity that is considered more general than `:e2` (for instance, this could be the case when `:e1` and `:e2` represent groups of subjects). (ii) Each and every association that `:expr2` has through the `pac:hasParameter` property must be logically inferable from a corresponding association of `:expr1`.

Name	Domain	Range
`pac:hasLocationParameter`	`pac:ContextExpression`	`pcm:Location`

pac:hasDateTimeParameter	pac:ContextExpression	pcm:DateTime
pac:hasConnectivityParameter	pac:ContextExpression	pcm:Connectivity

Table 14: Sub-properties of pac:hasParameter

We concentrate on the 2nd condition above. Checking whether this condition holds can be a rather inefficient process. Consider, for instance, the simple example of Table 13. Clearly, :expr2 cannot be considered to be subsumed by :expr1 for the former has as parameter a type of device (a tablet) whereas the latter has as parameter a location. Nevertheless, for this fact to be discovered in an automated manner, it must be verified that the individuals :Athens and :iPadPro9.7 are indeed mutually incomparable. Ontologically, this amounts to discovering that the two individuals are not instances of a common class from the Security Context Element depicted in Figure 2. This effectively means that the path of subclass relations that leads from the class pcm:Tablet (i.e. the immediate class to which the individual :iPadPro9.7 belongs) to the top-level class pcm:SecurityContextElement, and the corresponding path that leads from the class pcm:AbstractLocation (i.e. the immediate class to which the individual :Athens belongs) to pcm:SecurityContextElement must be traversed in order to ensure that they do not share any common classes. This is a computationally expensive process.

One way to reduce this computational cost is to define sub-properties of the pac:hasParameter property that directly associate a context expression with one of the top-level concepts of the pcm:SecurityContextElement depicted in Figure 2. Thus, when a context expression has as a parameter an instance of one of these subclasses, the association takes place through the appropriate sub-property rather than through the pac:hasParameter property. These sub-properties are shown in Table 14. For instance, the context expressions represented by the individuals :expr1 and :expr2 in the example of Table 13, will now be associated with their corresponding parameters through the sub-properties pac:hasLocationParameter and pac:hasConnectivityParameter respectively. In this way, the subsumption of :expr2 by :expr1 can be precluded from the outset, without having to traverse the aforementioned paths since now the two parameters are associated with the context expressions through different sub-properties of pac:hasParameter ruling out any subsumption relation between them. It is to be noted that the fact that two parameters are associated with a context expression through the same sub-property does *not* necessarily imply that the two parameters are mutually comparable.

6 ABAC REFERENCE IMPLEMENTATION WITH THE ADOPTION OF XACML 3.0

As already discussed, there are many reference implementations of the ABAC model. One example of an access control framework that is consistent with ABAC is the eXtensible Access Control Markup Language (XACML) [8]; another notable example is the Next Generation Access Control (a.k.a. NGAC) standard [14, 15]. In this work we opt for XACML as the basis of our implementation for the semantic authorisation engine. XACML is a widely diffused standard, deployed worldwide by many organisations such as banks, insurance companies, health care providers. etc. In contrast, NGAC has no practical reference implementation; XACML is superior to NGAC in the following aspects: a) separation of authorisation functionality from proprietary operating environment; b) attribute and policy management and c) operational efficiency.

6.1 A Generalised Ontological Template for Context Expressions

XACML describes both a policy language and an access control decision request/response language. Both languages use XSD notations; hence policy definition and request/response elements are serialised as XML elements. The policy language details general access control requirements, and has standard extension points for defining new functions, data types, combining logic, etc. The request/response language lets you form a query to ask whether or not a given action should be allowed, and interpret the result. The response always includes an

answer about whether the request should be allowed using one of four values: Permit, Deny, Indeterminate (an error occurred or some required value was missing, so a decision cannot be made) or Not Applicable (no policy available to this service addresses this request).

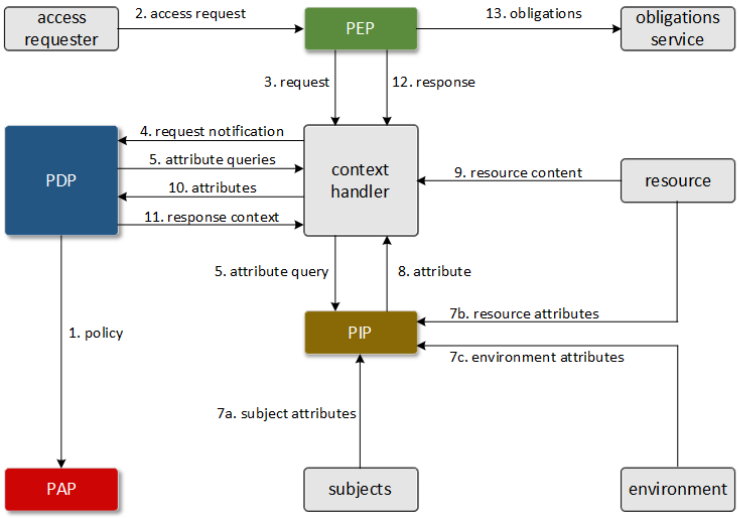

Figure 6: Extended XACML Flow & Architectural Components

The specification defines five main components (see Figure 6) that handle access decisions; namely Policy Enforcement Point (PEP), Policy Administration Point (PAP), Policy Decision Point (PDP), Policy Information Point (PIP), and a Context Handler.

The functional purpose of the main components is:

- The Policy Administration Point (PAP) provides an interface or API to manage the policies that are stored in the repository and provides the policies to the Policy Decision Point (PDP).
- The Policy Enforcement Point (PEP) is the interface to the external world. It receives the application specific access requests and translates them to XACML access control requests, then it denies or allows access based on the result provided by the PDP.
- Policy Decision Point (PDP) is the main decision point for the access requests. It collects all the necessary information from other actors and yields a decision.
- Policy Information Point (PIP) is the point where the necessary attributes for the policy evaluation are retrieved from several external or internal actors. The attributes can be retrieved from the resource to be accessed, environment (e.g. time), subjects, and so forth.

In this paper, we also propose a new approach that focuses on the enhancement of the Policy Decision Point as it is explained in Sections 6.2, 6.3. As already mentioned, XACML uses XSD notation in order to model the three basic artefacts which are required i.e. the policy, the request and the response. Thus, as depicted in Figure 7, three types of XML documents are processed or produced by an XACML engine in order to judge upon a decision: the Policy.xml which serialises an actual policy, the Request.xml which serialises an authorisation request and the Response.xml that serialises the output of the engine.

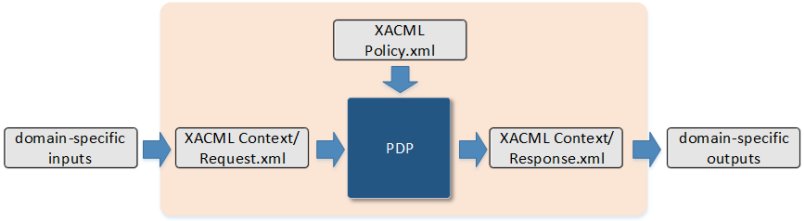

Figure 7: Usage of XML artefacts

6.2 The Usage of an Expert System as an Inference Engine

Expert Systems use knowledge representation to facilitate the codification of knowledge into a knowledge base which can be used for reasoning, i.e., we can process data with this knowledge base to infer conclusions. The basic components of an Expert System are presented in Figure 8. The two foundational concepts include Rules and Facts. Rules represent static knowledge (a.k.a. templates) while facts represent dynamic knowledge. In this reference implementation, we rely on Drools [16] expert system. The reason behind our choice is the outstanding performance of the engine [17] which makes it capable to support near real-time decision making.

The Rules are stored in the Production Memory and the facts again which these rules are evaluated by the Inference Engine are kept in the Working Memory. Facts are asserted into the Working Memory where they may then be modified or retracted. A system with a large number of rules and facts may result in many rules being true for the same fact assertion; these rules may be in conflict. The Agenda component (see Figure 8) manages the execution order of these conflicting rules using a Conflict Resolution strategy.

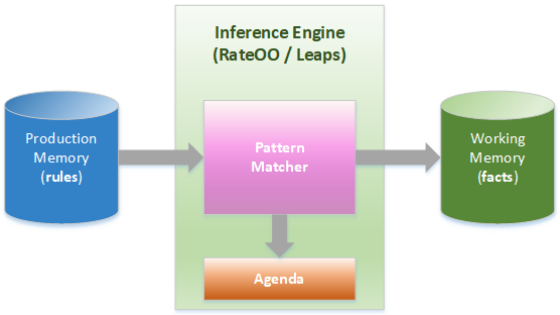

Figure 8: Expert system basic components (taken from [16])

The idea regarding attribute expansion and policy enforcement is straightforward: each inference type that is presented in Table 15 will be modeled in the production memory. Furthermore, the attributes that the Context Handler will indicate that are required will be transformed dynamically (per request) to facts that will be persisted on the working memory. In order to initialise Rules and Facts two separate parsers were developed that transform the JSON-LD entries in Facts and the reasoning expectations to Rules. There are two methods of execution for a rule system: Forward Chaining and Backward Chaining; systems that implement both are called Hybrid Chaining Systems. Forward chaining (see Figure 9) is "data-driven" and thus reactionary, with facts being asserted into working memory, resulting in one or more rules being concurrently true and scheduled for firing by the Agenda. Backward chaining is “goal-driven” and it is outside the scope of this document.

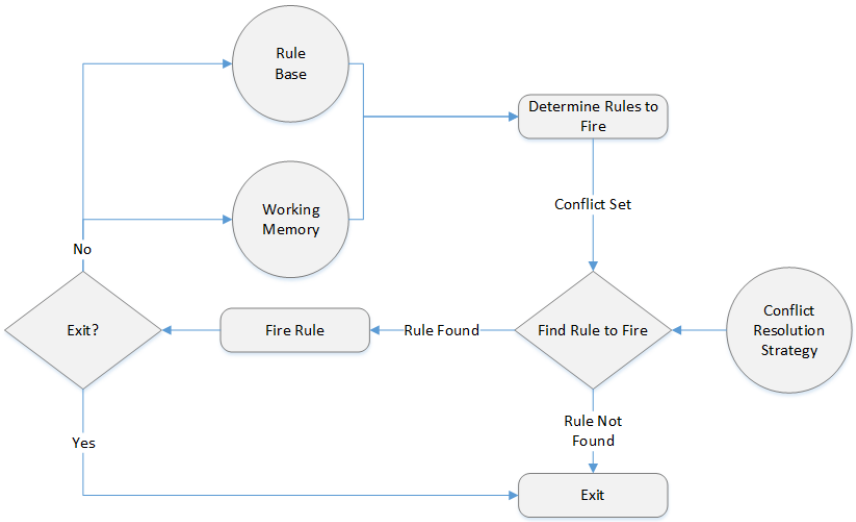

Figure 9: Forward chaining execution flow

6.3 Fusing the Expert System with ontological reasoning capabilities

In our approach we have designed and developed a dedicated parser called *ContextModel2ExpertSystemRules Parser* to enhance the expert system (i.e. Drools engine) with inferencing capabilities. More specifically, the goal is to populate the Production Memory of the expert system with knowledge coming from the Security Context Element (see Figure 2) in order to fuse it with semantic authorisation capabilities. To accomplish this goal, the parser injects into the production memory all the classes, properties and instances of the Security Context Element as knowledge facts (in the form of triples). This input is combined with a number of meta-rules that have been created and inserted into the production memory in order to constitute the expert system capable of inferencing additional facts at run-time (see the example in Table 11). In particular, the proposed semantic authorisation engine supports the following basic types of inferencing: *Property Transitivity*; *Sub-property Transitivity*; *Supertype Inheritance*; *Class Transitivity*; *Knowledge Expansion through Domain and Range Generalisation*.

We note that the use of an expert system along with a fusing technique that enhances its 'traditional' propositional logic reasoning with valuable ontological reasoning capabilities is considered by the authors as the most appropriate way for performing context-aware policy enforcement in the domain of cloud computing. The authorisation engines used especially in the cases of cloud applications, with the often unforeseen numbers of users and amounts of access requests, should be accurate and efficient. The option of exploiting any of the well-known Description Logic (DL) reasoners [18] in order to perform the necessary inferencing and policy enforcement, may have the outcome of improved context expressivity and thus accuracy but would significantly delay the outcome of access control decisions [19]. On the contrary, the proposed approach is efficient for access control purposes since it keeps the valuable knowledge expressed in semantics without having to perform any time-consuming queries at run-time.

In Table 15 below, we present the different kinds of inference types supported for ontological reasoning along with some examples. We begin with the *Property Transitivity* description: for any given knowledge triple (`s1, pred, o1`) that has as a predicate a transitive object property (`pred`), if there is at least one more knowledge triple with the same object property , i.e. (`s2, pred, o2`), and subject the same as the object of the first triple (`s2 equalsTo o1`) then a new, inferred triple should be created, having the same object property, the subject of the first triple and the object of the second triple, i.e. (`s1, pred, o2`). In a similar vein, the *Sub-property Transitivity* is described as follows: for any given object property (`pred`) that is sub-property of another object property (`pred.parent`), if there is at least one knowledge triple (`s, pred, o`) then there should be a new inferred triple with the same subject and object but with a property that is the parent of the given object property (`s, pred.parent, o`). The *Supertype Inheritance* is defined as follows: for any given instance of a class (`s, isA, clazz`) that has a parent there should be a new inferred instance of the parent class (`s, isA, clazz.parent`). The *Class Transitivity* is described as follows: for any given class that has a parent class (`a, subClassOf, b`) that is a sub-class of a third class (`b, subClassOf, c`) then there should be a new class that is a subclass of the third class (`a, subClassOf, c`). The *Knowledge Expansion through Range Generalisation* involves the following description: for any given knowledge triple (`s, pred, o`) with an object that is an instance of class (`o, isA, clazz`) that has a parent (i.e. using the supertype inheritance for o) then there should be a new inferred triple with the same object that is considered an instance of the parent class [(`s, pred, o`) where (`o, isA, clazz.parent`)]. The *Knowledge Expansion through Domain Generalisation* refers to the following: for any given knowledge triple (`s, pred, o`) with a subject that is an instance of class (`s, isA, clazz`) that has a parent (i.e. using the supertype inheritance for `s`) then there should be a new inferred triple with the same subject that is considered an instance of the parent class [(`s, pred, o`) where (`s, isA, clazz.parent`)].

Property Transitivity	

Example Facts	`:s a pcm:Subject;`
	`  pcm:isLocatedIn :Athens.`
	`:Athens a pcm:City;`
	`  pcm:isLocatedIn :Greece.`
	`:Greece a pcm:Area;`
	`  pcm:isLocatedIn :SouthEurope.`
	`:SouthEurope a pcm:Area.`

Example Inferred Facts

```
:Athens pcm:isLocatedIn :SouthEurope.
:s pcm:isLocatedIn :Greece.
:s pcm:isLocatedIn :SouthEurope.
```

Meta-rule

```
rule "Knowledge Expansion through Property Transitiveness"
  when
     $pred: ObjectProperty(transitive == true )
     $triple1: KnowledgeTriple( predicate == $pred ,
                                $subject1: subject,
                                $object1: object
               )
     $triple2: KnowledgeTriple( predicate == $pred ,
                                subject == $object1
               )
     not (
        exists(
           KnowledgeTriple(
                predicate == $pred ,
                subject == $triple1.subject,
                object == $triple2.object
           )
        )
     )
   then
     KnowledgeTriple newtriple = new KnowledgeTriple(
                                          $triple1.getSubject(),
                                          $pred ,
                                          $triple2.getObject()
                                      );
     insert(newtriple);
end
```

Sub-property Transitivity

Example Facts

```
pcm:areaContainsRegion rdfs:subPropertyOf pcm:areaContainsArea.
:Sachara a pcm:Region.
:Africa a pcm:Area;
  pcm:areaContainsRegion :Sachara.
```

Example Inferred Facts

```
:Africa pcm:areaContainsArea :Sachara.
```

Meta-rule

```
rule "Knowledge Expansion through Sub-Property Transitiveness"
  when
     $pred: ObjectProperty( parent != null )
     $triple: KnowledgeTriple( predicate == $pred ,
                               $subject1: subject ,
                               $object1: object
               )
     not (
       exists(
          KnowledgeTriple( predicate == $pred.parent ,
                           subject == $triple.subject ,
                           object == $triple.object
          )
       )
     )
  then
     KnowledgeTriple newtriple = new KnowledgeTriple(
                                                $triple.getSubject(),
                                                $pred.getParent(),
                                                $triple.getObject()
```

	); insert(newtriple); end
Supertype Inheritance	
Example Facts	pcm:City rdfs:subClassOf pcm:Area. :Athens a pcm:City.
Example Inferred Facts	:Athens a pcm:Area.
Meta-rule	rule "Supertype Inheritance Inference" when $instance: InstanceOfClazz(clazz.parent !=null) not (exists (InstanceOfClazz(name == $instance.name , clazz == clazz.parent))) then InstanceOfClazz newinstanceofclazz = new InstanceOfClazz($instance.getName(), $instance.getClazz().getParent()); insert(newinstanceofclazz); end
Class Transitivity	
Example Facts	pcm:Continent rdfs:subClassOf pcm:Area. pcm:Region rdfs:subClassOf pcm:Continent.
Example Inferred Facts	pcm:Region rdfs:subClassOf pcm:Area.
Meta-rule	rule "Class Transitiveness Inference" when $clazz: Clazz(parent != null , parent.parent != null) not (exists (Clazz(name==$clazz.name, parent == $clazz.parent.parent))) Clazz newclazz = new Clazz($clazz.getName(), $clazz.getParent().getParent()); insert(newclazz); end
Knowledge Expansion through Range Generalisation	
Example Facts	pcm:Area pcm:areaContainsArea pcm:Area. pcm:Country rdfs:subClassOf pcm:Continent. pcm:Continent rdfs:subClassOf pcm:Area. :Europe a pcm:Continent; pcm:areaContainsArea :Greece. :Greece a pcm:Country.
Example Inferred Facts	:Europe a pcm:Continent; pcm:areaContainsArea :Greece. :Greece a pcm:Area.
Meta-rule	rule "Knowledge Expansion through Range Generalisation" when $triple: KnowledgeTriple() $objectinstance: InstanceOfClazz(name == $triple.object.name,

```
                                                clazz != $triple.object.clazz
                                )
                    not (
                      exists(
                         $newtriple: KnowledgeTriple(
                                        predicate == $triple.predicate ,
                                        subject == $triple.subject,
                                        object.name == $triple.object.name ,
                                        object.clazz != $triple.object.clazz
                                )
                      )
                    )
                 then
                    KnowledgeTriple newtriple = new KnowledgeTriple(
                                                        $triple.getSubject() ,
                                                        $triple.getPredicate() ,
                                                        $objectinstance
                                                );
                    insert(newtriple);
end
```

Knowledge Expansion through Domain Generalisation	

Example Facts	`pcm:Area pcm:areaContainsArea pcm:Area.`
	`pcm:Country rdfs:subClassOf pcm:Continent.`
	`pcm:Continent rdfs:subClassOf pcm:Area.`
	`:Europe a pcm:Continent;`
	`  pcm:areaContainsArea :Greece.`
	`:Greece a pcm:Country.`

Example Inferred Facts	`:Europe a pcm:Area;` `  pcm:areaContainsArea :Greece.`

Meta-rule	`rule "Knowledge Expansion through Domain Generalisation "`
	`  when`
	`  $triple: KnowledgeTriple()`
	`  $subjectinstance: InstanceOfClazz(`
	`  name == $triple.subject.name,`
	`  clazz != $triple.subject.clazz`
	`  )`
	`  not (`
	`  exists(`
	`  $newtriple: KnowledgeTriple(`
	`  predicate == $triple.predicate ,`
	`  object == $triple.object,`
	`  subject.name == $triple.subject.name ,`
	`  subject.clazz != $triple.subject.clazz`
	`  )`
	`  )`
	`  )`
	`  then`
	`  KnowledgeTriple newtriple = new KnowledgeTriple(`
	`  $subjectinstance ,`
	`  $triple.getPredicate() ,`
	`  $triple.getObject()`
	`  );`
	`  insert(newtriple);`
	`end`

Table 15: Fusing Drools with inferencing capabilities (meta-rules are expressed as Drools rules)

We note that on top of these meta-rules that are injected in the production memory of the expert system for supporting the basic types of ontological inferencing, we have created a number of meta-rules for checking the consistency of the knowledge triples that are fed into the expert system. These meta-rules are used for checking the consistency of:

- Knowledge Triples based on Domain Restrictions.
- Knowledge Triples based on Range Restrictions.
- Sub-property Definition based on Domain Restrictions.
- Sub-property Definition based on Range Restrictions.

Based on this approach, our enhanced expert system is able to inject to its working memory all the inferred facts necessary for correctly evaluating the active context-aware access control policies. For example, the inferred facts described in Table 1 of Section 3 can be produced without the use of a time-consuming DL reasoner but through the efficient assessment of the relevant meta-rules inserted into the production memory of the system. Specifically, only the *Property Transitivity* meta-rule is adequate for "realizing" that since `Athens` is located in `Greece` (which is located in `South Europe`) then `Athens` is also located in `South Europe` (see also Table 15). In addition, since the `Subject` that is requesting access to sensitive data is located in `Athens` (which the expert system just inferred that is located in `South Europe`) then the `Subject` is also located in `South Europe`, allowing for the successful evaluation of the relevant policy and resulting to the access permission.

7 RELATED WORK

This section provides brief overviews of existing approaches to access control in cloud environments; it also outlines existing approaches to context modelling, as well as to the declarative representation of policies.

7.1 Access Control Schemes and Approaches

Recently, several access control schemes for safeguarding sensitive data in the cloud –hence alleviating the security concerns associated with cloud storage– have been proposed. Several schemes focus on secure data sharing among a group of users [20–23]. These generally rely on computationally-expensive re-encryptions of the shared data in order to protect them against security threats from internal users. An interesting approach that does not require re-encryption is the Secure Data Sharing in Clouds (SeDaSC) methodology proposed in [24]. In particular, SeDaSC provides protection against insider threats through efficient forward and backward access control schemes. More specifically, the SeDaSC methodology is based on a trusted cryptographic server (CS) which is responsible for encrypting sensitive data files, each with a single symmetric key, before storing them in the cloud; the CS is also responsible for determining whether a particular user (a data requestor) can read, or write to, a sensitive data file. To this end, the CS employs access control lists (ACLs) defined for each data file by the file's owner. It also provides a key-partitioning scheme whereby, for each user included in the ACL of a data file, the corresponding symmetric key used for encrypting the file is divided into two portions such that both of them are required for decrypting the data; the one portion is kept at the CS, whereas the other one is securely sent to the user. Through a user's portion of the key, the CS can determine whether a user is eligible to read, or write, a sensitive data file; it can also protect against insider security threats by enabling forward access control that precludes users that are removed from an ACL from accessing the corresponding data file; finally, it enables backward access control by forbidding new users that are added to an ACL to view prior updates to the corresponding data file. In contrast to the work reported in this paper, however, the SeDaSC methodology –as well as the approaches reported in [20–23]– does not take into account the contextual information upon which an access control decision may depend; in addition, access control decisions are taken solely on the basis of syntactically-parsed information from the ACLs without any possibility for semantic inferencing; last but not least, it lacks any means of reasoning about the correctness and consistency of the access control policies.

Other approaches to cloud-based access control focus on exploiting trust relationships between the data owners and data requestors [25–27]. The work in [27] in particular proposes a multi-dimensional scheme that is based on individual trust values that are calculated by the data owner, as well as on public reputation values that are provided by one or more reputation centres (RCs). More specifically, the data owner encrypts the data with a single symmetric key K which is then divided into a number of distinct parts $K_0, K_1, \dots, K_n$ $(n \geq 0)$ such that

the decryption of the data requires all $n + 1$ key parts. K_0 is encrypted by the data owner with a public attribute key that corresponds to a particular trust attribute of interest, whereas each of $K_1, \ldots, K_n$ is encrypted with the public key of a distinct RC. The encrypted data and keys are uploaded to the cloud. Each time now a user requests access to the encrypted data, the CSP forwards the request to the RCs and to the data owner in order to determine whether the user enjoys the required reputation and trust for accessing the data – a determination based on minimum trust and reputation thresholds set by the data owner. If the user is deemed eligible to access the data, the RCs and the data owner securely send the appropriate keys to the user who then proceeds to decrypt and access the data. The proposed scheme is flexible in that it can be based solely on individual trust values (calculated by the data owner), or solely on public reputation values (provided by the RCs); it can also be based on any number of RCs (naturally, the greater the number of RCs the more accurate the reputation value). In contrast to the work reported in this paper, however, trust-based access control schemes generally ignore the contextual information pertaining to an access request that often needs to be taken into account in order to arrive at an access decision. They also lack the means of performing any semantic inferencing during policy enforcement or of reasoning about the correctness and consistency of the access control policies.

Trust-based access control schemes can impact the privacy of the trustor (data owner) as the evaluation of trust may reveal sensitive information about the trustor (such as his/her preferences, opinions, beliefs and interests). It can also threaten the privacy of the RCs that provide trust and reputation evidence. Safeguarding privacy is therefore of utmost importance for the viability of trust-based access control schemes. To this end, the work in [28] proposes two schemes to preserve privacy in trust evaluation. The schemes are based on two independent entities that are assumed not to collude with each other: the Authorised Proxy (AP), which is responsible for access control and management of aggregated trust evidence data, and the cloud-based Evaluation Party (EP) that processes the data collected from a number of trust evidence sources. The EP processes the data in an homomorphically-encrypted form and yields an encrypted trust "pre-evaluation result". Subsequently, when a user requests the pre-evaluation result, the EP first checks with the AP whether the user is eligible to access this result; if the check is positive: either (i) the AP re-encrypts the pre-evaluation result (which has already been obtained from the EP) such that it can be decrypted by the requestor and sends it to the requestor (1st scheme); or (ii) the AP is prevented from obtaining the pre-evaluation whilst still allowing its decryption by the requestor (2nd scheme). The 1st scheme is more computationally-efficient than the 2nd one, whereas the 2nd one is more secure as the AP is not fully trusted.

An entirely different approach is proposed in [29] which focuses on hardening the network functions virtualisation infrastructure (NFVI) in CSP data centres. The NFVI is the set of all hardware and software components required for the deployment of virtualised network functions (VNFs) such as load balancers, firewalls, IDSs, etc. The approach introduces the NFVI Trust Platform (NFVI-TP): a middleware that provides a root trusted module (RTM) that ensures that every component deployed on top of the NFVI-TP is trustworthy (e.g. by verifying that it is provided by a trustworthy party). To this end, the NFVI-TP dynamically deploys security functions such as resource access functions, data access functions, encryption/decryption and authentication functions, etc., as well as trust evaluation functions, trust management functions, and recommender functions. Through the use of these functions, the NFVI-TP ensures that the various VNFs deployed on top of it perform in a secure and trustworthy manner. No attempt is made, however, to take into account any contextual information pertaining to access requests.

7.2 Modelling Context in Access Control

A number of approaches to context modelling have been proposed. In [30, 31] detailed reviews of context models are provided that range from key-value models, to graphical models, mark-up schemes, object-oriented models, logic-based models and ontology-based models. In [32], Miele et al. propose a context model approach that was initially developed for mobile devices and later extended for capturing the knowledge that lurks in service-based applications [33]. In [34], an ontological model of the W4H classification for context is proposed. W4H stands for "who, where, when, what, how" and provides a set of generic classes, properties, and relations that exploit the five semantic dimensions of identity, location, time, activity and device profiles. A similar approach is reported in [35], where the 'five Ws' of context are identified: Who, What, Where, When, and Why. In [36], ContextUML is proposed – an approach that uses a UML-based modelling language specifically designed for Web services. ContextUML considers that context contains any information that can be used by a Web service to adjust both its execution and its output.

Exploiting context in access control mechanisms is a clear direction of on-going research. Even dedicated context-aware extensions to traditional access control models (e.g. Role-based Access Control - RBAC) either do not cover all the aspects of the contextual information required with a reusable and extensible security related context model, or are proven cumbersome to maintain in dynamic environments where potential requestors are not known at design-time and often change their roles [37]. On the other hand, the ontological models that exist (e.g. [34]) do not cover all the security requirements associated with the lifecycle of a cloud application (i.e. both bootstrapping and operation phases). Usually, they fail to cover the full range of contextual elements that are associated with the security enhancement of the sensitive data managed by the cloud applications, or they are driven by heavy inferencing that is inefficient [38].

7.3 Modelling Policies

Turning now to the semantic representation of policies and policy rules, a number of relevant approaches have been proposed in the literature [39–41]. These generally rely on the expressivity of DLs, and particularly on OWL, for capturing the various knowledge artefacts that underpin the definition of a policy. In [39], KAoS is presented – a generic framework offering: (i) a human interface layer for the expression of policies that constrain the actions that an agent is allowed to perform in a given context; (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 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 [40] Rei is proposed – a framework for specifying, analysing and reasoning about policies. Rei adopts OWL-Lite 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 a set 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*. These variables are purportedly required for expressing policy rules in which no concrete values are provided for one or more of the contextual attributes – e.g. rules of the form "subject s is allowed to access object o only when s is located in the *same area* as another subject s'". 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 [41] 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, Conceptual Graphs (CGs) [42] may be used for capturing the various knowledge artefacts, and the properties thereof, involved in the definition of an access control policy. In particular, an approach analogous to the one proposed in [42] may be utilised whereby both the knowledge artefacts that characterise an access request, as well as the ones that characterise an access control policy are modelled in terms of CGs. Then, in order to determine whether the policy is applicable to the request, the degree of similarity between the two CGs is calculated possibly through the use of semantic inferencing; if a match is determined, the policy is deemed applicable. Nevertheless, CGs are not currently as widespread as other semantic technologies (such as OWL) and therefore generally lack the extended tooling support that these technologies enjoy (e.g. in terms of graphical editors and reasoning mechanisms that perform semantic inferencing).

Finally, markup languages such as RuleML [43], XACML [8], SAML [44] and WS-Trust [45] provide declarative formalisms for the specification of policies. Nevertheless, they do not provide any means of capturing the knowledge that dwells in policies. This brings about the following disadvantages: (i) it precludes any form of semantic inferencing when evaluating access request, as well as when identifying inter-policy relations; (ii) it

leads to ad-hoc reasoning about policy compliance, one which is tangled with the particular vocabularies that are utilised for articulating the rules according to which the reasoning takes place.

8 CONCLUSIONS

This paper has proposed a generalised ontological model for the semantic representation of context-aware access control policies. The model is founded on the basis of a set of relevant *knowledge artefacts* that accurately capture a wide range of contextual attributes that must be taken into account in order to permit, or deny, an access request. The model bears the following seminal characteristics. (i) It is capable of taking into account the context pertaining to *any* entity that is deemed relevant to a request and not just to the subject of are request. (ii) It enables stakeholders to accurately define –by priming the HLO with suitable constraints– the *structure* by which their access control policies must abide and thus infuse into these policies their particular security and business requirements. (iii) It enables *reasoning* –performed automatically by a *policy validator*– about the well-formedness of a particular access control policy, i.e. about its abidance by the constraints stipulated by the stakeholder in the HLO. Moreover, the proposed generalisation paves the way for automated reasoning regarding the identification of potential *inter-policy relations*, such as contradicting or subsuming polices, that may affect the effectiveness of the policies.

Another important aspect of this work is the exploitation of the proposed ontological model through the implementation of a reference authorisation engine that adheres to the ABAC paradigm. More specifically, we have devised and implemented a context-aware access control engine that essentially materialises one of the core artefacts of XACML 3.0, namely the PDP. Moreover, we have exploited a widely-adopted expert system (Drools) and enhanced it with reasoning capabilities by injecting into its production memory a series of meta-rules. These meta-rules substitute some of the most valuable functionalities of DL reasoners with respect to ontological reasoning, for they provide the advantage of rapidly producing inferred facts. Such inferred facts, along with the use of the proposed ontological template, enable the efficient evalution of context-aware access control policies in a manner appropriate for highly-dynamic cloud environments.

We are currently in the process of constructing an editor for facilitating the priming of the HLO with well-formedness constraints. The editor parses the underlying context model and prompts the user to specify which concepts must, may or must not be embodied in an ABAC rule, as well as their corresponding allowable cardinalities. Moreover, for each concept the editor prompts the user to select its allowable values, i.e. all those instances that may participate in the formation of the rule.

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] *What's Hindering the Adoption of Cloud Computing in Europe?* Available online: https://blog.cloudsecurityalliance.org/2015/09/15/whats-hindering-the-adoption-of-cloud-computing-in-europe/. Cloud Security Alliance.

[2] Veloudis, S., Verginadis, Y., Patiniotakis, I., Paraskakis, I., Mentzas, G., 2016. Context-aware Security Models for PaaS-enabled Access Control. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER'16),* April 23-25, 2016, Rome, Italy.

[3] Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller R., and Scarfone K., 2014. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. NIST.

[4] Veloudis, S., Paraskakis, I., Petsos, C., Verginadis, Y., Patiniotakis, I., & Mentzas, G. (2017). An Ontological Template for Context Expressions in Attribute-Based Access Control Policies. In *Claus Pahl, Donald Ferguson, Jorge Cardoso, Markus Helfert, Víctor Méndez Muñoz (Eds.) Proceedings 7th International Conference on Cloud Computing and Services Science (CLOSER 2017) Porto, Portugal, April 24-26* (pp. 123–134). SCITEPRESS - Science and Technology Publications.

[5] PaaSword Deliverable 2.1, 2015. Available online: https://www.paasword.eu/deliverables/.
[6] RDF 1.1 Turtle, 2014. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.
[7] Veloudis, S. and Paraskakis, I. (2016). Defining an Ontological Framework for Modelling Policies in Cloud Environments. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Luxembourg City* (pp. 277–284). IEEE Computer Society.
[8] OASIS, 2013. *OASIS eXtensible Access Control Markup Language (XACML).* Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.
[9] PaaSword Deliverable 2.2, 2015. Available online: https://www.paasword.eu/deliverables/.
[10] OWL 2 Web Ontology Reference New Features and Rationale. W3C Recommendation, 2012. Available online: https://www.w3.org/TR/2012/REC-owl2-new-features-20121211/.
[11] Veloudis, S., Paraskakis, I., & Petsos, C. (2017). Foundations for Designing, Defining, Validating and Executing Access Control Policies in Cloud Environments. In Flavio De PaoliStefan SchulteEinar Broch Johnsen (Ed.), *Service Oriented and Cloud Computing Proceedings of the 6th IFIP WG 2.14 European Conference, ESOCC 2017, Oslo, Norway, September 27-29, 2017* (pp. 75–82,). LNCS, 10465.
[12] Horrocks, I., Kutz, O., Sattler, U.:. The even more irresistible *SROIQ*. In: Doherty, P., Mylopoulos, J., Welty, C. A. (eds.) *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning* (KR'06), pp 57—67, AAAI Press (2006).
[13] Tao, J., Sirin, E., Bao, J. and McGuinness, D. L.: Integrity Constraints in OWL, *In Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10),* Atlanta, Georgia, USA, July 11-15 (2010)
[14] INCITS, 2016. INCITS 499: Information technology - Next Generation Access Control - Functional Architecture. Available online at: https://standards.incits.org/apps/group_public/project/details.php?project_id=1648.
[15] Ferraiolo, D., Chandramouli, R., Kuhn, R., & Hu, V., 2016. Extensible access control markup language (XACML) and next generation access control (NGAC). In Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control (pp. 13-24). ACM.
[16] Drools 2017. Drools Expert System. Available Online: http://www.drools.org.
[17] Bobek, S., Misiak, P., 2017. Framework for Benchmarking Rule-Based Inference Engines. In *Proceedings of the International Conference on Artificial Intelligence and Soft Computing ICAISC 2017,* pp 399-410.
[18] List of Reasoners. Available online at: http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/. Accessed on 28 August 2017.
[19] Bock, J., Haase, P., Ji, Q., & Volz, R., 2008. Benchmarking OWL reasoners. In Proceedings of ARea2008-Workshop on Advancing Reasoning on the Web: Scalability and Commonsense, Tenerife.
[20] L. Xu, X. Wu, and X. Zhang, "CL-PRE: A certificateless proxy re-encryption scheme for secure data sharing with public cloud," in *Proc. 7th ACM Symp. Inf., Comput. Commun. Security*, 2012, pp. 87–88.
[21] S. Seo, M. Nabeel, X. Ding, and E. Bertino, "An Efficient Certificate-less Encryption for Secure Data Sharing in Public Clouds," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 9, pp. 2107–2119, Sep. 2013.
[22] A. N. Khan, M. M. Kiah, S. A. Madani, M. Ali, and S. Shamshirband, "Incremental proxy re-encryption scheme for mobile cloud computing environment," *J. Supercomput.*, vol. 68, no. 2, pp. 624–651, May 2014.
[23] Y. Chen and W. Tzeng, "Efficient and provably-secure group key management scheme using key derivation," in *Proc. IEEE 11th Int. Conf. TrustCom*, 2012, pp. 295–302.
[24] M. Ali, R. Dhamotharan, E. Khan, S. U. Khan, A. Vasilakos, K. Li, and A. Y. Zomaya. "SeDaSC: Secure Data Sharing in Clouds", IEEE Sys. J. vol. 11, no 2, June 2017.
[25] G. Lin, D. Wang, Y. Bie, M. Lei, "MTBAC: a mutual trust based access control model in cloud computing", China Communications, vol. 11, no. 4, pp. 154 – 162, 2014
[26] Z. Yan, X. Li, R. Kantola, "Personal data access based on trust assessment in mobile social networking", in Proc. of IEEE TrustCom2014, 2014, pp. 989 - 994.
[27] Z. Yan, X. Li, M. Wang and A. V. Vasilakos, "Flexible Data Access Control Based on Trust and Reputation in Cloud Computing," in *IEEE Transactions on Cloud Computing*, vol. 5, no. 3, pp. 485-498, July-Sept. 1 2017.
[28] Zheng Yan, Wenxiu Ding, Valtteri Niemi, and Athanasios V. Vasilakos. 2016. Two Schemes of Privacy-Preserving Trust Evaluation. *Future Gener. Comput. Syst.* 62, C (September 2016), 175-189.
[29] Z. Yan, P. Zhang, A. V. Vasilakos, "A security and trust framework for virtualized networks and software-defined networking", Security Comm. Networks 2016, vol. 9, pp3059–3069.
[30] Strang, T., Linnhoff-Popien, C., 2004. A Context Modeling Survey. *In Workshop on Advanced Context Modelling, Reasoning and Management, (UbiComp'04)* - The Sixth International Conference on Ubiquitous Computing. Nottingham, England.
[31] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., & Riboni, D., 2010. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 161-180.
[32] Miele, A., Quintarelli, E., Tanca, L., 2009. A methodology for preference-based personalization of contextual data. In *ACM Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'09)*, pp. 287-298, Saint-Petersburg, Russia.

[33] Bucchiarone, A., Kazhamiakin, R., Cappiello, C., Nitto, E., & Mazza, V., 2010. A context-driven adaptation process for service-based applications. In *ACM Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems (PESOS'10)*, pp. 50-56, Cape Town, South Africa.

[34] Truong, H.-L., Manzoor, A., Dustdar, S., 2009. On modeling, collecting and utilizing context information for disaster responses in pervasive environments. In *ACM Proceedings of the first international workshop on Context-aware software technology and applications (CASTA'09)*, pp. 25-28, Amsterdam, The Netherlands.

[35] Abowd, G., & Mynatt, E., 2000. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction (TOCHI)* - Special issue on human-computer interaction in the new millennium, 29-58.

[36] Sheng, Q., & Benatallah, B., 2005. ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services Development. In *Proceedings of the International Conference on Mobile Business (ICMB'05)*, pp. 206-212, IEEE Computer Society.

[37] Heupel, M., Fischer, L., Bourimi, M., Kesdogan, D., Scerri, S., Hermann, F., Gimenez, R., 2012. Context-Aware, Trust-Based Access Control for the di.me Userware. In *Proceedings of the 5th International Conference on New Technologies, Mobility and Security (NTMS'12)*, pp. 1-6, Istanbul, Turkey, IEEE Computer Society.

[38] Verginadis, Y., Mentzas, G., Veloudis, S., Paraskakis, I., 2015. A Survey on Context Security Policies. In *Proceedings of the 1st International Workshop on Cloud Security and Data Privacy by Design (CloudSPD'15)*, co-located with the 8th IEEE/ACM International Conference on Utility and Cloud Computing, Limassol, Cyprus, December 7-10.

[39] Uszok, A., Bradshaw, J., Jeffers, R., Johnson, M., Tate, A., Dalton, J. and Aitken, S., 2005. KAoS Policy Management for Semantic Web Services. *IEEE Intel. Sys.*, vol. 19, no. 4, pp. 32 – 41.

[40] 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)

[41] 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).

[42] Z. Fu, F. Huang, X. Sun, A. Vasilakos and C. N. Yang, "Enabling Semantic Search based on Conceptual Graphs over Encrypted Outsourced Data," in *IEEE Trans. on Services Computing*, vol. PP, no. 99, pp. 1-11.

[43] Specification of Deliberation RuleML 1.01, 2015. Available online: http://wiki.ruleml.org/index.php/Specification_of_Deliberation_RuleML_1.01.

[44] Security Assertions Markup Language (SAML) Version 2.0. Technical Overview, 2008. Available online: https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf

[45] WS-Trust 1.3, 2007. Available online: http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.doc.

APPENDIX

The table below presents formal definitions –expressed in OWL 2– of the meanings intended for the abbreviations introduced through the classes `pac:XContextExpression` (`X ::= AND | OR | XOR | NOT`) of Figure 2. The identifiers prefixed with ":" represent anonymous individuals; `_:p1` and `_:p2` represent (arbitrary) instances of the Security Context Model (i.e. instances of the classes depicted in Figure 2).

Abbreviation	**Definition**

`:expr a pac:ANDContextExpression;`
`  pac:hasParameter _:p1;`
`  pac:hasParameter _:p2.`
	`:expr a owl:intersectionOf pac:ContextExpression _:x.`
	`_:x a owl:intersectionOf _:x1 _:x2.`
	`_:x1 a owl:Restriction;`
	`  owl:onProperty pac:hasParameter;`
	`  owl:hasValue _:p1.`
	`_:x2 a owl:Restriction;`
	`  owl:onProperty pac:hasParameter;`
	`  owl:hasValue _:p2.`

`:expr a pac:NOTContextExpression;`
`  pac:hasParameter _:p1;`
	`:expr a owl:intersectionOf pac:ContextExpression _:x.`
	`_:x a owl:complentOf _:x1.`
	`_:x1 a owl:Restriction;`
	`  owl:onProperty pac:hasParameter;`
	`  owl:hasValue _:p1.`

`:expr a pac:ORContextExpression;`	`:expr a owl:intersectionOf pac:ContextExpression _:x.`

pac:hasParameter _:p1; pac:hasParameter _:p2.	_:x a owl:Restriction; owl:onProperty pac:hasParameter; owl:someValuesFrom owl:oneOf _:p1 _:p2.
:expr a pac:XORContextExpression; pac:hasParameter _:p1; pac:hasParameter _:p2.	:expr a owl:intersectionOf pac:ContextExpression _:x. _:x a owl:intersectionOf _:x1 owl:complentOf _:x2. _:x1 a owl:Restriction; owl:onProperty pac:hasParameter; owl:someValuesFrom owl:oneOf _:p1 _:p2. _:x2 a owl:intersectionOf _:x3 _:x4 _:x3 a owl:Restriction; owl:onProperty pac:hasParameter; owl:hasValue _:p1. _:x4 a owl:Restriction; owl:onProperty pac:hasParameter; owl:hasValue _:p2.

Interpretation of `pac:XContextExpression`