An Ontological Template for Context Expressions in Attribute-Based Access Control Policies

Simeon Veloudis[1], Iraklis Paraskakis[1], Christos Petsos[1], Yannis Verginadis[2], Ioannis Patiniotakis[2] 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

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

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

Abstract: By taking up the cloud computing paradigm enterprises are able to realise significant cost savings whilst increasing their agility and productivity. However, due to security concerns, many enterprises are reluctant to migrate their critical data and operations to the cloud. 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 *ontological template* for the *semantic representation* of *context expressions* in access control policies. This template is underpinned by a suitable set of interrelated concepts that generically capture a wide range of *contextual knowledge* that must be considered during the evaluation of policies.

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 of 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 (CSA, 2015). A promising approach to alleviating these concerns is to assist application developers in infusing adequate *access control policies* in cloud applications for safeguarding their data against unauthorised accesses (Veloudis et al., 2016). 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, 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 (Veloudis et al., 2016) outlined the construction of a generic ontological model for access control policies, one that bears the following characteristics. Firstly, it is underlain by a suitable Context-aware Security Model – an extensible framework of relevant interrelated concepts that capture a wide range of relevant contextual attributes, thus embracing the attribute-based access control (ABAC) scheme (Hu et al., 2014). Secondly, it uses a generic and extensible formalism for expressing access control policies, one which unravels the definition of a policy from the code employed for enforcing it, bringing about the following seminal advantages: (i) it allows 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) it forms an adequate basis for reasoning *generically* about the correctness and consistency of the policies, hence about the effectiveness of the access control that they ultimately exercise.

Nevertheless, the ontological model devised in (Veloudis et al., 2016) assumes that the contextual attributes articulated in an access control policy are invariably associated with the subject of a request. It therefore ignores 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).

This paper proposes an extension to the ontological model outlined in (Veloudis et al., 2016) 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 provides the means to declaratively capture, in terms of a suitable ontological model, the *knowledge* that lurks behind the various contextual attributes that are associated with the entities relevant to a request. This is a crucial feature for the following reasons: (i) It enables the evaluation of a request against an access control policy to be performed, and reasoned about, at the *semantic level*. For instance, referring to the example above, suppose that, at the time a request is issued by the subject `s`, the object `o` is reported to reside in a data centre in Athens, Greece. Clearly, `o` satisfies the contextual condition set by the policy (Greece is an EU country). For this to be inferred, however, the *knowledge* that lurks behind the contextual attributes that participate in the definition of the relevant context needs to be accurately captured. (ii) It paves the way for performing automated reasoning about potential *inter-policy relations* such as the identification of contradicting or subsuming polices. 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`).

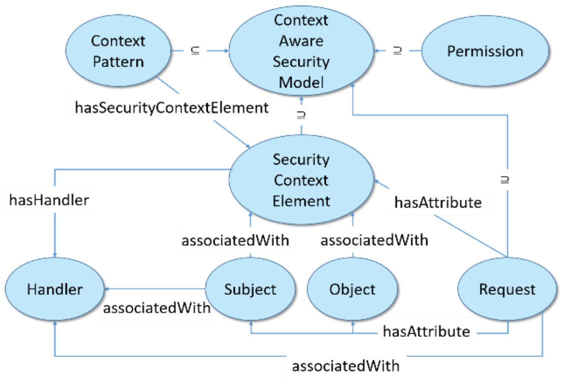


Figure 1: Context-aware security meta-model (namespace prefixes are omitted to reduce clutter).

The rest of this paper is structured as follows. Section 2 presents the ontologically-expressed *context 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 proposes an ontological template for access control rules and, crucially, for the context expressions on which these rules rely. Section 5 outlines how context-based inferencing can be performed in order to identify inter-policy relations. Section 6 presents related work and, finally, Section 7 presents conclusions.

2 MODELLING CONTEXT

Figure 1 depicts an updated meta-model that captures the main facets of the Context-aware Security Model presented in (Veloudis et al., 2016). The main change with respect to the meta-model of (Veloudis et al., 2016) is the extraction of the classes `pcm:Object`, `pcm:Subject`, `pcm:Request` and `pcm:Handler` from the class `pcm:SecurityContextElement` (the namespace prefix `pcm`, as well as all other namespace prefixes encountered in this section, are defined as part of the Context Model (PaaSword Deliverable 2.1, 2015)). As discussed in Section 4, this change simplifies the incorporation of context in access control policies.

In addition, as outlined in Section 3, it renders semantic inferencing at the level of requests simpler to comprehend and exploit.

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.
- `pcm:Object` – Describes the protected resources – e.g. relational or non-relational data, files, software artefacts that manage sensitive data.
- `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 Context Model. 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 handler include, for example, authentication handlers, request handlers, location handlers, IP-address-to-city handlers, etc.
- `ppm:Permission` – This class refers to the allowed actions that an individual 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

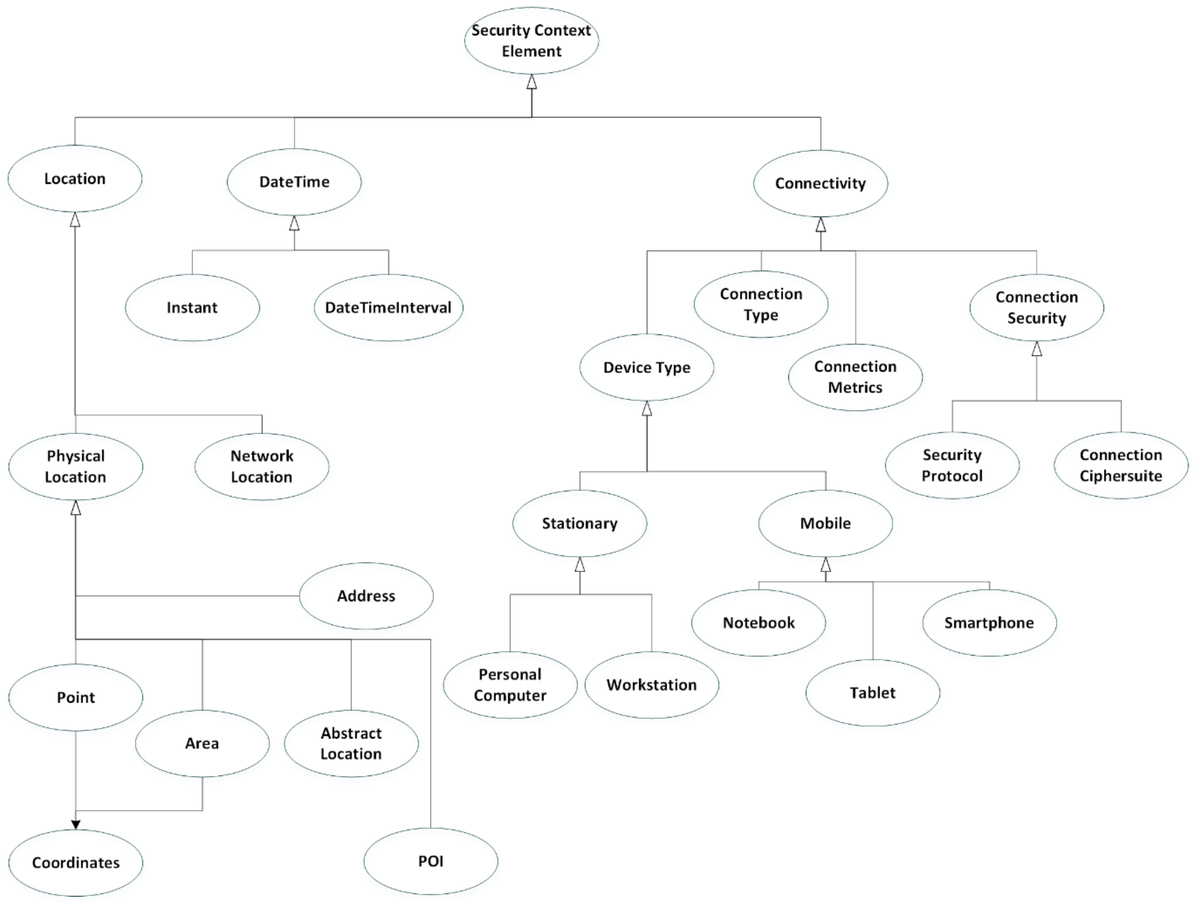


Figure 2: Security Context Element overview diagram (namespace prefixes are omitted to reduce clutter)

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.

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` and `pcm:NetworkLocation`. A physical location may involve: an address, a geographical position, an area, an abstract location and/or a point of interest 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 an 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_GCM_SHA256 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 aims at: (i) interrelating a request with its relevant subjects and objects; (ii) interrelating the subjects and objects of a request, as well as the request itself, with contextual attributes drawn from the `pcm:SecurityContextElement` class. 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 (and not to the subjects or objects that are associated with a request – e.g. the date/time at which a request takes place) are piggy-backed to a request through the object property `pcm:hasAttribute` which interrelates the classes `pcm:Request` and `pcm:SecurityContextElement`.

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

Table 1: Inferred facts expressed as RDF triples (Turtle notation (RDF 1.1 Turtle, 2014)).

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.`

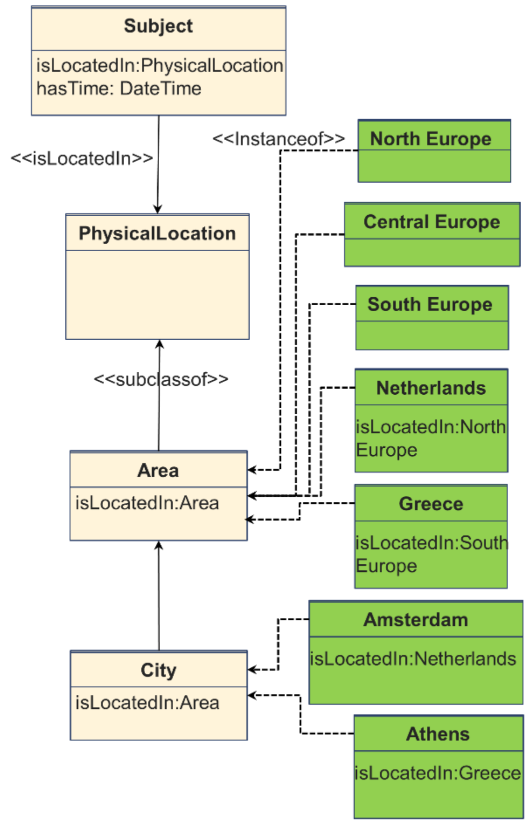


Figure 3: Inferencing based on property transitivity

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 city level. 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.

4 INCORPORATING CONTEXT IN ACCESS CONTROL POLICIES

4.1 An Ontological Model for ABAC Policies

Figure 4 depicts the ontological model for ABAC policies proposed in (Veloudis et al., 2016). Following the XACML standard (OASIS, 2013), each ABAC policy comprises one or more ABAC rules. An ABAC rule is associated with a set of relevant *knowledge artefacts* (see Table 2) that need to be taken into account for deciding whether an access request must be permitted or denied. In this respect, ABAC rules can be regarded as *knowledge containers* for their encompassing policies.

Ontologically, an ABAC rule takes the form of an instance of the class `pac:ABACRule` depicted in Figure 4 (the `pac` namespace prefix is defined as part of the ontological model for ABAC policies (PaaSword Deliverable 2.2, 2015)). The knowledge artefacts attached to an ABAC rule are described generically in terms of the *ontological template* of

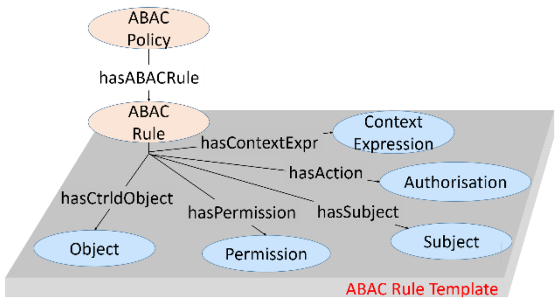


Figure 4: Ontological model for ABAC policies (namespace prefixes are omitted to reduce clutter).

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

Knowledge artefact	Description	Ontological representation	Associating property
Controlled object	Identifies the sensitive object on which access is requested.	Instance of the `pcm:Object` class outlined in Section 2.	`pac:hasControlledObject` <u>Domain</u>: `pac:ABACRule` <u>Range</u>: `pcm:Object`
Authorisation	Determines the type of authorisation granted (either 'permit', or 'deny').	Instance of the class `pac:Authorisation` (either the instance `pac:permit` or the instance `pac:deny`).	`pac:hasAuthorisation` <u>Domain</u>: `pac:ABACRule` <u>Range</u>: `pac:Authorisation`
Action	Identifies the operation requested to be performed on the controlled object.	Instance of the class `ppm:DataPermission` (see Section 2).	`pac:hasAction` <u>Domain</u>: `pac:ABACRule` <u>Range</u>: `ppm:DataPermission`
Actor	Identifies the entity requesting access to the controlled object.	Instance of the `pcm:Subject` class (see Section 2).	`pac:hasActor` <u>Domain</u>: `pac:ABACRule` <u>Range</u>: `pcm:Subject`
Context expression	A propositional logic expression that identifies the contextual conditions that must be satisfied in order to permit (or deny) a request.	Instance of the class `pac:ContextExpression` outlined in Section 4.2.	`pac:hasContextExpression` <u>Domain</u>: `pac:ABACRule` <u>Range</u>: `pac:ContextExpression`

Figure 4. More specifically, each class of this template identifies a particular knowledge artefact, whilst each object property associates such an artefact with an ABAC rule. Table 2 briefly elaborates on the classes and properties of the ontological template of Figure 4. In the remaining of this paper, we shall focus on the *context expression* artefact.

4.2 An Ontological Model for ABAC Policies

In (Veloudis et al., 2016), context expressions take the form of reified versions of the ontological template depicted in Figure 5. More specifically, a context expression is represented by an instance of the class `pac:ContextExpression`. The various contextual attributes that are bound by a context

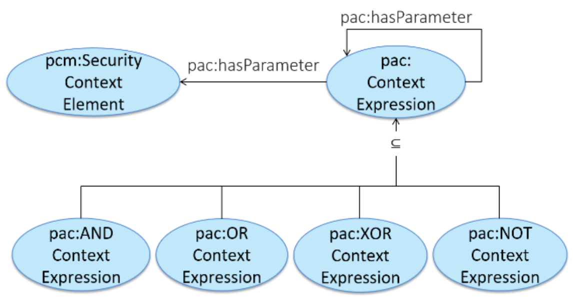

pac:hasParameter
pcm:Security Context Element
pac:hasParameter
pac: Context Expression
⊆
pac:AND Context Expression
pac:OR Context Expression
pac:XOR Context Expression
pac:NOT Context Expression

Figure 5: Ontological template for context expressions

expression take the form of *parameters* of the expression. Ontologically, these parameters are represented as instances of the classes of the Security Context Element depicted in Figure 2. They are associated with the individual that represents a context expression through the object property `pac:hasParameter`.

The parameters of a context expression may be combined through the usual logical connectives. To this end, the classes `pac:XContextExpression` (where `X` stands for one of `AND`, `OR`, `XOR`, `NOT`) are provided (see Figure 5). These are subclasses of the `pac:ContextExpression` class. Their intended meaning is as follows: if a context expression is represented by an instance of the class say `pac:ANDContextExpression`, its parameters, i.e. the contextual attributes associated with it through the `pac:hasParameter` property, are interpreted as being pairwise *conjuncted*. Likewise, if a context expression is represented by an instance of the class say `pac:ORContextExpression`, its parameters are interpreted as being pairwise *disjuncted*. Analogous interpretations apply to the rest of the classes. Table 3 presents an example context expression that is represented by the individual `:expr`. The expression 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

Table 3: Context expression example.

```
: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.
```

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.

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 the range of the object property `pac:hasParameter` (see Figure 5). The example of Table 4 shows a recursively-defined context expression that includes the context expression represented by the individual `:expr1` as a parameter.

Nevertheless, context expressions that take the form of instantiations of the model depicted in Figure 5 make no provisions of indicating the entity with which they are associated. In this respect, they are assumed to be invariably associated with the subject of a request. As discussed in Section 1, this constitutes a limitation for it ignores the fact that contextual attributes may need to be associated with other entities such as the object of a request, the request itself, or with any other entity whose context is deemed relevant for deciding whether to permit or deny a request. In this respect, Section 4.3 below presents an extension to the ontological template for context expressions of Figure 5 that allows contextual attributes to be attached to *any* entity that is associated with a request.

4.3 A Generalised Ontological Template for Context Expressions

A straightforward approach to allowing a context expression to be attached to any entity that is associated with 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 he the actual subject of the request; (ii) the controlled 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.

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

Table 4: Recursive context expression example. :para1 and :para2 are defined as in Table 3.

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

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

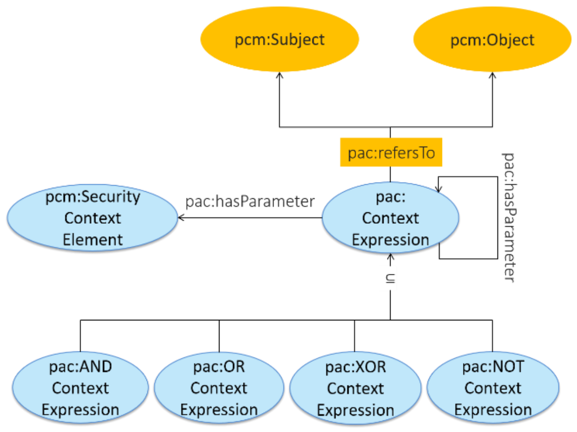
```
:rule1 a pac:ABACRule;
   pac:hasAction :Read;
   pac:hasActor :s;
   pac:hasAuthorisation pac:positive;
   pac:hasControlledObject :o.
:s pac:hasContextExpression :expr1.
:expr1 a pac:ContextExpression;
   pac:hasParameter :para2.
:rule2 a pac:ABACRule;
   pac:hasAction :Read;
   pac:hasActor :s;
   pac:hasAuthorisation pac:positive;
   pac:hasControlledObject :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 6: Using pac:hasContextExpression twice (:para2 is defined as in Table 2).

```
:rule1 a pac:ABACRule;
   pac:hasAction :Read;
   pac:hasActor :s;
   pac:hasAuthorisation pac:positive;
   pac:hasControlledObject :o.
   pac:hasContextExpression :expr1.
:s pac:hasContextExpression :expr1.
:expr1 a pac:ContextExpression;
   pac:hasParameter :para2.
:rule2 a pac:ABACRule;
   pac:hasAction :Read;
   pac:hasActor :s;
   pac:hasAuthorisation pac:positive;
   pac:hasControlledObject :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.
```

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.

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.



pcm:Subject
pcm:Object
pac:refersTo
pcm:Security Context Element
pac:hasParameter
pac: Context Expression
pac:hasParameter
⊆
pac:AND Context Expression
pac:OR Context Expression
pac:XOR Context Expression
pac:NOT Context Expression

Figure 6: Extended ontological template

A more elegant solution is to extend the ontological template of Figure 5 through the introduction of a new object property, namely pac:refersTo (see Figure 6). 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. 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

Table 7: Associating context using the extended model of Figure 6(:para2 is defined as in Table 3).

```
:rule1 a pac:ABACRule;
   pac:hasAction :Read;
   pac:hasActor :s;
   pac:hasAuthorisation pac:positive;
   pac:hasControlledObject :o;
   pac:hasContextExpression :expr1.
:expr1 a pac:ContextExpression;
   pac:hasParameter :para2;
   pac:refersTo :s.
:rule2 a pac:ABACRule;
   pac:hasAction :Read;
   pac:hasActor :s;
   pac:hasAuthorisation pac:positive;
   pac:hasControlledObject :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 8: Associating context using the extended model of Figure 6. `:para1`, `:para2` and `:para3` are defined as in Table 3; `para4` specifies the type of device (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.
```

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`.

Finally, it is 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`.

5 OPTIMISING CONTEXT-BASED INFERENCING AT THE POLICY LEVEL

One of the main virtues of declaratively specifying (through the ontological template of Figure 6) 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.

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`

Table 9: Checking property subsumption

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

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`.

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 9. 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 10. For instance, the context expressions represented by the individuals `:expr1` and `:expr2` in the example of Table 9, 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.

Table 10: Sub-properties of `pac:hasParameter`

Name (`pac` prefix)	Domain (`pac` prefix)	Range (`pcm` prefix)
`hasLocationParameter`	`ContextExpression`	`Location`
`hasDateTimeParameter`	`ContextExpression`	`DateTime`
`hasConnectivityParameter`	`ContextExpression`	`Connectivity`

6 RELATED WORK

A number of approaches to context modelling have been proposed in the literature, recognising the necessity of formally capturing knowledge in order to drive interactions in service-based applications. In (Strang & Linnhoff-Popien, 2004; Bettini et al., 2010) 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. Miele et al., (2009) 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 (Bucchiarone et al., 2010). In (Truong et al., 2009) 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 (Abowd and Mynatt, 2000), where the 'five Ws' of context are identified: Who, What, Where, When, and Why. In (Sheng, 2005), 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 (Heupel, 2012). On the other hand, the ontological models that exist (e.g. (Truong et al., 2009)) 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 (Verginadis et al., 2015).

Turning now to the representation of policies and policy rules, a number of relevant approaches have been proposed in the literature (Uszok et al., 2005; Kagal et al., 2003; Neijdl et al., 2005). These generally rely on the expressivity of Description Logics (DLs), and particularly on OWL (2004), for capturing the various knowledge artefacts that underpin the definition of a policy. In (Uszok et al., 2005), 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 (Kagal et al., 2003) 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 (Nejdl et al., 2005) 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, markup languages such as RuleML (2015), XACML (OASIS, 2013), SAML (2008) and WS-Trust (2007) 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.

7 CONCLUSIONS

This paper has proposed an *ontological template* for the semantic representation of context expressions in access control policies. We argue that such a template facilitates developers in expressing *effective* security policies which give rise to security controls that are appropriate for dynamic and heterogeneous cloud environments. The proposed template is founded on the basis of relevant *knowledge artefacts* that accurately capture a wide range of contextual attributes that must be taken into account during the evaluation of a policy. One of the virtues of the proposed ontological template is that it enables the evaluation of a request against an access control policy to be performed, and reasoned about, at the *semantic level*; furthermore, our ontological template paves the way for the performance of automated reasoning about potential *inter-policy relations* such as the identification of subsuming polices.

Another seminal advantage offered by the proposed template is the fact that it is expressed in a generic, interoperable and extensible RDF vocabulary that lends itself to, and therefore paves the way for, a series of *correctness checks* that are performed automatically by a policy validator. These checks aim at assessing the validity of a policy with respect to a higher-level ontology (HLO) that captures all those knowledge artefacts that a policy must, may, or must not, comprise. These correctness checks are clearly of utmost importance for they increase assurance on the effectiveness of the policies.

Currently, we are in the process of constructing the policy validator and the HLO. The constraints expressed through the HLO are articulated on the basis of the Integrity Constraints (IC) semantics for OWL 2 proposed in (Tao et al., 2010). We also intend to construct an editor for priming the HLO.

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

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.

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.

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.

CSA, 2015. *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.

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.

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.

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.

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).

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.

OWL Web Ontology Language Reference. W3C Recommendation, 2004. Available online: http://www.w3.org/TR/owl-ref/.

PaaSword Deliverable 2.1, 2015. Available online: https://www.paasword.eu/deliverables/.

PaaSword Deliverable 2.2, 2015. Available online: https://www.paasword.eu/deliverables/.

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

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

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

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.

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.

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)

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.

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.

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

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.

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