A Generic Framework for Representing Context-aware Security Policies in the Cloud

Simeon Velouds[1], Yiannis Verginadis[2], Ioannis Patiniotakis[2], Iraklis Paraskakis[1] and Gregoris Mentzas[2]

[1]South East European Research Centre (SEERC), The University of Sheffield, International Faculty CITY College, 24 Prox. Koromila St., 54622, Thessaloniki, Greece
{sveloudis, iparaskakis}@seerc.org
[2]Institute of Communications and Computer Systems, National Technical University of Athens, Athens, Greece
{jverg, ipatini, gmentzas}@mail.ntua.gr

Abstract. Enterprises are increasingly embracing cloud computing in order to reduce costs and increase agility in their everyday business operations. Nevertheless, due mainly to confidentiality, privacy and integrity concerns, many organisations are reluctant to migrate their sensitive data to the cloud. In order to alleviate these security concerns, this paper proposes the *PaaSword framework*: a generic PaaS solution that provides capabilities for guiding developers through the process of defining appropriate *policies* for protecting their sensitive data. More specifically, this paper outlines the construction of an extensible and declarative formalism for representing policy-related knowledge, one which disentangles the definition of a policy from the code employed for enforcing it. It also outlines the construction of a suitable Context-aware Security Model, a framework of concepts and properties in terms of which the policy-related knowledge is expressed.

Keywords: Context-aware security, Ontologies, Linked USDL, Policies, Access Control, Data privacy, Security-by-design

1 Introduction

There is generally consensus among analysts that cloud computing is being adopted by enterprises at an ever-increasing pace [1]. The main force that drives this trend is the new economy-based paradigm that cloud computing introduces [2] which enables significant cost savings, whilst accelerating the deployment and utilisation of new applications [3]. Nevertheless, the increasing adoption of cloud computing transforms the enterprise IT environment into a matrix of interwoven infrastructure, platform and application services that are delivered remotely, over the Internet, by diverse service providers [4]. These services may span not only different technologies and geographies, but also entirely different domains of ownership and control. This creates a set of unprecedented security vulnerabilities stemming mainly from the fact that corporate data reside in externally-controlled servers or untrusted cloud providers. Exploiting these

This work has been published in Markus Helfert, Donald Ferguson, Victor Méndez Muñoz, Jorge Cardoso (Eds): "*Cloud Computing and Services Science: 6th International Conference, CLOSER 2016, Rome, Italy, April 23-25, 2016, Revised Selected Papers*", pp. 339-359, Springer International Publishing, 2017, DOI: 10.1007/978-3-319-62594-2_17

vulnerabilities may result in data confidentiality and integrity breaches [5]. Evidently, the benefits offered by the cloud computing paradigm cannot fully materialise without addressing these new security challenges [6].

A promising approach to alleviating the security concerns associated with cloud computing is to assist application developers in defining effective security controls for safeguarding the sensitive data accessed through the cloud applications that they develop. To this end, in [6] we proposed the *PaaSword framework*, a generic PaaS solution that provides capabilities for guiding developers through the process of defining appropriate *policies* for protecting sensitive data. More specifically, three are the main kinds of policy that the PaaSword framework aims at supporting: (i) *Data encryption policies*, which determine the strength of the cryptographic protection of a sensitive object; (ii) *data fragmentation and distribution policies*, which determine the manner in which sensitive data objects are fragmented and distributed to different physical servers for privacy reasons; (iii) *access control policies*, which determine when to grant, or deny, access to sensitive data.

In order to *effectively* guide developers in defining security policies, the PaaSword framework bears two seminal characteristics. Firstly, it hinges upon an adequate scheme that takes into account the inherently dynamic and heterogeneous nature of cloud environments. Secondly, it captures the *knowledge* that lurks behind such a scheme (e.g. actions, subjects, locations, environmental attributes, etc.) using a generic and extensible formalism, one which can be tailored to the particular needs of different cloud applications. The first characteristic calls for the incorporation of the notion of *context* in policies, i.e. the consideration of dynamically-changing contextual attributes that may characterise data accesses. It therefore involves the development of a re-usable and generic *Context-aware Security Model* which goes beyond the traditional context-insensitive security (e.g. DAC, MAC, RBAC [7]). The second characteristic calls for the adoption of a declarative approach to modelling policy-related knowledge, one which is orthogonal to the code of any particular cloud application and which can be easily adapted to suit the needs of any such application.

The aim of this paper is twofold. On the one hand, it outlines the construction of the Context-aware Security Model. On the other hand, it outlines the construction of an extensible and declarative formalism for representing policy-related knowledge, one which disentangles the definition of a policy from the code employed for enforcing it, bringing about the following advantages: (i) it allows the policy-related knowledge to be extended and instantiated to suit the needs of a particular application, independently of the code employed by the application; (ii) it forms an adequate basis for reasoning generically about the correctness and consistency of the security policies, hence about the effectiveness of the security controls that these policies give rise to.

The rest of this paper is organised as follows. In Section 2, we elaborate on a *context-aware security model* that will be used as the underlying vocabulary for describing the three kinds of policy that the PaaSword framework supports. In Section 3, we introduce a *policy model* that allows for the semantic description of the policies. In Section 4, we present a case study that demonstrates the use of the context-aware security model, as

well as of the policy model. Finally, Section 5 briefly discusses relevant work and Section 6 concludes the paper by presenting the next steps for the implementation and evaluation of the proposed approach.

2 Context-aware Security Model

In this section, we present a context-aware access model, which can be used by the developers in order to annotate database Entities, Data Access Objects (DAO) or any other web endpoints that give access to sensitive data managed by cloud applications. This context model conceptualises the aspects, which must be considered during the selection of a data-access policy. These aspects may be any kind of information which is machine-parsable [8]; indicatively they may include the user's IP address and location, the type of device that s/he is using in order to interact with the application as well as his/her position in the company. These aspects can be interpreted in different ways during the security policy enforcement. In particular, the context-aware access model can set the basis for determining which data is accessible under which circumstances.

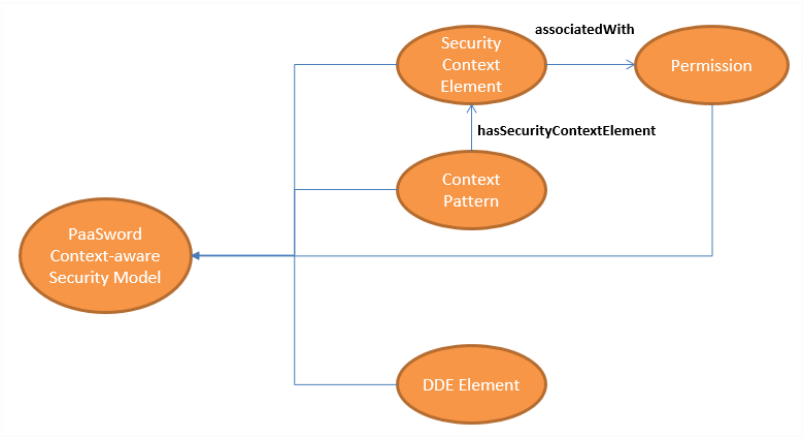

Fig. 1. Context-aware security meta-model

2.1 Context-aware Security Meta-Model

In Fig. 1, we present a meta-model that captures the main facets of the `Context-aware Security Model` along with their associations. Specifically, this model comprises of two different kinds of facets that may give rise to:

- Dynamic security controls – These controls grant or deny access to sensitive data on the basis of dynamically evolving contextual attributes, which are associated with the entity requesting the access. The relevant model facets are:
  - `Security Context Element`
  - `Permission`

—Context Pattern

- Static security controls - These controls are independent of any dynamically evolving contextual attributes. They mainly correspond to the distribution and cryptographic protection features that certain data artefacts must have and affect the bootstrapping phase of a cloud application. The relevant model facet is the:
  —Data Distribution and Encryption Element (DDE)

According to this meta-model, instances of these aforementioned facets formulate the Context-aware Security Model. Furthermore, Context Pattern Elements are directly associated to Security Context Elements (through the hasSecurityContextElement property) in order to be defined, while the latter can be associated with certain Permission Elements. Due to space limitations, we discuss only the context model facets that are relevant to access control.

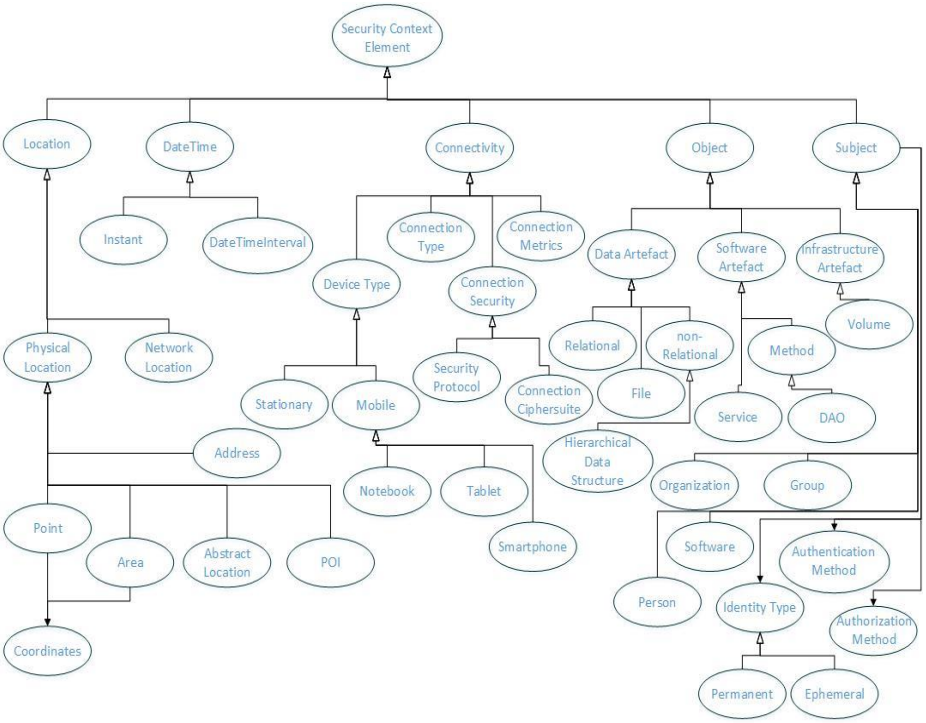

Fig. 2. Security Context Element Overview

2.2 Context Model Facets

This section provides an elaboration of the initial set of facets that have been included in the part of the model that gives rise to dynamic security controls. We note that all these model facets are focused on the aspects relevant to access control for cloud services.

Security Context Element.
The `Security Context Element` refers to the following five top-level concepts (see Fig. 2):

- `Location` - This class describes a physical and/or a network location where data are stored or from which a particular entity is requesting to access data.
- `DateTime` - This class describes the specific chronological point expressed as either instant or interval that characterises an access request (extends `owl-time:TemporalEntity`).
- `Connectivity` - This class captures the information related to the connection used by the Subject for accessing sensitive data (see Fig. 2).
- `Object` - This class refers to any kind of artefacts that should be protected based on their sensitivity levels. These artefacts may refer to (non-) relational data, files, software artefacts that manage sensitive data or even infrastructure artefacts used.
- `Subject` - An instance of this class represents the agent seeking access to a particular data artefact. This can be an organization, a person, a group or a service (extends `foaf:Agent, goodrelations:BusinessEntity, goodrelations:ProductOrService`).

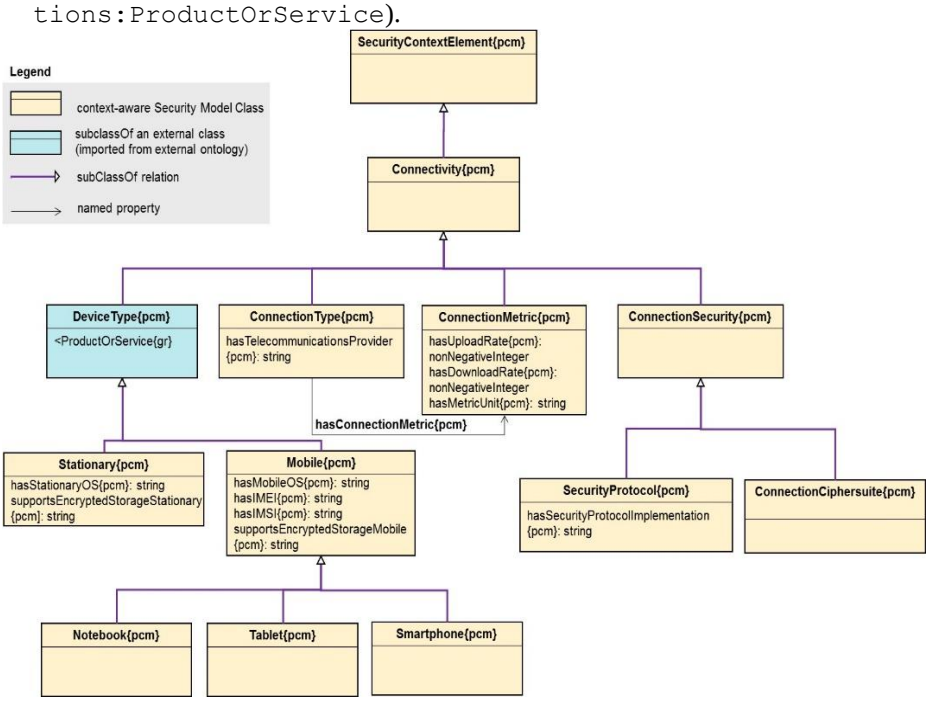

Fig. 3. UML Class diagram for the Connectivity context element

In Fig. 3, we provide further details regarding the `Connectivity` top level concept that include subclasses, imported or extended external classes, data and object properties. The identifier `pcm` (stands for PaaS Control Model) recognises the namespace underlying the classes and properties of the proposed vocabulary. Due to space limitations

the details of all the top level concepts are not explained in this paper but they are available in the following URL: http://imu.ntua.gr/software/context-aware-security-model.

Context Pattern.
The next facet of this model is the `Context Pattern model` that includes the following top-level concepts:

- `Location pattern` - It refers to recurring motives of data accesses that are recognized with respect to the Location context element.
- `DateTime pattern` - It refers to recurring motives of data accesses that are recognized with respect to the DateTime context element.
- `Connectivity pattern` - It refers to recurring motives of data accesses that are recognized with respect to the Connectivity context element.
- `Object pattern` - It refers to recurring motives of data accesses that are recognized with respect to the Object context element.
- `Permission pattern` - It refers to recurring motives of data accesses that are recognized with respect to the Permission element.
- `Access Sequence Pattern` - It refers to data accesses that are recognized by any preceding access actions made by a particular Subject (extends `Kaos:AccessAction`).

For the above vocabulary, we use the identifier pcpm (stands for PaaS Context Pattern Model) for recognising the respective namespace of underlying classes and properties.

Permission.
Another important facet is the `Permission model` that involves the following top-level concepts (see Fig. 6):

- `Data Permission` - This class refers to any action allowed by a Subject upon a data entity (extends schema.org:Action)
- `DDL Permission` - This class reveals the data definition language (DDL) related actions on a specific Object.

The `Data Permission` involves four subclasses:

- `Datastore Permission` – It describes any action allowed by a Subject upon a data entity in a datastore (e.g. Search, List, Select, Insert, etc.)
- `File Permission` - It describes any action allowed by a Subject upon a file (e.g. Read, ChDir, Move, Delete, etc.)
- `WebEndpoint Permission` – It describes any web endpoint related action that is allowed upon a data artefact (e.g. Get, Put, Post, Delete).
- `Volume Permission` - It refers to any access permission to a dedicated infrastructure artefact.

The `DDL Permission` involves two subclasses:

- `Datastore DDL Permission` – It describes any DDL related permission on a datastore (e.g. Create, Alter, Drop).
- `File System Structure Permission` - It describes any DDL related permission on a file (e.g. CreateDir, RenameDir, CopyDir, DeepCopyDir, ChOwner, etc.).

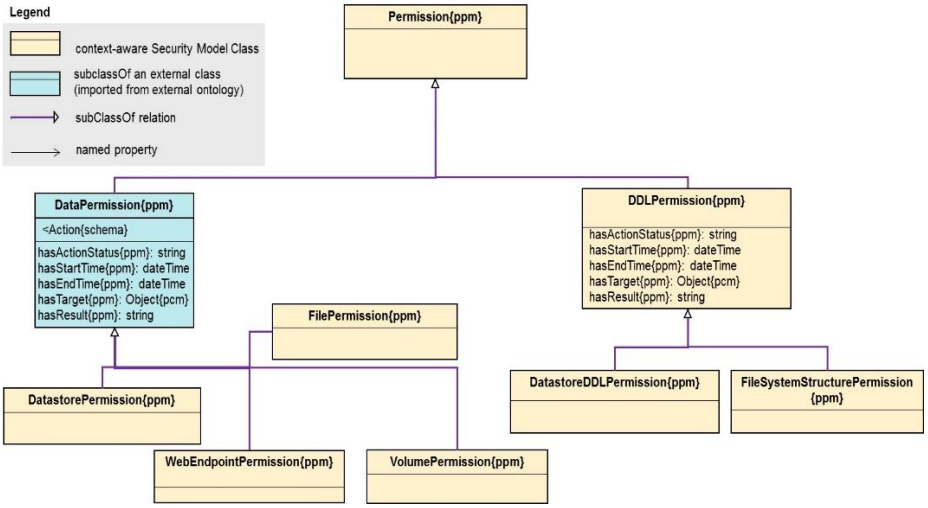

Fig. 4. UML Class diagram for the Permission context element

For the above vocabulary, we use the identifier ppm (stands for PaaS PaaS Permission Model) for recognising the respective namespace of underlying classes and properties. In Section 3, we demonstrate the way that these contextual elements that give rise to dynamic security controls, can set the basis for developing a policy model for paas-enabled access control.

3 Policy model for PAAS-enabled access control

As already mentioned in Section 1, three are the main types of security policy that the PaaSword framework aims at supporting: (i) *Data encryption* policies. These determine the strength of the cryptographic protection that each sensitive object enjoys for confidentiality reasons. They give rise to security controls enforceable during bootstrapping of a cloud application. (ii) *Data fragmentation and distribution* policies. These determine the manner in which sensitive data objects must be fragmented and distributed to different physical servers for privacy reasons. They too give rise to security controls enforceable during application bootstrapping. (iii) *Access control* policies. These are essentially ABAC policies that determine when to grant, or deny, access to sensitive data on the basis of dynamically-evolving contextual attributes associated with the en-

tity requesting the access. Context awareness is deemed of utmost importance for leveraging the security of cloud-based applications which by definition operate in dynamic and heterogeneous environments. Access control policies give rise to security controls dynamically enforceable during application execution time. Due to space limitations, in this paper we only consider access control policies.

3.1 Access Control Policy Model

We argue that, in order to aid application developers in defining effective ABAC policies for any kind of sensitive data, the PaaSword framework must be underpinned by an underlying ontological model, one which bears the following characteristics:

- It is founded on a framework of relevant interrelated concepts which capture all those knowledge artefacts that are required for describing an ABAC policy. Such a framework is provided by the vocabulary outlined in Section 2.
- It uses an extensible formalism for accommodating the framework of interrelated concepts, hence expressing ABAC policies. Such a representation disentangles the definition of a policy from the code employed for enforcing it, offering the following seminal advantages: (i) It allows the framework of relevant interrelated concepts to be extended and instantiated, independently of the code employed by the application. Such an extension/instantiation aims at customising the framework to the particular needs of a given application. (ii) It forms an adequate basis for reasoning generically about the correctness and consistency of the ABAC policies, hence about the effectiveness of the security controls that these policies give rise to.

ABAC Policy Rules.
Following an approach inspired by the XACML standard [9], an ABAC policy comprises one or more rules. A rule is the most elementary structural element and the basic building block of policies. A generic template for ABAC rules is provided in Table 1.

Table 1. Generic ABAC rule template

[actor] with [context expression] has [authorisation] for [action] on [controlled object]

The template defines a generic structure, in terms of relevant attributes, to which all ABAC rules in the PaaSword framework adhere. It comprises several attributes which are further elaborated below.

- *actor* identifies the subject who may request access to perform an operation on a sensitive object; it draws its values from the `pcm:Subject` class of the Security Context Element model defined in Section 2.2.
- *context expression* is a Boolean expression which identifies the environmental conditions that must hold in order to permit, or deny, the performance of an operation on a sensitive object. Context expressions are further elaborated in the following subsection.

- *authorisation* determines the type of authorisation (positive i.e. 'permit', or negative i.e. 'deny') that is granted.
- action identifies the operation that may, or may not, be performed on the protected sensitive object; it draws its values from the `ppm:Permission` class of the Security Context Element model defined in Section 2.2.
- *controlled object* identifies the sensitive object on which access is requested; it draws its values from the `pcm:Object` class of the Security Context Element model defined in Section 2.

In our ontological model, an ABAC rule takes the form of an instance of the class `pac:ABACRule` (see Fig. 5) of our framework. A number of object properties are attached to this class which are intended to capture the aforementioned attributes. As depicted in Fig. 5, these associate the `pac:ABACRule` class with an appropriate framework of relevant classes from the vocabulary of Section 2.2 which adequately capture the attributes of the ABAC rule template. The identifier `pac` (stands for PaaS Access Control) recognises the namespace underlying the classes and properties of the proposed ontological model.

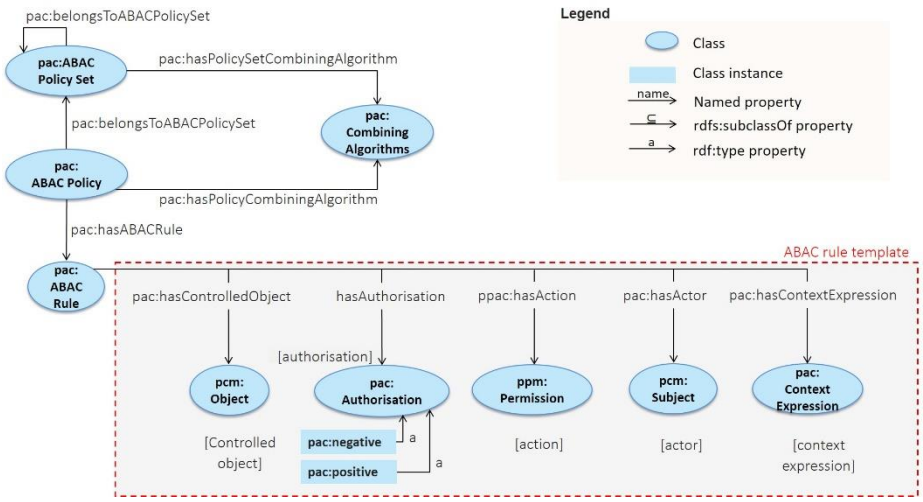

Fig. 5. ABAC ontological model

Context Expressions.
A context expression takes the form of an instance of the class `pac:ContextExpression` of our framework (see Fig. 5). It specifies a number of constraints on the values of one or more instances drawn from the vocabularies `pcpm:ContextPattern` and `pcm:SecurityContextElement` defined in Section 2.1. The class `pac:ContextExpression` is associated with these vocabularies through the object properties `pac:hasPatternParameter` and `pac:hasParameter` respectively depicted in Fig. 6. As we would expect, a context expression may combine two or more constraints using logical connectives (conjunction, disjunction, exclusive disjunction,

negation). In order to capture such combinations of constraints, the `pac:ContextExpression` class encompasses a subclass for each logical connective. A context expression may be defined recursively, in terms of one or more other context expressions. This is captured by associating the `pac:ContextExpression` class with itself through the properties `pac:hasParameter` and `pac:hasPatternParameter` (see Fig. 6).

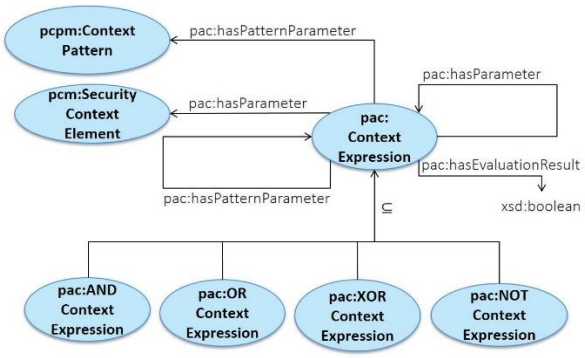

Fig. 6. Context expression ontological model

ABAC Policies and Policy Sets.
In our ontological model, an ABAC policy takes the form of an instance of the class `pac:ABACPolicy` which represents It is associated with the rules that it comprises through the property `pac:hasABACRule`. An ABAC policy may comprise a multitude of ABAC rules which potentially evaluate to different (and conflicting) access control decisions. This calls for a combining algorithm which reconciles the different decisions and determines an overall decision for the entire policy [9]. An example of a combining algorithm is the 'deny-overrides' algorithm, whereby a policy evaluation resolves to 'deny' if at least one of its constituent rules evaluates to 'deny', or if none of them evaluates to 'permit'. A combining algorithm takes the form of an instance of the class `pac:CombiningAlgorithms` depicted in Fig. 5. A combining algorithm is attached to an ABAC policy through the property `pac:hasPolicyCombiningAlgorithm`.

Following an approach inspired by the XACML standard [9], access control policies are grouped into *policy sets*. In our ontological model, a policy set takes the form of an instance of the class `pac:ABACPolicySet` (see Fig. 5). A policy is associated with its enclosing policy set through the property `pac:belongsToABACPolicySet`. A policy set may exhibit a hierarchical structure and comprise one or more other ABAC policy sets. This recursive inclusion is captured by rendering the `pac:belongsToABACPolicySet` property applicable to ABAC policy sets too (see Fig. 5). ABAC policy sets are also associated with combining algorithms. As in the case of policies, these reconcile the potentially different access control decisions to which the policies comprising a policy set may evaluate.

It is to be noted here that analogous policy models have been devised for the rest of the policy types outlined at the beginning of Section 3.

3.2 Access Control Policies in Linked USDL

Section 3.1 outlined a model for the generic representation of ABAC policies. This section demonstrates how this model can be incorporated into the ontological framework provided by Linked USDL [10], and in particular, into USDL-SEC – Linked USDL's security profile (USDL stands from Unified Service Description Language). By capitalising on USDL-SEC, our approach avoids the use of bespoke, non-standards-based, ontologies for the representation of ABAC policies (see Section 5.2 for a relevant outline of such ontologies). Instead, it is based on a diffused ontological framework which has recently attracted considerable research interest.

In addition, the adoption of Linked USDL brings about the following advantages [11]: (i) Linked USDL relies on existing widely-used RDF(S) vocabularies (such as GoodRelations, FOAF and SKOS), whilst it can be easily extended through linking to further existing, or new, RDF(S) ontologies. In this respect, it promotes knowledge sharing whilst it increases the interoperability, reusability and generality of our framework. (ii) By offering a number of different profiles, Linked USDL provides a holistic and generic solution able to adequately capture a wide range of business details. This is important for our work as it allows us to capture the business aspects of the security policies encountered within our framework. (iii) Linked USDL is designed to be easily extensible through linking to further existing, or new, RDF(S) ontologies. This is particularly important for our model as it facilitates seamless integration with the Context-aware security model of Section 2. (iv) It provides ample support for modelling, comparing, and trading services and service bundles. It also provides support for specifying, tracking, and reasoning about the involvement of entities in service delivery chains. This is important for our work for it allows comparisons to be drawn between different policy models that may potentially be offered through our framework.

The USDL-SEC Vocabulary.
USDL-SEC identifies five top-level concepts: *Security Profile*, *Security Goal*, *Security Mechanism*, *Security Technology* and *Security Realization Type* (see Fig. 7). The *Security Profile* is a root concept that encompasses the different security profiles to which a cloud service, or application, may adhere. Each security profile is associated with one or more security goals. This gives rise to the *Security Goal* concept which encompasses a number of sub-concepts each representing a distinct security goal. A complete list of all security goals provided by USDL-SEC is depicted in Fig. 7. Each security goal is associated with one or more security mechanisms through which it is implemented. This gives rise to the *Security Mechanism* concept which encompasses a number of sub-concepts each representing a particular kind of security mechanism – a complete list of all the security mechanism kinds provided by USDL-SEC is depicted in Fig. 7. Each security mechanism is associated with one or more security technologies through

which it is realised, giving rise to the *Security Technology* concept. In addition, a security mechanism is associated with a particular layer of the ISO/OSI protocol stack at which it is realised (e.g. the network or the application layer). This gives rise to the concept *Security Realization Type* that specifies such a layer.

The concepts and their associations identified above are formalised in terms of classes of the ontology, and each concept association takes the form of an object property[1]. In fact, four object properties are introduced: `hasSecurityGoal` which associates a security profile with its corresponding security goal; `isImplementedBy` which associates a security goal with the mechanism that achieves it; `isRealizedByTechnology` which associates a security mechanism with the technology that implements it; `hasSecurityRealizationType` which associates a security mechanism with the ISO/OSI layer at which it is realised. In addition, the fact that a concept forms a sub-concept of another concept is captured through the SKOS `broader` property.

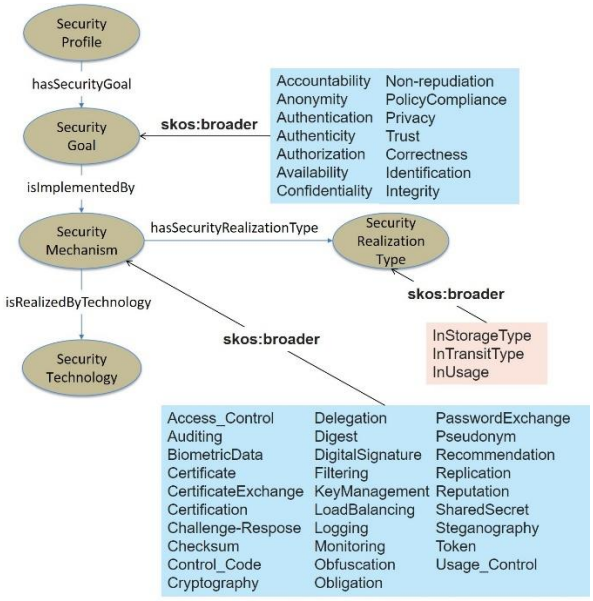

Fig. 7. USDL-SEC

The above framework of classes and properties lays the foundations for constructing a set of ontological templates suitable for the semantic representation of the three kinds of security policy that the PaaSword project supports. The following subsection, provides an account of how this framework is reified in order to give rise to an ontological template for the representation of ABAC policies. Analogous accounts apply to the

[1] All USDL-SEC classes and properties are prefixed with the `usdl-sec` namespace. To avoid notational clutter, this namespace is omitted here.

other two kinds of policy, namely Data Encryption policies and Data Fragmentation and Distribution policies.

Incorporating ABAC Policies into USDL-SEC.
At the highest level of abstraction, the ABAC policy model forms, essentially, a particular security profile to which a cloud application may adhere. In this respect it is modelled as an instance of USDL-SEC's `SecurityProfile` class, namely `pac:PaaSAccessControlProfile`. A security profile is associated, through the object property `hasSecurityGoal`, with one or more security goals from the USDL-SEC class `SecurityGoal`. In the case of ABAC policies, the security goal is *authorisation*. This is modelled in Fig. 8 by associating the instance `pac:PaaSAccessControlProfile` with an instance, say `pac:AccessControlGoal`, of the `Authorization` class through the property `hasSecurityGoal`. The `Authorization` class forms a sub-concept of `SecurityGoal`.

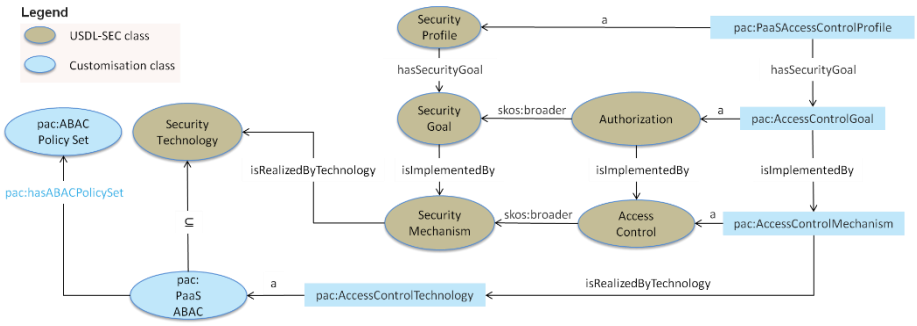

Fig. 8. USDL-SEC customisation (only classes and properties used in this paper are depicted)

The authorisation goal is achieved by means of a suitable access control mechanism. USDL-SEC provides a layer of abstraction, namely the concept `SecurityMechanism`, for the specification of such a mechanism. In particular, it provides the class `AccessControl`, a sub-concept of `SecurityMechanism`, an instance of which, say `pac:AccessControlMechanism`, represents the access control mechanism offered by the PaaSword framework. This instance is associated with the `pac:AccessControlGoal` instance through the property `isImplementedBy`.

The access control mechanism represented by the instance `pac:AccessControlMechanism` is realised by means of some underlying concrete security technology. USDL-SEC provides a layer of abstraction, namely the concept `SecurityTechnology`, for the specification of such a technology. In our model, the access control mechanism is realised by the access control technology provided by the PaaSword framework. This is modelled by introducing the `pac:PaaSABAC` subclass (see Fig. 8), along with the instance `pac:AccessControlTechnology` which represents this access control technology. This instance is associated with the access control mechanism through the property `isRealizedByTechnology` (see Fig. 8).

The pac:PaaSABAC subclass is associated, through the property pac:hasABAC-PoliceSet, with the class pac:ABACPolicySet (the top concept of the ABAC policy model of Fig. 5). This essentially captures the fact that the access control mechanism is realised through the policies encompassed in one or more ABAC policy sets.

It is to be noted here that the policy models devised for the rest of the policy types outlined at the beginning of Section 3 are incorporated into USDL-SEC in an analogous manner.

4 Case Study

The purpose of this section is to demonstrate how the Context-aware Security model of Section 2 and the policy model of Section 3 can be further reified in order to give rise to concrete security policies. In particular, we present a number of policies – and their constituent rules – that form part of a fictitious, albeit realistic, scenario. This scenario is deliberately kept simple and it is by no means intended to form a fully-fledged use case. It is to be noted here that we are currently obliged to confine ourselves to such simple scenarios since all the involved individuals, and their relevant properties, must be entered into the model manually. Clearly, for a fully-fledged use case, this process would require a large amount of tedious work; consideration of such use cases is therefore deferred until an editor that automates this process is developed.

4.1 The Car Park Scenario

A company has been contracted to develop a smart system for managing vehicle traffic and parking spaces in a European city. As part of this system, cameras are installed in a number of privately-owned long-stay car parks. We assume that, in addition to providing real-time footage of the car park premises for security purposes, the cameras also capture and store the following data: (i) Which vehicle (if any) occupies a particular parking space; vehicles are identified by their registration plate numbers. (ii) The date a vehicle entered the car park; we assume that in long-stay car parks the exact time of entry, or exit, of a car is not of interest as costs are typically charged on a per-day basis. (iii) The date a vehicle exited the park. We assume that these data are stored in the database table ParkingPositions depicted in Fig. 9.

SpaceId	RegistrationPlate	EntryDate	ExitDate	CostPerDay(€)
1001	NIP5146	17/11/2015	21/11/2015	7
1002	NZY8547	13/11/2015	24/11/2015	7
⋮	⋮	⋮	⋮	⋮

Fig. 9. The ParkingPositions table

In order to compress costs and ensure storage elasticity, it has been decided to migrate the data contained in these tables to the cloud. To this end, SIEMENS' R&D

department undertook the development of a cloud application, namely AppDB, through which car park administrators can obtain access to these data. The application is available for Windows, Mac OS and Android devices.

In order to alleviate security concerns, the application was developed with the aid of the framework presented in Section 3, which allowed the incorporation of a number of security controls during application design time. These controls implement a set of security policies which take the form of reifications of the ABAC policy model presented in Section 3 (see Fig. 5).

4.2 ABAC Policy

As part of the long-stay car park scenario we require that a user of the AppDB application, typically an employee of a car park should only be allowed read/write access to the table of Fig. 9 from specific locations and during specific hours. This requirement aims at preventing situations whereby the user of the AppDB accesses the table from a public place thus giving the opportunity to a third party to inadvertently, or on purpose, look at – or even alter – the data stored in the table.

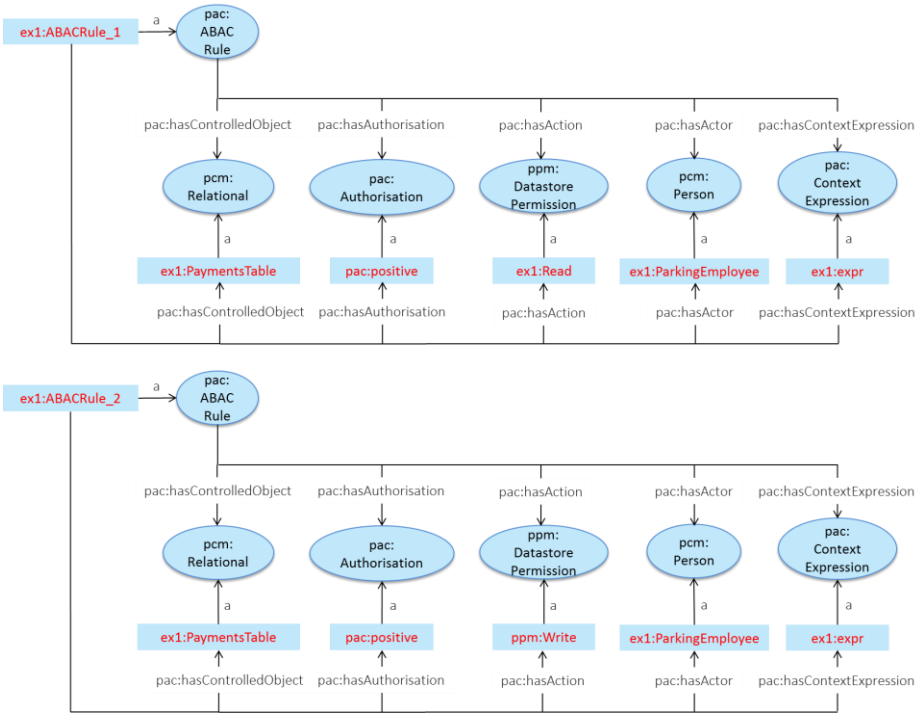

Fig. 10. Instantiated ABAC rule template

This requirement gives rise to two access control rules, one allowing read access from specific locations and during specific times, and one allowing write access from

the same locations and during the same times. These rules are represented by the individuals ex1:ABACRule_1 and ex1:ABACRule_2 respectively depicted in Fig. 10. These rules are further elaborated below. ABAC rule 1 takes the form:

```
ex1:ParkingOwner with ex1:expr has pac:positive for
ppm:Read on ex1:PaymentsTable
```

It reifies the generic rule template of Fig. 5. The rule is associated with a number of contextual attributes through the properties depicted in Fig. 10. These attributes are represented by the individuals illustrated in Fig. 10 and detailed in Table 2.

Table 2. ABAC rules contextual attributes

Object property	Individual	Instance of	Description
pbe:has Controlled Object	ex1:Payments Table	pcm: Relational	Associates ex1:ABACRule_1 with the relational table of Fig. 9 that it protects.
pbe:has Authorisation	pac:positive	pac: Authorisation	Associates ex1:ABACRule_1 with the positive (permit) authorisation.
pac:hasAction	ppm:Read	ppm:Datastore Permission	Associates ex1:ABACRule_1 with the read operation.
pac:hasActor	ex1:Parking Employee	pcm:Person	Associates ex1:ABACRule_1 with a parking employee.
pac:has Context Expression	ex1:expr	pac:Context Expression	Associates ex1:ABACRule_1 with a context expression that restricts the locations and the times from which the relational table of Fig. 10 can be accessed.

ABAC rule 2 takes the form:

ex1:ParkingOwner with ex1:expr has pac:positive for ppm:Write on ex1:PaymentsTable

This rule is associated with the same contextual attributes as the ones of ABAC rule 1.

4.3 Context Expression

As already indicated, the rules represented by the individuals ex1:ABACRule_1 and ex1:ABACRule_2 involve a context expression which restricts the locations from which, and the times during which, the relational table of Fig. 9 can be accessed. This context expression is represented by the individual ex1:expr depicted in Fig. 10. This individual is an instance of the class pac:ANDContextExpression (see Fig. 6) and therefore the parameters that it involves are logically conjuncted. These parameters are associated with ex1:expr through the object property pac:hasParameter. They are modelled in Fig. 11 by the individuals ex1:EmployeeWorkingHours and ex1:expr1. The former individual is an instance of the class pcm:DateTimeInterval introduced by the Context-aware Security model of Section 2. It is restricted to the required working hours (e.g. say 09:00 to 17:00) through the properties pcm:hasBeginning, pcm:hasEnd and pcm:hasTimeZone. The former two properties associate ex1:EmployeeWorkingHours with the xsd:dateTime values T09:00:00 and T17:00:00 respectively. The latter property associates ex1:EmployeeWorkingHours with the xsd:string value +02:00.

The latter individual, namely ex1:expr1, is yet another context expression modelled as an instance of the class pac:ORContextExpression. This means that its two parameters, namely ex1:Parking_1 and ex1:Parking_2 (see Fig. 11), are logically disjuncted. These parameters are instances of the class pcm:Point from the Context-aware Security model presented in Section 2. They represent the two parking buildings from the long-stay car park scenario.

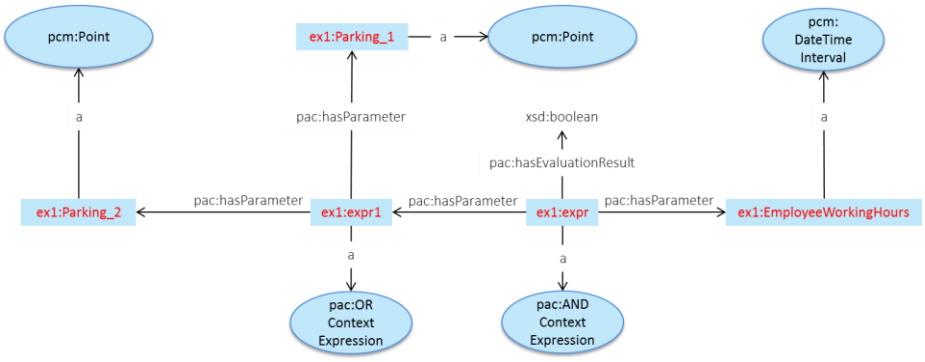

Fig. 11. Context expression for ex1:ABACRule_1 and ex1:ABACRule_2

5 Related work

The research outlined in this section is divided into two main categories: (i) works that are related to the Context-aware Security of Section 2; works that pertain to the Policy Model of Section 3.

5.1 Research related to the Context-aware Security Model

In the literature, there is a plethora of context models. For example, [12] and [13] review models of context that range from key-value models, to mark-up schemes, graphical models, object-oriented models, logic-based models and ontology-based models. An interesting context model is the one proposed in [14], which was initially developed for mobile devices and later extended for the use in service-based applications in [15]. Another example is the one in [16] who developed an ontological model of the W4H classification for context. The W4H ontology provides a set of general classes, properties, and relations exploiting the five semantic dimensions: identity (who), location (where), time (when), activity (what) and device profiles (how). Furthermore, authors exploited the concepts of the W4H ontology by including domain-independent common context concepts from existing work; e.g. FOAF, vCard, the OWL-Time Ontology, etc. The five dimensions of context have been also pointed out earlier by Abowd and Mynatt [17] who stated that context should include the 'five W': Who, What, Where, When, and Why. For example, by 'Who', they mean that it is not enough to identify a person as a customer; the person's past actions and service related background should also be identified for better service provision. 'What' refers to the activities conducted by the people involved in the context and interactions between them. 'Where' represents location data. 'When' is related to time. 'Why' specifies the reason for 'Who' did 'What'. 'Why' represents a complicated notion and acts as the driving force for context sensitive information systems. In addition to that, from the literature review we found interesting efforts that concerned modelling languages, which take context explicitly into account. The first such effort was ContextUML a UML-based modelling language that was specifically designed for Web service development and applies model-driven development principles; see [18]. In a Web-service-based environment, ContextUML considers that context contains any information that can be used by a Web service to adjust its execution and output.

The need for the exploitation of context in the access control mechanisms is quite evident from the state-of-the-art. Nevertheless, we found that even dedicated context-aware extensions to traditional access control models (e.g. Role-based Access Control - RBAC) either do not cover all the contextual elements with a reusable security related context model or are proven hard to maintain in dynamic environments where users often change roles or are not known a priori [19]. On the other hand, pure ontological models (e.g. [16], or even Attribute-based Access Control (ABAC) approaches (e.g. [20]) they do not seem to cover all the security requirements associated with the lifecycle of a cloud application (i.e. bootstrapping and run-time). Specifically, either they do not cover the full range of contextual elements that are associated with all the security

aspects of sensitive data managed by cloud applications or they are based on heavy inferencing that is considered as inefficient for such dynamic environments [21].

5.2 Research related to the Policy Model

Turning now to policies and policy-based applications, syntactic descriptions promote a declarative approach to policy expression, one which aims at replacing a trend whereby policies are encoded imperatively, as part of the same software that checks for their compliance. Several markup languages have been proposed for the declarative description of policies, some prominent examples being [22], [9], [23] and [24]. These generally provide XML-based syntaxes for expressing policy rules and sets. Nevertheless, such syntactic descriptions fail to capture the knowledge lurking behind policies. In this respect, they are merely data models that lack any form of semantic agreement beyond the boundaries of the organisation that developed them. Any interoperability relies on the use of vocabularies that are shared among all parties involved in an interaction.

In order to overcome the aforementioned limitations, semantically-rich approaches to the specification of policies have been brought to the attention of the research community. These generally embrace Semantic Web representations for capturing what we term action-oriented policies, i.e. policies which control when a particular actor or subject can perform a specified action on, or through the use of, a particular resource. These approaches typically employ ontologies in order to assign meaning to actors, actions and resources. Several works in the area of semantic policy representation have been reported in the literature [25], [26] and [27]. In [25], the authors presented KAoS – a general-purpose policy management framework which exhibits a three-layered architecture comprising:

- A human interface layer, which provides a graphical interface for policy specification in natural language.
- A policy management layer, which uses OWL [28] to encode and manage policy-related knowledge.
- A policy monitoring and enforcement layer, which automatically grounds OWL policies to a programmatic format suitable for policy-based monitoring and policy enforcement.

In [26] the authors proposed Rei – a policy specification language expressed in OWL-Lite [28]. It allows the declarative representation of a wide range of policies which control which actions can be performed, and which actions should be performed, by a specific entity. Furthermore, it defines a set of concepts (rights, prohibitions, obligations, and dispenations) for specifying and reasoning about access control rules. In this respect, it provides an abstraction which allows the specification of a desirable set of behaviours which are potentially understandable – hence enforceable – by a wide range of autonomous entities in open and dynamic environments.

In [27], the authors recognise that cloud computing, and in particular the concept of multi-tenancy, calls for policy-driven access control mechanisms. They propose an ontology-based framework to capture the common semantics and structure of different

types of access control policies (e.g. XACML policies, firewall policies, etc.), and facilitate the process of detecting anomalies in these policies. Their ontology captures the underlying domain concepts involved, the policy structure and the policy attributes. Particular types of access control policies are obtained by appropriately instantiating the ontology.

6 Conclusions

We have presented suitable vocabularies of concepts and properties, namely the Security Context Element, the Context Pattern and the Permission, which adequately captures the knowledge lurking behind ABAC policies. We have also proposed a generic ontological model for the abstract representation of ABAC policies, which disentangles the definition of a policy from the actual code, employed for enforcing it, bringing about the advantages outlined in Section 3.1. The model is underpinned by the Security Context Element vocabulary, and is incorporated into the ontological framework offered by USDL-SEC (Linked USDL's security profile). Such a model forms the basis of the PaaSword framework – essentially a security-by-design framework which aims at aiding cloud application developers in defining effective access control policies for any kind of sensitive data.

Any effective use of the ABAC policy model requires a mechanism through which it can be suitably customised in order to allow for the specification of concrete ABAC policies. Such a customisation amounts to an extension and/or instantiation of the abstract classes and properties presented in Section 3. It is the responsibility of such a mechanism to ensure that this extension/instantiation takes place according to a set of predefined governance policies. In the future, we intend to investigate the construction of a higher-level ontological framework that will generically accommodate these governance policies and thus pave the way for the construction of a generic customisation mechanism that can be easily adapted to the particular needs of the potential adopter of our framework.

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. The authors would like to thank the partners of the PaaSword project (www.paasword.eu) for their valuable advices and comments.

References

1. Cisco, 2011. Cloud: What an Enterprise Must Know, Cisco White Paper.
2. Vaquero, L.M., Rodero-Merino, L., Caceres, J. and Lindner, M., 2008. A break in the clouds: Towards a cloud definition. SIGCOMM Comput. Commun. Rev., vol 39, no 1, pp. 50 — 55.
3. Micro, T., 2010. The Need for Cloud Computing Security. Trend Micro.

4. NIST, 2011. Cloud Computing Reference Architecture, National Institute of Standards and Technology.
5. CSA, 2013. The Notorious Nine. Cloud Computing Top Threats in 2013. Cloud Security Alliance.
6. Verginadis, Y., Michalas, A., Gouvas, P., Schiefer, G., Hübsch, G., Paraskakis, I., 2015a. PaaSword: A Holistic Data Privacy and Security by Design Framework for Cloud Services. Proceedings of the 5th International Conference on Cloud Computing and Services Science (CLOSER 2015), May 20-22, Lisbon, Portugal.
7. Ferrari, E., 2010. Access Control in Data Management Systems. Synthesis Lectures on Data Management, Morgan & Claypool, Vol. 2, No. 1, p. 1-117.
8. Dey, A. K., 2001. Understanding and Using Context. In Personal and Ubiquitous Computing Journal, vol. 5, no. 1, p. 4-7.
9. 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.
10. Linked USDL, 2014. Available online: http://linked-usdl.org/.
11. Pedrinaci, C., Cardoso, J. and Leidig, T., 2014. Linked USDL: a Vocabulary for Web-scale Service Trading. In 11th Extended Semantic Web Conference (ESWC).
12. 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.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.
20. Jung, C., Eitel, A., Schwarz, R., 2014. Cloud Security with Context-aware Usage Control Policies. In Proceedings of the INFORMATIK'14 Conference, pp. 211-222.

21. Verginadis, Y., Mentzas, G., Veloudis, S., Paraskakis, I., 2015b. 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.
22. Specification of Deliberation RuleML 1.01, 2015. Available online: http://wiki.ruleml.org/index.php/Specification_of_Deliberation_RuleML_1.01.
23. 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
24. WS-Trust 1.3, 2007. Available online: http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.doc
25. 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.
26. Kagal, L., Finin, T. and Joshi, A., 2003. A Policy Language for a Pervasive Computing Environment. In 4th IEEE Int. Workshop on Policies for Distributed Systems and Networks (POLICY '03).
27. Hu, H., Ahn, G.-J. and Kulkarni, K., 2011. Ontology-based policy anomaly management for autonomic computing. In 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)
28. OWL Web Ontology Language Reference. W3C Recommendation, 2004. Available online: http://www.w3.org/TR/owl-ref/.