Context-aware Policy Enforcement for PaaS-enabled Access Control

Yiannis Verginadis, Ioannis Patiniotakis, Panagiotis Gouvas, Spyros Mantzouratos, Simeon Veloudis, Sebastian Thomas Schork, Ludwig Seitz, Iraklis Paraskakis, and Gregoris Mentzas

Abstract— It is generally conceded that, due to security and privacy concerns, enterprises and users are reluctant to embrace the cloud computing paradigm and hence benefit from the cost reductions and the increased flexibility or business agility that this paradigm brings about. These concerns stem mainly from the significantly-expanded attack surfaces that result from the heterogeneous nature of cloud services and the dynamicity inherent in cloud environments. In order to alleviate these concerns, effective and flexible access control approaches are required to consider the contextual parameters that characterise data access requests in the cloud. In this respect, this work presents PaaSword: a novel holistic access control framework—essentially a PaaS offering—that extends the popular XACML standard with semantic reasoning capabilities that support the federation of effective context-aware access control policies and their infusion into cloud applications with minimal manual intervention and effort. To determine the performance of our solution, a comparative evaluation test is presented and discussed, against a well-known reference implementation of the XACML standard, namely the open source WSO2 Balana engine.

Index Terms— access control, cloud computing, context-aware policy enforcement

1 Introduction

By enabling ubiquitous access to shared pools of distributed and configurable resources, cloud computing represents a significant shift from the traditional client/server paradigm towards service-based architectures that offer theoretically boundless scalability and a flexible pay-per-use model [1]. Evidently, such a shift brings about significant advantages for users and enterprises in terms of cost, flexibility and business agility. In particular, it greatly facilitates small and medium enterprises (SMEs) in dealing efficiently and effectively with the data storage and processing challenges that they may be facing.

Nevertheless, despite the compelling benefits, the majority of enterprises are still to cross Moore's chasm [2] with respect to cloud adoption. More specifically, less than 20% of enterprise applications run in the cloud, with 29% of the enterprises reporting security concerns such as data breaches, insufficient access management, insecure APIs and account hijacking [3] as significant averting factors for migrating their data and operations to the cloud [4]. Furthermore, these concerns are fuelled by a series of recent security attacks that gained increased publicity, such as the iCloud hack that leaked hundreds of personal photos of celebrities [5], or the ADP breach that exposed payroll and tax data of nearly 640,000 companies in the US [6]. Moreover, governmental legislations regarding data privacy and data location, such as the EU's General Data Protection Regulation [7], present an additional source of concern for enterprises which are now faced with severe legal and financial consequences if data confidentiality is breached, or if cloud providers move regulated data across national borders [8].

Clearly, these concerns must be alleviated if enterprises and users are to embrace the cloud paradigm and benefit from the manifold advantages that it brings about. This can only be achieved if appropriate *security policies* are infused into cloud applications in order to restrict access to sensitive data [9]. Nevertheless, in an inherently dynamic cloud world where data persist over distributed and ubiquitously accessible computing resources, these policies, if they are to be effective, must be able to take into account the varying *contextual circumstances* surrounding data access requests and affecting their permissibility. This calls for context-aware access control policies, i.e. policies capable of tying the permission, or denial, of an access request to a plethora of heterogeneous attributes that synthesise the situation, or context, of one or more entities deemed relevant to the request—e.g. the subject or object of the request, or even the request itself [10], [11].

The burden of defining and implementing such sophisticated access control policies typically falls on application developers, raising concerns about the degree to which these policies sufficiently and adequately address the entire gamut of contextual attributes that need to be considered for protecting the sensitive data. In this regard, we argue that a promising approach to alleviating the security concerns associated with the adoption of cloud computing is to offload part of this burden from the application developers. To this end, this work proposes PaaSword: an innovative security-by-design framework, essentially a PaaS offering, for facilitating developers in infusing appropriate context-aware access control policies into cloud applications. More specifically, PaaSword constitutes a holistic framework based on the popular

- *Y. Verginadis, I. Patiniotakis and G. Mentzas are with the Institute of Communications and Computer Systems, National Technical University of Athens, Athens, Greece. E-mails: {jverg, ipatini, gmentzas}@mail.ntua.gr.*
- *P. Gouvas and S. Mantzouratos are with the Ubitech Ltd., Athens, Greece. E-mails: {pgouvas, smantzouratos}@ubitech.eu.*
- *S. Veloudis and I. Paraskakis are with the South East European Research Centre, The University of Sheffield, International Faculty CITY College, Thessaloniki, Greece. E-mails: {sveloudis, iparaskakis}@seerc.org.*
- *S. T. Schork is with CAS Software AG, Karlsruhe, Germany. Email: sebastian.schork@cas.de.*
- *L. Seitz is with the RISE SICS, Stockholm, Sweden. Email: luwig.seitz@ri.se*

XACML standard [12] that provides: i) the necessary toolset for facilitating developers in defining context-aware access control policies that are specifically tailored to the particular needs of their cloud applications; these policies take the form of code-level Data Access Object (DAO) annotations; and ii) a middleware for interpreting, monitoring and enforcing these annotations dynamically, during application runtime, as well as for governing them.

One of the main strengths of the PaaSword framework is that it is underpinned by a generic representation of policies, one that uses an ontological template for capturing the various concepts, and their interrelations, that are involved in the definition of a policy; this template is malleable in the sense that it can be suitably extended with concepts and relations that reflect the particular needs of the underlying domain of application. In this respect, PaaSword promotes a clear separation of concerns by unravelling the representation of policies from the actual code employed for enforcing them, whilst accurately capturing the knowledge that lurks behind policies. This brings about the following seminal advantage. It enables—by virtue of semantic inferencing—the generation of new knowledge on the basis of the knowledge already encoded in the policies. Therefore, it can successfully tackle situations in which the contextual information piggybacked on an access request does not necessarily match, at the syntactic level, the corresponding information encoded in the policies. For example, if a policy states that a sensitive data object (say o) is only readable by requests that originate from within the EU, then a request that originates from a location in, say, Belgium will be permitted to read o, as semantic inferencing allows the generation of the (new) knowledge that the request indeed originates from within the EU. In this regard, the PaaSword framework allows for the definition of interoperable policies i.e. polices enforceable across the diverse administrative domains that a cloud environment may span. Moreover, through the incorporation of a suitable production system, PaaSword is able to monitor and enforce the corresponding policies, hence perform the aforementioned semantic inferencing, with an acceptable performance penalty.

In addition, the generic policy representation underpinning the PaaSword framework offers one more important advantage. It lends itself to a series of automated checks regarding: (i) the well-formedness of policies, i.e. whether policies include all the (contextual) information required for granting, or denying, access to sensitive data; (ii) the disclosure of any inter-policy relations such as subsumption and contradiction. In this respect, this representation paves the way for the construction of a generic mechanism for policy governance, one that enables the organisations adopting the PaaSword framework to create and manage their security policies according to predefined rules and regulations. This is of utmost importance for it increases our assurance on the effectiveness of the policies. The implementation details of such a governance mechanism are, however, beyond the scope of this paper.

The rest of this paper is structured as follows. Section 2 presents an overview of the PaaSword framework and outlines the generic policy representation that underpins it. Section 3 presents the policy enforcement middleware that implements PaaSword's context-aware access control scheme. Section 4 presents the annotation interpretation mechanism and Section 5 discusses the main aspects of the policy enforcement business logic. Section 6 provides a comparative evaluation of the PaaSword framework against a widely-adopted implementation of the XACML standard. Finally, Section 7 briefly discusses related work and Section 8 concludes the paper and outlines future work.

2 The PaaSword Framework

This section presents an overview of the PaaSword framework (Section 2.1) and outlines the ontological template upon which it is founded (Section 2.2). In particular, with respect to the ontological template, an outline of the Context-aware Security Model that articulates the various concepts underpinning the template is provided.

2.1 A Security-by-Design Framework

The PaaSword framework addresses the semi-honest adversarial model discussed in [13] whereby a malicious cloud provider is assumed to correctly follow the specification of an underlying protocol whilst, at the same time, is able to intercept messages in order to disclose sensitive data [14], [15]. The framework offers, as a service, the following security-related features:

i. Transparent key usage for efficient authentication of the subject of an incoming access request.
ii. Annotation capabilities at the level of DAOs in the form of an IDE plugin that guide developers into the process of articulating all those access control policies that are required for protecting their sensitive data in the cloud.
iii. Dynamic interpretation of the DAO annotations into policy enforcement rules.
iv. Governance and quality control of the annotations and the respective policies that they implement.
v. Formulation and implementation of the overall policy enforcement business logic.

This work focuses on aspects ii., iii. and v. above which are inextricably linked to the implementation of PaaSword's access control scheme. As discussed in Section 1, this scheme must enable the expression of *context-aware access control policies* and it is hence based on the Attribute-based Access Control (ABAC) model—a model capable of taking into account all those attributes that synthesise the context of one or more entities that are deemed relevant to an access request [16]. Nevertheless, this model presents a crucial limitation: it lacks the means of addressing any form of *interrelation* between the considered attributes (e.g. whether the possession of one attribute by an entity also implies the possession of another). It therefore precludes any kind of *semantic inferencing* during the evaluation of an access request on the basis of the information encoded in these attributes. This means that the contextual information piggybacked on an access request must necessarily match, at the syntactic level, the corresponding information encoded in the policies. Naturally, this creates the burden of having to define fine-grained access control policies that cover the potentially different contexts that may be attached to the entities that are related to a request. For example, a policy stating that a sensitive data object (say o) is only readable

by requests that originate from within the EU is insufficient, by itself, for deciding whether a request originating from a location in, say, Belgium should be permitted or denied: a second, finer-grained policy that states that requests originating from a location in Belgium are permitted to read `o`, is required. Moreover, the interpretation of contextual attributes at a purely syntactic level impedes the federation of access control across different actors who are likely to use different attribute vocabularies; interoperability thus depends on ad-hoc mechanisms for translating attributes across different actor domains, a process that is resource- and time-consuming, as well as error-prone, in practice.

In contrast, the access control scheme offered by the PaaSword framework extends the ABAC model with the ability of fusing *semantic knowledge* into the process of deciding whether to permit, or deny, a request. This allows the generation of new knowledge through *semantic inferencing* on the information residing in the contextual attributes that pertain to a request—e.g. in the example above, the knowledge that the request indeed originates from within the EU. This absolves application developers from the burden of defining fine-grained access control policies such as the one in the example above.

2.2 Extending ABAC with a Context-aware Policy Model

The PaaSword framework extends the ABAC model through the introduction of a generic *Policy Model*, one that adheres to the popular XACML standard [12] and views ABAC policies as finite, non-empty, sets of ABAC *rules*. A rule is the most elementary structural element and the basic building block of policies. It is abstractly described in terms of the *ontological template* depicted in Table 1. This template comprises the following concepts: i) `actor` identifies the subject requesting access to perform an operation on a sensitive object; note that an instantiated ABAC rule can involve the `any_actor` instance for describing rules that do not target any specific actor; ii) `context expression` identifies the environmental conditions that must hold in order to permit, or deny, the performance of an operation on a sensitive object; iii) `authorisation` determines the type of authorisation (positive i.e. 'permit', or negative i.e. 'deny') that is granted; iv) `action` identifies the operation that may, or may not, be performed on the protected sensitive object; and v) `controlled object` identifies the sensitive object on which access is requested.

The ontological template of Table 1 is underpinned by an extensible *Context-aware Security Model* (hereafter referred to as *Context Model* for simplicity) that captures—in terms of ontological classes and properties—the various concepts, and their interrelations, involved in the definition of a policy. These concepts may involve any kind of contextual information that is machine-parsable [11] and pertains to an access request; for example, they may include the network and physical location of the subject that issues an access request, the type of device that is used for issuing the request, as well as the position (or role) that this subject occupies in a company.

TABLE 1
ABAC RULE TEMPLATE

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

The Context Model (CM) incorporates various facets including: i) the *Subject* class that incorporates concepts for describing the entity requesting access (e.g. whether it is a person, a software agent, an organisation, a group, etc.), as well as any other entity that is deemed relevant to an access request. Note that the concept `actor` of the ontological template of Table 1 draws its instances from this class; ii) the *Object* class that incorporates concepts for describing the (sensitive) object on which access is requested (e.g. whether it is a relational or non-relational data object, a file, etc.). Note that the concept `controlled object` of the ontological template of Table 1 draws its instances from this class; iii) the *Security Context Element* class that incorporates concepts for describing the context of an entity relevant to a data access request such as location, date and time, type of connectivity, etc. Note that the concept `context expression` of the ontological template of Table 1 interrelates instances from this class with instances from the classes Subject and Object above; and iv) the *Permission* class that incorporates concepts for describing the semantics of the type of access sought (e.g. read-only access, read/write access, etc.) on the sensitive data. Note that the concept `action` of the ontological template of Table 1 draws its instances from this class. A more elaborate account of the aspects of the CM can be found in [17], [18].

The use of such an ontological template for describing ABAC rules, hence ABAC policies, essentially disentangles the definition of a policy from the actual code employed for enforcing it. This brings about the following seminal advantages: (i) it allows the performance of semantic inferencing, hence the generation of new knowledge, on the basis of the information already encoded in existing policies; (ii) it achieves a clear decoupling between the policy decision and policy enforcement points (PDP and PEP respectively), a decoupling essential for generating dynamically, during application runtime, fresh access control policies that capture the new knowledge generated through semantic inferencing; (iii) it forms an adequate basis for reasoning generically about the well-formedness of the security policies, i.e. whether they include all the information required for granting, or denying, access to sensitive data, as well as about inter-policy relations such as subsumption and contradiction; (iv) it facilitates the overall governance of policies. For more information on the Policy Model, the interested reader is referred to [9], [17], [18].

3 POLICY ENFORCEMENT MIDDLEWARE

This section presents an overview of the policy enforcement middleware that implements PaaSword's context-aware access control scheme. Fig. 1 depicts the main components of this middleware which are detailed below.

The *CM Editor* enables the relevant stakeholders (e.g.

DevOps, product manager, cloud application developer) to customise the CM such that it adequately supports the expression of policies in the underlying domain of application. The role/position of the stakeholder depends, each time, on the hierarchical structure advocated by the organisation adopting the PaaSword solution, as well as on the particular rights and obligations that have been assigned to the various roles/positions that comprise this hierarchy. This entails the population of the model's classes with appropriate instances, as well as the potential addition of new classes and/or properties—these are intended to capture any aspects of the underlying domain of application that must be taken into account by the policies and which have not been included in the generic version of the CM offered by the PaaSword framework.

The *Access Policy Editor* enables the relevant stakeholders to create new, or modify existing, context-aware access control rules and hence policies. The newly-created or modified rules abide by the ontological template of Table 1 and are founded upon an instantiated version of the CM, one that has been created through the use of the CM Editor.

The *Annotations Governance Mechanism* enables the governance of the code-level annotations that express access control policies. Such governance entails automated checks regarding: (i) the well-formedness of the newly-created or modified policies, i.e. whether they include all the (contextual) information required for granting, or denying, access to sensitive data; (ii) the disclosure of any inter-policy relations such as the subsumption, or contradiction, of the newly-created or modified policies with other existing policies. In addition, it entails the imposition of potential restrictions on the entities that are allowed to create and/or modify policies (e.g. DevOps, product manager, cloud application developer), as well as on the (contextual) circumstances under which such activities may take place. The details of the implementation of this mechanism are beyond the scope of this paper.

The *PaaSword IDE Plugin* enables the cloud application developer to insert code-level annotations that express access control policies through the use of Eclipse's popular web-integrated environment.

The *Access Policy Enforcement Mechanism* (also referred to as semantic authorisation engine) implements the policy enforcement business logic according to the XACML architecture, as well as the processing model for evaluating access requests. It involves two main components:

- A *Production System* for monitoring and evaluating the access control policies in an efficient manner. This system comprises the following three primary elements: the production memory, which contains a set of access control policy rules; the working memory, which stores data representing facts and assertions about the contextual attributes encoded in access control policy rules; the inference engine, which discerns and executes the policy rule applicable to a particular access request[1]. It is worth noting here that the production system employed by the reference implementation of the PaaSword framework is built around the Drools[2] Business Rule Management System due to its performance capabilities;
- The *ContextModel2ExpertSystemRules* parser which is responsible for infusing inferencing capabilities into the production system. More specifically, it translates the semantic knowledge captured in the CM into rules that are subsequently fed into the production system's memory.

The *Facts Mechanism* feeds the working memory of the Production System with real-world facts, i.e. with the values currently assumed by the contextual attributes encoded in an access control policy. Clearly, these values are essential

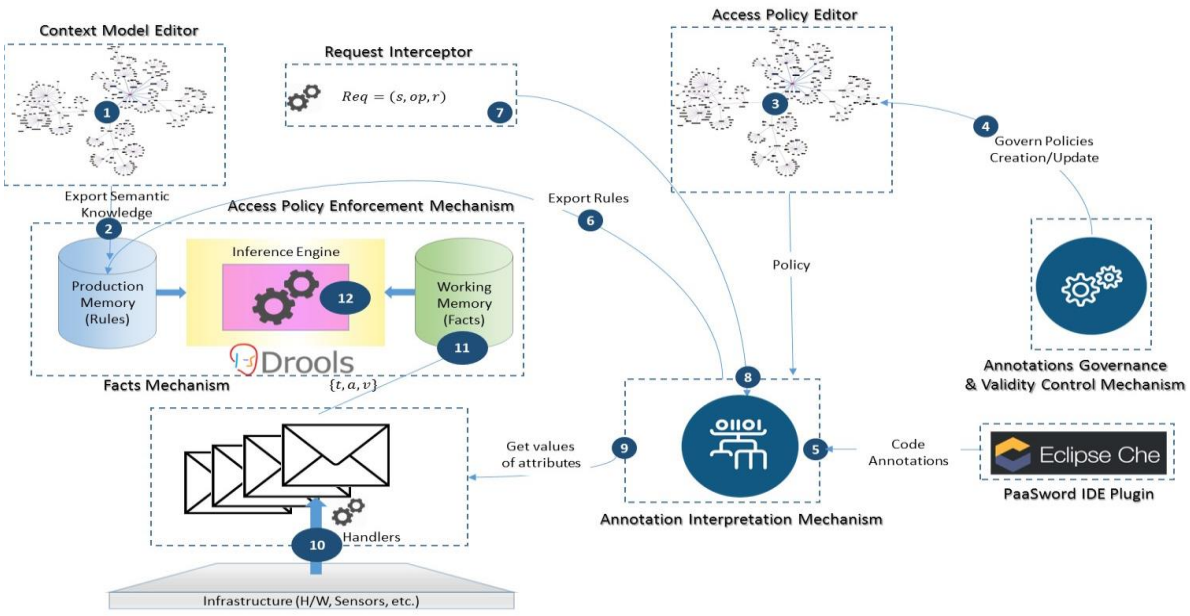


Fig. 1. Policy Design & Enforcement Related Components

[1] In case two or more rules are applicable, conflict resolution is performed on the basis of XACML's combining algorithms [2] in order to discern a single ultimately applicable policy rule.

[2] http://www.drools.org/

for the evaluation of the policy; they are retrieved dynamically, during application runtime, by invoking a number of appropriate handlers (see Section 4.2 for more details).

The *Annotation Interpretation Mechanism* undertakes the task of interpreting the code-level policy annotations into XACML-based enforceable policies. It comprises two main components:

- The *AttributesLookUp* component which is responsible for informing the Facts Mechanism of which particular attribute values it should retrieve;
- The *PolicyModelBootstrapping* component which is responsible for the actual translation of the code-level policy annotations into rules expressed in the jargon of the production system.

Fig. 1 provides a high-level description of the execution flow of the aforementioned middleware components, unveiling their use as well as their interactions. For the remaining of this section we describe the main steps of this flow. These steps are clustered according to the main execution phases of a PaaSword-enabled cloud application, namely pre-bootstrapping, bootstrapping and runtime.

The *Pre-bootstrapping phase* involves the following steps: i) through the use of the Context Model Editor, the CM is updated, instantiated, serialised in an appropriate format and persisted (see Fig.1, step 1); ii) knowledge is extracted from the Context Model and expressed in the form of Production Memory rules (see Fig.1, step 2); iii) a set of appropriate access control policies is created by instantiating the PaaSword Policy Model (see Fig.1, step 3) and subsequently validated and persisted (see Fig.1, step 4) in an appropriate serialised format; iv) the PaaSword annotations that implement these policies are inserted into the cloud application code (see Fig.1, step 5); and v) suitable handlers are created and associated with the appropriate classes of the CM. More specifically, each handler (e.g. geolocation) is associated with the particular class of the CM (e.g. location) that represents the attribute for which the handler will provide measurements.

The *Bootstrapping phase* involves the following steps: i) access policies are fed into the Annotation Interpretation Mechanism and translated into Production System rules (see Fig.1, step 6); and ii) appropriate handlers are then selected for providing the necessary data for evaluating the contextual attributes encoded in the policies.

The *Run-time phase* involves the following steps: i) an incoming access request is intercepted (see Fig.1, step 7) captured as a triple (`s, op, r`) that corresponds to a subject (`s`) that requires to perform an operation (`op`) on a certain protected resource (`r`); ii) the request is fed into the Annotation Interpretation Mechanism where the contextual information encoded in the request is extracted (see Fig.1, step 8); iii) the access control attributes whose values need to be resolved are recognised on the basis of the applicable access control policies[3] and the corresponding handlers are queried (see Fig.1, step 9); iv) the values provided by the handlers are aggregated, semantically uplifted on the basis of the knowledge captured in the CM (see Fig.1, step 10), and ultimately uploaded to the working memory of the production system (see Fig.1, step 11); and v) the relevant access control policies are evaluated and the access control decision is issued (see Fig.1, step 12). Note that a description of the steps ensuing a successful, or not, access request are beyond the scope of this paper.

4 Annotation Interptetation

4.1 Annotations

Two distinct kinds of annotation are discerned: *@PaaSwordPEP* and *@PaaSwordEntity*. *@PaaSwordPEP* annotations are the vehicle through which access control policies are infused into the code of cloud applications. In particular, these annotations decorate the DAO-implementing Java classes and methods of a cloud application with the rules, policies and policy sets defined through the Access Policy Editor (see Section 3). Thus, when an access request is received during application runtime, the PaaSword framework evaluates these rules, policies and policy sets and invokes the corresponding DAO only if a permit decision is resolved. This evaluation is performed against the attribute values that are piggybacked on the access request. The definition of a *@PaaSwordPEP* annotation that associates a DAO with a policy set, policy or rule is provided in Table 2.

@PaaSwordEntity annotations are, on the other hand, used at the class-level in order to distinguish those classes that are handled as PaaSword *entities*, i.e. as containers of

TABLE 2
@PaaSwordPEP Annotation

```
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface PaaSwordPEP {
    String value() default '';
    Type type() default Type.POLICY;
    public enum Type {
        RULE, POLICY, POLICY_SET
    }
}
```

code that feed data to an underlying database during application bootstrapping time. This database may be fragmented and distributed over a number of different physical servers for privacy reasons. This gives rise to yet a third kind of annotations, namely the *@PaaSwordDDE* annotations which are responsible for specifying the manner in which sensitive data are fragmented and persisted over different physical servers. *@PaaSwordDDE* annotations shall not further concern us here.

4.2 Security Model Editors

1) CM Editor

As mentioned in Section 3, the CM Editor enables relevant stakeholders (e.g. DevOps, product manager, cloud application developer) to suitably instantiate and customise the CM in order to support the expression of access control policies in the underlying domain of an application. Recall

[3] An access control policy is deemed applicable if it is designated to protect the controlled object targeted by the access request.

from Section 2.2 that the CM constitutes the semantic background against which access control policies are defined. It provides the underlying ontological infrastructure, i.e. the concepts, properties and instances in terms of which the *attributes* of a policy are specified. For example, consider an ABAC policy that protects a particular data object by enumerating the locations from which this object is accessible. Such a policy hinges upon a *location* attribute and thus requires the introduction in the CM of a corresponding `Location` concept along with its relevant properties and instances.

In addition, the CM Editor allows the association of CM concepts, hence of policy attributes, with appropriate *handlers*. A handler is essentially a software routine that provides real-time measurements of the current value of an attribute. For instance, in the case of the `Location` attribute, one or more handlers must be specified for providing the whereabouts of a subject that issues an access request to a particular data object.

*2) Access Policy Edi*tor

Once the semantic background is reified, the Access Policy Editor assists relevant stakeholders in creating and/or modifying access control policies in a controlled and rule-based manner. In particular, the Access Policy Editor takes into account a set of *constraints* that define the admissible structure of a policy. These constraints form essentially a set of meta-policies that articulate the 'ingredients' of an access control policy, i.e. all those attributes along with their allowable values or value ranges, that a policy must, may or must not articulate[4]. Based on these constraints, the Access Policy Editor exposes a suitable GUI that guides a user through the process of creating or updating a policy by providing, at each juncture of this process, the allowable values that may be selected. In this respect, the Access Policy Editor advocates a *type-safe* approach to policy creation and modification (depicted in Fig. 2 as a typeSafety interface). Table 3 summarises the main functionalities of

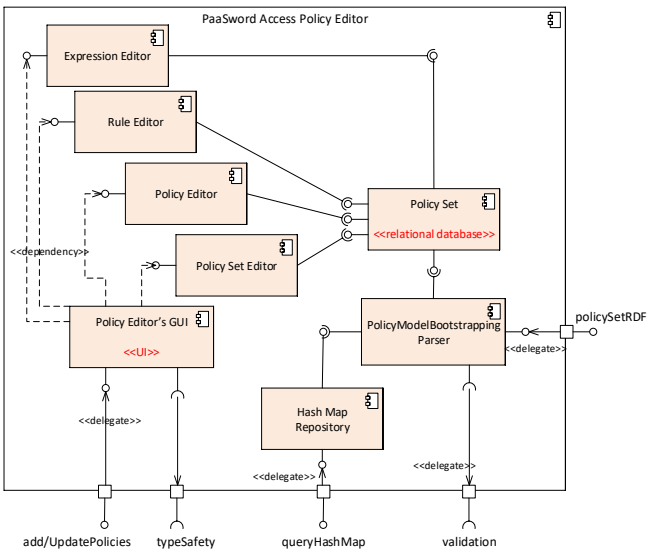


Fig. 2. Access Policy Editor

TABLE 3
Access Policy Editor Operations

Functionality	Responsible subcomponent
Facilitates a user in creating a new, or modifying an existing, *context expression* (see Section 2.2).	*Expression Editor*
Facilitates a user in creating a new, or modifying an existing, *policy rule* by providing its identifier, controlled object, action, actor, context expression and, of course, decision.	*Rule Editor*
Facilitates a user in creating a new, or modifying an existing, *policy* by providing its identifier, as well as its pertinent rules and combining algorithm.	*Policy Editor*
Facilitates a user in creating a new, or modifying an existing, *policy set* by providing its identifier, as well as its pertinent policies and combining algorithm. The resulting policy set is serialized in JSON and persisted in the *Policy Set* relational database subcomponent.	*Policy Set Editor*

the Access Policy Editor, along with the particular subcomponents that are responsible for delivering these functionalities. These subcomponents are depicted in the UML component diagram of Fig. 2.

In addition, the Access Policy Editor comprises the *PolicyModelBootstrapping Parser*, a subcomponent responsible for converting the newly-created or updated policies from the JSON serialisation in which they are persisted in the *Policy Sets* database (see Table 3), to equivalent RDF/TTL and XACML serialisations; these are then exported to the PaaSword Models triple store[5] and to the Policy Administration Point (PAP) [12] respectively. The former serialisation is suitable for validating the policies against the constraints regarding the admissible structure of a policy, whereas the latter is suitable for deploying and managing the policies during application runtime. Table 4 presents an example of a policy expressed in RDF/TTL whereby a

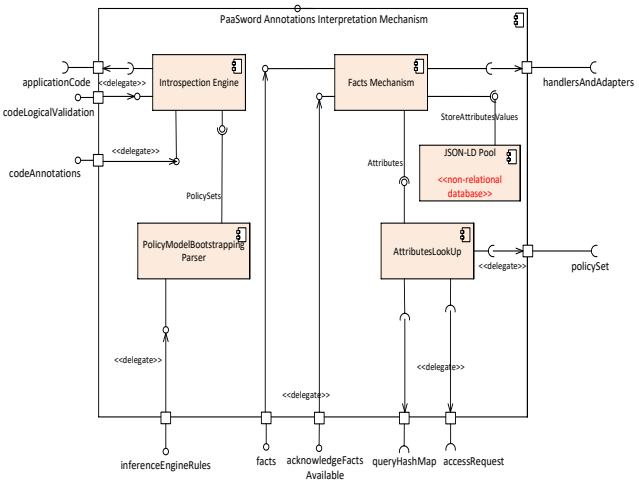


Fig. 3. Annotation Interpretation Mechanism

[4] These constraints are represented ontologically. The manner in which these constraints are formulated and represented shall not concern us here.

[5] The Apache Jena Fuseki (https://jena.apache.org/documentation/fuseki2/) triple store is used in particular.

TABLE 4
SAMPLE POLICY RULE IN RDF/TTL

```
# Definition of Policy Rule: Rule_1
ex:Rule_1              a       pac:ABACRule ;
  dcterms:identifier   '3'^^xsd:string ;
  rdfs:label    'Rule_1'^^xsd:string ;
  pac:hasControlledObject
   'eu.paasword.examples.CarPark.LogEntry';
  pac:hasAuthorisation        pac:permit ;
  pac:hasAction          ex:Write ;
  pac:hasActor  ex:Guard ;
  pac:hasContextExpression
   ex:IN_WorkHoursAndDays .
# Definition of Context Expression: IN_Work-
ing_Hours_and_Days
ex:IN_Working_Hours_and_Days  a
  pac:ANDContextExpression ;
  pac:hasParameter     ex:IN_Working_Hours ;
  pac:hasParameter      ex:IN_Working_Days ;
  dcterms:identifier   '4'^^xsd:string ;
  rdfs:label
   'IN_Working_Hours_and_Days'^^xsd:string .
```

subject acting in the capacity of the role 'Guard' can gain 'write' access to the resource `CarPark.LogEntry` only during working days and hours (this condition is defined in terms of the context expression `ex:IN_Work-HoursAndDays`).

The *PolicyModelBootstrapping Parser* is also burdened with the task of discerning the set of attributes that a newly-created or updated policy comprises and storing them in a hash map along with the controlled object that these attributes are designated to protect. More specifically, each pair in the hash map comprises the URI of a controlled object that is protected by a policy, as well as a list of ontological classes from the CM that represent the attributes of the policy (stored in the Hash Map Repository as depicted in Fig.2). These are effectively the attributes that need to be evaluated each time an access request that targets the particular controlled object is received[6]. An example hash map is provided in Table 5; it associates a specific data object—the controlled object `CarPark.LogEntry`—with a number of classes from the CM, namely the classes `Subject`, `DateTimeInterval`, `Physical Location` that need to be evaluated each time an access request that targets `CarPark.LogEntry` is received. This evaluation need is acquired by querying the Hash Map Repository usined the the *queryHashMap* interface as depicted in Fig.2.

TABLE 5
EXAMPLE HASH MAP

```
eu.paasword.examples.CarPark.LogEntry \
   http\://www.paasword-project.eu/ontologies/
   casm/2016/05/20#DateTimeInterval\
   http\://www.paasword-project.eu/ontologies/
   casm/2016/05/20#Subject\
   http\://www.paasword-project.eu/ontologies/
   casm/2016/05/20#PhysicalLocation
```

[6] Clearly, upon receipt of an access request, the attributes that need to be evaluated in order to decide whether to permit, or deny, the request must be quickly determined; this justifies our choice of storing these attributes in a hash map indexed by the identifiers of the controlled objects.

4.3 Annotation Interpretation Mechanism

The ultimate goal of the Annotation Interpretation mechanism is to transform *@PaaSwordPEP* annotations into XACML-based enforceable access control policies. More specifically, this mechanism offers the following functionalities. i) It introspects the source code of a PaaSword-enabled application and determines whether it contains valid *@PaaSwordPEP* annotations; this functionality is offered by the *Introspection Engine* subcomponent depicted in Fig. 3. ii) It interprets *@PaaSwordPEP* annotations into appropriate inference engine rules and persists them in the Production Memory of the Production System; this functionality is offered by the *PolicyModelBootstrapping Parser* component[7] (see Fig. 3). iii) It parses the inference engine rules in the Production System and discerns all those contextual attributes which must be evaluated on the receipt of an access request in order to decide upon its permissibility; this functionality is offered by the *AttributesLookUp* subcomponent (see Fig. 3). iv) It feeds the Working Memory of the Production System with facts, i.e. with the current values of all those contextual attributes discerned by the *AttributesLookUp* subcomponent; this functionality is offered by the Facts Mechanism subcomponent (see Fig. 3).

We next provide brief accounts of the four subcomponents that offer these functionalities.

1) Introspection Engine

Performs a series of correctness checks that aim at determining the *logical validity* of *@PaaSwordPEP* annotations. By 'logical validity' we refer here to certain characteristics of an annotation such as, for example, the uniqueness of the application name appearing in an annotation, or whether an annotation decorates a REST or web end point method. Logical validation takes place through introspection at the Bytecode level. Note here that the structural validity of an annotation is not checked for if an application compiles correctly, structural validity is assured from the outset.

2) PolicyModelBootstrapping Parser

As depicted in Fig.4, upon the creation or update of an access control policy, this parser undertakes the task of translating the relevant rules in RDF triples for validation purposes. Once validated, it parses the arguments of each *@PaaSwordPEP* annotation in order to guarantee that this policy is used on a certain application and translates it into rules expressed in the Drools jargon. In parallel, it exports them using XACML notation and creates hash maps for fast retrieval of the necessary context attribute values, once a request is intercepted at run-time. More specifically, this parser retrieves from the PaaSword Model triple store the RDF/TTL-serialisation of each rule, policy or policy set that is referenced from within a *@PaaSwordPEP* annotation (e.g. PaaSwordPEP (Type.Policy, "policy1")) and translates it into one or more inference-engine production rules.

3) AttributesLookUp

Identifies all those attributes that need to be evaluated in order to determine whether an access request to a particular controlled object can be granted. More specifically,

[7] Note that this is the same component as the one encountered in the Access Policy Editor mechanism.

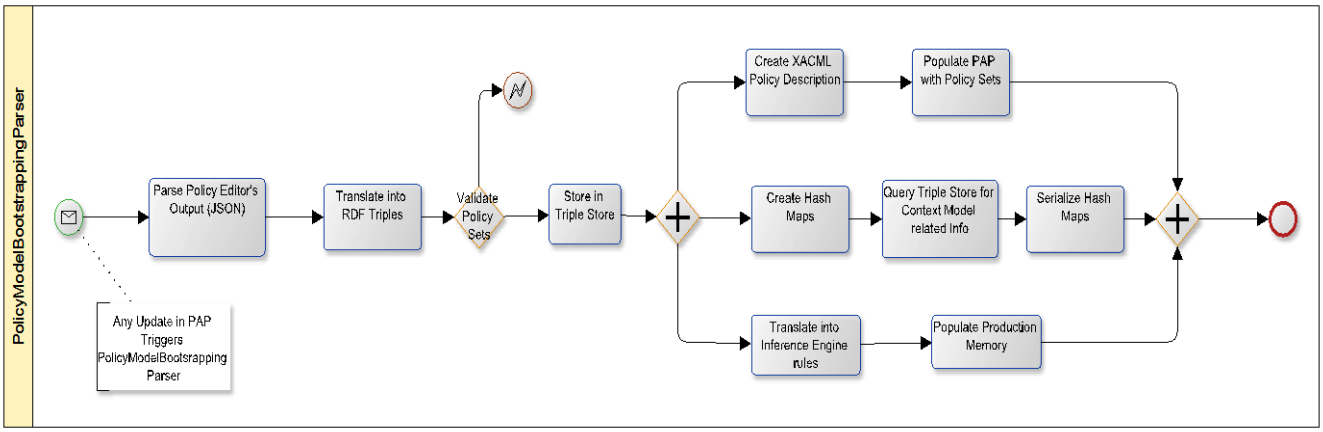


Fig. 4. PolicyModelBootstrapping Parser Workflow

it intercepts all incoming access requests and determines, for each one of them, the id of the controlled object that it targets. It then uses this id as a search term in order to look up into the Hash Map repository all those ontological classes from the CM that represent attributes designated to protect that controlled object. Finally, it invokes the Facts Mechanism and passes to it these URIs as arguments.

4) Facts Mechanism

Retrieves the current values of all attributes whose URIs have been received from the AttributesLookUp component. These values are derived from a variety of sources including, for example, sensors, cell phones, relevant databases and even the request itself. In order to effectively handle input from such diverse sources, the Facts Mechanism employs a set of *handlers*—i.e. software routines that federate the raw data produced by these sources and extract the required attribute values. Note that each CM class that represents an attribute is associated with at least one handler. The Facts Mechanism also employs a set of *adapters* that semantically uplift the attribute values emitted by the handlers by constructing appropriate instances in the corresponding CM classes. For example, a handler might emit the current location of an entity involved in an access control rule in the form of latitude and longitude while an adapter might perform reverse geocoding for inferring the relevant city or country. This information takes the form of an instance of the CM class `Location`. Finally, the Facts Mechanism populates the working memory of the Production System and triggers the PaaSword Policy Enforcement Mechanism which is responsible for evaluating access requests (see Section 5).

5 Policy Enforcement

5.1 Policy Enforcement Business Logic & Implementation

The PaaSword Policy Enforcement mechanism is responsible for evaluating access requests against existing policy sets and deciding whether to permit, or deny, these requests. The basic component that materialises the policy enforcement business logic is the *Semantic Authorization Engine* (see Fig. 5)—a dedicated Production System that comprises, at its very core, an Inference Engine that matches real-world facts against production rules in order to decide whether to permit, or deny, access requests. As described in Section 4.3, the facts are produced by the Facts Mechanism and the various handlers that it employs, whereas the production rules are generated by the PolicyModelBootstrapping Parser which translates RDF/TTL-expressed rules, policies and policy sets from the PaaSword Models triple store into production rules.

As depicted in Table 6, a production rule is a two-part structure comprising an antecedent that articulates all those conditions that must be satisfied in order for the rule to be applicable, and a consequent that specifies the actions that are to be performed if these conditions are indeed satisfied—in our case these actions amount to the permission or denial of an access request. The rules are stored in the Production Memory component (see Fig. 5) whereas the facts reside in the Working Memory.

Two are the main methods of rule execution in a production system: *backward chaining* and *forward chaining* [19].

TABLE 6
Production Rule

When <*conditions*> ***then*** <*actions*>

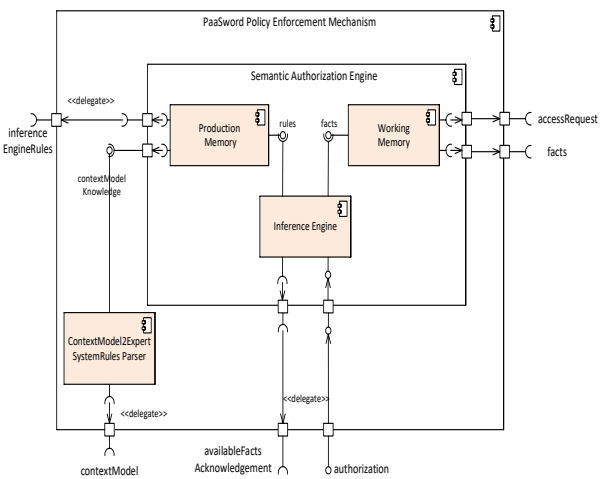


Fig. 5. Policy Enforcement Mechanism

The former implements a form of reverse engineering: it starts off with a particular conclusion (e.g. 'permit access') and reasons about the legitimacy of this conclusion on the basis of a ruleset that resides in the Production Memory. Such reasoning is useful when the goal is to identify, for example, all those conditions that must be fulfilled in order for a particular controlled object to be accessible.

The latter implements a form of deductive reasoning: it starts off with a particular scenario, e.g. with certain facts about the real world, and works its way through a given ruleset, attempting to generate inferences on the basis of the applicable rules; these inferences ultimately provide a conclusion. It is therefore better-suited to our work: each time an access request is received, a decision (either permit or deny) needs to be deduced on the basis of a given ruleset and certain facts about the real world. In the current PaaSword implementation, the Drools engine is used as a forward chaining engine.

5.2 Interaction with the Semantic Authorization Engine

The component diagram in Fig. 5 and the workflow diagram in Fig. 6, depict all the communication endpoints that the Semantic Authorization Engine uses in order to proceed with the rules triggering. The Semantic Authorization Engine makes use of the Annotations Interpretation Mechanism outlined in Section 4.3, in order to feed its production memory with the appropriate rules. Specifically, during engine bootstrapping, it parses all the policy rules and adds them to the corresponding knowledge base. Once a request is intercepted from a PaaSword-enabled application, the Semantic Authorization Engine makes use of the Facts Mechanism, in order to create extra facts that enrich further the working memory, thus allowing the proper evaluation of all the related policy sets to an incoming access request. For example, given the IP-address of a request, the Annotations Interpretation Mechanism returns the geographical area that the request comes from. When an access request to a protected resource is processed by the PaaSword framework, it is necessary to collect all relevant information available, in order to evaluate the related policy rules using the most complete and up-to-date information. This information may originate from various sources, such as the access request itself, data acquired from sensors and external sources or knowledge extracted from primitive information. All of them are handled by the Facts Mechanism and they are used to prime the Working Memory of the Inference Engine, which performs the actual evaluation of policy rules. For efficiency reasons, it is important to keep the minimum amount of data in the Working Memory, during the rules evaluation process. Our approach involves the identification, upon an access request, of which rules should be eventually used, based on the resource that is to be accessed, and the deduction of all the relevant attributes in order to populate the Working Memory only with the necessary facts.

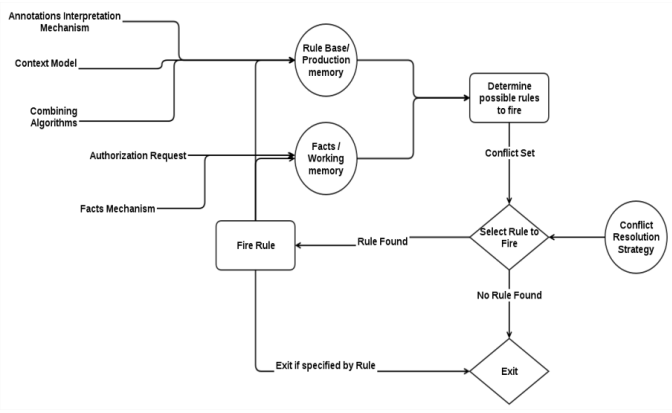


Fig. 6. Semantic Authorization Engine Forward Chaining Workflow

5.3 Interaction with the Context Model

An important task for successfully evaluating an incoming access request, without having to design fine-grained access control rules, is the population of the Production Memory with knowledge extracted from the Context Model. In the PaaSword framework, the use of semantics allows for the implementation of coarse-grained access control rules, alleviating the design-time effort with respect to coping with the burden of fine-tuning the attributes used in the rules and the attribute values acquired from the access request or from external sources. For example, the PaaSword adopter may design rules that restrict access to sensitive data based on the requestor's physical location defined at a country-level, while the available sources provide location information only at a city-level. PaaSword's innovative solution introduces the handling of the necessary semantics in an efficient way. Specifically, PaaSword CM supports basic RDF reasoning which can be mapped to a set of rules through the dedicated *ContextModel2ExpertSystemRules Parser*. This enables semantic reasoning within the Semantic Authorization Engine. The main goal concerns the efficiency in access control by keeping the valuable knowledge expressed semantically without having to perform SPARQL[8] queries at runtime. Using the *ContextModel2ExpertSystemRules Parser*, we parse the knowledge of CM offline and generate rules that support, but not limited to: Property Transitivity; Subproperty Transitivity; Supertype Inheritance; Class Transitivity and Member Induction.

For example (see Fig. 7) based on the CM the "Tablet", "Smartphone" and "Notebook" are subclasses of the class "Mobile" which is a subclass of the "Device Type". We consider for this example the: a) Supertype inheritance and b) Class Transitivity as it is presented in Table 7. During the Production Memory bootstrapping, the rules shown in Table 8 will be automatically added. All of them come directly from ContextModel2ExpertSystemRules Parser after parsing the CM itself. Based on this example, the platform intercepts an access request originating from the "SamsungN7000" (i.e. initial fact) which is a smartphone (instance of class `Smartphone`). After firing all the relevant rules that carry the knowledge of the CM into the expert system, two additional facts are inferred. First that the "SamsungN7000" is also a `Mobile` and second the "SamsungN7000" is an instance of the class `DevType`. The

[8] https://www.w3.org/TR/rdf-sparql-query/

added value of such an approach is that the administrator or the cloud application developer is not obliged to register a fine-grained access control rule at the level of sub-classes `Smartphone, Notebook` or `Tablet` depending on what is the expected value that will be acquired by the intercepted access request. We also note that defining coarse-grained access control rules becomes a necessity in cases that the intercepted contextual information is not known at design-time.

In a similar way, ContextModel2ExpertSystemRules Parser handles the Combining Algorithms. The Combining Algorithms per policy or policy set are also transformed into rules during knowledge base initialization. According to the XACML standard [12] there is a set of combining algorithms (e.g. Deny-overrides, Permit-overrides, First-applicable, etc.) that we support through this framework. For example the Deny-overrides combining algorithm treats the decisions in such a way that if any rule outcome is "Deny", then that decision prevails, although there might be "Permit" results as well.

6 Evaluation

We have set up a comparative evaluation test in order to determine the performance of our solution, compared to a reference implementation of the XACML standard, namely the open source WSO2 Balana[9] engine. For our initial evaluation test, we have created a single Policy Set comprising a single Policy with an increasing number of Rules and we have focused on measuring the policy evaluation time (i.e. execution time) and the RAM consumption. We have gradually increased the number of Rules from 1 to 50000 in a single Policy. These rules were kept as simple as possible, essentially concluding at a single match clause that refered to the inferred current location of the subject of a test request. For the PaaSword experimental set up, this corresponded to the use of a dedicated context handler (concerning location). Furthermore, in order to approximate a realistic use of XACML, we have conducted the equivalent tests using the Balana engine, in which the PDP performs external retrieval of attributes from an SQL database, dynamically during evaluation. The typical case for such external retrieval would be a request carrying the subject-identifier, and a policy requiring other subject attributes

TABLE 7
Inference Types to be Translated

Inference Type: **Supertype inheritance** Logical Formula: **{?A rdfs:subClassOf ?B. ?S a ?A} => {?S a ?B}.** *Example:* Smartphone rdfs:subClassOf Mobile. 'SamsungN7000' a Smartphone => 'SamsungN7000' a Mobile
Inference Type: **Class Transitivity** Logical Formula: **{?B rdfs:subClassOf ?C. ?A rdfs:subClassOf ?B} => {?A rdfs:subClassOf ?C}.** *Example:* Smartphone rdfs:subClassOf Mobile. Mobile rdfs:subClassOf DeviceType => Smartphone rdfs:subClassOf DeviceType

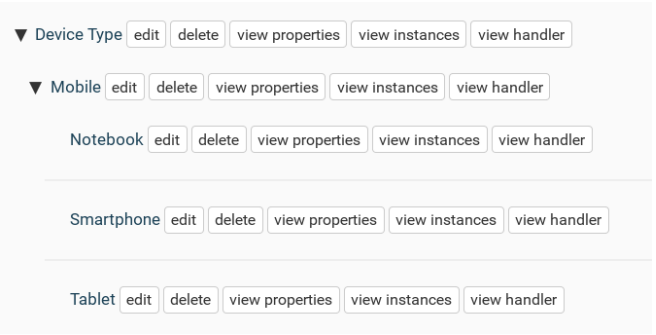


Fig. 7. Context Model Snapshot

(e.g. that require a mapping between subject's latitude and longitude to countries) for evaluation, that are not present in the request and that would have to be retrieved on the fly, based on the subject identifier. This setup of the Balana engine (called Balana++ for the purposes of our evaluation) is the closest possible emulation of the context-aware functionality offered by the PaaSword solution. This evaluation took place on a private Openstack[10] installation, using two identical virtual machines with 4VCPUs, 8GB RAM, 80GB Disk and operating system Ubuntu 16.04.3 LTS.

The measurements against the PaaSword Semantic Authorization Engine were performed for a single Policy containing `n` Rules. These Rules referred to a single Expression consisting of a single Condition requiring the Subject's location to be equal to a certain value. In order to eliminate variations in the execution time of a single request, the measurement script performed several iterations of the same requests and calculated the average mean. Since the PaaSword engine uses REST for both internal and external communication, a network transportation penalty was considered for the measurement of each request. Given that the Semantic Authorization Engine is not called directly but it is invoked by the PaaSword Controller which handles the policy enforcement points, especially smaller numbers of Rules are affected by the initial network transportation overhead. The average round trip time was measured by calling a reference endpoint on the PaaSword Controller and added to the penalty. By design, the PaaSword Semantic Authorization Engine uses a handler

TABLE 8
Drools Rules Based on RDFs Reasoning

```
rule 'SupertypeInheritanceSmartphone'
 when
  PCondElem(elemId='device' && value='Smartphone'
 then
  insertLogical(new PCondElem('device','Mobile'))
 end
rule 'ClassTransitivityMobile'
 when
  PCondElem(elemId='device' && value='Mobile')
 then
  insertLogical(new PCondElem('device','DevType')
 end
```

[9] https://github.com/wso2/balana

[10] https://www.openstack.org/

mechanism to resolve contextual information. These handlers are implemented by external information services which introduce again a network transportation overhead including the service's individual processing time. Their average duration was measured, multiplied by the total count of Rules within the scenario and again added to the penalty.

The results shown in Fig. 8 take penalties of approximately 4ms per Rule for the handlers and 71ms for the communication between measurement script and PaaSword Controller into account. It becomes evident by this evaluation that PaaSword solution, due to its advanced context-aware capabilities, proved to be slower than the Balana engine-based solution. Moreover, both solutions average execution times increase exponentially after the significant increase of deployed rules (>1000 rules). In Fig. 9, a similar trend can be reported for the memory consumed by each engine for evaluating the increasing number of rules. Especially, in the case of PaaSword such an exponential increase of the execution time can be addressed with certain optimisations on the engine that allow the parallelization of pattern matching actions (as seen in the second experiment that follows).

Moreover, as Balana or any other XACML engine doesn't support context inferencing, additional code was required in order to achieve similar capabilities with PaaSword. In real application scenarios such an implementation with Balana would be very difficult to maintain since updates or creation of new policies would require manual updates of this code. This fact constitutes a significant advantage of PaaSword paid with some time penalty as the number of rules increases. In addition, since the PaaSword engine features semantic access control based on an extensible model, each authorization request requires pre-processing by the PaaSword Controller before the Semantic Authorization Engine is called. This pre-processing step transforms the incoming request into a concrete set of addressable Rules to be handled by the engine. The Semantic Authorization Engine then creates additional negation rules as the basis of PaaSword's approach of a closed world assumption. While, in Balana, the deployed rules, include context expressions with one attribute and with a set target value that should be detected in order to provide an authorisation permit result.

Nevertheless, a second series of experiments were conducted to further study the behavior of PaaSword under various loads of simultaneous requests. This second round of experiments was conducted with an upgraded version of the PaaSword ABAC engine, which relied on a radically enhanced release of Drools reasoning engine (version 7.19.0.Final). The engine is now able to evaluate more business rules simultaneously by dividing the RETE and PHREAK[11] pattern-matching algorithms in independent partitions and evaluating them in parallel. This inherent capability improved throughput and reduced significantly the processing time per request.

The experiments were conducted as follows. The ABAC engine was provided with an initial set of knowledge triples for specific Subjects, Objects, Actions and their properties. After the initialization, many concurrent sets of authorization requests were submitted to the engine. These sets included 10, 100, 200, 300, 400, 500, 600, 700, 800, 900 and 1000 simultaneous requests. Every 100ms the engine was queried in order to infer the amount of requests that have been replied. A specific probe was installed to measure the end-to-end execution time along the memory consumption.

This flow was repeated for 4 different setups. The first setup includes an ABAC engine hosting only one rule file with one context expression. The second setup includes an ABAC engine hosting only one rule file with a complex context expression (with 10 attributes). The third setup includes an ABAC engine hosting 10 rules with a complex expression. Finally, the fourth setup relates to an engine hosting 10 policies with 10 rules each. Each request was accompanied by a random set of attributes and each experiment (e.g. 10 simultaneous requests for setup 1) was conducted 10 times in order to get unbiased results. Fig. 10 gives the average serving time of a single request, as the number of simultaneous requests increases (lines P-Setup 1-4). The corresponding times of Balana++ are also included for comparison (B-Setup 1-4). Executions with less than 300 simultaneous requests completed too fast (poll time was 100ms), hence their measurements have been considered inaccurate and have been omitted.

According to the Fig. 10, the average request serving time for PaaSword engine seems to converge as the number of requests increases. Contrary the corresponding

Fig. 8. PaaSword vs. Balana++ – Total execution time

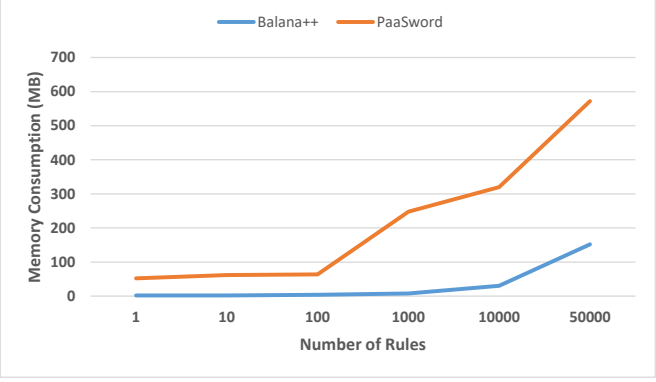


Fig. 9. PaaSword vs. Balana++ – Memory consumption

[11] https://docs.jboss.org/drools/release/7.19.0.Final/drools-docs/html_single/index.html

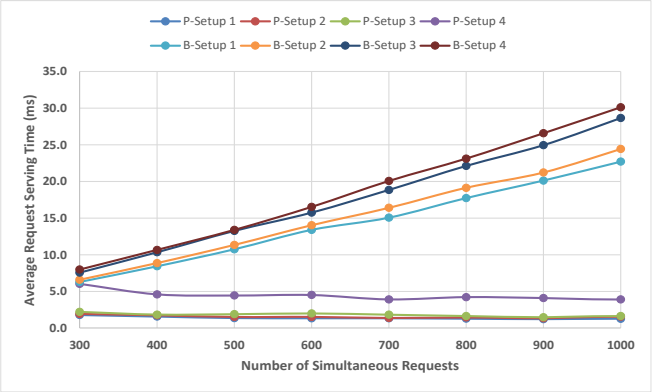


Fig. 10. PaaSword* vs. Balana++ – Average request serving time

readings of Balana++ linearly increase. PaaSword engine measurements span from 1.4ms (setup 1) up to 4.4ms (setup 4), and clearly outperform Balana++ as the parallelization capabilities of the upgraded and underlined Drools engine significantly improved the results.

7 Related Work

This section outlines related work for the main area of attention of this paper, namely access control models; it also outlines related work for policy expression and management as it forms a seminal part of our work.

7.1 Access Control Models

One can generally discern three "traditional" access control models [20]: Mandatory Access Control (MAC), Discretionary Access Control (DAC) and Role-Based Access Control (RBAC); these are also known as identity-based models for they identify users and resources based on their (unique) names [21]. Identity-based models are inherently inadequate for cloud computing as they can only fulfil security requirements in specific environments [22]; they are also context insensitive by-design. Several research efforts have been proposed aiming at extending these models with features that potentially render their use adequate in cloud settings [23]; these efforts are briefly outlined below.

7.1.1 Extending RBAC for Cloud Computing

In [24], Tianyi et al. propose the cloud optimized RBAC model (coRBAC) which inherits features from distributed RBAC (dRBAC). coRBAC merges dRBAC's decentralised authentication services and isolates different organizations by implementing an internal RBAC in each one of them. The approach heavily depends upon a Certificate Authority (CA) for issuing user certificates which may cause efficiency and scalability problems as a new certificate must be issued each time access is required; in addition, it constitutes a single point of attack.

In [25], the authors propose a Task-Role-Based Access Control (T-RBAC) scheme for cloud-based health care systems. T-RBAC enables dynamic activation and revocation of user permissions based on the task at hand, whilst it introduces a workflow authorization model for synchronizing workflow with authorization flow. Nevertheless, it fails to discern sensitivity levels for protected data objects, whilst it is not clear how authorization information may be meaningfully shared between different health institutions.

In [26], Wang et al. recognise that trust is a main concern in cloud computing due mainly to the large number of users and the diverse role classifications utilised across different clouds; they therefore propose an adaptive access algorithm that enriches RBAC with trust relationships between cloud service providers and cloud consumers. Trust levels are calculated dynamically, based on observed user behaviour.

In [22], Younis et al. propose an RBAC-based access control model for cloud computing in which users are classified based on their jobs and are thus placed in a security domain relative to their role; each role comprises a set of tasks, and each task has its individual security classification. Dynamic and random behaviours of users are considered by introducing a risk engine, whereas security tags are issued in untrusted or semi-trusted environments that comprise a user's role, classification, and permissions.

In [27], Hummer et al. propose an RBAC-based model for defining and enforcing Identity and Access Management (IAM) policies in cross-organizational SOA business processes. The processes are described using the WS-BPEL standard, whilst the access policies (roles, permissions and mutual exclusion relations between roles) are expressed in a domain-specific language (DSL) for abstracting away from technological details and involving domain experts in the security modeling process. At deployment time, the WS-BPEL process is instrumented with special activities to ensure its compliance to the IAM policies at runtime

A main drawback of the aforementioned approaches compared to the work presented here is that they fail to effectively integrate, hence take into account, dynamically-evolving contextual information in access control decisions. As already stated, this is an important prerequisite for coping effectively with security challenges in the cloud domain, especially in cases where a lot of context manifestations should be considered [13].

7.1.2 Infusing Context Awareness

In an attempt to enhance security in remote service accesses, a number of efforts have focused on extending identity-based models with location-awareness [20, 28]. Location-aware access control (LAAC) enables access decisions to take into account the physical location from which access requests originate. Nevertheless, even though LAAC models have been studied extensively [28], they generally lack the ability to consider any contextual attributes other than the subjects' physical location (and, of course, its credentials). To overcome this limitation, several apporaches that attempt to integrate a wider range of contextual attributes have been proposed. In [29], the authors propose a scheme that also considers the temporal characteristics of a request. In [16], Hu et al. propose the Attribute-based Access Control (ABAC) scheme which is capable of taking into account any attribute that synthesises the context of an entity that is deemed relevant to a request; notably, ABAC has gained increased popularity due to its diffused XACML implementation that has been endorsed as an OASIS standard [12].

In a similar vein, the authors in [30] propose the Context-Aware RBAC (CA-RBAC) model for pervasive applications that is able to integrate and use contextual information generated from ambient sensors in an "active space". Covington et al. –motivated by the needs of intelligent home applications– introduce the notion of context-aware access control (CAAC) [31]; they propose a set of services that are security-enabled based on the location of not only the subject of a request, but also of the targeted object; however, as opposed to the work reported here, the proposed model fails to account for the specicifity of spatial information such as the multi-granularity of the position. In [32], Y. Yang et al. propose an IAM Architecture for cloud computing, based on several security services offered to cloud customers; they seem to lack, however, the foundation in established and diffused access control standards, such as XACML, and instead invent new terminology for well-defined IAM concepts. In [33], Lodderstedt et al. propose SecureML, an RBAC-based modelling language for integrating access control information into application models expressed in UML; the authors introduce the notion of authorization constraints that are expressed in the Object Constraint Language (OCL) and articulate contextual conditions that must be fulfilled for granting access to sensitive resources.

A main drawback of the aforementioned approaches compared to the work presented here is their inability to define a flexible and extensible underlying context model that would permit the definition of relationships between contextual attributes, thus paving the way for automated reasoning about the satisfaction of access control policies in the face of evolving and dynamically-generated contextual conditions; naturally, as discussed in Section 2, this renders these schemes overly rigid for the requirements of cloud-based systems as the contextual information piggybacked on access requests must necessarily match, at the syntactic level, the corresponding information encoded in the policies.

To overcome this limitation, the access control models proposed in [34—37] advocate two main design guidelines: (i) they incorporate context-awareness to control resource access and to enable dynamic adaptation of policies depending on context changes; (ii) they employ semantic technologies for context/policy specification to allow high-level description and reasoning about context and policies[12]. Compared to our work, however, none of these approaches has been designed to provide fine-grained data access control, e.g. by providing the ability to specify different access rules for different rows of a database. Moreover, as opposed to our work, none of these approaches provides any means for generically describing access control policies which crucially precludes any automated checks regarding: (i) the well-formedness of policies, i.e. whether policies include all the (contextual) information required for granting, or denying, access to sensitive data; (ii) the disclosure of any inter-policy relations such as subsumption and contradiction. Such checks are clearly of utmost importance for they increase our assurance on the effectiveness of the policies, especially of complex ones that integrate a multitude of contextual attributes. Last but not least, as opposed to our work, none of these approaches addresses the scalability issues that semantic reasoning brings about.

7.1.3 Attribute-Based Encryption

Recently Attribute-Based Encryption (ABE) has gained popularity as a data access control mechanism, where encryption keys are derived based on contextual attributes that characterise a user, thus allowing any user with the right attributes to successfully decrypt and access the encrypted files. The ABE paradigm has been applied to cloud computing in several different approaches, e.g. [38—42]. ABE-based solutions have three major limitations compared to our approach: First, ABE is only applicable for data access control; other resources (e.g. services, actuators) cannot be controlled with ABE since no additional enforcement mechanisms beyond encryption are typically employed. Second, when a user's attributes are revoked in an ABE system, all files associated to the attributes have to be re-encrypted and the keys to be updated. This puts heavy computation and key update requirements on the data owner or the system to which the owner has delegated this task. Last but not least, ABE-based solutions are bound to predefined sets of attributes which renders them overly rigid for the requirements of cloud-based systems as the contextual information piggybacked on access requests must necessarily match, at the syntactic level, the corresponding attribute values articulated by the encryption scheme.

Table 9 summarises how the works outlined in this section, compare to our work. The comparison is based on five distinct criteria that are seminal for judging the suitability of an approach for cloud settings. Based on this table, it becomes evident that none of the reviewed works are able to demonstrate support that satisfies all these five criteria. The work introduced in this paper addresses all these aspects and with respect to the context sensitivity, it is able to cope with dynamically-generated contextual attributes.

7.2 Policy Management

As shown in [43], a major weakness of contemporary open-source registry and repository systems is that policy definition and policy enforcement are entangled in the implementation of a single software component: the rules that a policy comprises are typically encoded in an imperative manner, as part of the same code that checks for potential policy violations. This has a number of negative repercussions among which is the lack of portability and the lack of explicit representation of policy relationships. To overcome this weakness, the research community has considered approaches that promote declarative policy representations; these approaches may be discerned into purely syntactic and semantic ones.

[12] On a similar note, the XACML Technical Committee discussed possible approaches of enriching ABAC with semantic technologies (see https://wiki.oasis-open.org/xacml/XACMLandRDF); however, these discussions have yet to lead to the development of any standard documents.

7.2.1 Syntactic Descriptions

Syntactic descriptions were introduced along with the Service Oriented Architecture (SOA) model as part of a standardisation efforts aiming, primarily, at facilitating interoperable data exchanges in interactions. In the realm of policies and policy-based applications, syntactic descriptions promote a declarative approach to policy expression, one which aims at tackling the disadvantages of encoding policies imperatively [18]. A number of formalisms that advocate the use of markup languages for the declarative representation of policies have been proposed (RuleML[13], XACML [12], WS-Trust [44]). Although they succeed in disentangling the representation of policies, from the code employed by an application for enforcing them, these formalisms lack any form of semantic agreement outside the confines of the organisations that created them. Any interoperability, thus hinges on ad-hoc vocabularies that are shared by the various stakeholders that participate in an interaction. This inevitably brings about the following disadvantages: (i) it restricts the portability and reusability of policies; (ii) it hampers the determination of inter-policy relations; (iii) it leads to ad-hoc reasoning about policy compliance, one which is perplexed with the particular vocabularies that are utilised for expressing the rules according to which the reasoning takes place; (iv) it hinders the performance of rule-based policy governance.

7.2.2 Semantic Approaches

To overcome these disadvantages, a number of semantically-rich approaches to the specification of policies have been proposed [45, 46]. These generally embrace Semantic Web representations (e.g. OWL[14]) for capturing the knowledge artefacts that underly policies. More specifically, these approaches employ ontologies in order to assign meaning to actors, actions and resources. Being a formal, explicit specification of a shared conceptualization [47], an ontology provides a flexible, formal, and unambiguous means of agreement upon the semantics of concepts, and their interrelations, in a given domain of discourse. In [45], the authors propose KAoS – a general-purpose framework for managing, monitoring and enforcing a wide range of policies. In [46], the authors propose POLICYTAB for facilitating trust negotiation in Semantic Web environments. The aforementioned semantically-enhanced approaches rely on bespoke, non-standards-based, ontologies for the representation of policies. This by definition restricts the generality, hence the portability and reusability of the policies that they represent. In contrast, our reliance on Linked-USDL raises this restriction. In addition, their reliance on OWL, despite the obvious benefits stemming from the rich set of properties that OWL offers, raises concerns about the degree to which these approaches are lightweight, hence their performance is questionable.

8 Conclusions

We have proposed, implemented and evaluated an approach based on the popular access control standard XACML, which adds semantic reasoning capabilities to the policy design, attribute gathering, and policy evaluation process. Using this approach, we can bridge the gap between syntactically different, but semantically equal attributes that are relevant for policy evaluation, thus furthering federation of policies, and related attribute gathering between different administrative domains.

Based on the performed comparative evaluation, two series of experiments were conducted. The first one involved an increasing number of rules to be evaluated while the second one involved an increasing number of simultaneous access requests. The PaaSword solution, proved to be slower than the Balana WSO2 engine in the first set of experiments. But, before conducting the second experiment PaaSword was upgraded based on the latest version of the underlying Drools engine. Based on the advanced parallelization capabilities of the rule engine PaaSword outperformed Balana++ with respect to the average request serving time for a static number of rules deployed.

TABLE 9
Comparison Criteria

Criteria		Yes	No
Context sensitivity	Location-only	[20, 28]	[22, 24–27]
	Predefined only contextual attributes	[31, 38–42]	
	Dynamically-generated contextual attributes	[16, 29, 30, 32, 33]	
Fine-grained access control (sensitivity differentiation of targeted objects)		[22, 41]	[16, 20, 22, 24–33, 38–42]
Semantic technologies to enable high-level description and reasoning about context and policies		[34–37]	[16, 20, 22, 24–33, 38–42]
Scalability issues stemming from semantic reasoning		None	[16, 20, 22, 24–42]
Automated checks regarding access control policy well-formedness and disclosure of inter-policy relations			

Acknowledgments

The research leading to these results has received funding from the EU's H2020 programme under grant agreement No 644814, the PaaSword project (www.paasword.eu).

References

[1] Cisco. Cloud: What an Enterprise Must Know, Cisco White Paper [Online]. Available: https://www.avitgroup.com/wp-content/uploads/2016/07/Cisco-Cloud-What-an-Enterprise-Must-Know.pdf, 2011.

[13] http://wiki.ruleml.org/index.php/SpecicationofDeliberationRuleML1:01

[14] https://www.w3.org/OWL/

[2] G. A. Moore, A. "Crossing the Chasm: Marketing and Selling Technology Products to Mainstream Consumers," *Harper Business*, 1991, New York, NY, ISBN:006-662002-3.

[3] CSA, "The Notorious Nine. Cloud Computing Top Threats in 2013," Cloud Security Alliance, 2013 [Online]. Available: http://www.cloudsecurityalliance.org/topthreats.

[4] RightScale, "State of the cloud report," *RightScale*, 2017 [Online]. Available: https://www.rightscale.com/2017-cloud-report.

[5] D. Povar, G. Geethakumari, "Digital Forensic Architecture for Cloud Computing Systems: Methods of Evidence Identification, Segregation, Collection and Partial Analysis. Information Systems Design and Intelligent Applications," in *Proceedings of Third International Conference*, India, 2016, Volume 1, 10.1007/978-81-322-2755-7_22.

[6] T. A. Hemphilla, P. Longstreetb, "Financial data breaches in the U.S. retail economy: Restoring confidence in information technology security standards, " *Technology in Society*, vol. 44, pp. 30-38, Feb. 2016, http://dx.doi.org/10.1016/j.techsoc.2015.11.007.

[7] F. Bieker, et al., "A Process for Data Protection Impact Assessment Under the European General Data Protection Regulation," *Annual Privacy Forum*, Springer International Publishing, LNCS 9857, pp. 21-37, 2016, DOI: 10.1007/978-3-319-44760-5_2.

[8] N. Paladi, A. Michalas, "One of our hosts in another country: Challenges of data geolocation in cloud storage," In *proceedings of the 4th International Conference on Wireless Communications*, Vehicular Technology, Information Theory and Aerospace Electronic Systems (VITAE), pp. 1-6, 2014, DOI:10.1109/VITAE.2014.6934507.

[9] Y. Verginadis, G. Mentzas, S. Veloudis, I. Paraskakis, "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 (UCC), IEEE/ACM, 2015.

[10] G. Abowd, E. Mynatt, "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, pp. 29-58, 2000.

[11] A. K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing Journal*, vol. 5, no. 1, pp. 4-7, 2001.

[12] E. Rissanen (ed.), "eXtensible Access Control Markup Language (XACML) Version 3.0," *Organisation for the Advancement of Structured Information Standards (OASIS)*, OASIS Standard, January 2013. [Online] Available: http://docs.oasis.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf.

[13] Y. Verginadis, A. Michalas, P. Gouvas, G. Schiefer, G. Hübsch, I. Paraskakis, "PaaSword: A Holistic Data Privacy and Security by Design Framework for Cloud Services, " In *proceedings of the 5th International Conference on Cloud Computing and Services Science (CLOSER 2015)*, Lisbon, Portugal, May 2015, DOI: 10.5220/0005489302060213.

[14] N. Paladi, A. Michalas, C. Gehrmann, "Domain based storage protection with secure access control for the cloud," In *proceedings of the International Workshop on Security in Cloud Computing*, ACM, New York, NY, USA, ASIACCS '14, 2014, DOI: 10.1145/2600075.2600082.

[15] N. Santos, K. P. Gummadi, R. Rodrigues, "Towards trusted cloud computing, " In *proceedings of the Conference on Hot Topics in Cloud Computing*, USENIX, Berkeley, CA, USA, HotCloud'09, 2009, URL: http://dl.acm.org/citation.cfm?id=1855533.1855536.

[16] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone, "Guide to Attribute Based Access Control (ABAC) Definition and Considerations," *NIST Special Publication 800-162*, 2014, URL: http://dx.doi.org/10.6028/NIST.SP.800-162.

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

[18] Y. Verginadis, I. Patiniotakis, G. Mentzas S. Veloudis, I. Paraskakis, "Data Distribution and Encryption Modelling for PaaS-enabled Cloud Security", In *proceedings of the 2nd International Workshop on Cloud Security and Data Privacy by Design (CloudSPD'16)*, co-located with the 8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2016), Luxembourg, December 12-15, 2016.

[19] K. Sachin, S. Jaloree, and R. S. Thakur, "Performance Comparison between Forward and Backward Chaining Rule Based Expert System Approaches Over Global Stock Exchanges," *International Journal of Computer Science and Information Security*, vol.14, no. 3, 2016.

[20] M. Decker, "Modelling of location-aware access control rules," *Handbook of Research on Mobility and Computing*: Evolving Technologies and Ubiquitous Impacts, IGI Global, pp. 912-929, 2011, DOI: 10.4018/978-1-60960-042-6.ch057.

[21] A. R. Khan, "Access control in cloud computing environment," *ARPN Journal of Engineering and Applied Science*, vol. 7, no. 5, pp. 613-615, 2012.

[22] Y. A. Younis A., K. Kifayat, M. Merabti. "An access control model for cloud computing." *J. Inf. Sec. Appl.* 19 (2014): 45-60.

[23] Y. Verginadis, A. Michalas, P. Gouvas, G. Schiefer, G. Hubsch, I. Paraskakis, "PaaSword: A Holistic Data Privacy and Security by Design Framework for Cloud Services," *Journal of Grid Computing*, pp. 1-16, 2017, DOI: 10.1007/s10723-017-9394-2.

[24] Tianyi Z, Weidong L, Jiaxing S, "An efficient role-based access control system for cloud computing", *In proceedings of the 11th International Conference on Computer and InformationTechnology*, IEEE, Pafos, Cyprus, pp. 97—102, 2011, DOI: 10.1109/CIT.2011.36.

[25] H. A. Jayaprakash Narayanan, M. H. Gunes, "Ensuring access control in cloud provisioned healthcare systems." In *proceedings of the 2011 IEEE Consumer Communications and Networking Conference (CCNC)*, IEEE, 2011, pp. 247–251, DOI: 10.1109/CCNC.2011.5766466.

[26] W. Wang, J. Han, M. Song, X. Wang, "The design of a trust and role-based access control model in cloud computing." In *proceedings of the 6th International Conference on Pervasive Computing and Applications*, IEEE, 2011, pp. 330–334, 2011, DOI: 10.1109/ICPCA.2011. 6106526.

[27] W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, S, Dustdar, "An integrated approach for Identity and Access Management in SOA Context," In *proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT'11)*, Innsbruck, Austria, pp. 21—30, 2011.

[28] A. Cleeff, W. Pieters, R. Wieringa, "Benefits of location-based access control: A literature study," In *proceedings of the IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing*, IEEE Computer Society, Washington, DC, USA, GREENCOM-CPSCOM'10, pp. 739-746, 2010, DOI: 10.1109/GreenCom-CPSCom.2010.148.

[29] S. M. Chandran, J. B. D. Joshi, "Lot-rbac: A location and time-based rbac model," In *proceedings of the 6th International Conference on Web Information Systems Engineering*, Springer-Verlag, Berlin, Heidelberg, WISE'05, pp. 361-375, 2005, DOI: 10.1007/11581062 27.

[30] D. Kulkarni, A. Tripathi, "Context-aware role-based access control in pervasive computing systems," In *proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, ACM, New York, NY, USA, SACMAT'08, pp. 113-122, 2008, DOI: 10.1145/1377836.1377854.

[31] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, G. D. Abowd, "Securing context-aware applications using environment roles," In *proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, ACM, New York, NY, USA, SACMAT'01, pp. 10-20, 2001, DOI:10.1145/373256.373258.

[32] Y. Yang, X. Chen, G. Wang, L. Cao, "An Identity and Access Management Architecture in Cloud", In *proceedings of the 7th International Symposium on Computational Intelligence and Design*, IEEE, Hangzhou, China, 2014, DOI: 10.1109/ISCID.2014.221.

[33] T. Lodderstedt, D. A. Basin, J. Doser, "Secureuml: A uml-based modeling language for model-driven security," In *proceedings of the 5th International Conference on The Unified Modeling Language*, Springer-Verlag, London, UK, UML'02, 2002, pp. 426-441.

[34] H. Shen, Y. Cheng, "A context-aware semantic-based access control model for mobile web services," *Advanced Research on Computer Science and Information Engineering, Communications in Computer and Information Science*, Springer Berlin Heidelberg, vol. 153, pp. 132-139, 2011, DOI: 10.1007/978-3-642-21411-0 21.

[35] A. Toninelli, R. Montanari, L. Kagal, O. Lassila, "A semantic context-aware access control framework for secure collaborations in pervasive computing environments," In *proceedings of the 5th International Conference on The Semantic Web*, Springer-Verlag, Berlin, Heidelberg, ISWC'06, pp. 473-486, 2006, DOI: 10.1007/11926078 34.

[36] A. S. M. Kayes, J. Han, A. Colman, "An ontology-based approach to context-aware access control for software services," In Lin X, Manolopoulos Y, Srivastava D, Huang G (eds) WISE, Springer, *Lecture Notes in Computer Science*, vol. 8180, pp. 410-420, 2013.

[37] L. Costabello S. Villata, F. Gandon, "Context-aware access control for rdf graph stores," Frontiers in Artificial Intelligence and Applications, *Raedt LD, Bessiere C, Dubois D, Doherty P, Frasconi P, Heintz F, Lucas PJF (eds) ECA*I, IOS Press, vol. 242, pp. 282- 287, 2012.

[38] G. Wang, Q. Liu, J. Wu, and M. Guo, "Hierarchical attribute-based encryption and scalable user revocation for sharing data in cloud servers," *Computers Security*, vol. 30, no. 5, pp. 320-331, 2011.

[39] M. Zhou, Y. Mu, W. Susilo, and J. Yan, "Piracy-preserved access control for cloud computing," In *proceedings of the IEEE TrustCom'11*, pp. 83-90., 2011.

[40] Z. Yan, X. Li, M. Wang, A. V. Vasilakos, "Flexible Data Access Control Based on Trust and Reputation in Cloud Computing," *IEEE Transactions on Cloud Computing*, vol. 5, no. 3, pp. 485-498, 2017.

[41] H. He, R. Li, X. Dong, Z. Zhang, "Secure, Efficient and Fine-Grained Data Access Control Mechanism for P2P Storage Cloud", *IEEE Transactions on Parallel and Distributed Systems,* vol. 25, no. 7, pp. 471—484, 2014.

[42] K. Yang, X. Jia, "Expressive, Efficient and Revocable Data Access Control for Multi-Authority Cloud Storage", *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 1735—1744, 2014

[43] D. Kourtesis, I. Paraskakis, "A registry and repository system supporting cloud application platform governance," In *proceedings of the International Conference on Service-Oriented Computing*, Springer-Verlag, Berlin, Heidelberg, ICSOC, pp. 255-256, 2012, DOI: 10.1007/978-3-642-31875-7 36.

[44] WS-Trust 1.3, OASIS [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.doc

[45] A. Uszok, J. Bradshaw, R. Jeffers, M. Johnson, A. Tate, J. Dalton, and S. Aitken, "KAoS Policy Management for Semantic Web Services," *IEEE Intelligent Systems*, vol. 19, no. 4, pp. 32-41, 2004.

[46] W. Nejdl, D. Olmedilla, M. Winslett, C. C. Zhang, "Ontology-Based policy specification and management, " In *proceedings of ESWC'05*, Springer-Verlag, Berlin, Heidelberg, pp. 290-302, 2005.

[47] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," *International Journal of Human-Computer Studies*, vol. 4, pp. 907-928, 1995, DOI: 10.1006/ijhc.1995.1081.

Yiannis Verginadis received his Ph.D. degree in Electrical and Computer Engineering from the National Technical University of Athens (NTUA), Greece in 2006. He is currently a senior researcher at the Institute of Communications and Computer Systems (ICCS). He has more than ten years of experience in R&D projects and presents a strong publication record in: information systems, BPM and cloud computing.

Ioannis Patiniotakis received his Ph.D. degree in Electrical and Computer Engineering from the NTUA, Greece in 2015. He is currently a researcher at the ICCS. Since 1998 he has been employed as a professional software engineer in major software and consulting firms in Greece. His research interests include semantic web, cloud computing and security, recommender and decision support systems.

Panagiotis Gouvas received his Ph.D. degree in Electrical and Computer Engineering from the NTUA, Greece in 2011. He is currently the R&D Director of Ubitech Ltd. He is a Member of the Technical Committee of OASIS Cloud Application Management for Platforms (Schema CAMP), and he is the Workshop Leader of DIN SPEC 91337 dealing with Unified Application Management Interface for Cloud Application Platforms.

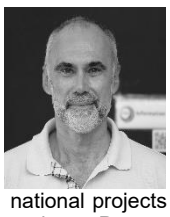
Spyros Mantzouratos currently works as senior researcher and integration engineer in Ubitech Ltd. He has an extensive experience in industry with respect to integration and development with a strong focus on cloud security. His research interests include cloud computing, access control and fog computing.

Simeon Veloudis holds a BSc and a PhD degree in Computer Science from the University of Reading. He has been employed for several years as a lecturer at various educational institutions. Currently he is a research associate at the South East European Research Centre (SEERC). His research interests lie in the realms of formal security modelling, in cloud computing, and in real-time safety critical systems.

Sebastian Thomas Schork received his BSc degree in computer science and his MSc in computer science with the emphasis on intelligent systems from Karlsruhe University of Applied Sciences, Germany, in 2015 and 2016. He is currently working as Software Engineer and Researcher on various research projects at CAS Software AG, Karlsruhe, Germany. His research interests include cloud security, data privacy and cloud computing.

Ludwig Seitz received his Ph.D.in Computer Science from the National Institute of Applied Sciences (INSA) of Lyon, France in 2005. He is currently a researcher at Swedish Institute of Computer Science (RISE SICS). He is heavily involved in standardisation, previously at OASIS (XACML standard) and currently at the IETF (working groups ACE and CoRE). His research interests include access control, information and usable security, and privacy.

Iraklis Paraskakis holds a PhD in IT and Education from the Open University (UK). He is currently a Senior Lecturer in the Department of Computer Science at CITY College, Greece. He is also a Senior Research Officer at SEERC, and is coordinating the Information & Knowledge Management Research Group. His research interests include Cloud computing, Governance and Quality Control, Service Oriented Computing, and information systems.



Gregoris Mentzas is a Professor at the School of Electrical and Computer Engineering and Director of the Information Management Unit (IMU). He serves as Director of the division of Industrial Electric Devices and Decision Systems. He holds a Ph.D. in Operations Research and Information Systems from NTUA in 1988. Prof. Mentzas has coordinated or participated in more than 40 international projects and is an Associate Editor in five scientific journals and was Program Committee member in more than 55 international conferences.