Authentic Refinement of Semantically Enhanced Policies in Pervasive Systems

Julian Schütte[1], Nicolai Kuntze[1], Andreas Fuchs[1], and Atta Badii[2]

[1] Fraunhofer Institute for Secure Information Technology SIT
[2] IMSS, University of Reading

Abstract Pervasive systems are characterised by networked heterogeneous devices. To fulfill the security requirements of an application, these devices have to abide by certain policies. However, as the contingent interaction between devices in all possible contexts within evolving pervasive systems devices cannot be known at development time, policies cannot be dedicated to concrete security mechanisms which might later not be supported by the devices present in the network. Therefore, policies need to be expressed at a more abstract level and refined appropriately to suit applicable mechanisms at run time. In this paper we describe how security policies can be combined with ontologies to support such an automated policy refinement. As thereby policy decisions depend on semantic descriptions, the correctness of these descriptions must be verifiable at a later time for policy decisions to be evidential. We therefore propose Trusted Computing-based approaches on generating proofs of correctness of semantic descriptions deployed in policies.

1 Introduction

Pervasive systems are dynamic infrastructures with heterogeneous devices unpredictably joining and leaving the network. Due to their complexity and heterogeneity, traditional access-control techniques and hard-coded mechanisms for communication protection come to their limits. Policies as a form of restricted high-level programming have been used for decades to regulate access to resources and have also been adapted for use with pervasive systems. Considering the great variety of platforms present in a pervasive system, the aspect of dynamic policy refinement gains more and more importance. Policy refinement denotes the process of deviating concrete applicable mechanisms from an abstract, high-level policy definition. Only if the level at which policies are specified is sufficiently abstract, will it be possible to adapt devices and services in a way to match the policy by finding a set of rules that is tailored to the capabilities of the platform. Such a set of rules is called a *refinement* of a high-level policy if its effect is an instance of the high-level policy effect. The integration of Semantic Web Technology (SWT) and policies is a promising approach in order to achieve a greater degree of abstraction, thereby facilitating the specification of policies and decoupling them from the actual enforcing platform. However, it must be considered that as soon as policies rely on external knowledge bases,

all information contained in the knowledge bases becomes security critical and thus its correctness must be verifiable. A way to provide proof of semantic device descriptions stated in ontologies is to let devices attest their semantic descriptions. Attestation in this context provides a verifiable reporting of a platform configuration that is based on an authenticated root of trust.
In this paper we propose combining semantic knowledge with policies to build a security policy architecture for pervasive systems. We further present an approach to apply attestation of properties for generating evidence of the information stored in knowledge bases in such a way that participants are able to verify the policy decision process and the semantic knowledge used in it.
The paper is structured as follows: in section 2 reviews work related to ours. In section 3 we then introduce an example scenario to motivate our approach. In section 4 we give an overview of the main building blocks of our approach. Then, in section 5 we introduce the policy model used and present the integration of semantic knowledge. In section 6 we explain how the correctness of semantic knowledge can be guaranteed and section 7 concludes the paper and outlines future work.

2 Related Work

Work related to ours is concerned with the integration of SWT into policies as well as refinement of high-level policies. The integration of semantic knowledge and security policies has been recognised as a promising approach for de-coupling the implementation from the actual behavior of a system. It has been shown that semantic information can be integrated into existing policy languages like XACML and WS-Policy [3,19]. Yet, these approaches do not propose using complex class expressions as policy elements and thus do not achieve the level of flexibility as provided by our proposed approach. Examples for policy frameworks based on semantic policy representation are KAoS [18] and Rein(n) [7,5]. Rein(n) is based on rules written in the RDF-based rule language *Notation3*[3] and has been used in [11] to implement a policy-controlled pervasive system. KAoS represents even the policy structure itself in description logic (i. e. in DAML, the antecedent of OWL) and makes use of the Java Theorem Prover[4] in order to integrate rules and variables which are not supported by plain DAML / OWL. In [16], an overview of these two approaches, as well as the policy framework Ponder2 is given. Ponder2 proposes a dedicated policy framework for pervasive systems [17,2] but does not integrate semantic knowledge. In contrast to that, the context-aware policy framework Proteus [15] features OWL for modeling context conditions which provide the basis for access control decisions. However, Proteus does not take refinements and negotiation of obligations into account. Refinement of semantic policies is considered in [6] where an approach on refining high-level policies is presented,yet, without integrating negotiation between

[3] http://www.w3.org/DesignIssues/Notation3
[4] http://ksl.stanford.edu/software/JTP/

multiple parties. Also, the authors focus on finding suitable web service compositions, in contrast to our approach which aims at finding appropriate security mechanisms. The aspect of policy negotiation is discussed in [8] where the authors propose a preferences model based on utility functions and description logics for agreeing on the most preferred setting. Although the authors do not consider the refinement process and focus on a slightly different setting than we do (e.g., different capabilties of endpoints are not discussed) the approach of negotiations based on individual preferences is promising and could be added as an extension to our solution. In previous work [21] we already showed how global preferences for certain policies can be used to optimize the overall system behavior in terms of security or performance.

3 Scenario

In order to illustrate the challenges we addressed we introduce the following example: imagine that Alice joins an intelligent office environment with her mobile phone A. She wants to use some of the resources provided by the environment, for example she needs to print a document using the printer in the hall. As Alice does not want anybody to intercept the document data while it is being sent to the spooler, she has set up a high-level policy pol_A stating that all communication to printing devices must be confidential. While most traditional policy languages would require Alice to specify which concrete actions should be taken (such as executing a certain encryption protocol), this is not possible in dynamic scenarios where at design time it is not known which actions are supported by the communication partners. Bob, the developer of the intelligent office, whishes to ensure that misuse of the printer B can be traced at a later time, so he wants clients to provide an identification that can be logged. Further, A and B have capabilities which are described in an ontology: capability c_A states that A runs an OSGi platform (and is thus able to dynamically load remote software modules). Capability c_B describes that the printer B is able to run a lightweight encryption protocol.

In order to realize this scenario, it is necessary to integrate semantic descriptions into policies and to use them for refining the high-level policies of Alice and Bob. The refinement has to state concrete security mechanisms which can be applied to both endpoints and fulfill the constraints set by capabilities c_A, c_B and policies pol_A, pol_B. Additionally, in some cases it might be required to verify the correctness of the semantic annotations of A and B. As the policy refinement and therewith the resulting security mechanisms depend on the constraints set by c_A and c_B, Alice must be able to verify that this description is truthful as otherwise semantic descriptions that have been tampered with will lead to an incorrect refinement of the high-level policies and thus to a possibly faulty policy decision. Here, the result of the policy refinement could be an obligation instructing the phone to load a module that implements a lightweight encryption protocol and can be executed by the printer, along with an identification protocol based on the IMEI of the phone. Although these mechanisms fulfil the requirements for confidentiality and identification, they are not the most secure solution but are rather a trade-off between the device capabilities and the security requirements. Finding this trade-off and verifying the correctness of the limiting device capabilities is the subject of our approach.

4 Building Blocks

In this section, we describe how information will be processed and which building blocks are involved.

4.1 Required Components

At first we assume devices come with a *Trust Anchor* (TA). The TA is a module that has the ability to attest the integrity and the state of the software running on the device by means of cryptographic signatures. It therefore measures the current platform configuration as a set of hash values of the currently running software modules and creates a blinded signature of it, called *Commitment*. As the integrity of the TA itself cannot be proven it has to be constructed in a way that other parties can trust its integrity, for example as a *(Mobile) Trusted Platform Module* (MTM, TPM), or as part of a virtualization container (i. e. integrated into a hypervisor kernel).
Second, devices must further be able to communicate via the built-in TA with a *Security Service*. The Security Service acts as a client to the TA and will create an attestation of the properties based on the previously generated commitment that can be considered as a confirmation which guarantees that the device runs a platform that has certain properties (called *capabilities* in this paper) but does not reveal the exact platform configuration (c.f. mapping *dev* in the next subsection). Further, we assume the existence of a common *knowledge base* (KB) providing semantic information about devices and their capabilities as well as about security mechanisms and their properties (c.f. mapping *sec* in the next subsection). Although information about security mechanisms is pre-defined and does not necessarily change while the system runs, information about devices is gained from semantic annotations of the services provided by devices (e. g. by means of SAWSDL [1]) and may thus be updated or changed during run time.
The policy infrastructure is based on the usual components *Policy Decision Points* (PDP) and *Policy Enforcement Points* (PEP), as described by PCIM and COPS. A PDP is a service that receives policy decision requests sent by PEPs, makes a decision about the request based on the policies stored within and sends back a policy decision. PEPs are attached to the services provided by devices and are mainly responsible for intercepting incoming and outgoing requests, sending policy decision requests to the PDP and enforcing the received policy decision using different plug-ins (*Enforcement Modules*). These main building blocks are connected to each other as depicted in Figure 1.

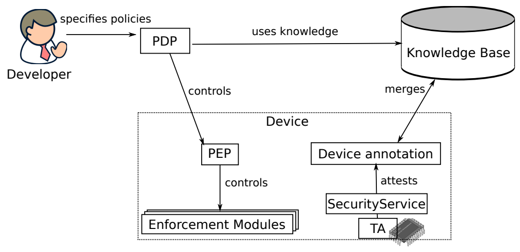

Figure 1. High-level building blocks

4.2 Information sources and mappings

After having introduced the main building blocks, we now describe how information is processed by them.

Knowledge Base The knowledge base *(KB)* takes the form of several ontologies, among them the Device Ontology (cf. Section 5.2) and the Security Ontology (cf. Section 5.2). Besides other information, KB provides the mapping *sec* which assigns concrete security mechanisms $s \in S$ to the security properties $p \in P$ they achieve and the platform capabilities $cap \in Cap$ which are required to execute them.

$$\{P : p_i, Cap : cap_j\} \hookleftarrow sec\,(S : s)$$

Device Annotations Other parts of the ontology, such as device-specific information, are provided at run time by the devices themselves in form of semantic annotations. These automatically generated facts must be verifiable at a later time as policy decisions depend on them. The mapping *dev* from a platform configuration c to the corresponding capabilities *cap* is denoted as

$$\{Cap : cap_i\} \hookleftarrow dev\,(C : c)$$

Security Policies The security policies in our approach regulate access requests to services depending on the requesting *subject*, the requested service (*resource*) and further *conditions* such as current context values, where subjects, resources and conditions can be specified based on facts from the ontology. The result of a policy decision will be a binary decision (deny/permit) and a set of requirements, stating security properties p_i which have to be fulfilled before the decision is enforced. A policy decision process *pol* can thus be denoted as the following operation *pol*:

$$([deny, permit]\,, \{P : p_i\}) \hookleftarrow pol\,(subject, resource, cond)$$

Refinement Process Based on the mappings *sec* and *pol* we define a refinement process ref of finding a security mechanism which matches device capabilities and security requirements stated by policies. The refinement process can also be formulated as a catenation of *sec* and *pol*, as shown in Figure 2:

$$(\{S : s_k\}) \hookleftarrow ref\,(\{Cap : cap_i\}\,, (subject, resource, cond))$$

$$ref = sec^{-1}\,(\{Cap : cap_i\}\,, pol(subject, resource, cond))$$

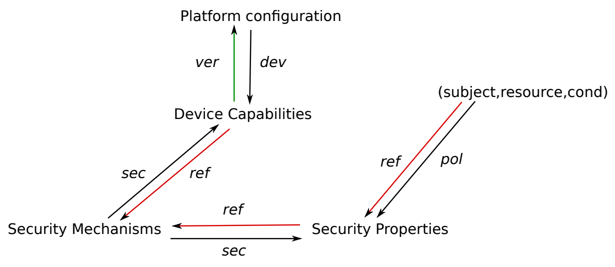

Figure 2. Refinement and verification process

Trusted Verification In addition to the policy refinement process, a verification process *ver* is needed, which allows a verifier to ensure that the semantic information used in the policy process is truthful, i. e. that the mapping *dev* is correct. That is, *ver* shall verify that a device fulfills the capabilities stated by its semantic annotation (i.e. verify an instance of *dev*). The verification process can be formulated as follows, where $\hat{C}$ denotes the set of attested platform configurations:

$$[true, false] \hookleftarrow ver\left(\hat{C} : c, \{Cap : cap_i \mid 0 \leq i \leq m\}\right)$$

5 Semantic access-control policies

In this chapter, we introduce the policy model that our approach relies on, provide a brief overview of the two most important ontologies of our knowledge base and describe the knowledge base is integrated into the policy decision process.

5.1 Policy Model and Decision Process

The authorisation policy model that we use is depicted by Figure 3. It is a simplification of the XACML policy model, so we were able to represent our policies in XACML syntax and use an existing policy decision engine[5] (with significant modifications, though) during the prototype implementation: an *AuthorisationPolicy* consists of a *MetaPolicy* (according to XACML's rule combining algorithm for resolving comflicts between positive and negative decisions) and a number of *AuthorisationRules*. Each AuthorisationRule consists of a *Subject* that requests access of a certain type (*AccessType*) to a *Resource* (similar to the XACML *target* element). Although the model is not limited to any resource or access type, for the sake of clarity in this paper we limit the usage of the *resource* field to identify services and the *access type* field to four different phases of a service invocation (*incoming* / *outgoing* and *request* / *response*). Further, an authorization rule contains a *condition* that must be true for the rule to be applicable and an *obligation* that must be fulfilled before the *Effect* of the rule is enforced. The *condition* refers to context information which can be used to describe the current situation and the obligation defines actions which can be

[5] SunXACML, http://sunxacml.sourceforge.net/

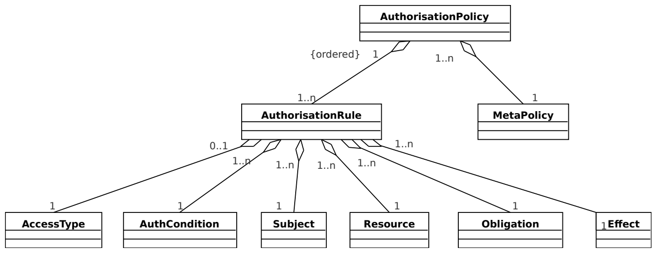

Figure 3. Model of authorisation policies

carried out by different *enforcement modules*. Effects supported by the policy model are either *permit* or *deny,* so that it is possible to specify positive as well as negative authorizations. By combining outgoing requests with negative authorizations, the policy model further covers refrain policies, i. e. policies that prevent a subject from sending out potentially harmful requests, for example, in order to protect it from revealing critical information to outsiders.

In order to integrate semantic information into policies, we represent Subjects and Resources in Description Logic (DL) using complex class expressions formulated in OWL Manchester syntax [4]. A complex class expression is a combination of OWL classes, properties, set operators and cardinality restrictions. So, referring to the example above, Alice's policy could describe the printer service by the complex class expression *(Subject* ***AND*** *Printer)* ***THAT*** *supportsSecurityMechanism* ***SOME*** *SecurityMechanism* ***THAT*** *supportsProtectionGoal* ***VALUE*** *confidentiality.*

When a device requests to access another device, the policy decision process is as depicted in Figure 4. The request from device A to B is intercepted by A's

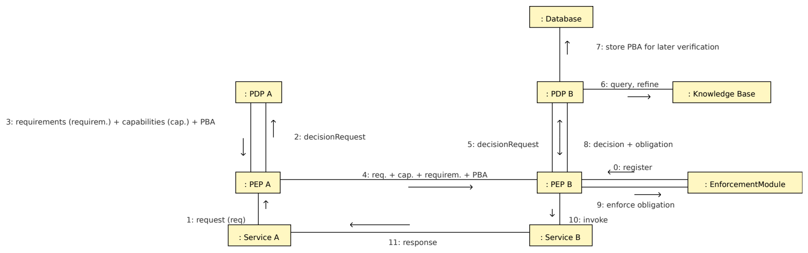

Figure 4. Collaboration of components for policy decision process

PEP, forwarded to its PDP where it is annotated with A's capabilities cap_A, an attestation $AT_A$of them and high-level requirements (e.g., *confidentiality,* in the example) and then sent to B where it is again intercepted and forwarded to B's PDP where the following refinement process takes place:

1. Given the subject (A), the access type (*call*), resource (B) and further context values, the PDP retrieves B's high-level requirements in form of security properties p from the policy.
2. PDP_B extracts A's capabilities *cap* from the annotated request.

3. PDP_B retrieves AT_A from the annotated request and verifies it as described in section 6.
4. Assuming the policy evaluates to *permit* in step (1), PDP_B finds an obligation *obl* defining a security mechanism which refines *req* and *cap* from A and B s.t.

$$\begin{aligned} O = &\ \mathit{SecurityProtocol} \sqcap \\ &\ \exists.\mathit{hasObjectiveStrengthRel}\left(\sqcap_i \left(p_i\right), \geqslant \varepsilon\right) \end{aligned}$$

5. PDP_B returns the policy decision (*permit*), the obligation *obl* and its own attestation of properties AT_B to the PEP which then sends AT_B, cap_B and p_B to A for later verification and enforces the obligation.

After this process, A and B are in possession of the following values: AT_A, AT_B, cap_A, cap_B, p_A, p_B and *obl*. The obligation *obl* will be sent to and enforced by the PEP that initially triggered the policy decision. The obligation *obl* is a result of the constraints set by the requirements and capabilities provided by the devices. It would thus be possible for a device to announce wrong capabilities in order to manipulate the refinement process in a way that weaker, possibly vulnerable, security mechanisms are chosen as *obl*, for example. Therefore, it must be possible at a later time to verify the correctness of device requirements and capabilities. By using attestation certificates AT_A and AT_B and mapping *dev* provided by KB, a verifier can validate that, at the time of the policy decision, device A and B have run a platform configuration that actually provides the announced requirements and capabilities. The next section describes how these attestations can be generated.

5.2 Knowledge Model

The policy refinement process makes use of semantic information which is represented in ontologies. Ontologies model knowledge in terms of classes which are abstract descriptions of entities (e. g., *SecurityProtocol*), relations beween them (e. g., *X supportsProtectionGoal Y*) and individuals which instantiate these classes (e. g., *OpenSSLv1*). The ontologies in our approach include the mappings *sec*, defined by a Security Ontology and *dev*, defined by device annotations and a Device Ontology. Besides these mappings, the Security- and Device Ontologies provide further information which can be used when specifying subjects, resources or conditions. For example, in our prototype we integrated the Pellet 2.1 reasoner in order to check whether subjects and resources contained in an access request are instance of the DL class expression stated in the policy.

Device Ontology In order to model the mapping *dev* we make use of the Device Ontology from the Hydra project[6] which is originally based on the FIPA device ontology[7] and the W3C DeliveryContext ontology[8] but has been largely

[6] http://hydramiddleware.eu
[7] http://www.fipa.org/specs/fipa00091/PC00091A.html
[8] http://www.w3.org/TR/dcontology/

extended to cover the needs of pervasive systems, such as modeling energy efficiency criteria, supported software libraries, etc. Further ontologies have been created and attached to the Device Ontology, including models of QoS parameters, services descriptions and possible malfunctions.

The relations and concepts and some individuals of the Device Ontology are pre-defined and fixed. Yet, as devices which are not known at design time need to be integrated into the knowledge base at run time, they provide descriptions of themselves in form of semantic annotations. The semantic annotation of a device comprises at least one individual of the main *HydraDevice* concept, possibly along with further properties. These semantic annotations are provided by a special service on the device and are retrieved and integrated into the knowledge base as the device joins the network. Referring to the example from section 3, the smart phone could be described by the following annotation (In Notation3. Namespaces have been omitted):

```
:AlicePhone a :Smartphone;
              :deviceId "1234";
              :hasHardware [a :DeviceHardware;
                              :availableMemory 2048;];
              :hasSoftware [a :softwarePlatform;
                              :hasVirtualMachine SUN Java CDC;
                              :hasModularisation FelixOSGi;];
              :info [a :InfoDescription;
                       :friendlyName "AlicePhone";
                       :modelDescr "G1DevPhone1.4";].
```

Security Ontology Information about security mechanisms is represented in the *Security Ontology*. Its main purpose is to provide the mapping *sec* from high-level security properties (e. g. protection goals) to specific security mechanisms in the form of implementation modules that can be applied at run time. It further provides additional information about these modules such as CPU and memory requirements and information about assurances of their security level, as provided by third parties such as FIPS [10] or Common Criteria. The concept *SecurityProtocol* denotes specific software modules that can be installed and started at run time. An estimation of their resource consumption is given by the two relations *requiresComputingPower* and *requiresMemory* and their security properties are modeled by the *SecurityObjective* concept which describes protection goals such as authentication, confidentiality or integrity. By the *ObjectiveStrengthRelation* each module is assigned a protection goal and a strength to which it supports this goal. That way, it can be expressed that a module is suited to achieve "high" confidentiality while it supports only "low" authenticity, for example.

6 Attestation of Properties

During the policy decision process a device claims to comply with certain properties. Each property claimed by the device refers to information in the ontologies which is the basis for policy decisions. It is therefore a requirement to authenticate devices with respect to their claimed properties in order to avoid that

devices induce wrong policy decisions by claiming false properties. This section introduces different methods to ensure that devices align with the information stated about them in the ontologies, i.e. that the device annotation mapping *dev* is correct. As introduced, we assume that devices contain a Trusted Computing Base (TCB) that consists of a Trust Anchor (TA) and the platform on which the TA is integrated. To create an authentic statement on the trustworthiness of the particular device and the provided properties the TCB can either monitor the behavior of the system that is running in an untrusted area or provide a proof of the state of the whole device. The former requires attestation limited to a trusted part and assumes certain trust boundaries, the latter requires proof on every piece of software running on the device.
The local monitoring approach assumes that a device consists of two different trusted domains. The TCB as a trusted domain must be enforced by means of, for example,a secure firmware or hardware rooted trust reporting as defined by the Trusted Computing Group (TCG) as explained below. This domain consists of the necessary infrastructure in order to monitor loading and execution of software component of the untrusted domain. Such monitoring may be implemented by several means, such as load time certificates, proof carrying code or inline monitoring.
An example for load time certificate validation is Reference Integrity Metrics (RIM) certificates as defined for Mobile Trust Modules [9]. In this concept, each software component is delivered with an appropriate certificate by a trusted party that attests the integrity of the software component. A software component may therefore only be loaded into the memory of the untrusted domain, if the provided certificate refers to the code to be executed and can be verified. An extension to this concept is the provision of properties of those software components instead of a general guarantee of trustworthiness, such that a PDP may check its policies accordingly. An alternative concept that does not require data other than the software itself is provided by inline monitoring of software properties. This however requires the PDP to transform its policies into a set of enforceable properties against the actual code that introduce checking of certain constraints during runtime, e.g. by means of aspect oriented programming.
The concept of the TCG, that is also required for the trusted domain of the monitoring approach, can be extended to the software component to be loaded itself. Based on a hardware root of trust, that is usually implemented within the pre-loader of the BIOS, every program code loaded into memory is hashed and stored to a Trusted Platform Module (TPM) before execution. Accordingly, the TPM holds the complete executional sequence of the platform from boot to present. This information can be reported to other parties, providing authentic evidence of the platform configuration and hence trustworthiness. These reports have been investigated widely in the scientific community lately [13]. Also the challenges of scalability [14] and reduction of processing overhead [20] have been targeted. However, especially the challenge of protecting the privacy in terms of user identification and device fingerprinting remain open, though WS-

Attestation [20] and Property-Based Attestation (PBA) [12] may be utilized in that perspective and will be considered as part of our future work.

7 Conclusion

In this paper, we have presented an approach for resolving abstract security policies of multiple domains with the help of semantic knowledge. This policy refinement process finds a set of applicable security mechanisms which matches the abstract security requirements stated by a developer and at the same time complies with the capabilities and restrictions stated by devices. We have presented the structure of the ontologies used to describe devices and security mechanisms, and explained the policy resolution process. In order to verify the correctness of the device descriptions used in the policy resolution process, we have proposed using different assurance techniques based on TPM attestations. While some approaches have been proposed previously on the integration of semantic knowledge and policies, we describe an architecture that integrates access-control and communication policies within a coherent protocol. In addition we consider validating the correctness of semantic knowledge used for policy decisions by means of trusted computing functions. A software architecture realizing the solution presented in this paper has been designed and a prototype including the semantic policy model has been implemented as part of the Hydra middleware. As part of our future work we intend to extend the protocol proposed herein towards a negotiation of quality-of-service parameters based on individual preferences, building the basis for a "self-protecting" system. Further, we will consider techniques for generating proofs of the correctness of semantic information based on Property-Based Attestation.

Acknowledgements The research reported in this paper has been supported by the Hydra EU project (IST-2005-034891).

References

1. Semantic Annotations for WSDL and XML Schema. W3C Recommendation, August 2007.
2. Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *Workshop on Policies for Distributed Systems and Networks (Policy '01)*, pages 18–39, 2001.
3. Rodolfo Ferrini and Elisa Bertino. Supporting RBAC with XACML+OWL. In *14th ACM symposium on Access control models and technologies (SACMAT)*, pages 145–154, 2009.
4. Matthew Horridge, Nick Drummond, John Goodwin, Alan L. Rector, Robert Stevens, and Hai Wang. The manchester owl syntax. In *CEUR Workshop Proceedings*, volume 216 of *OWLED*, 2006.
5. Lalana Kagal, Tim Berners-Lee, Dan Connolly, and Daniel Weitzner. Using semantic web technologies for policy management on the web. In *National Conference on Artificial Intelligence (AAAI)*, July 2006.

6. Torsten Klie, Benjamin Ernst, and Lars Wolf. Automatic policy refinement using owl-s and semantic infrastructure information. In *Proc. 2nd IEEE Int. Workshop on Modelling Autonomic Communications Environments (MACE)*, San Jose, US, October 2007.
7. Lalana Kagal. The rein policy framework for the semantic web, 2006. http://dig.csail.mit.edu/2006/06/rein/.
8. Steffen Lamparter and Sudhir Agarwal. Specification of policies for automatic negotiations of web services. In Lalana Kagal, Tim Finin, and James Hendler, editors, *Semantic Web and Policy Workshop*, pages 99–109, Galway, Ireland, November 2005.
9. TCG MPWG. The TCG mobile trusted module specification. *TCG specification version 0.9 revision*, 1.
10. National Institute of Standards and Technology. Security Requirements for Cryptographic Modules. Federal Information Processing Standards Publication 140-2, 2002.
11. Anand Patwardhan, Vladimir Korolev, Lalana Kagal, and Anupam Joshi. Enforcing Policies in Pervasive Environments. In *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, August 2004.
12. A.R. Sadeghi and C. Stüble. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *Workshop on New security paradigms*, pages 67–77, 2004.
13. R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. *Proc. of the 13th USENIX Security Symposium*, pages 223–238, 2004.
14. Frederic Stumpf, Andreas Fuchs, Stefan Katzenbeisser, and Claudia Eckert. Improving the scalability of platform attestation. In *Workshop on Scalable Trusted Computing (ACM STC '08)*, pages 1–10, Fairfax, USA, October 31 2008. ACM Press.
15. Alessandra Toninelli, Rebecca Montanari, Lalana Kagal, and Ora Lassila. Proteus: A semantic context-aware adaptive policy model. In *IEEE 2007 International Workshop on Policies for Distributed Systems and Networks (POLICY)*, Bologna, Italy, June 2007. IEEE Computer Society Press.
16. Gianluca Tonti, Jeffrey M. Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej Uszok. Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder. In *The Semantic Web (ISWC '03)*, volume 2870/2003, pages 419–437, 2003.
17. Kevin Twidle, Naranker Dulay, Emil Lupu, and Morris Sloman. Ponder2: A Policy System for Autonomous Pervasive Environments. In *The Fifth International Conference on Autonomic and Autonomous Systems (ICAS '09)*, April 2009.
18. Andrzej Uszok and Jeff Bradshaw. Kaos policies for web services. W3C Workshop on Constraints and Capabilities for Web Services, October 2004.
19. Kunal Verma, Rama Akkiraju, and Richard Goodwin. Semantic matching of web service policies. *Proceedings of the Second Workshop on SDWP*, 2005.
20. S. Yoshihama, T. Ebringer, M. Nakamura, S. Munetoh, and H. Maruyama. WS-attestation: efficient and fine-grained remote attestation on Web services. *International Conference on Web Services (ICWS '05)*, page 750, 2005.
21. Weishan Zhang, Julian Schütte, Mads Ingstrup, and Klaus M. Hansen. A Genetic Algorithms-based approach for Optimized Self-protection in a Pervasive Service Middleware. In *International Joint Conference on Service Oriented Computing (ICSoC '09)*, November 2009.