A Policy Specification Language for Composite Services

Muhammad Asim[a], Susana González Zarzosa[b], Qi Shi[a] and Bo Zhou[a,]
[a]School of Computing & Mathematical Sciences, Liverpool John Moores University
email:{m.asim, q.shi, b.zhou} @ljmu.ac.uk

[b]Atos Spain S.A., Madrid, Spain
email: susana.gzarzosa@atos.net

***Abstract*— Creating complex systems by combining smaller component services is one of the fundamental concepts in Service Oriented Architecture. Service compositions are built by combining loosely coupled services that are, usually, offered and operated by different service providers. While this approach offers several benefits, it makes the implementation and representation of the security requirements difficult. This paper reviews several requirement specification languages and analyses their suitability for composite services. A set of requirements is identified and a comparison between different specification languages is presented along with some conclusion on the suitability of each language in expressing security requirements for composite services.**

Keywords- policy languages; composite services; security; service-oriented computing

I. INTRODUCTION

Service-based applications are a new class of software systems that allow enterprises to offer their software systems as services by following the principal of *Service Oriented Architectures* (SOA). A service itself is a unit that offers certain functionality. If no single service can satisfy the functionality required by the user, then SOA allows multiple services to be composed to form a larger application in order to fulfil the user requirements. A SOA platform provides a foundation for modelling and composing multiple services in an ad hoc manner [1] [2].

Aniketos is an EU research project [3] that addresses trustworthy and secure service compositions with run-time monitoring and adaptation of services. One important task in the Aniketos project is to choose a specification language that is able to express security requirements, properties or policies for composite services. Also, it is a suitable policy language to specify what we need to monitor at runtime. Besides, the specifications should be able to be generated by both humans and software. In general, this language should serve to other purposes as well, e.g., it should specify the security requirements for a service (either desired by a consumer or advertised by a service provider). Naturally, we may use one language for requirements specification and another one for monitoring these requirements, but then there is a need for a transformation engine. Thus, one language for both purposes significantly reduces the complexity.

This paper reviews several security requirement specification languages and analyses their suitability for a modern, flexible, secure service platform. A set of requirements is identified and a comparison between different specification languages is presented along with some conclusion on the suitability of each language in expressing security requirements for services that are composite in nature. We use the Aniketos Platform as a reference point to discuss these languages and there suitability for composite services.

The paper is organized as follows: The next section presents the requirements for a specification language. Specification languages are discussed in Section 3. The suitability of the language ConSpec for the project Aniketos is discussed in Section 4. Section 5 presents the conclusion on specification language choice.

II. SPECIFICATION LANGUAGES REQUIREMENTS

In the context of the Aniketos Platform development, we are mainly looking for specification languages which are able to address the following requirements. The selected list of requirements is a result of analysis that has been carried out on more than fifty scenarios coming from three different domains (air traffic management, telecommunication and e-government) [4] [5].

- *(Rec-01) Cross-composite- The language for contract specification shall be able to express the properties for a hierarchical service. It should support both atomic and composite services.* Complex services often have a complex hierarchical structure. Thus, the contract specification language should be able to describe the desired and provided properties, taking into account that some parts of the service are provided by the services at the lower end of the hierarchy.

- *(Rec-02) Generalizable and Unambiguous- The language for contract specification shall be general enough to express requirements of various kinds.* Security requirements, which one would like to express with the language could be very different. These requirements may include presence of some countermeasures, various access control policies, well-known security properties, or a numerical security target (e.g., Risk level).

- *(Rec-03) Intelligible- There shall be no difference whether the set of policies is created by a human or*

software. The language should be easily interpretable both by humans and through automated means.

TABLE 1: CONTRACT/POLICY SPECIFICATION LANGUAGE REQUIREMENTS

Rec-01-01	The specification language should be able to express the scope of the policies to determine if it applies to a single or multiple executions of the same service.
Rec-02-01	The specification language should have unambiguous and restricted semantics to improve its clarity and simplicity.
Rec-02-02	The specification language should be able to represent state transitions.
Rec-03-01	The specifications should be able to be developed for integration with computer programs, i.e., Java.
Rec-03-02	The learning of the language should not require too much technical training in order to be able to express new requirements, properties or policies.

We could make these requirements even more specific as listed in Table 1.

III. SPECIFICATION LANGUAGES

In the literature, we can find a huge amount of work on policy specification languages as well as several taxonomies of these languages. We will start discussing some of these existing classifications that will help us in the search for a suitable specification language to be used in Aniketos and to choose the main potential candidate languages.

First, Bonatti et al. [6] differentiate the following groups of rule-based policy specifications performed by the REWERSE (Reasoning on the Web with Rules and Semantics) Project [7]. They differentiate the following groups of rule-based policy specifications:

1) *Logic-based policy languages:* focused on those languages with unambiguous semantics that enhance clarity, simplicity and modularity. The main advantages of these logic languages are: (i) they are very suitable for validation and verification; (ii) their declarative nature makes them expressive enough to formulate a wide range of policies with simplicity.In this category we find for example the eXtensible Access Control Markup Language (XACML) that is the standard for policy specification developed by the OASIS consortium.
2) *Action languages:* including those languages that can be used to represent actions, changes and their effects. Most of them describe dynamic situations according to a so-called state-action model. One of the most popular logic-based approaches of action languages is EventCalculus.
3) *Business rules:* based on those languages that are more concerned in the formulation of statements about how a business must be done or in other words, the guidelines and restrictions that apply to states and processes in an organization. They distinguish here three categories of rules: reaction rules ("ON event IF condition is fulfilled THEN perform action"), derivation rules (each rule expresses the knowledge that if one set of statements happens to be true, then some other set of statements must also be or become true) and integrity constraints (assertions that must be satisfied in all evolving states). One of the more relevant business process description languages is the Business Process Execution Language for Web Services (BPEL4WS).
4) *Controlled natural languages:* which are defined like "*subsets of natural languages whose grammars and dictionaries have been restricted in order to reduce or eliminate both ambiguity and complexity*". Therefore, this category would be included in what it is called "semantic languages". An example is PROTUNE that is the name of the policy language and meta-language developed in the REWERSE Project.

To summarize, from the analysis performed by REWERSE, we select the following potential languages for a further study taking into consideration the requirements indicated above for Aniketos:

- XACML
- Event Calculus
- Web Service Description Language (WSDL) /BPEL4WS
- PROTUNE (and other relevant semantic web languages)

In the PrimeLife Project [8], they define three types of policies that they considered important parts of any privacy policy that have to be covered by any policy language: (i) data handling; (ii) access control; and (iii) trust policies. The languages selected from the PrimeLife study are:

- XACML
- The Platform for Privacy Preferences (P3P)

Finally, we are going to analyse the Contract Specification Language (ConSpec) that is an automata-based policy specification language presented in the literature [9] as a potential language for specifying both policies and contracts in various security enforcement related tasks of the application lifecycle.

In the next subsections, we discuss in more detail each one of the selected policy languages that we have considered as candidates in Aniketos.

A. *eXtensible Access Control Markup Language*

eXtensible Access Control Markup Language (XACML) [22] is an Extensible Markup Language (XML) based language used to express and interchange access control policies. It is designed to express authorization policies in XML against objects that are themselves identified in XML. XACML is a general purpose policy language and it can be used to protect any resource type (i.e., not just data), but it is difficult to write XACML policies and even more difficult to reason over (i.e., it is unsatisfactory regarding requirement Rec-03-02). Therefore we could use this language in Aniketos project since it would allow encoding most of security properties that will be included into the Contract (requirement Rec-01), but we would need to "misuse" the constraint part of XACML policies since XACML is tailored towards Access Control policies.

B. *Event Calculus*

Event Calculus (EC) [10] is a first-order temporal logical language for representing actions and their effects that can be used to specify properties of dynamic systems, which change over time. Such properties are specified in terms of events and fluents. An event in EC is something that occurs at a specific instance of time (e.g., invocation of an operation) and may change the state of a system. Fluents are conditions regarding the state of a system, which are initiated and terminated by events. A fluent may, for example, signify that a specific system variable has a particular value at a specific instance of time or that a specific relation between two objects holds.

ecXML [11] is an XML formalisation of the Event Calculus that is used to describe how a contract's state evolves, according to events that are described in the contract. The main advantage of this language for Aniketos is that it is very suitable for runtime monitoring and can be used to represent properties, policies and contracts in a dynamic environment (Rec-01). But it is more oriented towards states and actions than services, and the syntax could become too complicated for compound services and expression of hierarchies (Rec-02). Moreover, it would require a big effort to accomplish requirement Rec-03 to automate the generation and runtime monitoring of these rules in Java code.

C. *Web Service Definition Language / Business Process Execution Language for Web Services*

The WSDL [12] is the World Wide Web Consortium (W3C) standard language for web service descriptions. It is an XML format used to create a flexible Service Level Agreement (SLA) for web services defining mutual understandings and expectations of a service between the service provider and service consumers. It uses a very limited syntax that defines services as collections of network endpoints or ports.

The Business Process Execution Language for Web Services (BPEL4WS) [13] is a language used for specifying business process behaviour based on Web Services, which was created to overcome the limits of WSDL. It allows building definitions of a business process (that can be either an executable itself or a business protocol) where both the process and its partners are modelled as WSDL services. The language is layered on top of several XML specifications (WSDL 1.1, XML Schema 1.0, and XPath1.0) but makes no use of semantic information.

This language is a service-oriented composition language that forms the base of Aniketos, but we want to express also security properties and trustworthiness (Rec-01). Consequently we need something that provides more information than BPEL4WS.

D. *PROVisional TrUst Negotiation*

PROvisional TrUst Negotiation (PROTUNE) [14] is a natural language for the specification of rule based policies on the semantic web defined by REWERSE [7]. It is a logic-based and declarative policy language that includes logical axioms to constrain the behaviour and how the web resources must be used. But the main feature of PROTUNE that makes it different from the previously discussed languages is that it is a semantic web language.

The semantic web languages are developed to allow intelligent agents in the semantic web to reason and make decisions policy-driven based on the knowledge it is provided by the semantics. Therefore, one of the main advantages of these semantic web languages for Aniketos is that it facilitates greater automatic machine interpretability of conditions, taking decisions and performing tasks (covering the requirement Rec-03). Besides, this kind of language provides an enormous expressivity and can be used to represent complex knowledge in a distributed environment and support classification in hierarchies (requirements Rec-01 and Rec-02). But this last feature is also a big drawback (high complexity) due to which it cannot be considered in the project Aniketos. Reasoning with a semantic web language is difficult and it requires a well-defined semantic that should be developed specifically for Aniketos. Furthermore, its high expressiveness can lead to non-standard formalism and sometimes to complexity in the reasoning.

The semantic web languages standardized by the W3C are (i) Resource Description Framework (RDF) [15] and (ii) Web Ontology Language (OWL) [16].

OWL includes more vocabulary and consequently extends the facilities offered by XML, RDF and RDF Schema (RDF-S) for expressing meaning and semantics

what makes it easier to represent machine interpretable content on the Web. In turn, OWL provides three increasingly expressive sublanguages: OWL Lite, OWL Description Logic (OWL DL), and OWL Full.

In the case of PROTUNE, the syntax is based on normal logic program rules. Finally, we can take into consideration two prominent semantic web languages based on OWL, which appear in much of the literature: Rei and KAoS [17][18]. Rei is a policy language based on OWL-Lite that includes logic-like variables to provide more flexibility in the specification of relationships that are not possible in OWL. For example, it is possible to define individual and group based policies that could be useful in large scale distributed environments for saving time. They are associated with agents, called subjects, by means of the *has* construct: *has(Subject, PolicyObject).*

KAoS is another policy language based on OWL with the following distinguishing features: (i) it does not assume that the policies are applied in homogeneous components: (ii) it supports dynamic runtime policy changes; (iii) the framework is extensible to different execution platforms; (iv) the KAoS framework is intended to be robust and adaptable in continuing to manage and enforce the policy of any combination of components.

E. Platform for Privacy Preferences

The P3P [19], published by the W3C, enables web sites to express their privacy practices in a standard format that can be retrieved automatically and interpreted easily by user agents. P3P user agents will allow users to be informed of site practices (in both machine and human readable formats) and to automate decision-making based on these practices when appropriate. But this option has been discarded for Aniketos because a report [20] on the assessment of P3P and Internet privacy finds that P3P fails to comply with baseline standards for privacy protection. It is a complex and confusing protocol that also fails to address many of the privacy problems. The report concludes that there is little evidence to support the industry claim that P3P will improve user privacy citing the widely accepted Fair Information Practices.

F. ConSpec Language

The ConSpec [9] language with its syntax shown in Fig. 1 is strongly inspired by the policy specification language PSLang, which was developed by Erlingsson and Schneider in [21] for runtime monitoring. However, even though ConSpec is a more restricted language than PSLang, it is expressive enough to write policies referring to multiple executions of the same application, as well as to executions of all applications of a system, in addition to policies about a single execution of the application and of a certain class object lifetime according to the *scope* of the policy.

Effectively, a ConSpec contract specifies a set of guards each with an associated set of reactions. A guard is defined as a method prototype. A reaction is a set of expressions specifying state changes, where the left hand side specifies the state before and the right hand side the state afterwards. Whenever the guard method is called in the code, the state expression is checked and, if the left hand side of the expression matches the current state, the right hand side expression is applied to update it. In the event that the state fails to match any of the left hand side expressions, the contract is considered to be violated. The following example states that, once the file *Secret.dat* has been opened, plaintext socket connections can no longer be used. Note that the skip keyword is used to represent no state change.

```
BEFORE File.Open(String path) PERFORM
     path == "Secret.dat" -> {private = true;}

BEFORE Socket.Send(String sd, String data)
PERFORM
     private == false -> {skip;}
```

Figure 1. Syntax of ConSpec

One of the attractive features of this approach is that the use of a finite state machine coupled with guards defined against explicit methods means that the ConSpec script defines not just the policy but also the means to identify it. However, we can also see from the above example that ConSpec was originally developed for use with single isolated pieces of software, written in a specific language (in the case of the Aniketos project, this is Java). This impacts the cross-composite requirements.

The language has therefore been extended to support composed services [1][2]. This can be achieved in one of two ways. First, a single ConSpec file can be defined to apply across a set of composed services. This requires there to be a single centrally managed finite automata state machine that all guard events refer back to. In this case, rather than specifying methods for the guards, a service identifier must also be specified. Service identifiers can also be passed as a parameter to the reaction, so that the state change can be predicated on service properties as well. In this case, earlier guards that identify particular functionality in a particular service can be used to correlate with guards identifying different functionality at a later time. It also allows more flexibility in defining contracts, since ideally the contract should be independent of the service composition that its applied to. Second, each service can be given its own ConSpec file. In this case there's effectively an automaton applied to each service. However, there needs to be correlation between the services, so a further central

automaton is needed at the composition level. State changes at the service level generate events, which are then matched against guards at the composition level which potentially update the central automaton. An attractive feature of using finite automata is that they are themselves compositional: this arrangement is equivalent to a finite automaton applied across all services. This allows cross-composition.

The policies written in ConSpec are easily interpretable by humans. It has a comparatively simple semantics, and is simple to learn. ConSpec is an automata-based language. Although this feature slightly reduces its expressiveness (in comparison with its predecessor PSLan [21], or other declarative languages as EventCalculus [10], XACML [22], PROTUNE [14], etc.), it allows automatic reasoning on it. For example, in the project we needed to check that requirements desired by a consumer could be fulfilled by a service provider. Furthermore, it is simple to define a policy decision point for monitoring purposes if an automaton is available. Finally, ConSpec defines different scopes of its application. Thus, we may define a policy for a single execution of a service or multiple executions. Overall, ConSpec provides an unambiguous, cross-composite and intelligible approach, which makes it a more suitable specification language for composite services.

IV. CONSPEC IN THE ANIKETOS PROJECT

Based on the above analysis, we selected the ConSpec language as a specification language for the Aniketos platform and extended it (as discussed above) to support the composite nature of services. In the scope of the Aniketos project, we have created a tool, which provides a graphical user interface for making and changing ConSpec policies. The tool is called a ConSpec Editor illustrated in Fig. 2.

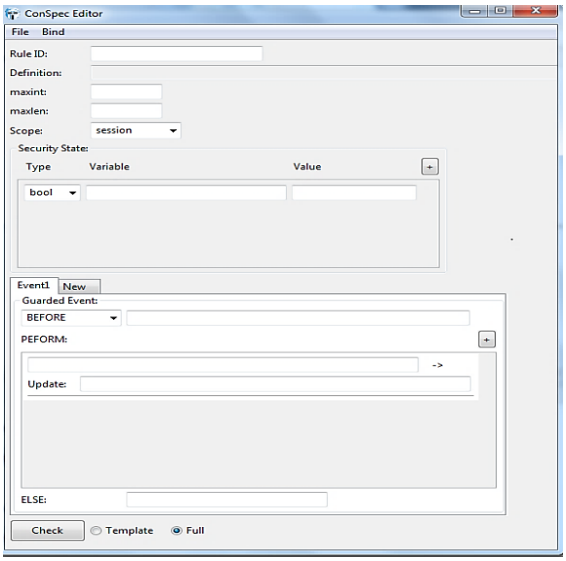

Figure 2. ConSpec editor

ConSpec policies can be created with the ConSpec Editor without knowing the ConSpec language. As an example, the ConSpec policies are used by a monitoring module developed as a part of the Aniketos project. The monitoring module is responsible for the runtime monitoring of a service to ensure that the service behaves in compliance with a pre-defined security policy. For more details about the monitoring framework, see [2].

Consider the following example where a service designer creates a travel booking composition that consists of several tasks, such as ordering, booking hotel, booking flight, payment and invoice, and each task is performed by a component service. The service designer might want that the payment service component should only be invoked when it has a trustworthiness value ≥ 90%. This requirement could easily be specified using the ConSpec language as shown in Fig. 3.

```
RULE ID Trustworthiness
SECURITY STATE
    String ServiceID=Payment;
   int trust_threshold = 90;
   /* assume trustworthiness is in [0%,..., 100%]*/

BEFORE invoke (serviceID)
PERFORM
   (eval_Trust(serviceID) >= trust_threshold) -> skip
                                  condition1 -> update
```

Figure 3. ConSpec policy example 1

The monitoring module in adherence to the policy monitors services to ensure that only a payment service with trustworthiness value ≥ 90% is used. In another example, where a service designer imposes the separation of duty constraint for a particular service composition, i.e., both service A and service B should be offered by different providers.

```
RULE ID SoD_Goal
SECURITY STATE
string serviceProvider = _;
string guardedTask1 = ServiceA;
string guardedTask2 = ServiceB;

BEFORE v#service.start
(string id, string type, int time, int date,
   string provider) PERFORM

(id == guardedTask1 || id == guardedTask2) &&
   serviceProvider == "_" -> {serviceProvider =
   provider; }

(id == guardedTask1 || id == guardedTask2) &&
   !(serviceProvider == "_") && !(provider ==
   serviceProvider) -> {skip}

   !(id == guardedTask1) && !(id == guardedTask2)
-> {skip}
```

Figure 4 . ConSpec policy example 2

The requirement for the above example can be specified in ConSpec as illustrated in Fig. 4.

V. CONCLUSION ON SPECIFICATION LANGUAGE CHOICE

The different languages discussed here exhibit interesting properties in relation to their suitability for composite service. However, comparing the requirements and needs that the Aniketos platform requires to express security policies and the previous descriptions of the different languages, we can conclude that ConSpec is the best solution for the main reasons summarized below:

- It is extended to offer unambiguous, cross-composite solutions with important elements of generalizability for composite services.
- It is developed as a language for representing security relevant behaviours of an application in terms of Java calls, which allows the rules to be generated and checked at runtime also by software or security automata.
- A policy written in the ConSpec language is easily interpretable by humans and the simplicity of the language allows a comparatively simple semantics and a reasonably fast learning curve.
- Although ConSpec does not allow any arbitrary type to represent the security state of a service, it includes tags for expressing security requirements in different stages of the application life cycle. It makes it possible to indicate constraints that can be applied to multiple executions of a service, as well as interactions with other services.

TABLE 2. MATCHING OF SPECIFICATION LANGUAGES TO REQUIREMENTS

Requirement	XACML	EC	WSDL/ BPEL4 WS	PROTUNE	OWL-based (Rei/KAoS)	ConSpec
Rec-01	X	X	X	X	X	X
Rec-01-01	X	X				X
Rec-02	X			X	X	X
Rec-02-01			X			X
Rec-02-02	X			X	X	X
Rec-03	X		X	X	X	X
Rec-03-01	X		X	X	X	X
Rec-03-02			X			X

Table 2 summarizes the requirements that are covered by each of the different languages presented above.

REFERENCES

[1] D. Llewellyn-Jones, M. Asim, Q. Shi, and M. Merabti, "Requirements for Composite Security Pattern Specification," Second International Workshop on Cyberpatterns 2013: Unifying Design Patterns with Security, Attack and Forensic Patterns, Abingdon, UK., 2013, pp. 70-77.

[2] M. Asim, D. Llewellyn-Jones, B. Lempereur, B. Zhou, Q. Shi, and M. Merabti, "Event Driven Monitoring of Composite Services,"The 5th ASE/IEEE International Conference on Information Privacy, Security, Risk, Washington D.C., USA, Sep 2013, pp. 550-557.

[3] Aniketos (Secure and Trustworthy Composite Services), http://www.aniketos.eu, retrieved: April, 2015.

[4] Aniketos Consortium, Deliverable D2.3: Models and methodologies for implementing Security-by-Contract for services, 2012,http://www.aniketos.eu/content/deliverables, retrieved: April, 2015.

[5] Aniketos Consortium, Deliverable D1.2: First Aniketos architecture and requirements specification, 2012, http://www.aniketos.eu/content/deliverables, retrieved: April, 2015.

[6] P. A. Bonatti et al. Rule-based Policy Specification: State of the Art and Future Work. Technical Report IST506779/Naples/I2-D1/D/PU/b1, Reasoning on the Web with Rules and Semantics, REWERSE, August 31st, 2004.

[7] REWERSE: Reasoning on the Web with Rules and Semantics, http://rewerse.net/, Retrieved: March, 2015.

[8] PrimeLife (Privacy and Identity Management in Europe for Life), Deliverable D5.1.1: Final requirements and state-of-the-art for next generation policies, August 2009, http://primelife.ercim.eu/, retrieved: March, 2015.

[9] I. Aktug and K. Naliuka, "ConSpec: A Formal Language for Policy Specification.", In Proceedings of the First International Workshop on Run Time Enforcement for Mobile and Distributed Systems , 2007, pp. 2-12.

[10] M. P. Shanahan, The Event Calculus Explained, in Artificial Intelligence Today, eds. M. J. Wooldridge and M. Veloso, Springer-Verlag Lecture Notes in Artificial Intelligence no. 1600, Springer-Verlag, pages 409-430, 1999.

[11] A. D.H. Farrell, M. J Sergot, M. Salle, and C. Bartolini, "Using the Event Calculus for the Performance Monitoring of Service-Level Agreements for Utility Computing" First IEEE International Workshop on Electronic Contracting (WEC'04), 2004

[12] Web Service Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl, retrieved: March, 2015.

[13] Business Process Execution Language for Web Services Version 1.1, http://public.dhe.ibm.com/software/dw/specs/ws-bpel/ws-bpel.pdf, 5 may 2003, retrieved: March, 2015.

[14] P.A. Bonatti, J.L. De Coi, D. Olmedilla, and L.Sauro, "PROTUNE: A Rule-based PROvisional TrUst Negotiation Framework", 2010.

[15] Resource Description Framework (RDF), http://www.w3.org/2001/sw/wiki/RDF, retrieved: Feb, 2015.

[16] OWL Web Ontology Language Overview, http://www.w3.org/TR/owl-features/, WC3 Recommendation February 2004, retrieved: Feb, 2015.

[17] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri and A. Uszok, "Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei and Ponder," In Proceedings of the 2nd International Semantic Web Conference (ISWC2003). Springer-Verlag, 2003.

[18] L. Kagal, T. Finin and A. Joshi, "Declarative Policies for Describing Web Service Capabilities and Constraints", Proceedings of the 6th international conference on E-Commerce and Web Technologies, 2005

[19] Platform for Privacy Preferences Project (P3P), http://www.w3.org/P3P, retrieved: Feb, 2015.

[20] Electronic Privacy Information Center and Junkbusters, "Pretty Poor Privacy: An Assesment of P3P and Internet Privacy" (June 2000), http://epic.org/reports/prettypoorprivacy.html, retrieved: Feb, 2015.

[21] U. Erlingsson. The inlined reference monitor approach to security policy enforcement. PhD thesis, Department of Computer Science, Cornell University, 2004.

[22] eXtensible Access Control Markup Language (XACML) Version 3.0 (http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf), retrieved: Feb, 2015.