

Graphical Interface for Ontology Mapping with Application to Access Control

Michał Drozdowicz¹(✉), Motasem Alwazir¹, Maria Ganzha^{1,2},
and Marcin Paprzycki¹

¹ Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland
{michal.drozdowicz,motasem.alwazir,maria.ganzha,
marcin.paprzycki}@ibspan.waw.pl

² Department of Mathematics and Information Sciences,
Warsaw University of Technology, Warsaw, Poland

Abstract. Proliferation of smart, connected devices brings new challenges to data access and privacy control. Fine grained access control policies are typically complex, hard to maintain and tightly bound to the internal structure of the processed information. We thus discuss how semantic inference can be used together with an intuitive ontology management tool to ease the management of Attribute Based Access Control policies, even by users not experienced with semantic technologies.

1 Introduction

Rise of the Internet of Things (IoT), results in growing interest in data access control. Obviously, assuring privacy and security of data is of utmost importance. However, creation of complex ecosystems results in need of establishing, which data is going to be exposed, to which stakeholder, why, when, for how long, etc. Recently (see, [2,3]), we have proposed an Attribute Based Access Control system utilizing semantic reasoning to enrich available information, when making access control decisions (SXACML). However, even IT professionals have limited knowledge of semantic technologies. Therefore, we consider how an ontology non-expert can effectively define and/or manage an ontology within the Policy Administration Point.

To this effect, Sect. 2 introduces the SXACML system, and provides a use case scenario. In Sect. 3, we outline the state-of-the-art in ontology modeling tools. Section 4, describes OntoPlay, a module that provides needed ontology management capabilities. Next, in Sect. 5, we outline how OntoPlay has been integrated with the SXACML.

2 SXACML

The eXtensible Access Control Markup Language (XACML; [1]) allows implementation of the Attribute Based Access Control (ABAC; [7]) mechanisms. In the XACML, attributes are grouped into four categories:

- *Subject* – the entity (possibly a person) requesting access,
- *Resource* – the entity, access to which is under control,
- *Action* – that the *Subject* requests to be performed on the *Resource*,
- *Environment* – other attributes that bring additional context.

The XACML specification defines also a reference architecture, comprised of:

- *Policy Enforcement Point* (PEP) – responsible for actual enabling or preventing access to the resource. It also coordinates the execution of, so called, *Obligations* – additional operations that should be performed when a decision has been made (e.g. logging the request for auditing purposes).
- *Policy Information Point* (PIP) – a source of values of attributes. Commonly handled by data stores, such as relational databases or LDAP directories.
- *Context Handler* – converts requests and responses between native formats and the XACML canonical representation and coordinates, with the PIP, gathering of the required attribute values.
- *Policy Decision Point* (PDP) – evaluates policies and issues the final authorization decision.
- *Policy Administration Point* (PAP) – defines, stores and manages policies.

In this context, in [2, 3], we introduced a semantics-driven implementation of the Policy Information Point. There, request attributes, combined with information stored in an ontology, allowed inferring additional data, necessary for access control decision. Advantages of such approach include, but are not limited to:

1. Simplified policies – information common to multiple policies can be extracted into the ontology.
2. Better support for Role Based Access Control.
3. More flexibility in defining relationships between concepts in policies.
4. Possibility of greater interoperability, by semantic mapping of disparate concepts in the request and in policies.

However, note that, while the first two points can be solved using only an ontology describing the domain of the organization and, perhaps, a simple static mapping of XACML terms into ontology concepts; the last points require a more robust and dynamic solution, allowing administrators to manage the mapping of concepts. Hence, we will now focus on defining additional relationships between domain concepts, and providing interoperability in the access control context.

Use case scenario. To put our work in a real-world context, consider a somewhat simplified example originating from the INTER-IoT project¹. Let us consider controlling access to facilities of a cargo port, and assume that policies, stating which persons and vehicles may access the port premises, have been defined using a Policy Administration Point. One of them states that drivers (and trucks) employed by Globex Corporation can access the area. We assume

¹ <http://www.inter-iot-project.eu/>.

that authentication mechanisms are in place, to verify that information provided by the drivers/trucks is correct and valid. Now, consider that one day transport of goods is handled by a subcontractor of Globex Corp – Stark Transport. Here, subcontracted drivers/trucks should also be granted access. This scenario involves two aspects that necessitate additional processing of request attributes in order to make the authorization decision:

1. Relationship between Stark Transport and Globex Corp cannot be stored in access policies (as Globex hired Stark “incidentally” to deal with shortage of trucks, and *only then* they should be granted access).
2. Stark Transport uses a slightly different terminology than Globex Corp and thus concepts from the XACML access request have to be mapped to these used in the port’s policies.

3 Existing Ontology Modelling Tools

Let us now assume that semantic technologies are to be used in this scenario. Hence, the system should facilitate defining relationships between concepts related to access control. Moreover, this function should be accessible to users unfamiliar with ontology modeling. Existing tools for defining concepts and relations between them, manipulation and searching of data represented as an ontology; can be divided into two groups: (i) ontology editors, and (ii) SPARQL query editors.

Ontology editors. Ontology editors are integrated development environment (IDE) for creating new and managing existing ontologies. Their GUI is provided as a desktop, or a web, application. The World Wide Web Consortium (W3C)², lists 12 Ontology Editors³. In addition to ontology creation and modification, they may provide extra capabilities. For instance, *Protégé*⁴, *WebProtege*⁵, *Cognitum*⁶ and *OWLGrEd*⁷ display a graphical representation of concepts and relations between them (as defined in the ontology). They also support defining SWRL Rules, use of reasoners, etc. Those editors (or others, see [5, 8]) are very useful for ontology engineers or developers during the stage of designing, updating and utilizing an ontology but they are not suitable for non ontology experts, even when it comes to simple tasks, such as creating an instance of a class. For example, in *Protégé*, which, according to its website, is trusted by more than 300,000 users, such simple tasks cannot be performed without knowing the elements of an ontology. It also requires the user to understand the structure of the loaded ontology to be able to add consistent taxonomies. In other words,

² <https://www.w3.org>.

³ https://www.w3.org/wiki/Ontology_editors.

⁴ <http://protege.stanford.edu/>.

⁵ <http://protege.stanford.edu/products.php#web-protege>.

⁶ <http://www.cognitum.eu/semantics/FluentEditor/>.

⁷ <http://owlgred.lumii.lv/>.

Protégé does not facilitate easy management and mapping of XACML concepts in the ontology, by administrators with minimal knowledge of ontologies.

SPARQL editors. Tools included in this category, e.g. *SPARQL Editor*⁸, *YASQE*⁹ and *Virtuoso*¹⁰ allow writing and executing SPARQL queries, to search and retrieve data from an ontology. They provide a simple user interface, for user to select the ontology she wants to query, and a text area to write the queries. A more user friendly way to write SPARQL queries is a GUI designed to support building them; found in tools like: Visual *SPARQL BUILDER*¹¹ and *Gruff*¹². Nevertheless, no simple way exists to use such editors by persons who do not know SPARQL. A work around would be to prepare queries, which user might need for mapping XACML concepts. However, this would mean that the code would have to be changed whenever the ontology is modified. Furthermore, available queries would work only with a given ontology (could not be reused).

Summarizing, a number of user interfaces to semantically demarcated information exists. However, none of them could serve as a lightweight front-end, allowing creation of nested descriptions of OWL individuals and class expressions. Moreover, all of them require the user to have understanding of ontologies.

4 OntoPlay

Let us start by observing that users of, for example, database-centered applications do not need to know about databases, tables and relations. The same is needed for ontology-driven systems. Here, front-end/GUI should be straightforward and should “hide” use of ontologies. It should guide “ontology-illiterate” users to define individuals and/or classes expressions to query the ontology. Moreover, in an “Open World assumption”, one might need to change the ontology, to model more concepts, or add new properties. To avoid changing the system whenever vocabularies are modified, the front-end should be ontology-agnostic (i.e. ontology change should result in a minimum changes to its code). The GUI should be rendered dynamically, based on the underlying ontology. For example, consider a system that uses the Pizza ontology¹³. The GUI should allow people, who do not understand semantics, to add a new instance of the class Pizza. It should also make it easy to find a pizza from Italy.

The OntoPlay (see, [4]) is a web-based front-end plugin satisfying these requirements. Its GUI contains the *condition builder*, which allows “any user” to create class conditions, or individuals, translated to OWL expressions, and merge them with the underlying ontology. In [4], we showed how it helps dealing with various scenarios involving ontologies. Let us stress that the OntoPlay operates with any OWL ontology, if it is syntactically and semantically correct.

⁸ <http://sparql.carsten.io>.

⁹ <http://yasqe.yasgui.org/>.

¹⁰ <https://dbpedia.org/sparql>.

¹¹ <http://leipert.github.io/vsb>.

¹² <http://franz.com/agraph/gruff>.

¹³ <http://protege.stanford.edu/ontologies/pizza/pizza.owl>.

Recently, the level of OWL expressiveness, in the OntoPlay, had been increased through handling relations, and individuals using *annotation* properties¹⁴ (OWL 2.0 entities). Here, user is to be able to use annotations “without knowing it”. Additional interfaces were put in the OntoPlay as well.

The main components of the new OntoPlay are the same as before: Client, Server and Gateway [4]. The Gateway was not changed. On the client side, the AngularJS¹⁵ was used to allow data binding between HTML elements and models defined using JavaScript. This was needed in the condition builder interface, for example, when user changes data property to object property, or creates nested class expressions. On the server side, using the functionality and the routing of the Play framework, web services were defined, to read the structure (or the data) in the ontology and return it in JavaScript Object Notation (JSON). Combining Angular with web services made the user interface more dynamic and efficient from a programming point of view. Examples of defined services are: get properties of a class, get range type of a property, get individuals in the range of a property, etc. In addition, the structure of the JSON build has been changed to include the *annotation* properties for the new created entity, and relations defined within it. Let us now enumerate new interfaces available in the OntoPlay and the functionality of each one of them.

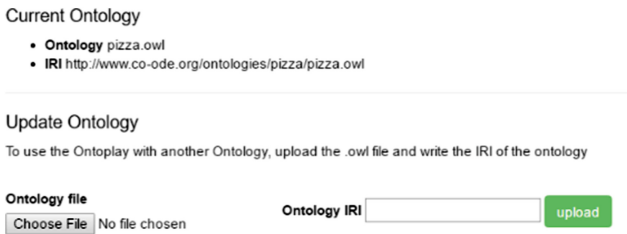


Fig. 1. Administrating the ontology connected to the OntoPlay

Managing ontology. A simple interface which allows the “system administrator” to manage the current ontology. The importance of this interface is that the Java code does not have to be changed when a different ontology is to be used. When using this interface (see, Fig. 1), the administrator can see the current ontology connected to the OntoPlay and update/replace it, if needed, by uploading the OWL file with the new ontology and writing its *IRI*.

Class instances. A web page displaying all individuals of a class, in the underlying ontology. Just like a system displaying data from the database, it allows users to perform operations: view, add, edit and delete data, defined in the ontology knowledge base. Figure 2 shows this view for the Country class defined in the Pizza Ontology.

¹⁴ https://www.w3.org/TR/owl2-syntax/#Annotation_Properties.

¹⁵ <https://angularjs.org/>.

Country

Uri: <http://www.co-ode.org/ontologies/pizza/pizza.owl#Country>

Super Class: [DomainConcept](#)

Properties: 2

Instances: 5

[Create New Individual](#)

5 Individuals
















	Local Name	Uri
  	France	http://www.co-ode.org/ontologies/pizza/pizza.owl#France
  	Italy	http://www.co-ode.org/ontologies/pizza/pizza.owl#Italy
  	England	http://www.co-ode.org/ontologies/pizza/pizza.owl#England
  	America	http://www.co-ode.org/ontologies/pizza/pizza.owl#America
  	Germany	http://www.co-ode.org/ontologies/pizza/pizza.owl#Germany

Fig. 2. Class interface displaying Countries defined in the ontology

Annotation properties. Observe that OWL does not put any constraints on the domain (it can be any IRI) and the range (any IRI or data literal) of the *Annotation Properties*. Some annotation properties are predefined by OWL (e.g. `owl:versionInfo`, `rdfs:label`, `rdfs:comment`, `rdfs:seeAlso` and `rdfs:isDefinedBy`), while custom ones can be added to an ontology. They can be added for several reasons. One of them is to define N-ary relations¹⁶. For example, one wants to establish the “amount” of a pizza topping. For example “a little” (or “a lot”) of black olive topping). To satisfy this, *withAmount* can be introduced as an annotation, to describe the relation `hasTopping`.

Predefined, and custom, annotation properties can be used to describe classes, data and object properties, individuals and even annotation properties. An ontology engineer may be able to use annotation properties in the context they were defined, meant for, without having constraints on the domain and range. On the other hand, user assumed here, would not know how to do it “the right way”. That is why we have introduced a mechanism to limit the annotation properties and make them available only where they are meant to be. To do that, a new admin panel (Fig. 3) has been developed to allow “system administrator” to restrict, which annotation property users can use to describe an individual, or a relation within an individual. Currently, *OntoPlay* supports only a data literal, as a value for any annotation property (custom or predefined). In the condition builder/individual creator interface, a new dialog is opened when the user chooses to annotate a property, or an expression. This dialog allows user to choose one of annotations pre-assigned to that property or class. Figure 4 illustrates how the user can specify that she wants a little fruit topping on her Pizza.

¹⁶ <https://www.w3.org/TR/swbp-n-aryRelations/>.

Here, **More** will allow her to choose one of default annotation properties and assign a value to it.

Annotation Property Configuration
Choose an annotation property to manage the configuration for that property

withAmount

Current Configuration

Name	Uri	Type	Input type	Action
hasTopping	http://www.co-ode.org/ontologies/pizza/pizza.owl#hasTopping	Object Properties	text	✕

Delete all

New Configuration

Component Type	Component	Input Type	
Object Properties	isBaseOf	text	Add

Fig. 3. Managing *withAmount* annotation property in the admin panel

5 Integration of Redesigned OntoPlay into the SXACML

Let us now discuss how the OntoPlay can be used in our use case scenario. As assumed, the semantic PIP uses OWL ontologies as the basis for inferring values of attributes that were not contained in the incoming access request, but required for policy evaluation. Typically, three ontologies are used by the system.

1. Request ontology – common for all scenarios where the SXACML is used. It contains vocabulary related to the XACML specification, such as Subject, Resource and Action.
2. Domain ontology – containing structured knowledge related to a particular organization or business. We assume it is modeled by an expert and filled with axioms from an external data source (database). This ontology needs not be directly related to the context of access control.
3. Mapping ontology – links the generic Request ontology with the Domain ontology. At the very least, it needs to define what, in domain terms, is the Subject and the Resource. It may also contain axioms defining relationships; e.g. hierarchies of resources and clustering of subjects into groups or roles.

While the first two ontologies are static or, at least, need not be modified together with the access control policies, the mapping ontology needs to be easy to update (also for an administrator unfamiliar with ontology modeling). To facilitate this, OntoPlay is configured to work with the Mapping Ontology, which in the beginning is empty, aside from the import statements referring to the Domain Ontology and the Request Ontology. This makes it possible to use any terms, from these ontologies, when creating class expressions in the UI. From the SXACML administration page it is then possible to define *equivalentClass* expressions, for the following classes declared in the RequestOntology.

- urn:oasis:names:tc:xacml:1.0:subject-category:access-subject
- urn:oasis:names:tc:xacml:3.0:attribute-category:resource
- urn:oasis:names:tc:xacml:3.0:attribute-category:action
- urn:oasis:names:tc:xacml:3.0:attribute-category:environment

To cover the scenario introduced in Sect. 2, we have modelled a simplistic domain ontology – Logistics Ontology – that represents knowledge concerning operations of the port. Here, note that in real-life, instead of creating a simple ontology, we would consider use of (possibly somewhat modified) existing transport and logistics ontologies (see, [6]). Integration of a more sophisticated ontology is planned, as future work, within the INTER-IoT project. The sample ontology consists of the following entities (we omit namespaces for brevity):

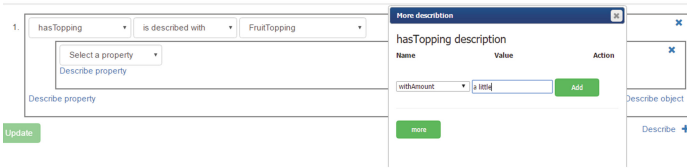


Fig. 4. Specifying the desired amount of the fruit topping using annotation property

- Person – an employee of the port or other institution.
- Company – a company employing or contracting persons.
- ContractedHaulier – a subtype of Company. A transport organization that has a fixed contract with the port authority.
- Location – an area of the port.
- Role – role of a person, e.g. a driver.
- Yard – the area within the port premises where truck and container movements occur. A subtype of Location.
- isHiredBy – a property describing the relationship between a Person and a Company.
- isContractedBy – a property describing the relationship between a Company and another Company. This property is transitive, therefore, a company contracted by company A that is contracted by company B, will also be inferred as contracted by company B. The ontology also contains a statement that Stark is subcontracted by Globex.

The policy that we focus on has the following structure:

- It specifies conditions for a Permit result
- The Subject needs to be of type HiredDriver
- The Resource needs to be Yard
- The Action needs to be Entry

Lastly, the request from the Stark Transport driver to access the facilities contains the following attributes (and their values):

- Subject category has two attributes: id = cristiano.cosio@starktransport.com, isHiredBy = Stark Transport
- Resource category has attribute id = InternalParking
- Action category has attribute id = Entry

We can see that the request does not use the same attribute set as the one used in the policy. Specifically, it does not refer to type HiredDriver and uses InternalParking instead of Yard. Here, note that the default XACML engine would not be able to deduct that Cristiano is an instance of type HiredDriver, therefore the permission would not be granted and our requirement of permission delegation would not be satisfied.

Let us demonstrate how we can employ semantic reasoning for this purpose. First of all, we need to make sure we map the generic subject category to the Person class from the Logistics Ontology. We do so by creating a class expression specifying that Subject is an equivalent class of Person. Subsequently, we should define another simple expression, tying together the concepts of Yard and InternalParking. This would be done in the same way – as an equivalentClass expression. Finally, we would like to specify the meaning of HiredDriver used in the policy, but not existing in the ontology itself. To this effect we create a class expression built of the following conditions:

1. HiredDriver is a subtype of Subject
2. it hasRole of Driver
3. it isHiredBy a ContractedHaulier OR it is hired by a Company that is subcontracted by a ContractedHaulier

The associated expression as defined in OntoPlay is shown in Fig. 5.

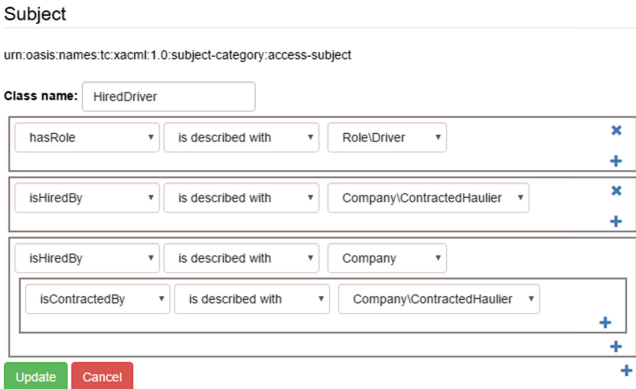


Fig. 5. Class expression defining the class HiredDriver

Based on this knowledge, the semantic Policy Information Point can infer that Cristiano matches the Subject conditions of the policy and that InternalParking matches the Yard requirement, therefore permitting the request.

6 Conclusions and Future Work

We have discussed how the redesigned OntoPlay, a simple and flexible ontology management UI component, can be utilized within an access control module to provide ontology mapping capabilities. The main improvement of this solution, when compared to other ontology editors, is that it is assumed that the user possesses minimal/no prior knowledge or experience with semantic technologies. We have illustrated how the combination of OntoPlay and semantic inference can simplify access control policies and improve the interoperability of the system. In the next steps, we plan to use transport and logistics ontologies listed in [6] and verify applicability of this approach to real-world use cases defined in the INTER-IoT project. We will also work on extending possibilities to easily map not only classes but also properties, thus enabling flexible attribute mapping.

Acknowledgments. Research presented in this paper has been partially supported by EU-H2020-ICT grant INTER-IoT 687283.

References

1. eXtensible Access Control Markup Language (XACML) version 3.0 (2013). <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html>
2. Drozdowicz, M., Ganzha, M., Paprzycki, M.: Semantic policy information point – preliminary considerations. In: Loshkovska, S., Koceski, S. (eds.) ICT Innovations 2015. AISC, vol. 399, pp. 11–19. Springer, Cham (2016). doi:[10.1007/978-3-319-25733-4_2](https://doi.org/10.1007/978-3-319-25733-4_2)
3. Drozdowicz, M., Ganzha, M., Paprzycki, M.: Semantically enriched data access policies in eHealth. *J. Med. Syst.* **40**(11), 238 (2016)
4. Drozdowicz, M., Ganzha, M., Paprzycki, M., Szymeja, P., Wasielewska, K.: OntoPlay - a flexible user-interface for ontology-based systems. In: AT, pp. 86–100 (2012)
5. Falco, R., Gangemi, A., Peroni, S., Shotton, D., Vitali, F.: Modelling OWL ontologies with graffoo. In: Presutti, V., Blomqvist, E., Troncy, R., Sack, H., Papadakis, I., Tordai, A. (eds.) ESWC 2014. LNCS, vol. 8798, pp. 320–325. Springer, Cham (2014). doi:[10.1007/978-3-319-11955-7_42](https://doi.org/10.1007/978-3-319-11955-7_42)
6. Ganzha, M., Paprzycki, M., Pawlowski, W., Szymeja, P., Wasielewska, K.: Semantic interoperability in the Internet of Things: an overview from the INTER-IoT perspective. *J. Netw. Comput. Appl.* **81**, 111–124 (2016)
7. Hu, V.C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K.: Guide to attribute based access control (ABAC) definition and considerations. NIST Spec. Publ. **800**, 162 (2014)
8. Petersen, N., Lange, C., et al.: TurtleEditor: an ontology-aware web-editor for collaborative ontology development. In: 2016 IEEE Tenth International Conference on Semantic Computing (ICSC), pp. 183–186. IEEE (2016)