FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Fine-grained authorization in Constrained RESTful Environments

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software and Information Engineering

by

Thomas Schmidleithner
Registration Number 1025525

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.-Prof. Dr. Wolfgang Kastner
Assistance: Dr. techn. Markus Jung

Vienna, March 16, 2015

Thomas Schmidleithner

Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Thomas Schmidleithner
Abelegasse 26/II/11, 1160 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) Thomas Schmidleithner

Abstract

Since the Internet of Things enables access to real-world objects which gather data about their environment, access to these objects has to be secured. Consequently, an authentication mechanism is needed to grant proper protection. Therefore, the ratified standard XACML, an access control policy language, is used to provide fine-grained access to these objects. The scope of this work is to extend an access control concept in order to authorize objects on a domain level for maintainability reasons. State-of-the-art technologies for a middleware and the access control concepts are presented. Furthermore, a Policy Decision Point which acts as an endpoint for authorization requests is implemented as a proof-of-concept for both protocols HTTP and CoAP.

Contents

CHAPTER 1

Introduction

The Internet has a massive and permanent influence on our everyday life. Within this global system of interconnected devices and networks, data generation and data exchange is one of the main tasks in a network of networks but this process is still dependent on the input of human beings. Regarding the term *Internet of Things* (IoT), in which a large number of (tiny) networked devices are communicating with each other, data are generated and shared between a variety of *things* or *objects*. Haller et al. [4] defined the term IoT as follows:

> "A world where physical objects are seamlessly integrated into the information network, and where the physical objects can become active participants in business processes. Services are available to interact with these 'smart objects' over the Internet, query their state and any information associated with them, taking into account security and privacy issues".

Considering potentialities, a wide spectrum of applications in different domains is offered by the IoT. Atzori et al. [1] grouped these domains in different categories, namely:

- **Transportation and logistics domain:** instrumenting vehicles with sensors, actuators and processing powers, roads and transported goods with tags and sensors, which allow sharing traffic and transportation information.
- **Healthcare:** tracking of objects and people such as staff and patients, identification and authentication of them as well as collecting and sensing data automatically.
- **Smart environment (home, office, plant) domain:** distributed sensors and actuators in buildings which automatically self-adjust environment settings and/or monitor and react on specific threshold settings.
- **Personal and social domain:** populate information from the recent past between social relationships. Information refers to traveling, losses (last recent location information of objects) and security concerning devices which have been removed from a restricted area.

1.1 Technologies for the Internet of Things

Advances in the field of the IoT have created a broad range of technologies in this area. A slight impression of the key and enabling technologies is provided by Atzori et al. who classified these technologies in two primary categories [1]:

Identification, sensing and communication technologies

As a key component, *RFID* tags which are characterized by a unique identifier and used for identifying real physical objects are mentioned. RFID is an abbreviation for radio-frequency identification and is used for identifying and tracking objects automatically.

Observing the environment with sensor networks and communicating sensing information between devices is another challenge in the IoT. Here, the wireless personal area network (WPAN) comes into play which is mainly based on the IEEE 802.15.4 standard with low-power and a low-bit rate communication [3].

Middleware

A software layer, which connects different enterprise applications and technologies in a distributed computing system to offer a single, standardized API is called a middleware. Hiding implementation details of different applications and technologies is one of the main tasks and therefore, it should be easier for the developer to focus on details in terms of the specific application instead of the entire environment. Furthermore, a middleware is important for linking modern applications with legacy systems.

1.2 Privacy Challenges in the Internet of Things

Having everything connected in the IoT is not only representing an advantage: objects must also be ensured to prevent unauthorized access and identification in privacy terms [14]. This authorization and authentication process is difficult to fulfill, even more difficult if such environments should be scalable. Thus, an advanced and fine-grained authorization concept is needed, which handles these problems [6]. This work should provide an overview of some related technologies for authentication in the Internet of Things, furthermore a fine-grained authorization approach is suggested and implemented.

1.3 Motivation

Such domains which were defined by Atzori et al. [1] entail different constraints that have to be considered. Power limitations and security authentication mechanisms are only a small proportion of these:

- **Low power in constrained environments:** As devices in the IoT will be more and more battery-operated, a power-efficient protocol has to be used. The *Constrained Application Protocol* (CoAP) by Shelby et al. [11], which is a proposed standard since June 2014, was designed for IP(v6) based communication using RESTful Web services for most of the constrained devices by the Internet Engineering Task Force (IETF) Constrained RESTful environments (CoRE) Working Group.

- **Authentication mechanism:** Communication between different things has to be ensured that unauthorized access and unauthorized identification is prevented. Certainly authorization and authentication in such environments can be difficult to establish considering the problem, that the access control policies are stored at a centralized machine and not redundantly on each device. Therefore, devices need to be verified for proper authorization rights which, especially in environments with small devices and weak processing power, is a challenging topic.

1.4 Aim of this Thesis

The scope of this work is to provide a fine-grained authorization mechanism for dealing with the control of objects and information in a constrained environment as a proof-of-concept. Furthermore, latency within constrained environments of this proof-of-concept should be evaluated for recursively capsulated objects. Additionaly, the current IoTSyS middleware has to be extended to allow authorization on HTTP and CoAP requests.

1.5 Structure of the Thesis

Chapter 2 provides state of the art concepts and technologies used within this work. For each of the technologies, a short overview is presented. In Chapter 3, we describe the fundamental access control concept which is used for the proof-of-concept described in Chapter 4. Chapter 5 provides an overview of the performance evaluation with various use cases in constrained environments. The last chapter contains the conclusion, summarizes the results and provides an outlook for possible extensions and further thoughts of this work.

CHAPTER 2

State of the Art Concepts and Technologies

2.1 Internet Protocol version 6

Due to the depletion of the address space of unallocated *Internet Protocol version 4* (IPv4) addresses, the *Internet Protocol version 6* (IPv6) was developed as the successor protocol to IPv4. By using a 128-bit address in IPv6 [2] instead of a 32-bit address which was used in IPv4, IPv6 expands the address space from 2^{32} to 2^{128} unique Internet addresses (approximately 3.4×10^{38} addresses). Thus, more levels of addressing hierarchy and a larger number of addressable nodes are supported. The stateless address auto-configuration simplifies address assignment by automatically configuring a host itself when connected to an IPv6 network. Another change to IPv4 is the simplified header. Some fields have been dropped, others have been made optional to reduce the bandwidth cost of IPv6. Furthermore, an IPv6 header allows greater flexibility for enabling new options and traffic handling, such as quality of service or “real-time” service with the new flow labeling capability. Support for authentication, data integrity and data confidentiality are also specified as an extension to IPv6. Identification of an IPv6 host is enabled by an IPv6 address which consists of eight blocks separated by colons. Each of these blocks contains four hexadecimal digits. The IPv6 header is shown in Figure 2.1.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  | IPv6
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
|         Payload Length        |  Next Header  |   Hop Limit   |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
|                                                               |  |
+                                                               +  |
|                                                               |  |
+                         Source Address                        +  |
|                                                               |  |
+                                                               +  |
|                                                               |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
|                                                               |  |
+                                                               +  |
|                                                               |  |
+                      Destination Address                      +  |
|                                                               |  |
+                                                               +  |
|                                                               |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.1: Standard IPv6 header without payload field [9]

2.2 6LoWPAN

The IPv6 over Low-Power Wireless Area Networks (*6LoWPAN*) concept is a standard for enabling IPv6 communication in networks with very limited processing capability and low-power devices. Shelby et al. [9] defined 6LoWPAN as follows:

> "6LoWPAN standards enable the efficient use of IPv6 over low-power, low-rate wireless networks on simple embedded devices through an adaptation layer and the optimization of related protocols."

Compared to the traditional IP protocol stack, the IPv6 protocol stack with 6LoWPAN has some different aspects which are shown in Figure 2.2. One of the main characteristics for 6LoWPAN is the header compression, which limits the payload size of packets provided by IEEE 802.15.4 and making it most efficient if all IPv6 packets can fit into a single IEEE 802.15.4 packet. Further, different aspects compared to the traditional IP protocol stack are enumerated as follows:

- **Support for IPv6 only** (implemented in the LoWPAN layer which is often part of the network layer).
- The **user datagram protocol** (UDP) is the most commonly used protocol in the transportation layer. Due to performance, efficiency and complexity reasons the Transmission Control Protocol (TCP) is typically renounced.

- **Control messaging** (e.g., ICMP echo, destination unreachable) is implemented by the Internet Control Message Protocol v6 (ICMPv6).
- **Application specific formats** or **binary formats** are often used for the application protocols.

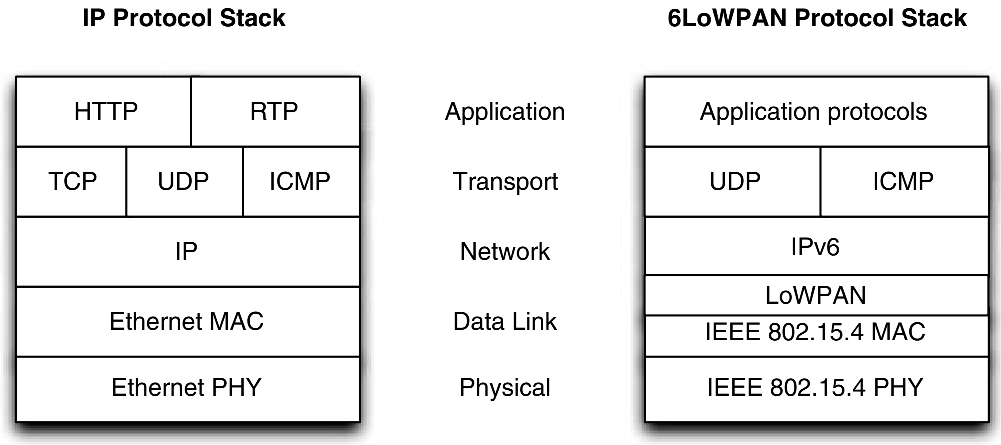


Figure 2.2: IP and 6LoWPAN protocol stacks [9]

2.3 Representational State Transfer

The architectural style of the World Wide Web is based on *Representational State Transfer* (REST) which follows a simple request/response pattern. On the one hand, a client accesses a resource on a server which is identified by a *Uniform Resource Identifier* (URI) by specifying an operation. On the other hand, the server sends back a response in a semistructured format like XML or JSON to the client. Mapped to HTTP, operations are `GET`, `PUT`, `POST` and `DELETE`, just to name the most important ones. The interaction between requests is stateless, thus, each request by a client has to contain all information which are necessary for the server to handle the request (like a session state). Compared to the SOAP- or RPC protocol, REST aims to provide a lightweight way for operating on distributed resources but is not an official standard.

2.4 Constrained Application Protocol

The *Constrained Application Protocol* (CoAP) by Shelby et al. [11] was designed for IP(v6) based communication using RESTful Web services for most constrained devices. It was released by the Internet Engineering Task Force (IETF)[1] Constrained RESTful environments (CoRE)[2] Working Group and is a proposed standard since June 2014. In the IoT, devices provide information for instance about their environment via sensors and actuators. This information should be

[1] `http://www.ietf.org`, access on 30 July, 2014

[2] `http://datatracker.ietf.org/wg/core/documents`, access on 30 July, 2014

accessible via a Web server running on these devices, but since IoT devices are often very small in sense of processing and memory capabilities, a lightweight protocol like CoAP is needed. Since CoAP is a RESTful protocol, it can be simply modified to enable HTTP support. The protocol supports the basic method types `GET`, `PUT`, `POST` and `DELETE` and also `OBSERVE` which allows subscription to another device to receive push notifications on resource changes. A CoAP request is sent by a client to request an action of a resource on a server and is similar to an HTTP request. The response is sent back by the server with an appropriate response code. As CoAP is bound to a datagram-oriented transport protocol such as UDP (which is connectionless) by default, these interchanges are asynchronously. Furthermore, unicast and multicast requests are also supported.

2.5 Open Building Information Xchange

Open Building Information Xchange[3] (oBIX) is an industry-wide initiative to define XML- and Web Service-based mechanisms for building control systems [13] and was developed by the Organization for the Advancement of Structured Information Standards (OASIS). oBIX is implemented as a RESTful approach, a resource centric architectural style for Web Services [7]. The aim of oBIX is that each device in a building automation environment should be represented as an oBIX object (which contains information about the environment) and accessible by a *URI* (Uniform Resource Identifier). In detail, oBIX is built on following concepts:
A flexible *object model* for describing data and operations is provided by oBIX. The object model defines a set of base objects with data types like booleans, numerics, strings, datetimes, enumerations and URIs. In oBIX, everything is realized as an object and every object can be accessed by a URI. Objects are mapped to physical entities and they represent their own state (e.g., a temperature sensor with a temperature value, a light with a boolean value). An object is based on the object model and can be encapsulated recursively with child objects. This gives the concept a high flexibility when dealing with compositions of complex building abstractions. Subtyping (*is-a*) as well as composition (*has-a*) are supported. Setting the *is* value of an object ensures that at least all fields of the according type are available. Choosing a semistructured language like XML as encoding syntax for oBIX objects makes requests and responses machine- and also human readable. Base object types like `int` are directly mapped to individual XML elements. Individual attributes for an object are subobjects and nested within their parent object. An example for an oBIX object is represented in XML Listing 2.1 which shows a smartmeter object that refers to the type `iot:SmartMeter` with a power and an energy value.

Listing 2.1: oBIX object

```xml
<obj href="smartmeter/" is="iot:SmartMeter">
  <real name="power" href="smartmeter/power" val="0.0"
    unit="obix:units/watt"/>
  <real name="energy" href="smartmeter/energy" val="0.0"
    unit="obix:units/kilowatthours"/>
</obj>
```

[3] See also http://www.obix.org/what.htm, accessed 14 February, 2014

An object must have a name to identify a subobject within a composite object (e.g., in Listing 2.1 the smartmeter has two values, every value with a distinct name like *power* and *energy*). A set of operations (`read`, `write`, `invoke` and `delete`) which are mapped to HTTP operations like `GET`, `POST`, `PUT`, `DELETE` enables control over these resources. Even custom operations are supported and have to be defined like any other custom object. Support for binary encoding allows a compact representation and makes oBIX well suited for 6LoWPAN devices and in general for low bandwidth usage.

A **history object** provides a list of timestamped changes correspondly to the object. As soon as an object changes one of its values, a new timestamped entry is added to this list. Querying this list is possible by setting a start time, an end time and the maximum number of entries in this list.

With the **lobby object**, an overview of all gateway handled oBIX objects is provided as a tree with concrete `href` values and can be accessed by the well-known address.

2.6 Access Control

Authorization and authentication has to be part of data sensitive systems. Such systems have to ensure that access is granted to authorized users and objects only. Ravi Sandhu and Perangel Samarati [8] described the access control concept as follows:

> "The purpose of access control is to limit the actions or operations that a legitimate user of a computer system can perform. Access control constrains what a user can do directly, as well as what programs executing on behalf of the users are allowed to do."

Various interconnected accessible objects in the IoT make it necessary to provide an access control concept which fits into the security and privacy prescription. Different access control requirements result in various access control approaches [10].

Mandatory Access Control

One access control mechanism is called *Mandatory Access Control* (MAC) which uses a *subject* to perform an operation on an *object*. Typically, a subject represents a user, a process or a thread. An object (also known as *target*) represents some sort of constructs such as files, devices, systems or others. This system-controlled policy mechanism restricts access to operations by subjects on selected objects. Subjects in such systems do not have the ability to override the policies and, therefore, have to be controlled by a security administrator.

Discretionary Access Control

The decision result for accessing a resource in the *Discretionary Access Control* concept is based on the identity of the subject which enables access control per subject. A subject can also pass the permissions to another subject, if the subject has the privilege to grant such permissions.

Role Based Access Control

In a system with *Role Based Access Control* (RBAC) permissions are associated with roles and each user is assigned to at least one role (or a set of roles). If the system has more different users than different roles, the management of permissions in such systems is rather simplified. This concept is a mitigated form of the Discretionary Access Control concept.

Attribute Based Access Control

An *Attribute Based Access Control* concept (ABAC) uses attributes which are combined to a policy. A policy contains types, such as user attributes, resource attributes and environment attributes. This modern concept allows fine-grained access control. One standard that implements this concept is called *XACML*.

2.7 eXtensible Access Control Markup Language

One implementation of the ABAC concept is called the *Extensible Access Control Markup Language* (XACML) which is a standard that defines a declarative access control policy language. Its schema is provided in XML [12]. An XACML-engine provides a request with information regarding the requested information and a policy with the constraints of an environment. As a result, the XACML-engine gives an output which is `Permit`, `Deny` or `NotApplicable`:

- `Permit` means that access to a requested resources is positively authorized and therefore the requested action can be performed.
- `Deny` is the decision result, if access to a requested action is not authorized and therefore access is denied.
- `NotApplicable` means that the engine could not find any policies for making a decision and, therefore, access is not possible.

An architectural view of the policy language model is shown in Figure 2.3.

History

The Organization for the Advancement of Structured Information Standards (OASIS) ratified eXtensible Access Control Markup Language v1.0 in February 2003. In February 2005, OASIS ratified XACML v2.0 which is currently used within this thesis.
The current version of XACML is v3.0 which was standardized by OASIS in January 2013.

XACML-request

An XACML-request is based on data values which are necessary for making an authorization decision. These data values are separated by different elements with `Subject`-, `Resource`-, `Action`- and `Environment` information. Every single element contains various information such as a `subject-id`, a `resource-id`, an `action-id` and can hold also custom

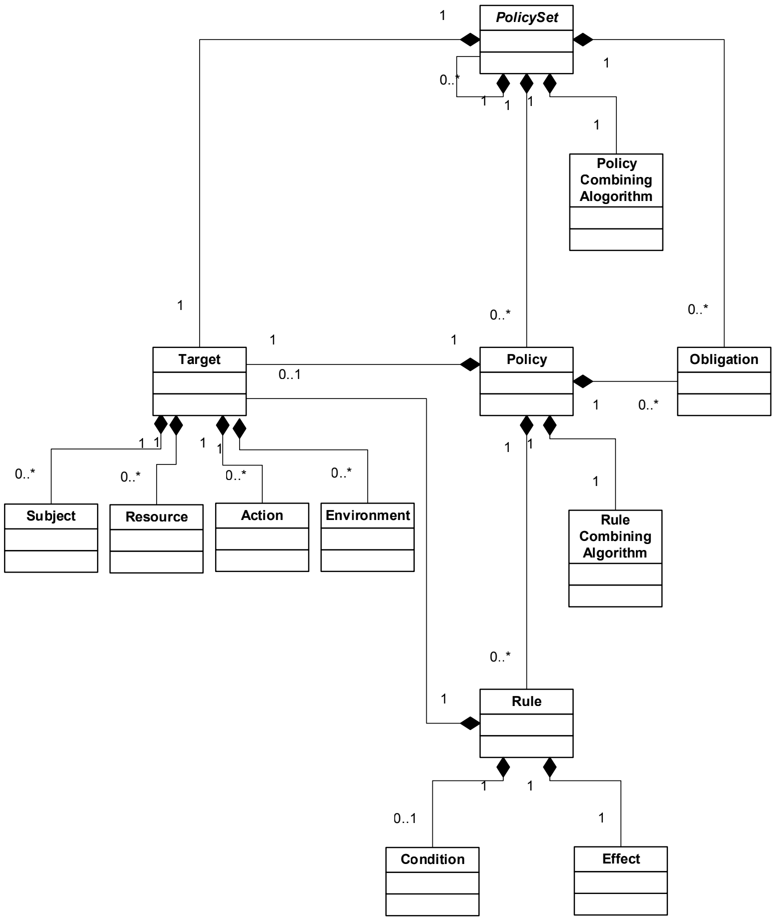


Figure 2.3: XACML Policy Language Model [12]

information. Listing 2.2 provides the basic structure of an XACML-request. Within a basic XACML-request, a subject (Line 6) requests a specific resource (Line 7) to execute an action (Line 8). Additionally, environment settings (Line 9) such as the current system time can be attached to the XACML-request.

Listing 2.2: Basic XACML-request

```
1  <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
4   xacml-2.0-context.xsd">
5
6    <Subject><Attribute AttributeId="..." /></Subject>
7    <Resource><Attribute AttributeId="..." /></Resource>
8    <Action><Attribute AttributeId="..." /></Action>
9    <Environment />
10
11 </Request>
```

XACML-policies

XACML-policies are separated by different `Policy`-elements within a `PolicySet`. A `PolicySet` can contain different `Policy`-elements which can also be recursively encapsulated and each `Policy`-element can contain different `Rule`-elements. Furthermore, the `Rule`-element can contain a `Description`- and a `Target`-element.

Listing 2.3: Basic XACML-policy

```xml
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
   http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-
   schema-os.xsd" PolicySetId="..." PolicyCombiningAlgId="...">

  <Target />

  <Policy PolicyId="..." RuleCombiningAlgId="...">
    <Description>The policy description</Description>
    <Target>...</Target>
    <Rule RuleId="..." Effect="Permit">
      <Condition>...</Condition>
    </Rule>
  </Policy>

</PolicySet>
```

XACML decision-making

In order to make decisions, we extract the XACML-request and apply it to the XACML-policies. As we have multiple policies in one policy set, the question about the final result of conflictive policy decisions arises. XACML specifies two combining algorithms for this problem:

- If at least one policy decision response is permit (and optional all others are deny) and the final evaluation output is permit, the algorithm is called `permit-overrides`.

$$\left(\bigwedge_{n>0} Permit_n\right) \wedge \left(\bigwedge_{m\geq 0} Deny_m\right) \Rightarrow Permit$$

- By analogy, if at least one policy decision response is deny (and optional all others are permit) and the final evaluation output is deny, the algorithm is called `deny-overrides`.

$$\left(\bigwedge_{n\geq 0} Permit_n\right) \wedge \left(\bigwedge_{m>0} Deny_m\right) \Rightarrow Deny$$

- If no policy is applicable, the decision response is $NotApplicable$.

Listing A.3 shows an XACML-decision response evaluated and positively emitted by the *Policy Decision Point* (PDP), which is responsible for evaluating access requests against authorization policies.

Listing 2.4: XACML decision response

```
<Response xmlns="..." xmlns:xsi="..." xsi:schmemaLocation="...">
  <Result>
    <Decision>Permit</Decision>
  </Result>
</Response>
```

A *Policy Enforcement Point* (PEP) can be used to intercept access requests and forwards a decision request to the PDP. Depending on the decision response, the PEP can execute any action.

Enterprise-Java-XACML compared to Sun's XACML implementation

The most popular implementation of XACML is realized by Sun. Compared to the Enterprise-Java-XACML implementation, the implementation of Sun has some missing features like caching of the policy evaluation result, or the missing policy search mechanism. On incoming requests, Suns XACML implementation applies each policy which results in heavily reduced performance for a high number of requests. Furthermore, it is not possible that multiple policies match a single request. Another problem is, that, without writing a wrapper, it is only possible to share policies in files.

CHAPTER 3

Access Control Concept

3.1 System Overview

The system is composed of constrained wireless motes, a gateway and a RPL border router. Policy decision requests can be performed from outside of the constrained network by a requestor but also inside by the motes. An overview is given in Figure 3.1.

- **Constrained wireless motes:** In a sensor network, a mote represents some type of sensor hardware. These motes are responsible for collecting information within its scope. Since most of these motes are very limited by their processing power and memory capabilities, it is not possible to perform powerful actions on them.
- **RPL border router:** The RPL border router provides the PEP. It is responsible for forwarding resource requests to the PDP, as soon as the policy decision arises, it can be forwarded to its motes.
- **Gateway:** The gateway provides the Policy Decision Point for granting authorization to specified resources based on the incoming XACML-request.

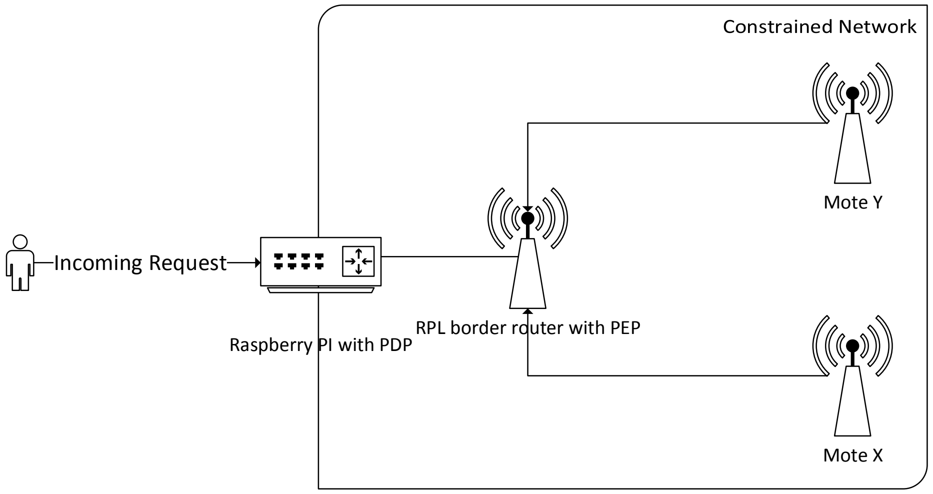


Figure 3.1: System Overview

3.2 Motivation to fine-grained XACML-policies

By the use of the oBIX standard, it is possible to fetch detailed information about the behavior of users. Thus, protection of data privacy is an important part of such environments. Therefore, the XACML policy language is used as a generic access control concept. To give the user a higher control of the corresponding oBIX objects, it is necessary to increase the generic access control to a fine-granular access control on an environment domain level.

3.3 Authorization with constraints on history objects

For making an authorization decision about requests on history objects, it is necessary to supplement the default decision request with additional start- and end date constraints. The start- and end date attributes can fit as part of one of the elements like `Subject`-, `Resource`-, `Action`- and `Environment` and just depend on the corresponding implementation and their policy.

Listing 3.1: Extended XACML-request (History)

```
<Attribute AttributeId="start-date"
  DataType="http://www.w3.org/2001/XMLSchema#dateTime">
  <AttributeValue>history-start-datetime-value</AttributeValue>
</Attribute>
<Attribute AttributeId="end-date"
  DataType="http://www.w3.org/2001/XMLSchema#dateTime">
  <AttributeValue>history-end-datetime-value</AttributeValue>
</Attribute>
```

The policy in Listing 3.2 can be wrapped within a `PolicySet`. The `Effect` attribute can either be `Permit` or `Deny`. It is possible to replace applying functions by other functions with the condition, that these operate with datetime values. With the `SubjectAttributeDesignator` the dynamic parameters are passed to the request (and could also be replaced by `ResourceAttributeDesignator`, `ActionAttributeDesignator` or `EnvironmentAttributeDesignator`). The custom namespace for the history constraints is `urn:tuwien:auto:iotsys`.

Listing 3.2: XACML-policy (History)

```
<Policy PolicyId="..." RuleCombiningAlgId="...">
  <Description/>
  <Target/>
  <Rule RuleId="..." Effect="...">
    <Condition>
      <Apply FunctionId="function:dateTime-greater-than-or-equal">
        <Apply FunctionId="function:dateTime-one-and-only">
          <SubjectAttributeDesignator AttributeId="start-date"
           DataType="http://www.w3.org/2001/XMLSchema#dateTime" />
        </Apply>
        <AttributeValue
         DataType="http://www.w3.org/2001/XMLSchema#dateTime">
        matching-datetime-value
        </AttributeValue>
      </Apply>
    </Condition>
  </Rule>
</Policy>
```

3.4 Authorization on a domain level

To allow domain control for oBIX objects, the IoTSyS-XACML module needs to be extended. Permitting or denying requests on a domain level is only possible, if the requested resource knows to which domain it belongs. Thus, iterating the basic oBIX lobby tree to get information about which domains the requested object belongs to, is the presented approach. Within this approach, a recursive function which tries to iterate through the tree is needed. The function iterates until it finds a domain object to which the requested oBIX object belongs to. The domain attribute can also be part of one of these elements such as `Subject-`, `Resource-`, `Action-` or `Environment`. The attribute element may contain various `AttributeValue` elements and each of these elements represents a domain from the iterated oBIX tree.

Listing 3.3: Extended XACML-request with a set of domains

```
<Attribute AttributeId="domain"
  DataType="http://www.w3.org/2001/XMLSchema#string">
  <AttributeValue>first-domain</AttributeValue>
  <AttributeValue>second-domain</AttributeValue>
  <AttributeValue>x-domain</AttributeValue>
</Attribute>
```

The policy below can be wrapped in a `PolicySet` with an attribute `PolicyCombiningAlgId` and as a value either `permit-overrides` or `deny-overrides`, the value for `Effect` can be Permit or Deny. A matching function (e.g., string-equal) can be chosen to match one or more domains. The custom namespace for the work with domain objects in Listing 3.4 is urn:tuwien:auto:iotsys.

Listing 3.4: XACML-policy (Domain)

```
<Policy PolicyId="..." RuleCombiningAlgId="...">
  <Description>Policy </Description>
  <Target />
  <Rule RuleId="..." Effect="...">
     <Target>
        <Resources>
           <Resource>
          <ResourceMatch MatchId="function:string-equal">
             <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#string">
                matching-domain
             </AttributeValue>
             <ResourceAttributeDesignator AttributeId="resource:path"
               DataType="http://www.w3.org/2001/XMLSchema#string"/>
             </ResourceMatch>
          </Resource>
           </Resources>
     </Target>
  </Rule>
</Policy>
```

CHAPTER 4

Proof-of-Concept

4.1 The IoTSyS integration middleware

A combination which provides the technologies from Chapter 2 in a middleware is realized in the *IoTSyS* middleware [5] which is defined as follows:

> "IoTSyS is an integration middleware for the Internet of Things. It provides a communication stack for embedded devices based on IPv6, Web services and oBIX to provide interoperable interfaces for smart objects. Using 6LoWPAN for constrained wireless networks and the Constrained Application Protocol together with Efficient XML Interchange an efficient stack is provided allowing using interoperable Web technologies in the field of sensor and actuator networks lightingand systems while remaining nearly as efficient regarding transmission message sizes as existing automation systems. The IoTSyS middleware aims at providing a gateway concept for existing sensor and actuator systems found in nowadays home and building automation systems, a stack which can be deployed directly on embedded 6LoWPAN devices and further addresses security, discovery and scalability issues."

oBIX gateway

IoTSyS provides a gateway with an OSGI based oBIX server, an HTTP Server, a CoAP Server, a RESTful Web service endpoint and a SOAP Web service endpoint to enable the necessary communication functionality and the appropriate protocol bindings HTTP, CoAP and SOAP.

For the evaluation of this fine-grained authorization concept, the IoTSyS middleware has to be extended to provide a working prototype with the ability, to analyze its performance. Since XACML is based on the XML standard, it is possible to implement this concept in another coding language than Java and is furthermore not restricted to the IoTSyS middleware.

4.2 Implementation

We extended the IoTSyS middleware by a PDP endpoint for interacting with HTTP and CoAP which implementation can be partly seen in Listing A.2. If an incoming request path on the Apache Tomcat server ends with PDP (e.g., http://GATEWAY/PDP), the request is intercepted by the PDP endpoint, which evaluates the incoming XACML-request against the XACML-policies and is responding with an XACML-decision response as an XML payload (see Listing A.7). The same holds for a CoAP-request (e.g., coap://GATEWAY/PDP), where the interception is implemented within the CoAP server (see Listing A.6).

To allow support for decision making on a domain level (such domains as e.g., base floor, lighting, heating) we need to bypass environment parameters to the XACML-request. Thus, extending the request template with domain attributes is a valid approach. This is illustrated in Listing 4.1.

Following source code is only a proof-of-concept and not intended to be used in production environments. In terms of future prospect, this code should be refactored by making use of an abstract class which implements methods for receiving the domains and domain objects to avoid code duplication.

Listing 4.1: Extended IoTSyS XACML-request template

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
      xacml-2.0-context.xsd">
  <Subject>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>{$SUBJECT}</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:ip-address"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>{$SUBJECT_IP_ADDRESS}</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:tuwien:auto:iotsys:subject:start-date"
      DataType="http://www.w3.org/2001/XMLSchema#dateTime">
      <AttributeValue>{$SUBJECT_HISTORY_START_DATETIME}</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:tuwien:auto:iotsys:subject:end-date"
      DataType="http://www.w3.org/2001/XMLSchema#dateTime">
      <AttributeValue>{$SUBJECT_HISTORY_END_DATETIME}</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:tuwien:auto:iotsys:subject:history-count"
      DataType="http://www.w3.org/2001/XMLSchema#integer">
      <AttributeValue>{$SUBJECT_HISTORY_COUNT}</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:tuwien:auto:iotsys:domain"
        DataType="http://www.w3.org/2001/XMLSchema#string">
      {$SUBJECT_DOMAIN}
    </Attribute>
  </Subject>
  <Resource>
    <Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>{$RESOURCE}</AttributeValue>
    </Attribute>

    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:protocol"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>{$RESOURCE_PROTOCOL}</AttributeValue>
    </Attribute>

    <Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:ip-address"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>{$RESOURCE_IP_ADDRESS}</AttributeValue>
    </Attribute>

    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:hostname"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>{$RESOURCE_HOSTNAME}</AttributeValue>
    </Attribute>

    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:path"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>{$RESOURCE_PATH}</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>{$ACTION}</AttributeValue>
    </Attribute>
  </Action>
  <Environment />
</Request>
```

Domain level access control

Having plenty of devices connected in a smart building (such as light in different rooms, boiler, cooler and much more), it could be useful to assign these devices to different domains. A domain could represent the lighting of the entire building, the lighting of the first floor or the heating of the building and every domain consists of a collection of such devices. By making use of such domains, it is possible to manage access policies on a domain level, which makes policy administration simpler and more efficient.

To decide whether it is possible to access a device, it is necessary to know to which domains the device belongs to. Thus, our approach is, to expand the oBIX tree to get these mappings between a domain and a device. Considering the fact, that the requested resource is not the root node of the tree, we need to recursively iterate through the oBIX tree in backwards direction. Hence, we cannot miss any object. Passing the domain to a query can be enabled by the extension of the XACML-request template with a new parameter *SUBJECT_DOMAIN* within the custom namespace `urn:tuwien:auto:iotsys:domain` which gets substituted by an *AttributeValue* for each domain related to the device in the oBIX tree. The request template is shown in Listing 4.1. Since the oBIX tree consists of different types of nodes and we are only interested in (parent) nodes, which are real devices (and not just instances of types such as `Domain` or `Unit`), we need to implement a subtype interface called `IDevice`, implementing the interface `IObj`. By inheriting each device (e.g., `Actuator`, `Sensor`) from `IDevice`, we can get all relevant instances from the oBIX tree. The rest of the objects can still implement the interface `IObj`. The UML diagram in Figure 4.1 provides an overview of the refactored oBIX hierarchy, nevertheless, objects that implement the interface `IObj` directly are not shown.

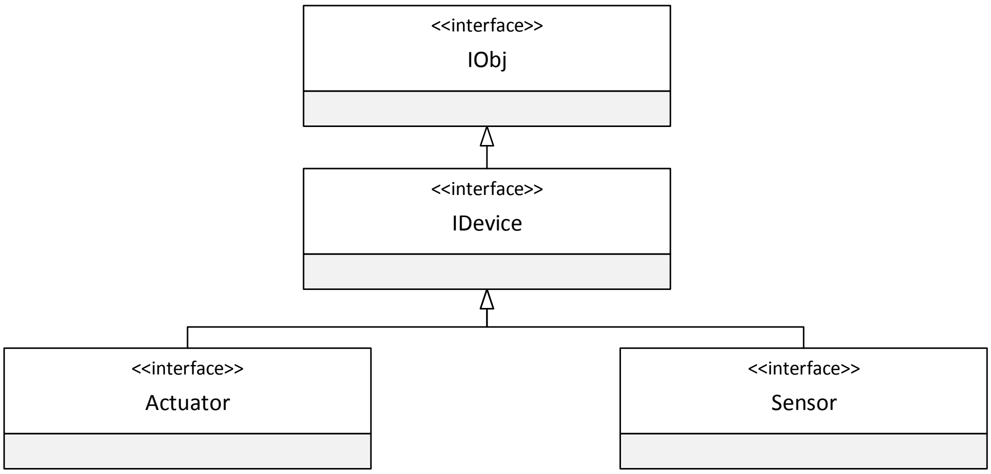


Figure 4.1: oBIX overview with new interface `IDevice`

Within the implementation, it is necessary to parse the related domains within the protocol related server (e.g., Tomcat, CoAP) where the oBIX lobby is utterly available.

XACML Policy Decision Point

As already seen in the system overview (Figure 3.1) it would be useful to provide a PDP for requesting authorization information by a PEP from outside the constrained network or directly by other motes. Therefore, a PDP has been implemented and enabled for `POST` requests of the protocols HTTP (accessible via `http://GATEWAY/PDP`) and CoAP (accessible via `coap://GATEWAY/PDP`).

Each `POST` request queried against the endpoint `/PDP` is intercepted by its protocol related server. The payload of the incoming request will be parsed by the `XACMLRequestParser` and a `Map` with the parameters is provided. For domain access control, the requested resource needs to be queried within the oBIX tree for fetching resource related domains. Now, the protocol related server intercepts the parameters to the `PDPInterceptor` which is responsible for

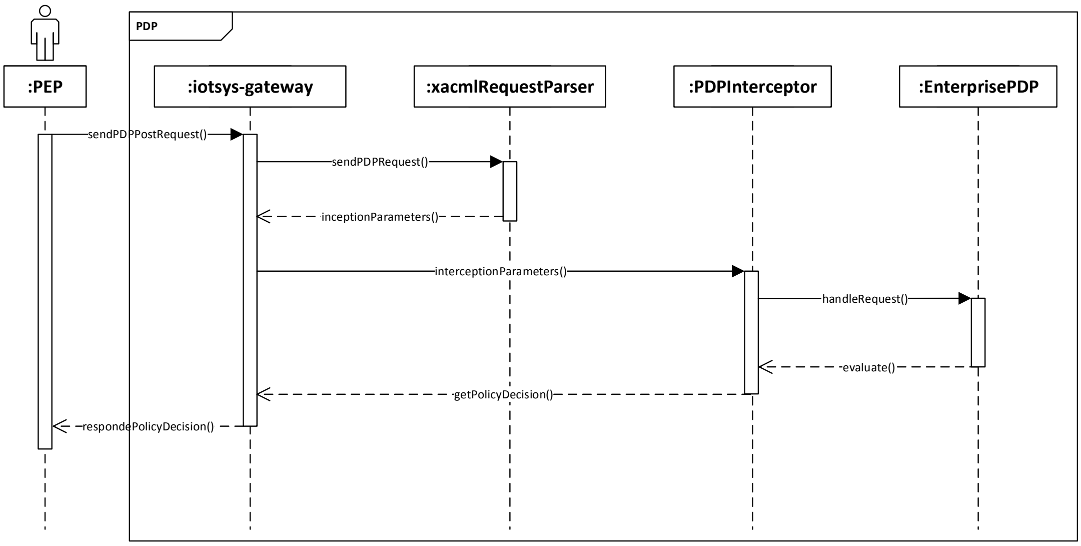


Figure 4.2: Sequence Diagram of a Policy Decision Request

execution of the evaluation process defined by the interface `PDP`, with the `EnterprisePDP` instance. After decision making, a valid decision response can be fetched by the `Response` object. At the end, the protocol related server provides the valid XACML-Response for the requesting party. An overview of the generic call sequence can be seen in Figure 4.2. Please note that the type of the protocols is hidden within the lifeline of the communication party *:iotsys-gateway*.

CHAPTER 5

Evaluation

To evaluate the Policy Decision Point with the protocols HTTP and CoAP, a benchmark tool[1] was developed to measure the response time depending on different Policy Decision requests. The benchmark tool is able to load one use case XACML-request scenario and sends a fixed number of requests to the PDP to evaluate the response time. Furthermore, different use cases have been compared by means of various XACML-requests and the corresponding response time. Additionally, a comparison of the results was drawn to see the impact of the implemented access control extensions.

5.1 Test Environment

For the evaluation of the performance, the extended IoTSyS middleware was deployed on a Raspberry Pi 2011.12 with a CPU frequency of 700 MHz and 512 MB memory, running a Raspbian platform.

5.2 Use Cases

We define some use cases in a typical home automation environment and evaluate them in the next section by means of their response time. All components are composed from oBIX objects.

Evaluation Scenario

Within a residential home (the oBIX name is *residentialHome* of type `knx:Part`), we define components as followed:

- **floorBasement** (`knx:Part`)
    - basementRoom (`knx:Part`): Represents the basement of the building

[1] `https://github.com/tschmidleithner/Benchmark`, access on 13th February 2015

        - boilerBasement (`iot:Boiler`)
        - coolerBasement (`iot:Cooler`)
        - smartBasement (`iot:SmartMeter`)
        - virtualLightBasement (`iot:LightSwitchActuator`)

- **floor1** (`knx:Part`): Represents the second floor of the building.
    - room1 (`knx:Part`)
        - virtualTemp1 (`iot:TemperatureSensor`)
        - virtualLight11 (`iot:LightSwitchActuator`)
    - copyRoom1 (`knx:Part`)
        - virtualLightCopyRoom (`iot:LightSwitchActuator`)
- **floor2** (`knx:Part`): Represents the second floor of the building.
    - flat1 (`knx:Part`)
        - boiler21 (`iot:Boiler`)
        - virtualLight21 (`iot:LightSwitchActuator`)
    - flat2 (`knx:Part`)
        - boiler22 (`iot:Boiler`)
        - virtualLight22 (`iot:LightSwitchActuator`)

To make access control more maintainable, some domains are defined to control a set of these components, defined as follows:

- **heating** (`knx:Domain`): Represents the heating of the overall building.
    - boilerBasement (`iot:Boiler`)
    - coolerBasement (`iot:Cooler`)
    - heatingfloor2 (`knx:Domain`): Represents the heating of the second floor.
        - boiler21 (`knx:iotBoiler`)
        - boiler22 (`knx:iotBoiler`)
- **lighting** (`knx:Domain`): Represents the lighting of the overall building.
    - lightingFloor1 (`knx:Domain`): Represents the lighting of the first floor.
    - lightingFloor2 (`knx:Domain`): Represents the lighting of the second floor.

The *residentialHome* with all related parts and components is built and wired together by the use of the `VirtualDeviceLoader` within the `iotsys-virtual` package for simulation purposes.

5.3 Benchmarks

Since this scenario offers various options to access the IoT devices, different access scenarios have been benchmarked. For improved readability, the XACML-policies are reduced to the minimum regarding namespaces and XML-syntax.

Deny lighting domain access

We define our XACML-policy in Listing 5.1 such that access to the domain *lightingfloor1* and also the *virtualLightBasement* is restricted.

Listing 5.1: Deny lighting domain access policy

```xml
<?xml version="1.0" encoding="utf-8"?>
<PolicySet PolicyCombiningAlgId="rule-combining-algorithm:
    deny-overrides:deny-overrides">
  <Target />
  <Policy RuleCombiningAlgId="rule-combining-algorithm:
      deny-overrides:deny-overrides">
    <Description>Deny access to virtualLightBasement</Description>
    <Target></Target>
    <Rule Effect="Deny">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch MatchId="function:string-equal">
              <AttributeValue
                  DataType="http://www.w3.org/2001/XMLSchema#string">
                /residentialHome/floorBasement/basementRoom/virtualLightBasement
              </AttributeValue>
              <ResourceAttributeDesignator AttributeId="resource:path"
                  DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </ResourceMatch>
          </Resource>
        </Resources>
      </Target>
    </Rule>
  </Policy>
  <Policy RuleCombiningAlgId="rule-combining-algorithm:deny-overrides">
    <Description>Deny access to domain lightingfloor1</Description>
    <Target></Target>
    <Rule Effect="Deny">
      <Condition>
        <Apply FunctionId="function:string-at-least-one-member-of">
          <SubjectAttributeDesignator AttributeId="subject:domain"
              DataType="http://www.w3.org/2001/XMLSchema#string"
              MustBePresent="false" />
            <Apply FunctionId="function:string-bag">
              <AttributeValue
                  DataType="http://www.w3.org/2001/XMLSchema#string">
                /residentialHome/lighting/lightingfloor1
              </AttributeValue>
            </Apply>
          </Apply>
        </Condition>
    </Rule>
  </Policy>
  <Policy RuleCombiningAlgId="rule-combining-algorithm:deny-overrides">
    <Description>Permit if no deny happened</Description>
    <Target></Target>
    <Rule Effect="Permit"></Rule>
  </Policy>
</PolicySet>
```

Deny heating domain access

We define our XACML-policy in Listing 5.2 such that access to the domain *heating* is denied, except for the boiler *boilerBasement* in the basement floor. Thus, every request of a heating- or cooling device will be denied except requests to the boiler in the basement floor.

Listing 5.2: Deny heating domain access policy

```
<?xml version="1.0" encoding="utf-8"?>
<PolicySet PolicyCombiningAlgId="rule-combining-algorithm:
    deny-overrides:deny-overrides">
  <Target />
  <Policy RuleCombiningAlgId="rule-combining-algorithm:permit-overrides">
    <Description>Deny access to domain heating except the boiler in the
        basement floor</Description>
    <Target></Target>
    <Rule Effect="Deny">
      <Condition>
        <Apply FunctionId="function:string-at-least-one-member-of">
          <SubjectAttributeDesignator AttributeId="subject:domain"
              DataType="http://www.w3.org/2001/XMLSchema#string"
              MustBePresent="false" />
          <Apply FunctionId="function:string-bag">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              /residentialHome/heating
            </AttributeValue>
          </Apply></Apply>
        </Condition>
    </Rule>
    <Rule Effect="Permit">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch MatchId="function:string-equal">
              <AttributeValue
                  DataType="http://www.w3.org/2001/XMLSchema#string">
                /residentialHome/heating/boilerBasement
              </AttributeValue>
              <ResourceAttributeDesignator AttributeId="resource:path"
                  DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </ResourceMatch>
          </Resource>
        </Resources>
      </Target>
    </Rule>
  </Policy>

  <Policy RuleCombiningAlgId="rule-combining-algorithm:deny-overrides">
    <Description>Permit if no deny happened</Description>
    <Target></Target>
    <Rule Effect="Permit"></Rule>
  </Policy>
</PolicySet>
```

Deny lighting domain access except for a subdomain

We define our XACML-policy in Listing 5.3 such that access to the domain *lighting* (the entire building) is restricted, except for the domain *lightingFloor1*.

Listing 5.3: Deny lighting domain access except for a subdomain

```
<?xml version="1.0" encoding="utf-8"?>
<PolicySet
    PolicyCombiningAlgId="policy-combining-algorithm:permit-overrides">
  <Target />

  <Policy PolicyId="restrict-domains-except-one:restrict-lighting"
    RuleCombiningAlgId="rule-combining-algorithm:deny-overrides">
    <Description>Deny access to domain lighting</Description>
    <Target></Target>
    <Rule RuleId="restrict-domains-except-one:restrict-lighting:rule"
      Effect="Deny">
      <Condition>
        <Apply FunctionId="function:string-at-least-one-member-of">
          <SubjectAttributeDesignator
            AttributeId="subject:domain"
            DataType="http://www.w3.org/2001/XMLSchema#string"
                MustBePresent="false" />
          <Apply
              FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
            <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#string">
              /residentialHome/lighting
            </AttributeValue>
          </Apply>
        </Apply>
      </Condition>
    </Rule>
  </Policy>

  <Policy PolicyId="restrict-domains-except-one:permit-lightingFloor1"
    RuleCombiningAlgId="rule-combining-algorithm:deny-overrides">
    <Description>Permit access to domain lightingFloor1</Description>
    <Target></Target>
    <Rule RuleId="restrict-domains-except-one:permit-ligntningFloor1:rule"
```

```
      Effect="Permit">
      <Condition>
          <Apply FunctionId="function:string-at-least-one-member-of">
            <SubjectAttributeDesignator
              AttributeId="subject:domain"
              DataType="http://www.w3.org/2001/XMLSchema#string"
                 MustBePresent="false" />
            <Apply FunctionId="function:string-bag">
              <AttributeValue
                 DataType="http://www.w3.org/2001/XMLSchema#string">
                /residentialHome/lighting/lightingfloor1
              </AttributeValue>
            </Apply>
          </Apply>
      </Condition>
    </Rule>
  </Policy>

  <Policy PolicyId="restrict-domains-except-one:restrict-rest"
    RuleCombiningAlgId="rule-combining-algorithm:deny-overrides">
    <Description>Deny if no permit happened</Description>
    <Target></Target>
    <Rule RuleId="restrict-domains-except-one:restrict-rest:rule"
      Effect="Deny">
    </Rule>
  </Policy>

</PolicySet>
```

5.4 Results

To evaluate the response time of the Policy Decision Point, requests with both HTTP and CoAP have been measured by the use of the benchmark tool mentioned above. Requests are measured by recording their runtime and those requests are performed one after the other. Since the presented access control concept is highly fine-grained, it would also be possible to define the IP-address of the requestor, the used protocol, a hostname, the request type and more. All these properties could be explicitly set within an XACML-policy and its corresponding XACML-request but for reasons of clarity and comprehensibility, the only changing parameter between these different requests is the resource path, whose parameter is in our case the most interesting aspect whereas the resource path is also used by requesting the regarding domain objects. All request examples below are based on the XACML-policies defined before, respectively. For the benchmark, we assume that requests come from outside of the constrained network, pretending to act like a PEP and perform requests to the PDP which is deployed on the Raspberry PI mentioned before.

Deny lighting domain access

To verify the policy decision response of this XACML-policy, a request shown in Listing 5.4 has been formulated to query if the lighting in the first floor is accessible (which should be denied because the domain *lightingFloor1* is restricted):

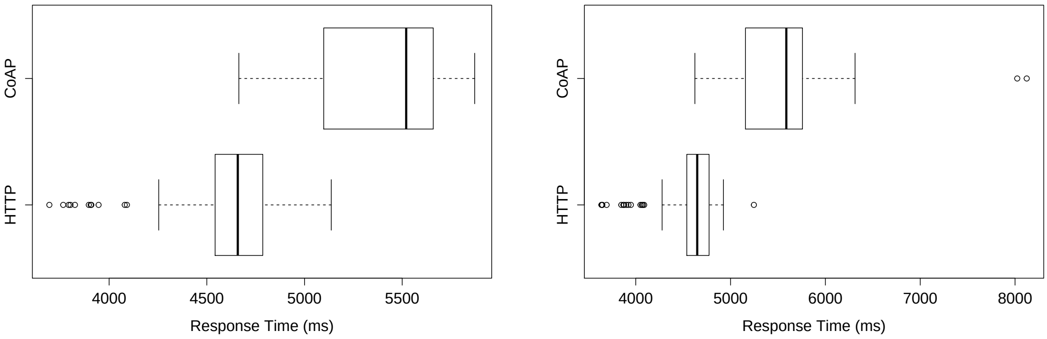
Listing 5.4: Request light access in the first floor

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
      xacml-2.0-context.xsd">
  <Subject>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>0:0:0:0:0:0:0:1</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:ip-address"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>0:0:0:0:0:0:0:1</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        http://localhost:8080/residentialHome/floor1/room1/virtualLight11
      </AttributeValue>
    </Attribute>

    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:protocol"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>http</AttributeValue>
    </Attribute>

    <Attribute
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:ip-address"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>127.0.0.1</AttributeValue>
    </Attribute>

    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:hostname"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>localhost</AttributeValue>
    </Attribute>

    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:path"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        residentialHome/floor1/room1/virtualLight11
      </AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>GET</AttributeValue>
    </Attribute>
  </Action>
  <Environment />
</Request>
```

The corresponding response is shown in Listing 5.5 for principle of completeness. Remaining responses in this chapter are not shown here. The benchmark is shown in Figure 5.1a. The statistical outliers next to the whiskers can be explained by caching mechanisms of the PDP. Therefore, those outliers next to the right whiskers have been observed at the beginning of each benchmark. Outliers next to the left whiskers are requests that have been cached already. This holds also for the rest of the benchmarks and is not mentioned again.

Listing 5.5: Response of light access first floor

```
<?xml version="1.0" encoding="UTF-16"?>
<Response xmlns="..." xmlns:xsi="..." xsi:schemaLocation="...">
  <Result>
    <Decision>Deny</Decision>
  </Result>
</Response>
```



(a) Request light access in the first floor

(b) Request light access in the second floor

Figure 5.1

Another request shown in Listing 5.6 may check if the light in the second floor is accessible (as this request is not restricted by any policy, the request will be permitted). The benchmark result is shown in Figure 5.1b

Listing 5.6: Request light access in the second floor

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
```

```
  <Subject><!-- ... --></Subject>
  <Resource>
    <!-- ... -->
    <Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        http://localhost:8080/residentialHome/floor2/flat1/virtualLight21
      </AttributeValue>
    </Attribute>

    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:path"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        residentialHome/floor2/flat1/virtualLight21
      </AttributeValue>
    </Attribute>
    <!-- ... -->
  </Resource>
  <Action><!-- ... --></Action>
  <Environment />
</Request>
```

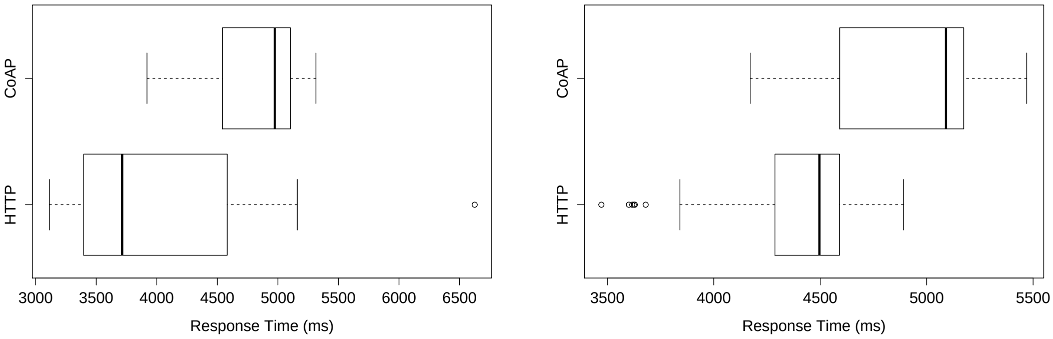
Deny heating domain access

A query for checking access for the climate exchange in the basement floor can be seen in Listing 5.7 where the access is denied. The results of the benchmark are shown in Figure 5.2a.

Listing 5.7: Request domain heating access cooler

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
  <Subject><!-- ... --></Subject>
  <Resource>
    <!-- ... -->
    <Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        http://localhost:8080/residentialHome/heating/coolerBasement
      </AttributeValue>
    </Attribute>

    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:path"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        /residentialHome/heating/coolerBasement
      </AttributeValue>
    </Attribute>
    <!-- ... -->
  </Resource>
  <Action><!-- ... --></Action>
  <Environment />
```

```
</Request>
```



(a) Request domain heating access cooler

(b) Request domain heating access boiler

Figure 5.2

A positively authorized query is shown in Listing 5.8 where the boiler is explicitly authorized in its policy. The according benchmark is shown in Figure 5.2b.

Listing 5.8: Request domain heating access boiler

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
  <Subject><!-- ... --></Subject>
  <Resource>
    <!-- ... -->
    <Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        http://localhost:8080/residentialHome/heating/boilerBasement
      </AttributeValue>
    </Attribute>

    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:path"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        /residentialHome/heating/boilerBasement
      </AttributeValue>
    </Attribute>
    <!-- ... -->
  </Resource>
  <Action><!-- ... --></Action>
  <Environment />
</Request>
```

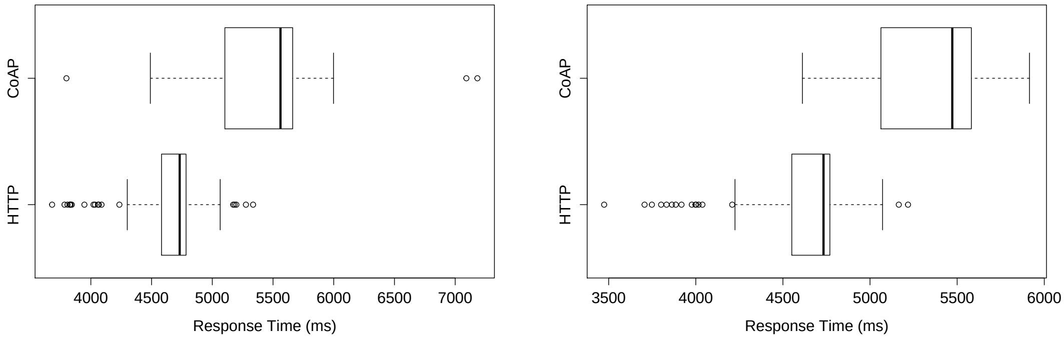
Deny lighting domain access except for a subdomain

Since this XACML-policy restricts access to the domain *lighting* a request shown in Listing 5.9 to the light in the second floor causes an access denied result. The benchmark is shown in Figure 5.3a.

Listing 5.9: Request Lighting Second Floor

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
  <Subject><!-- ... --></Subject>
  <Resource>
    <!-- ... -->
    <Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        http://localhost:8080/residentialHome/floor2/flat1/virtualLight21
      </AttributeValue>
    </Attribute>

    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:path"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        /residentialHome/floor2/flat1/virtualLight21
      </AttributeValue>
    </Attribute>
    <!-- ... -->
  </Resource>
  <Action><!-- ... --></Action>
  <Environment />
</Request>
```

A subdomain *lightingFloor1* of the domain *lighting* permits requests to this domain which decision request can be seen in Listing 5.10. The query targets the light of the copy room and the benchmark result is shown in Figure 5.3b.



(a) Request subdomain second floor

(b) Request light access copy room

Figure 5.3

Listing 5.10: Request domain lighting in first floor

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
  <Subject><!-- ... --></Subject>
  <Resource>
    <!-- ... -->
    <Attribute
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        http://localhost:8080/residentialHome/floor1/
        copyRoom1/virtualLightCopyRoom
      </AttributeValue>
    </Attribute>

    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:path"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>
        /residentialHome/floor1/copyRoom1/virtualLightCopyRoom
      </AttributeValue>
    </Attribute>
    <!-- ... -->
  </Resource>
  <Action><!-- ... --></Action>
  <Environment />
</Request>
```

Summary

It is obvious that all requests with HTTP show a faster response time compared to the CoAP requests, probably caused by the block fragmentation of the CoAP protocol. Generally speaking, the response time is also strongly dependent on the complexity of the XACML-policy and furthermore the number of linked domains. The more complex the oBIX lobby is, the slower the requests will be.

CHAPTER 6

Conclusion

In this work, a fine-grained authorization concept for the IoTSyS middleware has been presented. The implementation extends the existing IoTSyS middleware with access control on a domain level and on history objects. In addition, a Policy Decision Point for requesting authorization decision by a Policy Decision Request has been implemented in the protocols HTTP and CoAP. A use case scenario with sample requests has been defined and a benchmark for the comparison of these requests in the protocols HTTP and CoAP has been provided. The benchmark entails very different response times in comparison of HTTP and CoAP requests. Furthermore, the response time of requests is strongly dependent on the density of the oBIX tree and especially dependent on the number of existing domains. In general, domains in automation environments are a promising feature in sense of maintainability, given that granting access to e.g., the first floor alone is more comfortable compared to granting access for each object in the first floor.

Since this concept is only implemented in the protocols HTTP and CoAP, it would be useful to implement the access control concept in other protocols which can be easily done due to the use of the `PDPInterceptor`.

Another import feature would be a serverside GUI for the composition of XACML-policies. Due to complexity reasons of the XACML specification, it could be a challenging contribution, especially in terms of usability reasons.

At the moment, the interface `IDevice`, which is necessary for the domain determination, is only implemented in a few oBIX objects and should be extended to the rest of the oBIX domain belonging objects.

In avoidance of code duplication, the domain determination part in the server implementations could be outsourced in an abstract class like suggested in Appendix Listing A.5 but at the moment, the `CoAPServer` already extends the class `Endpoint`, therefore refactoring the code in future versions should be considered.

APPENDIX A

Listings

Listing A.1: PDP interface

```java
package at.ac.tuwien.auto.iotsys.xacml.pdp;

import java.util.Map;

import at.ac.tuwien.auto.iotsys.commons.interceptor.Parameter;

/**
 * Represents the interface of a policy decision point
 */
public interface Pdp {

  /**
   * Evaluates a decision request against the underlying XACML policy. The
   * three string attributes are mandatory. Additional parameters can be set
   * with the params Map.
   *
   * @param resource XACML resource-id
   * @param subject XACML subject-id
   * @param action XACML action-id
   * @param params additional parameters
   *
   * @return in case the request is evaluated to permit true, false
       otherwise.
   */
  public boolean evaluate(String resource, String subject, String action,
      Map<Parameter, String> params);
}
```

Listing A.2: EnterprisePDP

```java
package at.ac.tuwien.auto.iotsys.xacml.pdp;

```

```java
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Logger;

import an.xacml.adapter.file.XACMLParser;
import an.xacml.context.Decision;
import an.xacml.context.Request;
import an.xacml.context.Response;
import an.xacml.context.Result;
import an.xacml.engine.EvaluationContext;
import an.xacml.policy.AbstractPolicy;
import an.xacml.policy.Effect;
import at.ac.tuwien.auto.iotsys.commons.interceptor.Parameter;
import at.ac.tuwien.auto.iotsys.util.FileHelper;

/**
 * The enterprise PDP is responsible for handling
 * PDP request within the gateway. The policy is injected
 * by the PDPInterceptorSettings instance.
 */
public class EnterprisePDP implements Pdp {

  private Logger log = Logger.getLogger(EnterprisePDP.class.getName());

  private String resourcePrefix = "res/";
  private String requestTemplate = "";

  private static final String requestTemplatePath = "request/request.xml";

  private HashMap<String, String> policies = new HashMap<String, String>();

  public EnterprisePDP() {}

  /**
   * Constructor for the EnterprisePDP where a
   * directory string can be passed for the location
   * of the XML request template file.
   *
   * @param resourcePrefix the path prefix for the
   * request template to pass
   */
  public EnterprisePDP(String resourcePrefix) {
    this.resourcePrefix = resourcePrefix;
    try {
      String readFile = resourcePrefix + requestTemplatePath;
      log.info("Reading files from " + readFile);
      requestTemplate = FileHelper
          .readFile(readFile);
    } catch (IOException e) {
      e.printStackTrace();
    }

  }

  @Override
  public synchronized boolean evaluate(String resource, String subject,
     String action,
      Map<Parameter, String> params) {
    log.fine("Resource: " + resource);
    log.fine("Subject: " + subject);
    log.fine("Action: " + action);

    if (params == null) {
      params = new HashMap<Parameter, String>();
    }
    params.put(Parameter.RESOURCE, resource);
    params.put(Parameter.SUBJECT, subject);
    params.put(Parameter.ACTION, action);

    // Set default values if no value set
    if (!params.containsKey(Parameter.SUBJECT_HISTORY_START_DATETIME)) {
      params.put(Parameter.SUBJECT_HISTORY_START_DATETIME,
          "1970-01-01T00:00:00");
    }
    if (!params.containsKey(Parameter.SUBJECT_HISTORY_END_DATETIME)) {
      params.put(Parameter.SUBJECT_HISTORY_END_DATETIME,
          "2030-01-01T00:00:00");
    }
    if (!params.containsKey(Parameter.SUBJECT_HISTORY_COUNT)) {
      Integer max = (Integer.MAX_VALUE);
      params.put(Parameter.SUBJECT_HISTORY_COUNT, max.toString());
    }

    try {
      System.setProperty(XACMLParser.CONTEXT_KEY_DEFAULT_SCHEMA_FILE,
          resourcePrefix + "xacml-2.0-context.xsd");
      System.setProperty(XACMLParser.POLICY_KEY_DEFAULT_SCHEMA_FILE,
          resourcePrefix + "xacml-2.0-policy.xsd");

      String xacmlRequest = replaceParams(requestTemplate, params);
      log.info(xacmlRequest);

      // parse the XACML-request to a request instance
      Request req = XACMLParser.parseRequest(new ByteArrayInputStream(
           xacmlRequest.getBytes()));

      String policyFileName =
          PDPInterceptorSettings.getInstance().getPolicyFile();
      log.info("Using XACML-Policy File: " + policyFileName);
      String xacmlPolicy;
      synchronized (this) {
        if (!policies.containsKey(policyFileName)) {
          policies.put(policyFileName, FileHelper
              .readFile(resourcePrefix + "policies/" + policyFileName));
        }
        xacmlPolicy = policies.get(policyFileName);
      }
      log.info("Policy: " + xacmlPolicy);

      AbstractPolicy policy = XACMLParser
          .parsePolicy(new ByteArrayInputStream(xacmlPolicy.getBytes()));

      // evaluate the request against the policy
      Result result = policy.evaluate(new EvaluationContext(req));
      Response actualResponse = new Response(new Result[] { result });

      ByteArrayOutputStream tempOut = new ByteArrayOutputStream();
      XACMLParser.dumpResponse(actualResponse, tempOut);
      Decision d = result.getDecision();

      log.fine("decision request: " + d.toString());
      // deny request if value is not permit
      if (!d.equals(Effect.Permit)) {
        return false;
      }
      return true;

    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (Exception e) {
      log.severe("Exception occured ....");
      log.severe(e.getClass().getSimpleName());
      e.printStackTrace();
    }

    return false;
  }

  /**
   * Replaces the injected parameters of the template and
   * returns well-formed Policy Decision Request
   *
   * @param template the Policy Decision template
   * @param params injection parameters
   * @return the populated Policy Decision Request
   */
  private String replaceParams(String template, Map<Parameter, String> params) {
    if (params.containsKey(Parameter.SUBJECT_DOMAIN)) {
      String[] domainArr = params.get(Parameter.SUBJECT_DOMAIN).split(",");
      String strDomains = "";

      for (String d : domainArr) {
        strDomains += "<AttributeValue DataType=\"http://www.w3.org/2001/XMLSchema#string\">"
```

```
             + d + "</AttributeValue>" +
                 System.getProperty("line.separator");
      }

      template = template.replaceAll("\\{\\$" + Parameter.SUBJECT_DOMAIN +
          "\\}", strDomains);
    } else {
      template = template.replaceAll("\\{\\$" + Parameter.SUBJECT_DOMAIN +
          "\\}",
           "<AttributeValue
               DataType=\"http://www.w3.org/2001/XMLSchema#string\"></AttributeValue>");
    }

    for (Parameter r : params.keySet()) {
      if (!r.equals(Parameter.SUBJECT_DOMAIN)) {
        template = template.replaceAll("\\{\\$" + r + "\\}",
            params.get(r).trim());
      }
    }
    template = template.replaceAll("\\{\\$(.)*\\}", "");
    return template;
  }
}
```

Listing A.3: XACML response generation

```
package at.ac.tuwien.auto.iotsys.xacml.util;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.ls.DOMImplementationLS;
import org.w3c.dom.ls.LSSerializer;

import
    at.ac.tuwien.auto.iotsys.commons.interceptor.InterceptorResponse.StatusCode;

/**
 * Creates a well-formed XACML decision response.
 */
public class XacmlResponse extends AbstractXacml {
  private Document xacmlResponse;

  public XacmlResponse() {
    xacmlResponse = getBuilder().newDocument();
  }

  /**
   * Creates the Response {@link org.w3c.dom.Element} of the XACML decision
   * request depending on the given {@code StatusCode}.
   *
   * @param statusCode the status code
   * @return the XACML Decision Response
   */
  public Element getResponse(StatusCode statusCode) {
    Element request = xacmlResponse.createElementNS(URN_XACML_CONTEXT,
        "Response");

    Attr xsi = xacmlResponse.createAttribute("xmlns:xsi");
    xsi.setValue(XML_SCHEMA_INSTANCE);
    request.setAttributeNode(xsi);

    Attr schema = xacmlResponse.createAttribute("xsi:schemaLocation");
    schema.setValue(XML_SCHEMA_LOCATION);
    request.setAttributeNode(schema);

    Element result = xacmlResponse.createElement("Result");
    request.appendChild(result);

    Element decision = xacmlResponse.createElement("Decision");
    result.appendChild(decision);

    if (statusCode.equals(StatusCode.OK)) {
      decision.setTextContent("Permit");
    } else if (statusCode.equals(StatusCode.ERROR)) {
      decision.setTextContent("NotApplicable");
    } else {
      decision.setTextContent("Deny");
    }

    return request;
  }

  /**
   * Creates the Response {@link org.w3c.dom.Element} of the XACML decision
   * request depending on the given {@code StatusCode} and
   * returns the result as a string.
   *
   * @param statusCode the status code
   * @return String XACML Decision Response
   */
  public String getResponseString(StatusCode statusCode) {
    Element request = getResponse(statusCode);

    Document document = request.getOwnerDocument();
    DOMImplementationLS domImplLS = (DOMImplementationLS)
        document.getImplementation();
    LSSerializer serializer = domImplLS.createLSSerializer();

    return serializer.writeToString(request);
  }
}
```

Listing A.4: XACML request parser

```
package at.ac.tuwien.auto.iotsys.commons.util;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.HashMap;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

import at.ac.tuwien.auto.iotsys.commons.interceptor.Parameter;

/**
 * Helper class for extracting the parameters
 * of a given XACML-Request
 */
public class XACMLRequestParser {
  public static final String ACTION_ID =
      "urn:oasis:names:tc:xacml:1.0:action:action-id";
  public static final String RESOURCE_RESOURCE_ID =
      "urn:oasis:names:tc:xacml:1.0:resource:resource-id";
  public static final String RESOURCE_PROTOCOL =
      "urn:oasis:names:tc:xacml:1.0:resource:protocol";
  public static final String RESOURCE_IP_ADDRESS =
      "urn:oasis:names:tc:xacml:1.0:resource:ip-address";
  public static final String RESOURCE_PATH =
      "urn:oasis:names:tc:xacml:1.0:resource:path";
  public static final String RESOURCE_HOSTNAME =
      "urn:oasis:names:tc:xacml:1.0:resource:hostname";
  public static final String SUBJECT_HISTORY_COUNT =
      "urn:tuwien:auto:iotsys:subject:history-count";
  public static final String SUBJECT_TO_DATE =
      "urn:tuwien:auto:iotsys:subject:to-date";
  public static final String SUBJECT_FROM_DATE =
      "urn:tuwien:auto:iotsys:subject:from-date";
  public static final String SUBJECT_IP_ADDR =
      "urn:oasis:names:tc:xacml:1.0:subject:ip-address";
  public static final String SUBJECT_ID =
      "urn:oasis:names:tc:xacml:1.0:subject:subject-id";
  private String xacmlRequest;

  /**
   * Initialize the parser
   *
   * @param xacmlRequest the XACML-Request
   */
  public XACMLRequestParser(String xacmlRequest) {
    this.xacmlRequest = xacmlRequest;
  }

  private DocumentBuilder getBuilder() {
    DocumentBuilder builder = null;
    try {
      builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
    } catch (ParserConfigurationException e) {
      e.printStackTrace();
    }
    return builder;
  }

  /**
   * Parses the XACML-request and returns a map with
   * the parameters sent with the request
   *
   * Depending on the request, the map can contain keys
   * such as a subject with a subject IP address, history start time,
   * history end time, a history count, a resource with a hostname, an ip
   * address, a protocol and a resource id and an action parameter.
   *
   * @return map with parameters of the request
   */
  public HashMap<Parameter, String> parseRequest() {
    HashMap<Parameter, String> params = new HashMap<Parameter, String>();
    DocumentBuilder builder = getBuilder();
    InputStream stream = new ByteArrayInputStream(xacmlRequest.getBytes());

    Document document = null;
    try {
      document = builder.parse(stream);
    } catch (SAXException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }

    NodeList request = document.getElementsByTagName("Request");

    for (int i = 0; i < request.getLength(); i++) {
      NodeList nodeList = request.item(i).getChildNodes();
      for (int j = 0; j < nodeList.getLength(); j++) {
        if (nodeList.item(j).getNodeName().toUpperCase().equals("SUBJECT"))
            {
          // parse and map SUBJECT
          NodeList subject = nodeList.item(j).getChildNodes();
          for (int k = 0; k < subject.getLength(); k++) {
            Node attribute = subject.item(k);
            if (attribute.hasAttributes()) {
              NamedNodeMap attributes = attribute.getAttributes();
              for (int l = 0; l < attributes.getLength(); l++) {
                String nodeV = attributes.item(l).getNodeValue();
                if (nodeV.equals(SUBJECT_ID)) {
                  params.put(Parameter.SUBJECT,
                      attribute.getTextContent().trim());
                } else if (nodeV.equals(SUBJECT_IP_ADDR)) {
                  params.put(Parameter.SUBJECT_IP_ADDRESS,
                      attribute.getTextContent().trim());
                } else if (nodeV.equals(SUBJECT_FROM_DATE)) {
                  params.put(Parameter.SUBJECT_HISTORY_START_DATETIME,
                      attribute.getTextContent().trim());
                } else if (nodeV.equals(SUBJECT_TO_DATE)) {
                  params.put(Parameter.SUBJECT_HISTORY_END_DATETIME,
                      attribute.getTextContent().trim());
                } else if (nodeV.equals(SUBJECT_HISTORY_COUNT)) {
                  params.put(Parameter.SUBJECT_HISTORY_COUNT,
                      attribute.getTextContent().trim());
                }
              }
            }
          }
        } else if
            (nodeList.item(j).getNodeName().toUpperCase().equals("RESOURCE"))
            {
          // parse and map RESOURCE
          NodeList resource = nodeList.item(j).getChildNodes();
          for (int k = 0; k < resource.getLength(); k++) {
            Node attribute = resource.item(k);
            if (attribute.hasAttributes()) {
              NamedNodeMap attributes = attribute.getAttributes();
              for (int l = 0; l < attributes.getLength(); l++) {
                String nodeV = attributes.item(l).getNodeValue();
                if (nodeV.equals(RESOURCE_HOSTNAME)) {
                  params.put(Parameter.RESOURCE_HOSTNAME,
                      attribute.getTextContent().trim());
                } else if (nodeV.equals(RESOURCE_PATH)) {
                  params.put(Parameter.RESOURCE_PATH,
                      attribute.getTextContent().trim());
                } else if (nodeV.equals(RESOURCE_IP_ADDRESS)) {
                  params.put(Parameter.RESOURCE_IP_ADDRESS,
                      attribute.getTextContent().trim());
                } else if (nodeV.equals(RESOURCE_PROTOCOL)) {
                  params.put(Parameter.RESOURCE_PROTOCOL,
                      attribute.getTextContent().trim());
                } else if (nodeV.equals(RESOURCE_RESOURCE_ID)) {
                  params.put(Parameter.RESOURCE,
                      attribute.getTextContent().trim());
                }
              }

            }
          }
        } else if
            (nodeList.item(j).getNodeName().toUpperCase().equals("ACTION"))
            {
          // parse and map ACTION
          NodeList subject = nodeList.item(j).getChildNodes();
          for (int k = 0; k < subject.getLength(); k++) {
            Node attribute = subject.item(k);
            if (attribute.hasAttributes()) {
              NamedNodeMap attributes = attribute.getAttributes();
              for (int l = 0; l < attributes.getLength(); l++) {
                if (attributes.item(l).getNodeValue().equals(ACTION_ID)) {
                  params.put(Parameter.ACTION,
                     attribute.getTextContent().trim());
                }
              }
            }
          }
        }
      }
    }

    return params;
  }
}
```

Listing A.5: Helper for domain determination

```
// ...

public abstract class AbstractDomain {

  /**
   * Get the domains of the requesting resource
   * out of the oBIX lobby, depending on the
   * given URI.
   *
   * @param uri of the requesting resource
   * @return domains separated by commas
   */
  private String getDomainsOfRequestingResource(Uri uri) {
    ObjectBroker objectBroker = ObjectBrokerHelper.getInstance();
    Obj currentElement = objectBroker.pullObj(uri, false);

    // get requested object instance of IDevice
    Obj requestedResource = getParentObj(currentElement);

    ArrayList<String> domains = new ArrayList<String>();

    // check if requested oBIX object exists
    if (!(requestedResource instanceof Err))
    {
      Uri requestedObj = requestedResource.getHref();
      Object lobby = requestedResource.getRoot();
      Obj lobbyObj = (Obj)lobby;
      domains.addAll(getObjectDomains(requestedObj, lobbyObj.list()));
    } else {
      return "";
    }

  StringBuilder domainBuilder = new StringBuilder();
  for (String d : domains) {
    if (domainBuilder.length() > 0) {
      domainBuilder.append(",");
    }
    domainBuilder.append(d);
  }

  return domainBuilder.toString().trim();
}

/**
 * Returns a list of domains according to the requested object.
 *
 * @param requestedObj
 * @param children
 * @return a list of the domains
 */
private ArrayList<String> getObjectDomains(Uri requestedObj, Obj[]
    children)
{
  ArrayList<String> domains = new ArrayList<String>();

  for (Obj o : children)
  {
    if (o instanceof DomainImpl)
    {
      log.info("Domain found: " + o.getName());

      if (o.getByHref(requestedObj) != null)
      {
        domains.add(o.getNormalizedHref().getPath());
        log.info("Object " + requestedObj + " found in domain " +
            o.getNormalizedHref().getPath());

        // add parent domains
        Obj parent = o.getParent();
        if (parent != null)
        {
          domains.addAll(getParentsOfRequestedObject(parent));
        }
      }
    }
    domains.addAll(getObjectDomains(requestedObj, o.list()));
  }

  return domains;
}

/**
 * Get the parent object if current
 * object is not an instance of IDevice.
 * If the given object is no instance of
```

```java
   * IDevice, the request is applied recursively
   * until an instance of IDevice is found until
   * no more devices are in the tree.
   *
   * @param current
   * @return object
   */
  private Obj getParentObj(Obj current)
  {
    if (current instanceof IDevice)
      return current;

    Obj p = current.getParent();
    if (p != null)
      return getParentObj(p);

    return current;
  }

  /**
   * Returns a list of parent domains depending on the given
   * request object. Within this method, the given object
   * will be iterated to find instances of a domain.
   *
   * @param req
   * @return
   */
  private ArrayList<String> getParentsOfRequestedObject(Obj req)
  {
    ArrayList<String> domains = new ArrayList<String>();

    if (req instanceof DomainImpl)
    {
      log.info("Domain found: " + req.getNormalizedHref().getPath());
      domains.add(req.getNormalizedHref().getPath());
    }
    Obj p = req.getParent();
    if (p != null)
      domains.addAll(getParentsOfRequestedObject(p));

    return domains;
  }
}
```

Listing A.6: CoAP server implementation

```java
package at.ac.tuwien.auto.iotsys.gateway.obix.server;

// imports

public class CoAPServer extends Endpoint {

  //...

```

```
@Override
public void execute(Request request) throws IOException {
  String resourcePath = getResourcePath(request);
  log.info("Coap serving " + resourcePath + " for "
      + request.getPeerAddress().getAddress());

  // intercept PDP if request ends with PDP and request is a POST request
  if (resourcePath.endsWith("PDP") && request instanceof POSTRequest) {
    log.info("PDP interception");
    XACMLRequestParser xacmlRequestParser = new
        XACMLRequestParser(getPayload(request));
    HashMap<Parameter, String> params = xacmlRequestParser.parseRequest();
    String result = interceptPDP(params, request);
    log.info("XACML Decision Response: " + result);
    coapHelper.encodeResponse(result, request);
    request.sendResponse();
    return;
  }

  if (intercept(request)) return;

  // handle multicast requests first
  if (handleMulticastRequest(request)) return;

  LOG.info(String.format("Execution: %s", resourcePath));

  try {
    setCoAPResponse(request);
  } catch (URISyntaxException e) {
    e.printStackTrace();
  }

  if (request instanceof GETRequest) {
    if (request.hasOption(OptionNumberRegistry.OBSERVE)) {
      ObixObservingManager.getInstance().addObserver((GETRequest)
          request);
    }
  }

  request.sendResponse();
  log.info("Coap serving " + resourcePath + " for "
      + request.getPeerAddress().getAddress() + " done.");
}

 // ...

/**
 * Intercepts a PDP request and returns a Policy Decision Response.
 * The incoming parameters have to be parsed to a HashMap before
 * a decision can be evaluated.
 *
 * @param interceptorParams parsed request parameters
 * @param request the CoAP Request
```

```java
   * @return Policy Decision Response
   */
  private String interceptPDP(HashMap<Parameter, String> interceptorParams,
      Request request) {
    boolean interceptorsActive = Boolean.parseBoolean(PropertiesLoader
        .getInstance().getProperties()
        .getProperty("iotsys.gateway.interceptors.enable", "true"));

    if (!interceptorsActive || interceptorBroker == null
        || !interceptorBroker.hasInterceptors())
      return null;

    log.fine("Interceptors found ... starting to prepare.");
    String resourcePath = interceptorParams.get(Parameter.RESOURCE_PATH);

    InterceptorRequest interceptorRequest = new InterceptorRequestImpl();

    String domains = getDomainsOfRequestingResource(new Uri(resourcePath));
    if (domains.length() > 0) {
      interceptorParams.put(Parameter.SUBJECT_DOMAIN, domains);
    }

    interceptorRequest.setInterceptorParams(interceptorParams);

    log.fine("Calling interceptions ...");
    InterceptorResponse interceptorResp = interceptorBroker
        .handleRequest(interceptorRequest);

    if (!interceptorResp.getStatus().equals(StatusCode.OK)) {
      if (interceptorResp.forward()) {
        request.respond(CodeRegistry.RESP_FORBIDDEN,
            interceptorResp.getDecisionResponse(),
            MediaTypeRegistry.TEXT_XML);
      }
    } else {
      request.respond(CodeRegistry.RESP_VALID,
          interceptorResp.getDecisionResponse(),
          MediaTypeRegistry.TEXT_XML);
    }

    return interceptorResp.getDecisionResponse();
  }

  // ...
}
```

Listing A.7: Tomcat server implementation

```java
package at.ac.tuwien.auto.iotsys.gateway.obix.server;

// imports

public class TomcatServerNoSecurity {

```

```
// ...

public class ObixServlet extends HttpServlet {

    // ...

    @Override
  protected void doPost(HttpServletRequest req, HttpServletResponse resp)
       throws ServletException, IOException {

    StringBuffer obixResponse = null;
    Obj responseObj = null;
    String ipv6Address = "/" + getIPv6Address(req);
    String uri = req.getRequestURI();
    String data = getData(req);
    String resourcePath = getResourcePath(uri, ipv6Address);

    String response = "";
    // intercept PDP if request ends with PDP and request is a POST
        request
       if (uri.endsWith("PDP") && data != null) {
         log.info("PDP interception");

         XACMLRequestParser xacmlRequestParser = new
             XACMLRequestParser(data);
         HashMap<Parameter, String> params =
             xacmlRequestParser.parseRequest();

         // call interceptors
       response = interceptPDP(params, req, resp);
       log.info("XACML Decision Response: " + response);
       }

       // ...
  }

  /**
   * Intercepts a PDP request and returns a Policy Decision Response.
   * The incoming parameters have to be parsed to a HashMap before
   * its decision can be evaluated.
   * @param interceptorParams parsed request parameters
   * @param req the HTTP Request
   * @param resp the HTTP Response
   * @return Policy Decision Response
   */
  private String interceptPDP(HashMap<Parameter, String>
      interceptorParams, HttpServletRequest req, HttpServletResponse resp)
      {
    boolean interceptorsActive = Boolean.parseBoolean(PropertiesLoader
         .getInstance().getProperties()
         .getProperty("iotsys.gateway.interceptors.enable", "true"));

    if (!interceptorsActive || interceptorBroker == null
          || !interceptorBroker.hasInterceptors())
        return null;

      log.info("Interceptors found ... starting to prepare.");

      InterceptorRequest interceptorRequest = new InterceptorRequestImpl();

      String uri = interceptorParams.get(Parameter.RESOURCE_PATH);
      String domains = getDomainsOfRequestingResource(new Uri(uri));
      if (domains.length() > 0) {
        interceptorParams.put(Parameter.SUBJECT_DOMAIN, domains);
      }

      interceptorRequest.setInterceptorParams(interceptorParams);

      Enumeration<String> headers = req.getHeaderNames();

      while (headers.hasMoreElements()) {
        String k = headers.nextElement();
        interceptorRequest.setHeader(k, req.getHeader(k));
      }

      Enumeration<String> params = req.getParameterNames();

      while (params.hasMoreElements()) {
        String k = params.nextElement();
        interceptorRequest.setRequestParam(k, req.getParameter(k));
      }

      log.info("Calling interceptions ...");

      InterceptorResponse interceptorResp = interceptorBroker
          .handleRequest(interceptorRequest);



      if (!interceptorResp.getStatus().equals(StatusCode.OK)) {
        if (interceptorResp.forward()) {
          resp.setStatus(HttpStatus.SC_FORBIDDEN);
        }
      } else {
        resp.setStatus(HttpStatus.SC_OK);
      }

      resp.setContentType(MIME_XML);
      return interceptorResp.getDecisionResponse();
    }

      // ...
  }
}
```

List of Figures

Listings

Bibliography

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[2] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification, 1998.

[3] IEEE 802.15 Working Group for WPAN. IEEE802.

[4] Stephan Haller, Stamatis Karnouskos, and Christoph Schroth. The Internet of Things in an Enterprise Context. In John Domingue, Dieter Fensel, and Paolo Traverso, editors, *Future Internet – FIS 2008*, volume 5468 of *Lecture Notes in Computer Science*, pages 14–28. Springer Berlin Heidelberg, 2009.

[5] Markus Jung. IoTSyS - Internet of Things integration middleware. `http://code.google.com/p/iotsys/`.

[6] Markus Jung and Thomas Hofer and Wolfgang Kastner and Susen Döbelt. Protecting data assets in a Smart Grid SOA. *Journal of Internet Technology and Secured Transactions (JITST)*, 2:155 – 166, 2013.

[7] M. Neugschwandtner, G. Neugschwandtner, and W. Kastner. Web Services in Building Automation: Mapping KNX to oBIX. In *Industrial Informatics, 2007 5th IEEE International Conference on*, volume 1, pages 87–92, June 2007.

[8] R.S. Sandhu and P. Samarati. Access control: Principle and Practice. *Communications Magazine, IEEE*, 32(9):40–48, Sept 1994.

[9] Zach; Carsten Bormann Shelby. 6LoWPAN: The Wireless Embedded Internet, 2009.

[10] R. Shirey. Internet Security Glossary, Version 2. RFC 4949 (Informational), August 2007.

[11] The Internet Engineering Task Force. The Constrained Application Protocol (CoAP), 2014.

[12] The OASIS technical commitee. XACML: eXtensible Access Control Markup Language (XACML) 2.0, OASIS Standard, 2005.

[13] The OASIS technical commitee. oBIX 1.1 Committee Specification Draft 02 / Public Review Draft 02, 2013.

[14] International Telecommunication Union. ITU Internet Report 2005: The Internet of Things. 2005.