SitAC – A System for Situation-aware Access Control
Controlling Access to Sensor Data

Marc Hüffmeyer[1], Pascal Hirmer[2], Bernhard Mitschang[2], Ulf Schreier[1] and Matthias Wieland[2]
[1]*Hochschule Furtwangen University, Robert-Gerwig-Platz 1, Furtwangen im Schwarzwald, Germany*
[2]*Universität Stuttgart, Universitätsstraße 38, Stuttgart, Germany*
{*marc.hueffmeyer, ulf.schreier*}*@hs-furtwangen,*
{*pascal.hirmer, bernhard.mitschang, matthias.wieland*}*@ipvs.uni-stuttgart.de*

Keywords: Authorization, Attribute based Access Control, Situation-awareness, REST, Internet of Things.

Abstract: This paper addresses situation-aware access control for sensitive data produced in sensor networks. It describes how an attribute-based access control system can be combined with a situation recognition system to create a highly flexible, well performing, and situation-aware access control system. This access control system is capable of automatically granting or prohibiting access depending on situation occurrences and other dynamic or static security attributes. Besides a high-level architecture, this work also describes concepts and mechanisms that can be used to build such a system.

1 INTRODUCTION

This paper describes a situation-aware access control system to protect various kinds of RESTful services. The system is based on two major components: a situation recognition system and an attribute-based access control mechanism. A situation recognition system detects the occurrence of situations in so-called Internet of Things (IoT) environments. These environments comprise a number of devices that are usually attached with multiple sensors. Based on the sensors' values, the state of the environment can be detected. The aggregation of states leads to the derivation of *situations*. Hence, a situation can be defined as *"a state transition in an IoT environment"*. For example, the situation *"server room temperature overheated"* can be detected once temperature sensors reach a certain threshold. SitOPT (Franco da Silva et al., 2016; Hirmer et al., 2015; Wieland et al., 2015) is an example for such a situation recognition system, detecting situations through the aggregation of low-level sensor data. Attribute Based Access Control (ABAC) is an access control model that enables specifying rich variations of access rules. The main idea of ABAC is that any property of an entity can be used to determine access. We previously introduced RestACL (Hüffmeyer and Schreier, 2016c), an access control language based on the ABAC model. It has been designed to control access to RESTful services. The challenge in building a situation-aware access control system is the integration of those heterogeneous systems into one efficient and stable system.

A typical application scenario for a situation-aware access control system is the support of ambient assisted living. In such a scenario, possible emergency situations – like an elderly person falling down – can be detected through the aggregation of data produced by several sensors (e.g., acceleration and motion sensors). In case an emergency situation arises, a recognition system can automatically detect the situation and raise an alert to an emergency contact. However, since situation recognition cannot provide a completely reliable detection rate, it is good advice to double check whether the emergency situation really occurred. Therefore, an ambient assisted living home might be equipped with one or more cameras that observe the living environment. There are three different cases that target remote access to this camera: 1) An anxious family member might be privileged to access this camera permanently. 2) In case of an alert, a rescue service automatically gains temporary access to the camera to double check whether an emergency situation is given. This access privilege is dropped after a dedicated period of time. 3) In contrast to the first two cases, malicious persons must be permanently locked out. Imagine a burglar having access to the video camera. The burglar could easily spy out its victim, check where valuable items are kept and wait until the resident leaves its home to burglarize. Therefore, it is crucial to restrict access

Hüffmeyer, M., Hirmer, P., Mitschang, B., Schreier, U. and Wieland, M.
SitAC – A System for Situation-aware Access Control - Controlling Access to Sensor Data.
DOI: 10.5220/0006186501130125
In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*, pages 113-125
ISBN: 978-989-758-209-7

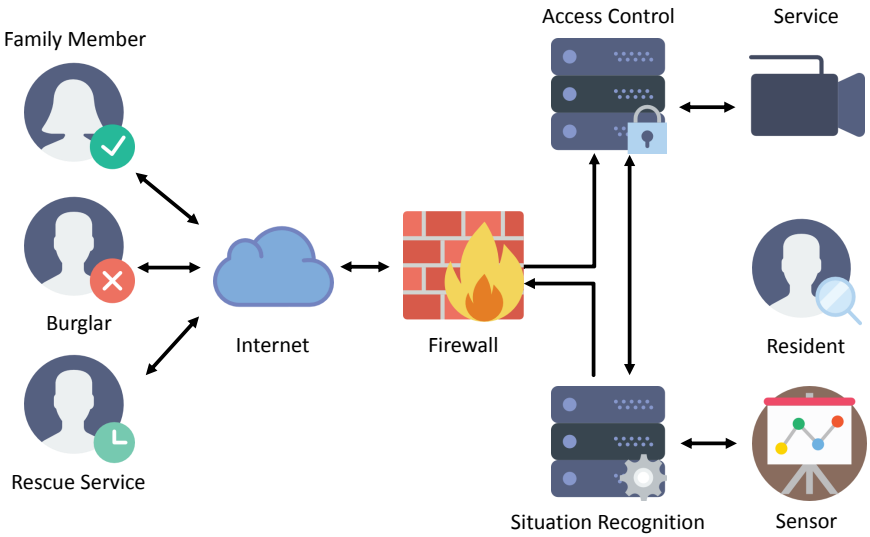

Figure 1: Scenario for situation-aware access control.

to the data produced by the camera. Figure 1 depicts this ambient assisted living scenario. The three different cases of access privileges (permanently granted, temporarily granted, permanently prohibited) in this scenario are indicated by the three mentioned users. All of them might access remote services in the living environment via the internet. A firewall is used to block non-authenticated users. Authenticated users send requests to the remote service. These requests are inspected and possibly rejected by an access control system. A situation recognition system is used to detect situations based on sets of sensor values. If a situation occurs, the system sends out notifications to registered users. For example, the family member and the rescue service might have been subscribed to receive notifications in case an emergency situation occurs. Also it is absolutely necessary that the situation recognition system informs the access control system about situation occurrences. Otherwise the access control system cannot act situation-aware.

Note that this paper introduces our approach for situation-aware access control based on the ambient assisted living scenario. However, there are several other application scenarios in which situation-aware access control is useful like, for example, Industry 4.0 scenarios as described in (Franco da Silva et al., 2016; Hirmer et al., 2015; Hirmer et al., 2016b).

The remainder of this paper is organized as follows: in Section 2, we describe the fundamentals of the situation recognition system (SitOPT) and the access control system (RestACL). In Section 3, an architecture, its components, and an API are introduced that enable situation-aware access control based on those systems. Situation registration and access procedures are discussed in detail in Section 4. Evaluation results are presented in Section 5. Finally, related work is referred to in Section 6 and a summary and an outlook to our future work are given in Section 7.

2 FOUNDATIONS

This section provides overviews over the SitOPT approach and over the RestACL access control system.

2.1 SitOPT

As depicted in Figure 2, the SitOPT approach (Hirmer et al., 2015; Wieland et al., 2015) offers several components for situation recognition. The two main components are the Situation Recognition Service (SitRS) and the Resource Management Platform (RMP). The SitRS is capable of detecting situations based on so called Situation Templates – a model for defining the conditions for occurring situations. More precisely, Situation Templates connect sensor values with conditions, which are then aggregated using logical operations such as AND, OR, or XOR. The root node of a Situation Templates is the actual situation to be defined. The RMP serves as gateway to the sensors and offers several adapters to bind different kinds of sensors and, furthermore, provides a uniform RESTful interface to access sensor data. The binding of devices equipped with sensors is conducted as described in (Hirmer et al., 2016b; Hirmer et al., 2016a).

A registered device can have one or more owners. Initially, only the owner has access to these devices. Once devices are bound, applications or users can register for situations to get notified on their occurrence. To realize this, first, a Situation Template

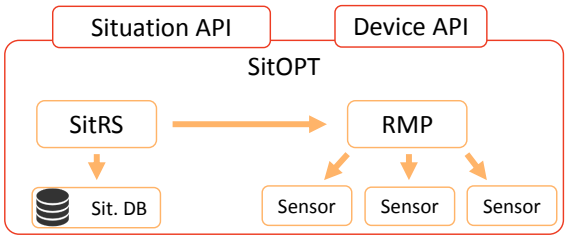

Figure 2: Architecture of SitOPT.

has to be modeled defining the situation's conditions. After that, it is used as input for the Situation Recognition Service, which transforms it into an executable representation (e.g., a CEP query) and executes it in a corresponding engine. On execution, situations are continuously monitored and registered applications or users are notified as soon as they occur. SitOPT offers two interfaces to upper-level applications. The Situation API allows registration on situations to get notified on their occurrence. The Device API allows registration of new devices to be monitored by the SitRS. In case a situation occurred, an alert is raised to all callbacks and the situation is written into a Situation Database. The Situation Database (SitDB) is a longterm storage for situation occurrences.

2.2 RestACL

Access control commonly answers the question which subjects might perform what actions on what objects (Ferraiolo et al., 2015). ABAC replaces directly referred entities like subjects or objects by attributes. ABAC mechanisms use so called categories to map attributes to entity types. For example, an attribute *name* can be mapped to a *subject* type.

Definition (Attribute): We define an attribute as a triple $a := (c,d,v)$ consisting of a category c, a designator d and a value v. Category, designator and value all have types like integers or character sequences.

Note that attributes are related to dedicated entities (e.g., an attribute *name* with the value *bob* is related to a human (the entity) with the name bob). Therefore, an entity consists of a set of attributes.

Definition (Entity): We define an entity $e := (\{a_1,...,a_n\})$ as the set of all attributes $a_1,...,a_n$ with $n \in \mathbb{N}$ belonging to the same category.

RestACL is an ABAC mechanism that targets authorization for RESTful services. Figure 3 shows the architecture of a RestACL access control system. Access logic is determined in policies located in a *Policy Repository*. A policy is a logical concatenation of expected attribute values. For example, a policy might declare that *"access is granted if attribute a_1 has value v_1 and attribute a_2 has value v_2"*. The identification of policies is done using so called *Domains* that describe mappings between resources and policies. An *Evaluation Engine* computes access decisions based on access requests derived from resource requests. The attributes that are required to perform the decision computation are delivered by one or more *Attribute Providers*. One kind of Attribute Provider stores attributes in a persistent manner. That means, the evaluation engine might pass an *id* attribute to an *Attribute Provider* and receive all known attributes of the entity that is related with the *id*. Another kind of attribute provider is needed to get information (such as the actual time) from the environment. A third kind is needed in order to get information from an external user identity management system. Further details about RestACL can be found in (Hüffmeyer and Schreier, 2016c). In (Hüffmeyer and Schreier, 2016a) it is shown that the division of resource-policy-mappings and the actual access control logic helps to create a much better scaling access control systems. This is especially required for the Internet of Things, respectively, a Network of Things because devices are added and removed frequently.

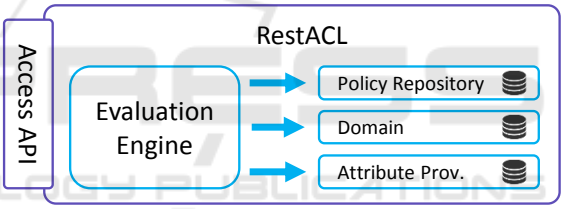

Figure 3: Architecture of RestACL.

An access control system that implements the language can be set up as a RESTful Service, too, enabling a self-protecting access control system. In this work, we present how such an access control system can be set up as a RESTful service itself. Access to web services (and the SitOPT system) is secured using access control services. These services also act as a user management component for the SitOPT system. Note that existing identity management solutions could be integrated, but since the access control mechanism is based on attributes, the access control services can perform the user management as well.

3 SitAC ARCHITECTURE

To ensure a fast and appropriate reaction to dedicated situations, users can register situations based on Situation Templates and get notified on their occurrence. On registration, users have to provide callback information that defines what subject is automatically in-

formed once a situation occurs. In the ambient assisted living scenario, for example, an elderly person might wear a necklace (a device) that includes a gyroscope and an accelerometer (the sensors). Through the aggregation of sensor data, it can be detected if the person falls to the ground. On detection, the situation recognition system raises an alert to all registered users. For example, if the accelerometer values show a peak and the gyroscope values oscillate, the elderly person might have fallen. But in some cases only the necklace might have fallen down. This can be classified as a *false-positive* situation detection. In order to double check whether an emergency situation occurred, the emergency contact requires remote access to the camera service mentioned in the scenario introduction (cf. Figure 1). If the camera pictures show the resident moving around, it is likely the case that there is not an emergency situation. The emergency contact can verify this with a phone call instead of immediately sending an ambulance. To enable this, dedicated services or sensor values, such as remotely-accessible cameras, must be accessible dependent on a certain situation for a limited group of people.

3.1 Requirements

An access control system for such a scenario must be very efficient and flexible. That means, the system must be capable to determine access decisions in short time to ensure that a service request provides up-to-date data. This must be guaranteed even for large amounts of services like cameras or any other web service. Furthermore, the system must be flexible in terms of adding and removing services, which means that the system must easily support the creation, change, and removal of services as well as policies that restrict access to these services and the produced data. Moreover, the access control system must be capable to express rich variations of access policies in order to embody multiple access control requirements. We identified four requirements a situation-aware access control system for RESTful services must fulfill. If the requirements are met, the benefits of REST are not violated and situation-aware access control can be implemented in an efficient manner.

(R1) *Request-Based Authorization*: If authorization is not done on a request base, for example, if access is granted or prohibited using tokens or along sessions, this might lead to access decisions that are not conform to the actual security policy. Imagine, a resource changes its state between two access requests from the same subject in a manner that the second request is denied while the first access was granted. Especially in a situation-aware context, the access rights might change if a dedicated situation occurs. If the second request is not independently evaluated to the first one, a (temporarily) unauthorized access might be the result or access might be prohibited even if the actual context would grant access. Therefore, authorization has to be done on a request base from the client perspective.

(R2) *Quick reaction*: The states of devices, sensors and other resources including their access rights can change frequently depending on the contextual environment (the situation). Hence, the access control system must be capable to quickly react to those changes. Imagine, a device or a sensor changes its state (e.g., a sensor value changes) and an access request is performed immediately after the state change. The system must respond with an access decision that belongs to the new state of the resource. Therefore, the system must support a tight integration of the situational context. In addition, a fast application of changes to the security policy and the given attributes must be guaranteed.

(R3) *Expressive strength*: A situation-aware access control system must enable high flexibility and fine-grained access control. As we can see from the ambient assisted living example, situation-dependent and situation-independent policies might coexist in the same context. Both types of policies must be supported by the access control system. Therefore, the access control system must support the application of rich variations of access rules based on various attributes of situation, subject, resource, environment or similar entities.

(R4) *Integration and administration*: The situation-aware access control system must support a tight integration of new devices, sensors, and Situation Templates as well as flexible administration of existing ones. Therefore, a carefully chosen set of initial policies and attributes must be created during the registration process. In addition, a highly structured and well-designed administration interface is required that enables access to management actions for humans as well as for automated systems.

3.2 SitAC Modeling with ABAC

Because ABAC is an ideal candidate for a flexible access control model, a situation-aware access control system can rely on the ABAC model. In situation-aware access control, a situation becomes an own category similar to subjects or resources. Entities of that category have dedicated attributes. For example, the SitOPT system notifies the Access Control System in case a situation has occurred and also in case a situation is no longer occurring. Therefore, a sit-

uation has *occurred* and *time* attributes that indicate whether a situation is currently occurring and the time at which the last change to the occurrence has happened. These attributes are delivered from the SitRS to the RestACL system. The RestACL system passes the attributes to its Attribute Providers to store them.

Because the SitAC system must be capable to grant access for a dedicated period of time after the occurrence of a situation, a situation also requires an *accessInterval* attribute. For example, during the situation registration process (cf. Section 4.3), a subject might create a policy that grants access if an emergency situation has occurred within the previous 20 minutes. The *accessInterval* attribute can be interpreted as a sort of counter that expires after a dedicated time period (measured from the moment the situation occurred). Once this timer is expired, access is not further granted or respectively prohibited. The *accessInterval* attribute is initially created during the registration process for a Situation Template but can be manipulated by the subject that registered the Situation Template at any time. Using the *occurred* attribute enables to revoke access rights before the *accessInterval* expires. For example, if the SitOPT system sends a notification that the emergency situation is no longer given, the *occurred* attribute is set to *false* and the situation-dependent access is revoked. That means, the decision whether a member of the rescue service can access the video camera depends, among others, on the attributes *occurred*, *time*, and *accessInterval* of the situation. For example, the following attributes are given.

Table 1: Situation Entity Example.

Att.	Category	Designator	Value
a_1	situation	id	123
a_2	situation	occurred	true
a_3	situation	time	12:00:00 01.01.2017
a_4	situation	accessInterval	1200000 ms

Note that a situation entity always must have the three attributes *occurred*, *time* and *accessInterval*. Otherwise, a situation-aware computation of access decisions is not possible. In addition, the time the access request arrives is required, too. The actual time is an attribute of the environment context. For example, if a request arrives at *12:10:00 01.01.2017* the environment time attribute is:

Table 2: Environment Attribute Example.

Att.	Category	Designator	Value
a_5	environment	time	12:10:00 01.01.2017

Note that access not only depends on the presence of dedicated attributes. Attribute values can also be related to each other. For example, the *environment time* attribute must provide a value that is between the *situation time* and the *situation time* plus the *accessInterval*. Access is granted right at the moment the situation occurred (indicated by the situation's *time* attribute). For example, we can say that:

$$v_{a_2} = true \tag{1}$$

and

$$v_{a_3} < v_{a_5} < v_{a_3} + v_{a_4} \tag{2}$$

must be fulfilled in order to grant access. In Section 4.3, a detailed example is given how these attributes and the relations are expressed within a policy.

3.3 Architecture

Figure 4 depicts the components that are required to combine the access control and situation recognition systems into a situation-aware access control system. The architecture is divided into three layers that are built on top of physical and environmental objects in the real world: a service layer, a security layer, and a client layer. Every layer provides RESTful services to the upper layer, respectively, to the public.

The service layer covers the services that need to be protected. For example, in the ambient assisted living scenario, the camera service is located in the service layer. In general, the service layer carries any type of RESTful services that requires permanent or situation-dependent protection. Note that the SitRS provides various services to manage situation recognition. For example, these services include sensor or Situation Template registration as well as access to the RESTful services provided by the RMP to access devices. These are located on the service layer as well. As described previously, situations can be registered including contact information (callbacks) that describe which subjects need to be informed in case of a situation occurrence. The SitRS periodically monitors whether the situation occurred.

The security layer is responsible for guaranteeing that only privileged subjects have access to the general services and the SitOPT system. Therefore, Enforcement points inspect every incoming request to the general services as well as the SitOPT system and consult the RestACL system whether the request can be permitted. Therefore, an Enforcement point formulates an access request and passes it to the Access API of the RestACL system. The access request contains at least the resource and subject identifiers and optionally a situation identifier. The RestACL system then identifies the policies that need

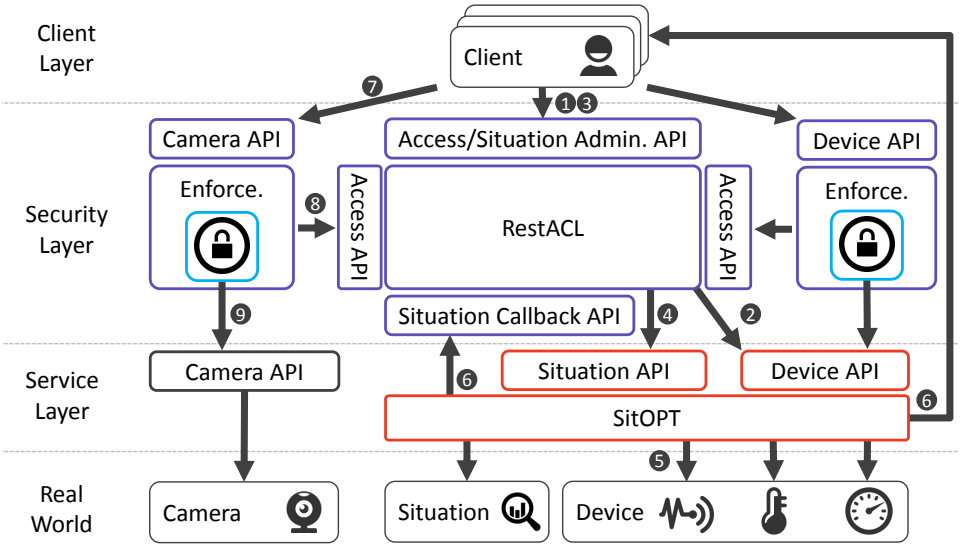

Figure 4: Architecture for situation-aware access control.

to be evaluated. During the evaluation process, the RestACL system uses the identifiers from the access request to load a list of attributes for each entity from its Attribute Providers. The actual access decision is computed based on the identified policies and the given attributes. This decision is returned back to the Enforcement points. That means, the Enforcement points provide the same service interface as their related web services. They only add the execution of access control logic to these services. The Enforcement points either rejects or forwards a request.

If a client registers devices, sensors or Situation Templates using the Access/Situation Administration API, the security layer checks if the client is allowed to perform this action (e.g., register a sensor for a dedicated device). If the client has the permission to perform the action, the RestACL system forwards the request to the SitOPT system. In order to provide situation-aware access decisions, the RestACL system registers itself as a situation callback for any situation. That means, the SitRS informs not only dedicated clients about all changes in the occurrence of a situation but also the RestACL system. The RestACL system then updates the related situation entities stored in its Attribute Providers and once an access request arrives, the RestACL system can perform a situation-aware policy evaluation.

As depicted in Figure 4, the situation-aware access control system works as follows: 1) A client registers devices (including sensors) at the Access/Situation Administration API. The RestACL system creates initial attributes for these devices and sensors (cf. 4). 2) The RestACL system forwards the creation request to the SitOPT system. 3) A client registers a Situation Template at the Access/Situation Administration API. The RestACL system creates a situation entity for this Situation Template. 4) The RestACL system forwards the Situation Template creation request to the SitOPT system and registers itself as a callback. 5) The SitOPT system monitors the devices (respectively their sensors). 6) If the situation occurs, the SitRS informs all callbacks (clients as well as the RestACL system) about the occurrence. 7) A client tries to access a device or another resource like the camera. 8) The related Enforcement point creates an access request and sends it to the RestACL system. 9) Depending on the result, the Enforcement point either forwards the request to the resource or rejects the client's request.

As part of the integration work for the two systems, an administration component is required that enables various types of clients to control policies, attributes and their assignments to devices and sensors. For example, human users might want to register devices and manage their access rights through a web application while other machines might want to register or deregister devices in an automated fashion using basic REST calls. The Access and Situation Administration component creates initial policies, attributes and resource assignments during the registration procedure for devices, sensors or Situation Templates. In addition, this component offers an API for policy, attribute and situation administration. Note that this component is not part of the generic ABAC system but is rather tailored for the situation-aware system. Details about the registration procedure are described in Section 4.

Table 3: SitAC services.

Action	API	Description
Register situation	Situation	Clients want to describe situations in which they are called back.
Deregister situation	Situation	Clients want to deregister situations in which they are called back.
Register device	Situation	Clients want to register devices.
Deregister device	Situation	Clients want to deregister devices.
Register sensor	Situation	Clients want to register sensors associated with a device.
Deregister sensor	Situation	Clients want to deregister dedicated sensors.
Access sensor values	Device	Clients want to access sensors resp. a snapshot of the actual values produced by the sensor.
Access services	Service	Clients want to access (read, update or delete) services like the camera for which situation-aware access is granted.
Register services	Access	Clients want to register services for which situation-aware access should be granted.
Register user	Access	Clients want to register themselves.
Create attributes	Access	Clients want to create attributes for themselves as well as attributes for the devices, sensors and services that they own.
Update, delete attributes	Access	Clients want to update or delete attributes that are used to determine access to devices and sensors.
Create policies	Access	Clients want to create policies that are used to determine access to devices and sensors. The policies contain the actual access logic for dedicated sensors.
Update, delete policies	Access	Clients want to update or delete policies that are used to determine access to devices and sensors.
Assign policies	Access	Clients want to change the mapping between policies and devices or sensors. Note that policies can be assigned to many devices, sensors or services while attributes are related to exactly one entity.

3.4 SitAC Services

Since the situation-aware access control system offers multiple RESTful services, a client can perform several actions in such an environment. Firstly, the client can perform situation recognition related operations like the registration of devices, sensors, or situations. Of course, the client must only perform those actions the subject is privileged for. Secondly, besides these situation recognition related operations, the client can also perform a set of access control related actions. Finally, the client can access the web services in case that the subject is privileged. Table 3 lists the different types of operations that can be performed by a client.

Services with the *Situation* type are offered by the SitOPT system. The security layer intercepts requests to those services and adds the execution of access control logic. If a client wants to perform one of those operations and the access control system permits the execution, the client's request is forwarded to the SitOPT system. Note that the RMP offers additional operations to update sensor values. Those operations are only accessible for the sensors but not from public. Therefore, the access control system does not mirror these operations to the public. Since one resource is directly mapped to one sensor and must only be updated by this sensor, there is no need for a fine-grained access control mechanism.

Actions with type *Access* are provided on the security layer. A client can change access restrictions to a user's devices, sensors or services/resources. Therefore, the client might assign new attributes to its own devices, sensors or services/resources using a `/attributes` subresource that is created for every resource. For example, a client might add a new attribute *location* to a device and refer this attribute in an access policy. In addition, the client can create new policies and assign those policies to the devices, sensors and services. Therefore, another subresource is created for each resource: the `/access` subresource.

Finally, actions of type *Service* are application service related operations and enable users to register services/resources that require situation-dependent access protection.

The access control system offers a REST interface that provides the aforementioned actions. Because the access control system is based on the RestACL language, which is designed to protect RESTful Services, the access control services can not only protect the SitOPT system and the web services, but also their

own REST API.

4 WALKTHROUGH OF THE EXAMPLE SCENARIO

To be able to perform authorization, it is essential to know the involved entities. In a resource-oriented environment, there are at least two involved entities: the requested resource and the requesting subject. While the resource can be identified using its URI, the requesting subject must authenticate itself to the Access Control Services.

4.1 Authentication

Authentication can be done using standardized HTTP authentication methods like basic or digest authentication or any other authentication method. Note that authentication is not required for every API call because some operations like the registration of a new device are not restricted to dedicated user. In case the subject does not authenticate itself and the operation requires authentication, the Firewall will return a HTTP 401 response. Note that user name and password do not need a special treatment in the access control system. They are regular attributes assigned to a dedicated subject. Once a user has authenticated him/herself, the access control system executes the access logic to determine whether the requested operation (e.g., read sensor data or update sensor access) must be performed. Therefore, the Access Control Enforcement point creates an access request containing the address of the requested resource (e.g., `/devices/1/sensors/accelerometer`), the access method (e.g., GET), and a subject identifier (e.g., `/users/1`). The access control system uses these properties to load various attributes from the Attribute Provider that are used in the evaluation process for an access request. For example, a subject might have an attribute *type* with the value *rescue* or a sensor (a resource) might have an attribute *location*. Given the identifier of the sensor and the identifier of the user, both can be uniquely identified and attributes like the previously mentioned ones can be loaded from an Attribute Provider. Note that the Access Control Services can employ multiple Attribute Providers for one request. Even external attribute sources like an external identity management are possible.

4.2 Thing, Sensor and Service Registration

If a user decides to create a new device composition that should be monitored for dedicated situations, the user must first register all devices and their sensors (cf. Figure 4 - Message 1). Therefore, the user sends exactly one registration request for each device to the access control system. This is done using a HTTP POST request as indicated below. The request must contain a unique device identifier, a list of device owners and optionally a device description. The RestACL system creates a new policy and limits access to the subjects of the owner list. After that, the system creates several new entries for the device within the Domain. Besides an entry for the device itself, the same policy is used to ensure that only one of the owners can register new sensors with that device or register new attributes. Finally, another entry ensures that only the owners can change the access rights for that device. In addition to the Domain entries, an owner list attribute is assigned to the device at the Attribute Provider. Note that a user can register multiple owners for one device. In such a case, all owners have access to the device and can change the access policies. Note that all these operations are executed within the RestACL system. Once the device is created at the RestACL system, the system forwards the creation request to the SitOPT system (cf. Figure 4 - Message 2).

For example, the elderly person from the ambient assisted living scenario might want to register its necklace at the SitOPT system. This can either be done by an expert or in an automated fashion. Note that an automated registration has not been implemented yet but is part of our future work. The person might be identified with the id *1* and therefore might be addressed using the path `/users/1`. The registration application needs to send a POST request to the RestACL system containing a unique device id and an owner list in the payload of the POST request (Message 1). Note that in this example the elderly person is the device owner.

```
POST /devices HTTP/1.1

{
  "deviceId" : "1234",
  "deviceDescription" : "necklace with sensor",
  "deviceOwners" : ["/users/1"]
}
```

When the request arrives, the RestACL system creates a new access policy (as indicated below) that grants access in case that the requesting subject is one of the subjects from the owner list. This policy is stored in the Policy Repository.

```
{
  "id": "P1",
  "effect": "Permit",
  "priority": "1",
  "condition": {
    "function": "equal",
    "arguments": [{
      "category": "subject",
      "designator": "uri"
    },{
      "value": "/users/1"
    }]
  }
}
```

In a second step, this policy is assigned to the new device (the necklace). Therefore, the RestACL system creates a new Domain entry (as indicated below) and stores the entry to its Domain database.

```
{
  "path": "/devices/1234",
  "access": [{
    "methods": ["GET"],
    "policies": ["P1"]
  }]
}
```

As a third step, the same policy is assigned to the `/sensor` subresource path of the necklace, too (as indicated below). The sensor subresource (e.g `/devices/1234/sensors`) must be used to register sensors like the accelerometer or the gyroscope with the necklace.

```
{
  "path": "/devices/1234/sensors",
  "access": [{
    "methods": ["POST"],
    "policies": ["P1"]
  }]
}
```

The same policy is also assigned to the `/attributes` subresource of the new device (e.g, `/devices/1234/attributes`). This subresource enables clients to store new attributes or to update attribute values at the Attribute Provider. In case that only external Attribute Providers are used, the binding between the attribute registration path and the policy is not required. But since the reference implementation uses its own Attribute Provider, clients need an API to update their own attributes as well as their devices' and sensors' attributes. Access to this API must be limited, too. Therefore, the binding is required.

In the fifth step, the policy is assigned to the `/access` subresource of the new device (e.g. `/devices/1234/access`). This subresource enables clients to assign new access policies to the device. For example, if the elderly person wants to grant access to the rescue service, a new policy for the device must be set using this URI.

Lastly, the new device gets an owner attribute. This attribute must be stored together with all other attributes of the device (like the mandatory *id* attribute) at the Attribute Provider.

```
{
  {
  "category" : "resource",
  "designator" : "id",
  "value" : "/devices/1234"
  },
  {
    "category" : "resource",
    "designator" : "deviceOwners",
    "value" : ["/users/1"]
  }
}
```

The service registration and the sensor registration procedures are very similar to the device registration procedure and therefore are not discussed in detail. But there is one big difference that should be noted: sensors are associated with devices and therefore can only be registered if the access policies for the associated device are fulfilled. Sensor registration is done using a POST request to the sensor list of a device (e.g., `/devices/1234/sensors`). The RestACL system checks whether access to the sensor list is permitted (checking the policies that have been assigned as described above). Note that during the device registration, an initial assignment is created (step 3) that restricts access to that sensor list to the device owners. Therefore, the owners can determine who might register new sensors for that device. Besides the execution of access logic, the registration process for devices and sensors is the same. That means, also for sensors a new policy is created that initially restricts access to the device owners and this policy is assigned to the sensor resource itself as well as to the `/attributes` and `/access` subresources of the sensor.

4.3 Situation Policy Example

If a subject wants to restrict access to a dedicated service in a situation-aware fashion, the subject must register a Situation Template. To do so, the subject must have the permission to read any of the sensor values defined in the template, otherwise the situation cannot be registered. During the registration process, the RestACL system creates a situation entity (and stores it to its Attribute Provider). This entity is used to store situation-aware access information. For example, the entity stores the information if the situation has occurred and the point of time it has occurred. These information can be taken into account during the evaluation procedure of an access request. For example, the RestACL policy shown below is used in the am-

bient assisted living scenario to protect access to the camera service.

```
{
  "id": "PEmergency",
  "description": "A policy that grants access
in case of a emergency situation. The policy
can be applied to the camera resource."
  "effect": "Permit",
  "priority": "2" ,
  "compositeCondition": {
    "operation": "AND",
    "conditions": [{
      "function": "equal",
      "arguments": [{
        "category": "subject",
        "designator": "type"
      } ,{
        "value": "rescue"
      }]
    },{
      "function": "equal",
      "arguments": [{
        "category": "situation",
        "designator": "occurred"
      },{
        "value": "true"
      }]
    },{
      "function": "between" ,
      "arguments": [{
        "category": "situation",
        "designator": "time"
      },{
        "category": "environment",
        "designator": "time"
      },{
        "function": "add",
        "arguments": [{
          "category": "situation",
          "designator": "time"
        },{
          "category": "situation",
          "designator": "accessInterval"
      }]
    }]
  }
}
```

The policy grants access in case that three conditions are fulfilled: 1) The requesting subject must have a *type* attribute with the value *rescue* indicating that the subject is a member of an emergency service. 2) the situation must have occurred. That means, the situation recognition service has informed the Access Control Services that the actual sensor value composition can be interpreted as the emergency situation. 3) The time the request arrived (indicated by the *environment time* attribute) must be in between the time at which the *situation occurred* and the time of the occurrence (indicated by the *situation time* attribute) plus the situation's *accessInterval* attribute.

The second and third condition require that situation attributes must be up-to-date. Therefore it is crucial that the SitRS informs the RestACL system in case that the occurrence of a situation has changed (either the *situation occurred* attribute value switches from *false* to *true* or vice versa). The SitRS is capable of informing one or more callback receivers in case that the situation occurs. In order to provide up-to-date situation attributes, the RestACL system registers itself as a callback for each situation. If a situation occurs, the SitRS informs the RestACL system, which forwards the updated situation entity information to the Attribute Provider.

4.4 Service Access

In the above mentioned scenario, standardized REST clients can be used to request service data from default web services. Those web services are protected by an upstream Enforcement Point. That means, the client performs a resource request , e.g., a HTTP *GET*) request to the camera service and the Enforcement Point intercepts it (cf. Figure 4 - Message 7). After a successful authentication procedure, the Enforcement Point executes access logic by sending an access request to the RestACL system and enforcing the access decision (cf. Figure 4 - Message 8). The access request is derived from the resource request and enriched with attributes stored at the Attribute Provider. The enforcement is done by either forwarding the resource request to the web service (cf. Figure 4 - Message 9) or rejecting the initial resource request (e.g., returning a HTTP 403 response). Forwarding or rejection of the resource requests depends on the actual access decision of the RestACL system. This procedure allows to treat every request individually in terms of access control. This ensures that requirement **(R1)** is fulfilled and proper access decisions can be guaranteed in a situation-aware fashion.

5 EVALUATION

We used two methods to evaluate our approach: 1) We evaluated whether our system meets the requirements described in Section 3.1. 2) We did a non-formal verification that the system produces the expected results. Therefore, the system has been used to evaluate the ambient assisted living scenario. The interested reader is referred to (Hüffmeyer and Schreier, 2016a) for further performance evaluations of the RestACL system for increasing numbers of resources.

The situation-aware access control system is capable of handling each service/resource request individ-

Table 4: Access types.

Access Type	s_1	s_2	s_3	s_4
Permanently grant	permit	permit	permit	permit
Permanently forbid	deny	deny	deny	deny
Temporary grant	deny	permit	deny	deny
Temporary forbid	permit	deny	permit	permit

ually. Every single remote request is forwarded to the access control system to ensure that the system cannot be bypassed. Because the system does not employ concepts like tokens, we can ensure that every request is evaluated individually by the access control system as required in (**R1**). This suits with the stateless communication principles of REST and of the situation-aware context very well.

For validation purposes both, the situation recognition system and the attribute-based access control system have been set up on the same computer. Running both systems on the same computer offers the benefit that network runtimes can be eliminated. That means, there is no additional delay between the time a situation occurred and the time access is granted. Therefore, granting and revoking access permissions can be reduced to setting attribute values. This can be performed in less than 1 ms, because it is a very basic operation. This ensures that proper access rights are assigned immediately after a situation occurrence and before a situation related access request can arrive. Therefore, we can see **(R2)** as fulfilled.

We created a solution that allows the coexistence of situation-dependent and situation-independent access policies. Both types of policies are not limited to a dedicated area, because they can also include any type of attributes. This enables a very flexible access control system that can be used in various fields like ambient assisted living, Industry 4.0, smart cities or any event-driven environment. Attributes and their categories can be freely selected and logically combined to express rich variations of access rights as required in (**R3**).

Section 4 describes the registration procedure in detail. Using this procedure guarantees that devices and sensors are never accessible without any access control inspections as required by **(R4)**.

In Section 1, we mentioned that there are different types of access cases. Access decisions are either permanently or temporary and access decisions either grant or prohibit access. That means, we conducted several tests that proved whether the system works properly. For each access type (permanently granted, permanently forbidden, temporarily granted, temporarily forbidden), it must be guaranteed that the system computes the right access decision before a situation has occurred (s_1), after the situation has occurred (s_2), after the access end has arrived (s_3) and after the situation occurrence has switched back (s_4). Table 4 lists the expected access decision for each test case. We could successfully verify that the system works properly with these test cases.

6 RELATED WORK

Situation-aware access control is a topic that has only been rarely discussed. However, there are a few publications that are associated.

A specialized Situation-Aware Access Control model is described in (Peleg et al., 2008; Beimel and Peleg, 2011). In this work, a health-care oriented model is presented. They describe a situation schema that is used to compute an access decision. The schema contains for example Patients, Data-Requesters or Health Records. Access is granted depending on several contextual factors rather than only depending on the role of the Data-Requester. They describe a specialized solution in medical environments, which extends the role-based access control model to include a medical context. So, this work presents a specialized solution rather than a generic approach for situation-aware access control.

Situation-aware Access Control for autonomous decentralized systems is described in (Yau et al., 2005). In this approach, a role-based access control model is used to enforce access control in dynamic and large scale information systems. The authors extend the hierarchical role-based model described in (Ahn and Sandhu, 2000). They introduce situation constraints that are applied to the relation between Users and Roles as well as the relations between Roles and Permissions. However, role-based access control requires careful role engineering and might lead to role over-engineering if a great diversity of access conditions must be expressed (Jin et al., 2012; Yuan and Tong, 2005).

The eXtensible Access Control Markup Language (XACML) is an OASIS standard that describes one way to apply ABAC. XACML policies are arranged in tree structures and XACML engines have to traverse multiple branches of that tree for every access

request. While XACML can be seen as a compositional approach that computes the union of several access rules that can be applied to a request (Ramli et al., 2012), RestACL divides the security policy of an application into access control logic on the one side and the identification of policies on the other side. RestACL was designed to fit the characteristics of RESTful Services such as unique identifiers, a uniform interface for resources and hypermedia as the engine of application state (HATEOAS) (Hüffmeyer and Schreier, 2016a; Hüffmeyer and Schreier, 2016b). Therefore, RestACL supports the requirements of the application scenario in a much more natural fashion when compared with XACML.

Glombiewski et al. (Glombiewski et al., 2013) present an approach similar to the situation recognition system of SitOPT by integrating context from a wide range of sources for situation recognition. However, they do not provide any abstraction for situation modeling, i.e., situations need to be defined using complex CEP queries. This proves difficult, especially for domain experts, e.g., in ambient living scenarios, who do not have extensive computer science knowledge. Furthermore, (Hasan et al., 2011) propose to use CEP along with a dynamic enrichment of sensor data in order to realize situation-awareness. In this approach, the situations of interest are directly defined in the CEP engine, i.e., the user formulates the situations of interest using CEP query languages (Hasan et al., 2011). A dynamic enrichment component processes and enriches the sensor data before the CEP engine evaluates them against the situations of interest. In the SitOPT approach, we provide an abstraction by Situation Templates (Häussermann et al., 2010) and a graphical interface (Franco da Silva et al., 2016) for situation modeling.

Many similar approaches exist that use ontologies for situation recognition (Wang et al., 2004). However, these approaches are either focused on specific use case scenarios (Brumitt et al., 2000) or cannot provide the efficiency required by real-time critical scenarios (Wang et al., 2004; Dargie et al., 2013), e.g., in the ambient assisted living scenario where situations need to be recognized timely. These limitations regarding efficiency occur in machine learning approaches (Attard et al., 2013), too. In contrast, the SitOPT approach offers high efficiency by recognizing situations in milliseconds (Franco da Silva et al., 2016) instead of seconds or even minutes as reported in (Wang et al., 2004; Dargie et al., 2013). This enables applicability in time-critical real-world scenarios in which recognition times are of vital importance.

The authors of (Yazar et al., 2014) present a low-cost Ambient Assisted Living system using vibration and passive infrared sensors for falling detection and other use-cases. The system works in real-time, but security aspects are not considered at all. On the one hand this fall detection system could be used in our use-case and on the other hand our SitAC approach would improve the security of the presented system.

7 SUMMARY AND FUTURE WORK

This work introduces a situation-aware access control system to protect sensor data and other RESTful services. It describes a way how situation-aware access control can be implemented using the generic attribute-based approach by introducing the situation as a dedicated category. The ABAC system is loosely coupled to a situation recognition system that provides up-to-date situation information.

The designed and implemented solution provides situation-aware access control including request-based authorization, the application of frequent policy and situation changes, and the expressive strength of an attribute-based access control system. Besides these benefits, the system performs very well in tests. Because of its generic design based on the ABAC model, the system can be used in different scenarios. We described an ambient assisted living scenario, but the application context of situation-aware access control is widely spread. As we already mentioned in the beginning, other interesting areas are industry 4.0, smart cities or any other event-driven environment. We conduct a comprehensive evaluation of the SitAC system in those environments in our future work.

In our future work, we want to ease the registration and administration processes. As we have previously mentioned in the example scenario, experts are required to perform the registration of devices and sensors. This is acceptable in machine-to-machine scenarios, where machines are the clients of the situation-aware access control system. Such systems are usually set up once by experts. It is not suitable, for example, in the ambient assisted living scenario because the clients are usually no experts. Therefore, we want to develop mechanisms that simplify the registration as well as administration processes to better support non-expert clients. The simplification of those processes must also target the task of policy management to enable clients to easily manage the access policies of their resources. Besides that, we want to have more detailed analysis of other scenarios like Industry 4.0 and investigate whether the presented solution fits newly upcoming requirements.

REFERENCES

Ahn, G.-J. and Sandhu, R. (2000). Role-Based Authorization Constraints Specification. *ACM Transactions on Information and System Security, Vol. 3, No. 4*.

Attard, J., Scerri, S., Rivera, I., and Handschuh, S. (2013). Ontology-based situation recognition for context-aware systems. In *Proceedings of the 9th International Conference on Semantic Systems*, pages 113–120. ACM.

Beimel, D. and Peleg, M. (2011). Using OWL and SWRL to represent and reason with situation-based access control policies. *Data & Knowledge Engineering*, Vol. 70, Issue 6.

Brumitt, B., Meyers, B., Krumm, J., Kern, A., and Shafer, S. (2000). EasyLiving: Technologies for Intelligent Environments. In *Handheld and Ubiquitous Computing*. Springer Berlin Heidelberg.

Dargie, W., Mendez, J., Mobius, C., Rybina, K., Thost, V., Turhan, A.-Y., et al. (2013). Situation Recognition for Service Management Systems Using OWL 2 Reasoners. In *Proceedings of the 10th IEEE Workshop on Context Modeling and Reasoning 2013*, pages 31–36. IEEE Computer Society.

Ferraiolo, D., Kuhn, R., and Hu, V. (2015). Attribute-Based Access Control. In *Computer, Vol.48*. IEEE Computer Society.

Franco da Silva, A. C., Hirmer, P., Wieland, M., and Mitschang, B. (2016). SitRS XT – Towards Near Real Time Situation Recognition. *Journal of Information and Data Management*.

Glombiewski, N., Hoßbach, B., Morgen, A., Ritter, F., and Seeger, B. (2013). Event Processing on your own Database. In *BTW workshops*, pages 33–42.

Hasan, S., Curry, E., Banduk, M., and O'Riain, S. (2011). Toward Situation Awareness for the Semantic Sensor Web: Complex Event Processing with Dynamic Linked Data Enrichment. *SSN*, 839:69–81.

Häussermann, K., Hubig, C., Levi, P., Leymann, F., Simoneit, O., Wieland, M., and Zweigle, O. (2010). Understanding and designing situation-aware mobile and ubiquitous computing systems. *Proc. of intern. Conf. on Mobile, Ubiquitous and Pervasive Computing*, pages 329–339.

Hirmer, P., Wieland, M., Breitenbücher, U., and Mitschang, B. (2016a). Automated Sensor Registration, Binding and Sensor Data Provisioning. In *Proceedings of the CAiSE'16 Forum, at the 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016)*.

Hirmer, P., Wieland, M., Breitenbücher, U., and Mitschang, B. (2016b). Dynamic Ontology-based Sensor Binding. In *Advances in Databases and Information Systems. 20th East European Conference, ADBIS 2016, Prague, Czech Republic, August 28-31, 2016, Proceedings*, volume 9809 of *Information Systems and Applications, incl. Internet/Web, and HCI*. Springer International Publishing.

Hirmer, P., Wieland, M., Schwarz, H., Mitschang, B., Breitenbücher, U., and Leymann, F. (2015). SitRS - A Situation Recognition Service based on Modeling and Executing Situation Templates. In Barzen, J., Khalaf, R., Leymann, F., and Mitschang, B., editors, *Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing*, volume RC25564 of *Technical Paper*. IBM Research Report.

Hüffmeyer, M. and Schreier, U. (2016a). Analysis of an Access Control System for RESTful Services. *ICWE '16 - International Conference on Web Engineering*.

Hüffmeyer, M. and Schreier, U. (2016b). Formal Comparison of an Attribute Based Access Control Language for RESTful Services with XACML. *SACMAT '16 - Symposium on Access Control Models and Technologies*.

Hüffmeyer, M. and Schreier, U. (2016c). RestACL - An Attribute Based Access Control Language for RESTful Services. *ABAC '16 - Proceedings of the 1st Workshop on Attribute Based Access Control*.

Jin, X., Krishnan, R., and Sandhu, R. (2012). A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. *DBSec '12 - Proceedings of the 26th Annual Conference on Data and Applications Security and Privacy*.

Peleg, M., Beimel, D., Dorib, D., and Denekamp, Y. (2008). Situation-Based Access Control: Privacy management via modeling of patient data access scenarios. *Journal of Biomedical Informatics*, Vol. 41, Issue 6.

Ramli, C. D. P. K., Nielson, H. R., and Nielson, F. (2012). The Logic of XACML. In *Lecture Notes in Computer Science - Formal Aspects of Component Software*. Springer.

Wang, X., Zhang, D. Q., Gu, T., and Pung, H. (2004). Ontology Based Context Modeling and Reasoning Using OWL. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*. IEEE Computer Society.

Wieland, M., Schwarz, H., Breitenbücher, U., and Leymann, F. (2015). Towards Situation-Aware Adaptive Workflows. In *Proceedings of the 13th Annual IEEE Intl. Conference on Pervasive Computing and Communications Workshops: 11th Workshop on Context and Activity Modeling and Recognition*. IEEE.

Yau, S. S., Yao, Y., and Banga, V. (2005). Situation-aware access control for service-oriented autonomous decentralized systems. In *Proceedings of the 2005 International Symposium on Autonomous Decentralized Systems*, ISADS '05.

Yazar, A., Erden, F., and Cetin, A. E. (2014). Multi-sensor ambient assisted living system for fall detection. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP14)*, pages 1–3. Citeseer.

Yuan, E. and Tong, J. (2005). Attribute based access control (ABAC) for Web services. *ICWS '05 - International Conference on Web Services*.