Symbiosis of smart objects across IoT environments

688156 - symbIoTe - H2020-ICT-2015

Resource Trading, Security and Federation Mechanisms

The symbIoTe Consortium

Intracom SA Telecom Solutions, ICOM, Greece
Sveučiliste u Zagrebu Fakultet elektrotehnike i računarstva, UNIZG-FER, Croatia
AIT Austrian Institute of Technology GmbH, AIT, Austria
Nextworks Srl, NXW, Italy
Consorzio Nazionale Interuniversitario per le Telecomunicazioni, CNIT, Italy
ATOS Spain SA, ATOS, Spain
University of Vienna, Faculty of Computer Science, UNIVIE, Austria
Unidata S.p.A., UNIDATA, Italy
Sensing & Control System S.L., S&C, Spain
Fraunhofer IOSB, IOSB, Germany
Ubiwhere, Lda, UW, Portugal
VIPnet, d.o.o, VIP, Croatia
Instytut Chemii Bioorganicznej Polskiej Akademii Nauk, PSNC, Poland
NA.VI.GO. SCARL, NAVIGO, Italy
Universität Zürich, UZH, Switzerland

For more information on this document or the symbIoTe project, please contact:
Sergios Soursos, INTRACOM TELECOM, souse@intracom-telecom.com

Document Control

Title: D3.2 - Resource Trading, Security and Federation Mechanisms

Type: Public

Editor(s): Jose Antonio Sanchez / Joaquin Iranzo Yuste

E-mail: jose.sanchezm@atos.net / joaquin.iranzo@atos.net

Author(s): Joaquin Iranzo (Atos), Jose Antonio Sanchez (Atos), Joao Garcia (UW), Pietro Tedeschi (CNIT), Giuseppe Piro (CNIT), Gennaro Boggia (CNIT), Giuseppe Bianchi (CNIT) Michał Pilc (PSNC), Mikołaj Dobski (PSNC), Vasilis Glykantzis (ICOM), Christoph Ruggenthaler (AIT), Pavle Skocir (UniZG-FER)

Doc ID: D3.2 - Resource Trading, Security and Federation Mechanisms.doc

Amendment History

Version	Date	Author	Description/Comments
v0.1	23/10/2017	Jose Antonio Sanchez	Initial ToC and assigned sections.
V0.2	13/11/2017	All	First version of the content to be consolidated
V0.3	24/11/2017	All	Consolidation of Federation, security and introduction
V0.4	04/12/2017	All	Consolidation of Bartering section and alignment with the Federation
V0.5	11/11/2017	Joaquin Iranzo Yuste	Consolidated version to be reviewed internally by the WP3 partners.
V0.6	13/12/2017	Jose Antonio Sanchez	Contributions by partners completed
V0.7	14/12/2017	Mikołaj Dobski, Pietro Tedeschi	Grammar corrections
V0.8	14/12/2017	Jose Antonio Sanchez	Consolidated version ready for final review
V0.9	15/12/2017	Michał Pilc	Proof reading
V1.0	22/12/2017	Jose Antonio Sanchez	Final version
V1.1	05/01/2018	Sergios Soursos	Final editing for submission-ready version

Legal Notices
The information in this document is subject to change without notice.
The Members of the symbIoTe Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the symbIoTe Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Table of Contents

Table of Figures

Table of Tables

(This page is left blank intentionally.)

1 Executive Summary

This deliverable describes the architecture and mechanisms that allow a set of IoT platforms to form federations in which they may share resources directly with minimal control from a central entity, i.e., the symbIoTe Core Services instance. Furthermore it explains the different forms of resource sharing, be it freely or through bartering, as well as the security features that we have implemented in order to achieve this result.

In symbIoTe, a federation is a set of IoT platforms that collaborate to share their IoT resources with the rest of the group, providing added value to native and adapted applications by broadening the number and diversity of resources that a single platform cannot achieve on its own. By joining one or more of such federations, a symbIoTe-enabled platform will offer the possibility to create new types of applications that expand beyond its own initial capabilities and it will do so in a secure, controlled and reliable way.

Platforms may choose to share their resources freely with the rest of the federation, by guaranteeing certain Quality of Service levels to be provided during access to these resources, or by bartering them for future access to another platform's resources. For resources shared for free, Service Level Agreements will guarantee that the access to these resources meets a minimum level of quality related to availability and performance, so we can rule out the possibility of platforms sharing only the less valuable or even the malfunctioning devices that they may have. For bartering, resource access is guaranteed as long as the rest of the platforms in the federation allow access to some of their resources in a fair way and failing to do so will affect their reputation.

All interactions between the federated platforms happen without a central registry and access is granted directly from platforms to applications without any kind of intermediary or centralized registry. This means that security needs to be enhanced in order to be sure that resources are discoverable and accessible only by the authorized users who are registered and recognized by, at least, one of the platforms inside a federation. By extending the symbIoTe security framework to work in a purely distributed environment, we are able to provide fine grain access control that is based on three pillars:

- Certificates for authentication of both applications and components,
- JSON Web Tokens (JWT) for authorization,
- A challenge-response protocol to verify the authenticity of both sides in every interaction.

Finally, an anomaly detector monitors the interactions between the platforms in the federation and, together with resource health metrics and access statistics, feeds the Trust Manager component, which will assign trust values to resources and foreign platforms as a whole. This will further enhance the reliability and security of the federations, and helps to identify unreliable resources and platforms.

This document provides a detailed description of the interactions and mechanisms of platform federations in symbIoTe and introduces the components that we have designed in order to achieve them. It starts by providing a general overview of the federation concept in symbIoTe, with details about the main processes like federation management, resource sharing, trust calculation or Quality of Service assessment, to later give a detailed explanation of the bartering and trading implementation. Finally, we provide a comprehensive description of the security features that we have implemented in order to execute all this interactions in secure and reliable way.

2 Introduction

2.1 Purpose of the Document

The purpose of this deliverable is to provide a detailed description of the implementation of federation mechanisms in symbIoTe, its advantages and features with special emphasis on the resource sharing and bartering functionality and the security issues associated to it. They will be discussed in detail, providing implementation solutions for each one of these aspects. This deliverable also serves as the basis for the future work towards the final implementation of symbIoTe federations.

2.2 Structure and Overview

Section 3 summarizes the concepts associated to symbIoTe architecture of levels and domains, putting a special attention to the Level 2 of compliance as it describes the main features, interactions, components and adaptations needed to achieve it, both in the centralized symbIoTe Core as well as in each individual platform that wants to be federated.

Section 4 introduces the concept of federations and the mechanisms that we will use to implement them. These mechanisms will include the management of Quality of Service parameters through Service Level Agreements and the reputation calculation for platforms and resources.

Section 5 talks about bartering and trading mechanisms and provides a detailed description of the implementation for bartering, including sequence diagrams and architectural components and interactions.

Section 6 describes the security issues relevant to platform federations and the mechanisms that we are implementing to overcome them.

Section 7 gives conclusions and highlights future work in all relevant tasks towards the final implementation of platform federations within symbIoTe.

At the end of the document, three annexes provide highly detailed descriptions of concepts presented at high level within the deliverable.

3 Architecture for IoT platform federations

This section will describe the high-level architecture that we envision for IoT platform federations based on the principles and components provided by symbIoTe. This architecture will allow these platforms to share resources among them in a reliable, controlled and secure way with minimal intervention of a centralized core. Furthermore, with some work from platform owners, native applications already existing could benefit from symbIoTe, broadening the number of resources they can access in a transparent way. We will describe how this can be achieved first, by providing an overview of the symbIoTe concepts of domains and levels of compliance, then focusing on Level 2, which is the one allowing platform federations and finally, by describing the components that make all this possible.

3.1 Overview

The symbIoTe vision is based on two concepts: domains and levels of compliance. There are four different domains and four levels of compliance. Although the numbers match, they are not equivalent one to one. While the notion of domains is closer to architectural concepts and location of components, levels of compliance refer to different interoperability features achieved by the different platforms collaborating through the symbIoTe stack.

There are four domains in symbIoTe which are shown in Figure 1.

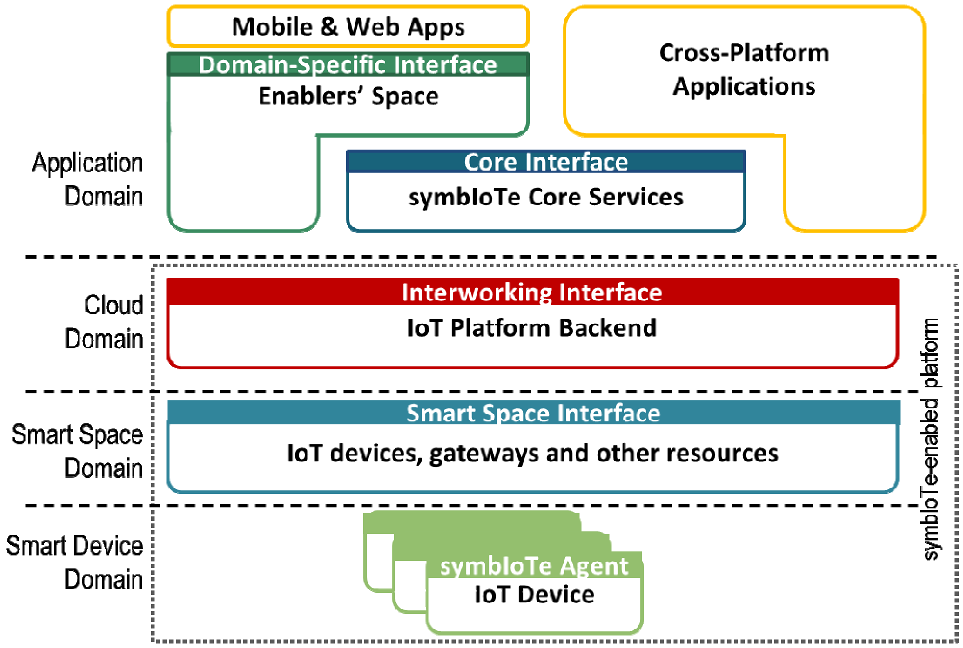

Figure 1. symbIoTe General Architecture

- **Application Domain:** Refers mainly to the Core Services and allows platforms to advertise IoT resources in a central registry. Domain specific enablers provide extra functionalities for concrete use cases.
- **Cloud Domain:** Provides standardized access to IoT platforms through the Interworking API.

- **Smart Space Domain:** Provides discovery and registration services for IoT devices in smart spaces.
- **Smart Device Domain:** Provides services and capabilities needed by IoT devices to successfully roam between smart spaces.

A more complete description of this domains and their architecture can be found in Deliverable D1.4 [1].

This is one of the pillars of the symbIoTe architecture. Another one is the levels of compliance, which are shown in Figure 2.

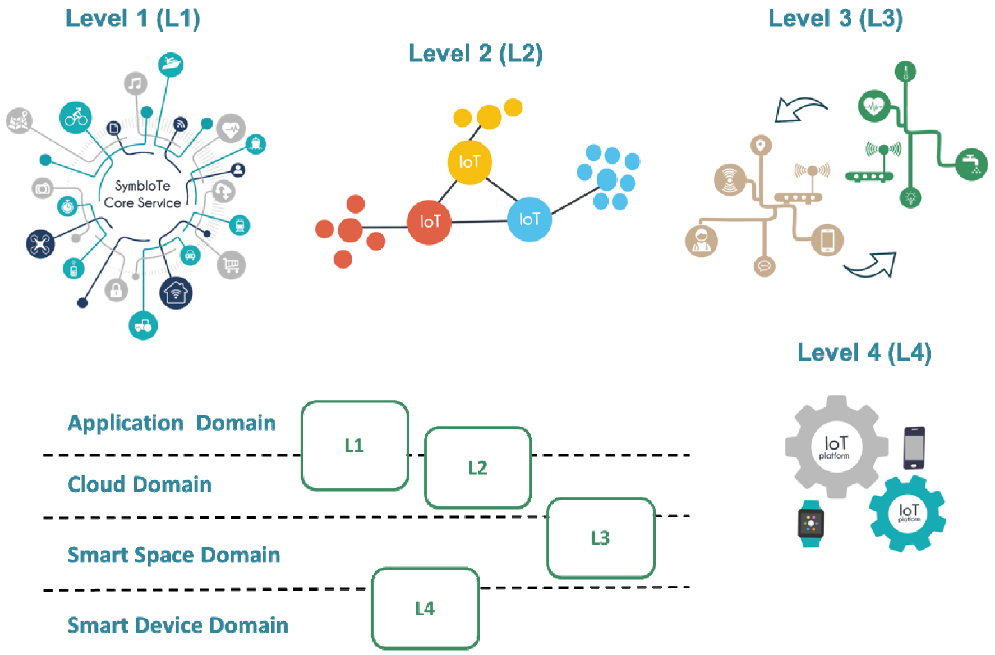

Figure 2 Levels of compliance in symbIoTe

We distinguish four levels of compliance [1]:

- **Level 1 (L1) - Syntactic and semantic interoperability:** Platforms provide an interworking interface, which allows them to advertise their resources in the core, providing access to third parties through the core services and achieving syntactic and semantic interoperability;
- **Level 2 (L2) - Enterprise interoperability:** Platforms compliant to Level 2 will be capable of forming federations among themselves, allowing them to share resources without the aid of symbIoTe Core Services;
- **Level 3 (L3) - Dynamic smart spaces:** Platforms integrate symbIoTe components which allow them to communicate between different smart spaces;
- **Level 4 (L4) - Roaming devices:** Platforms compliant with Level 4 allow their devices to roam between smart spaces using the symbIoTe services.

As shown in Figure 2, domains and levels are related such as one level of compliance might use components and services spread across different domains. For a more complete description of the different levels of compliance please see Deliverable 1.4 [1] as in this document we will focus on the question how platforms can achieve Level 2 compliance.

3.2 Compliance Level 2

As stated above, platforms compliant to Level 2 will be able to form federations. This will free them from a central IoT resources repository and it will also provide means for resource bartering and Quality of Service control. As in any decentralized system, security issues need to be addressed with utmost care and symbIoTe will provide mechanisms to share resources directly among platforms in a reliable, controlled and secure way.

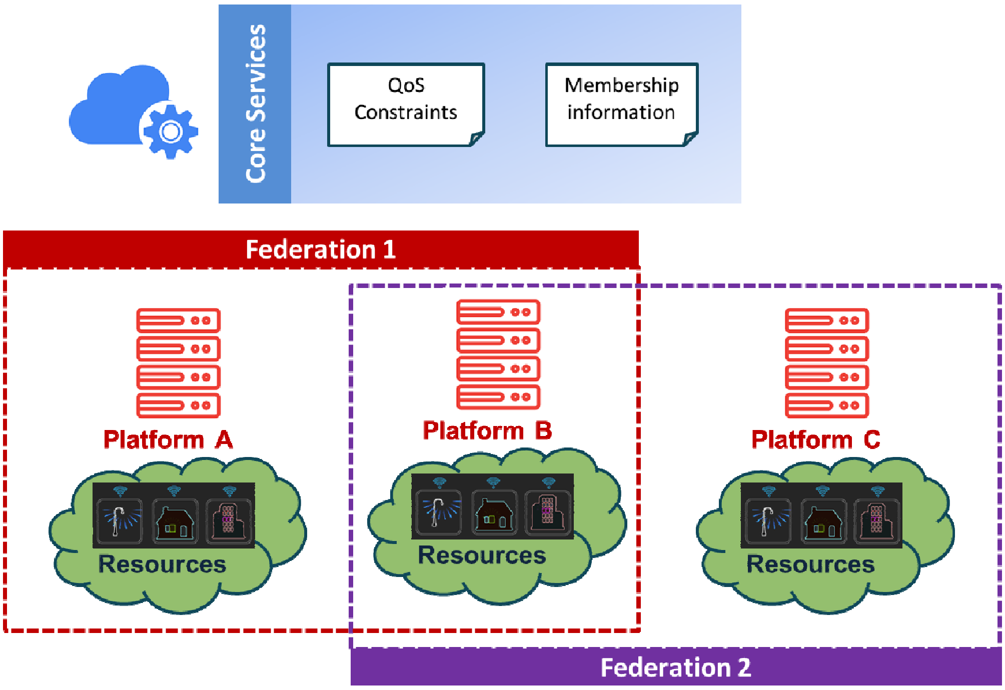

Figure 3 symbIoTe federations

Figure 3 shows a high-level overview of several platforms participating in federations. In the Core Services, we store information about QoS constraints and federation parameters themselves such as membership. As shown in the figure, a platform may belong to more than one federation at the same time and may choose to share some resources with either one or both.

Federations are created by an administrator, defining Quality of Services rules that the resources shared among the federation must comply with.

When a platform joins the federation at a later time, it will receive the information that will trigger some interactions among components of the Cloud Domain to enforce the compliance with the required Service Level Agreement (SLA).

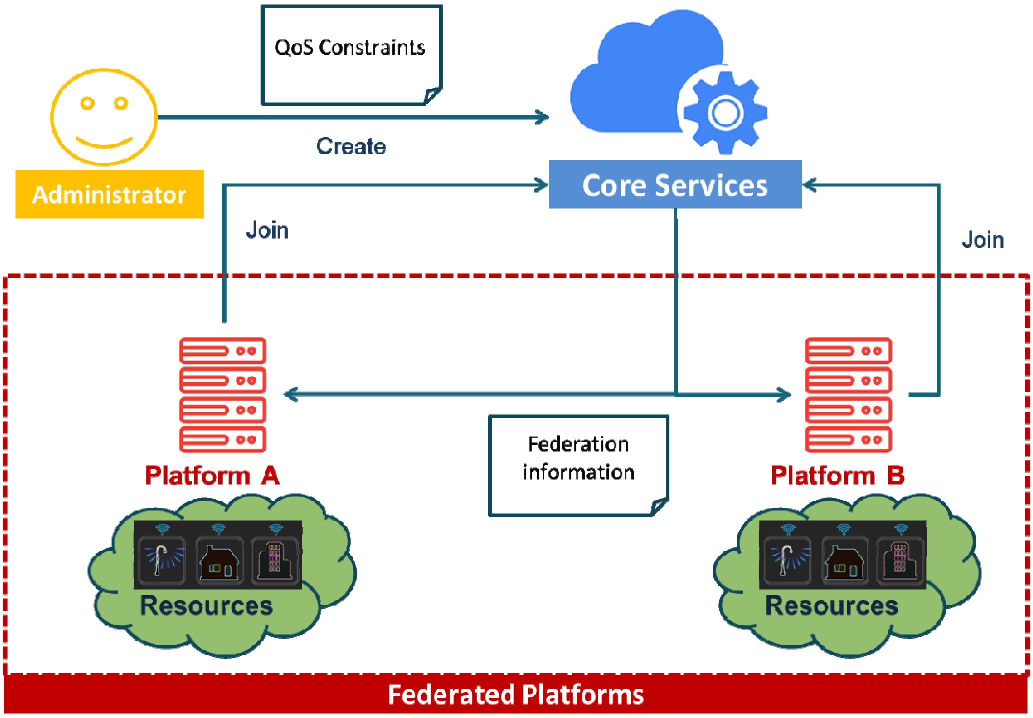

Figure 4. Creating a federation

Once a platform becomes a member of a federation, it may share some of its resources. Contrary to the Level 1 compliance [1][3], there isn't any centralised resource registry. Resource metadata is kept on the local platform side and distributed with the rest of the platforms through notifications.

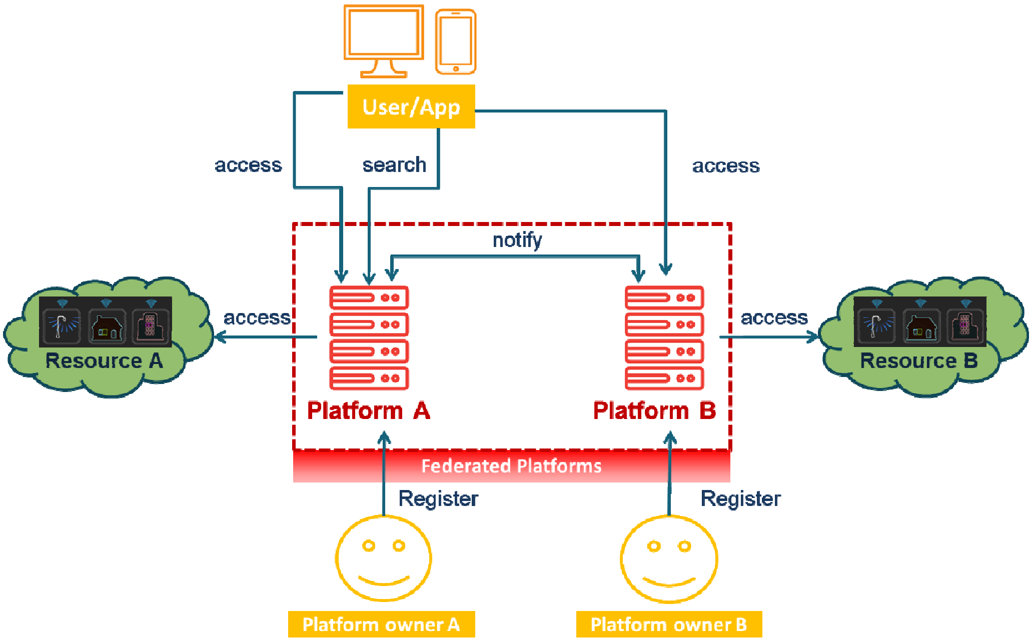

Figure 5. Resource Management

As shown in Figure 5, an application looks for resource metadata in its local platform but it will receive notifications about available resources from any other platform in the federation as well. As in Level 1, the access to resources is performed directly on the platform that owns/manages the resource.

At a high level, *federation management*, *resource sharing* and *resource access* are the three basic interactions within Level 2 compliant platforms. In order to support them, there are a series of components in both the Application domain and the Cloud domain that need to be either created or modified compared to the Level 1 solution.

3.3 Component description

To achieve Level 2 compliance, we reuse and extend several components of Level 1 architecture that is defined and described in deliverables D1.4 [1] and D2.5 [3].

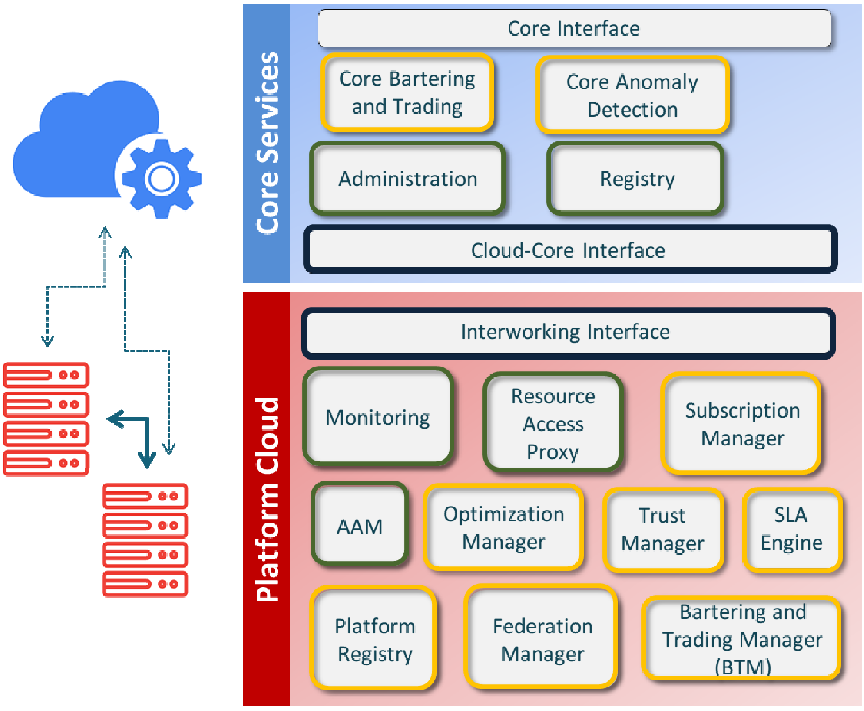

Figure 6. L2 Components

Figure 6 shows components placed in the Application and Cloud domains. New components are highlighted in yellow, while updated ones are highlighted in green.

- **Administration:** GUI based administration tool. Updated to allow the management of federations;
- **Core Bartering and Trading:** Performs the validation of resource bartering interactions in a centralized way;
- **Core Anomaly Detector:** Detects 0-day attacks (targeting vulnerabilities in our solutions that are unknown by the deployment time) and other types of security violations by using a signatureless approach;
- **Registry:** Stores information about federations such as membership and Quality of Service constraints;
- **SLA engine:** Hosts and monitors Service Level Agreements signed by the platforms when they join a federation;

- **Monitoring:** Gathers metrics from resources managed by the platform. Updated to monitor and aggregate metrics relevant to SLAs that will guarantee a certain level of QoS;
- **Resource Access Proxy:** Enables symbIoTe compliant access to resources within an IoT platform or enabler acting as a platform, updated to enable access with coupons for bartering;
- **Subscription Manager:** This is a new component that acts as a notification producer and receiver for events regarding resource sharing and updating across platforms in a federation.
- **Authentication and Authorization Manager (AAM):** Provides tokens and certificates that allow applications to search and access resources and components to communicate between them in a secure way. It has been updated to work locally on platforms and independently from the symbIoTe Core.
- **Optimization Manager:** Supports the suggestion of equivalent resources (from other platforms) if these resources are collocated in the same federation to optimize power/energy consumption, apply load balancing of equivalent resources, and/or enable global cost reduction within the federation. This is done by finding about similar resources within the federation that can be used indistinctly in case of need.
- **Trust Manager:** Supports a platform owner or application in taking informed decisions about foreign resource selection or platform interactions by calculating resources and platforms trust levels based their properties and interactions.
- **Platform Registry:** Enables the registration of IoT Devices and (Composite) IoT Services, which are offered by IoT platforms to be discoverable by other federated IoT platforms (L2 compliance).
- **Federation Manager:** Manages all required federation information needed at the platform level by passing the information to the rest of the components as required.
- **Bartering and Trading Manager:** Manages the bartering and trading between IoT platforms as far as this can happen in a decentralized way.

A more complete description of these components can be found in D1.4 [1] and they will be further explored and described as well in the rest of the sections of this deliverable.

3.4 Architecture adaptations

We have made some adjustments to the architecture compared to the initial one described in D1.4 [1]. The goal was to design federations in a decentralized way and to decrease the dependency on the central Core for basic functions such as SLA enforcement and bartering interactions. The outcomes of this effort are the following:

- The SLA Manager now resides in the Cloud Domain, at the platform side, while before it was hosted in the symbIoTe Core. This decreases the traffic going from platforms to the core, as we now do not need a centralized monitoring component for SLA enforcement. Also, the number of SLAs to be monitored gets lower, as each platform has to monitor one SLA per federation it belongs to, while before the core had to validate one SLA per platform and federation in each cycle, which put a greater burden on Core Services.

- The Core Bartering and Trading component now acts as a centralized validator of coupons, but not an issuer. This removes the need for global bartering and trading rules and provides autonomy for each platform to decide which resources are shared for bartering under its own terms.

The obvious disadvantage of this improved solution is that platform owners need to configure an extra microservice for their platforms, but we believe that the advantages outweigh the disadvantages in this case, given the improvement on autonomy and performance gained by a pure distributed approach.

4 Platform Federation Mechanisms

4.1 Introduction

The symbIoTe platform federation management ensures efficient and secure inter-platform communication and collaboration between multiple L2 compliant IoT platforms and connected native applications. In contrast to L1, the sharing of resources within the setup federation is handled directly between the involved platforms and does not require interactions with the symbIoTe Core Services.

However, the administration (creation, update, deletion) of each federation object is still done centrally by the Administration component in the Core domain. This approach allows a central coordination of the administered federation object and provides a user interface to platform owners when setting up general rules for L2 functionalities.

In the following subsections, we describe the overall federation administration and management workflow by outlining the available features and interactions. The setup and distribution of federations as well the registration and sharing of resources within the federation are discussed. Next, SLA handling, trust calculation and resource recommendation as additional features available within the symbIoTe L2 solution are introduced and key functionalities and concepts are highlighted.

4.2 Federation Management

This section describes the federation management approach and workflow in the symbIoTe ecosystem enabling organisational interoperability between symbIoTe-compliant IoT platforms with minimal dependency on a central system.

4.2.1 Federation Management workflow

The entire workflow for managing (creating, updating, deleting) federations is split in two major steps involving components in the Core and Cloud Domain:

1. **Administration of federation object properties and platform members**
   The first step deals with the manual setup and configuration of required properties per federation executed by an authenticated platform owner. After successful login, the federation administration view is opened and the following information has to be inserted according to the defined Meta Information Model object described in D2.4 [2]:
   - **Informative federation name** to allow easy lookup and classification of the intended purpose of the new federation,
   - **Public flag** indicating if the federation is visible in all search requests or is visible only to platforms which are federation members,
   - **SLA template** which describes the agreed conditions and requirements for all current and future platforms planning to join the federation (see section 4.4) and
   - **Members list** that include all IoT platforms participating in this federation.

   The membership in the federation is considered as by invitation only. Thus, only platforms which are already members in the federation may add or remove other IoT platforms. If the member list is modified, each affected platform owner will receive a notification in the administration component to approve a new federation join or leave request.

Currently, we assume that the consent for these changes have already been agreed upon and signed offline between all involved parties. Thus, we do not support any automatic voting or veto mechanisms without any manual interaction of the platform owners.

2. **Distribution of updates to and within federated IoT platforms**
   After successful validation and verification of the federation object, all affected platforms are notified. Next, the changed information from the Administration component, which is published in the Core, is pushed to all platform Federation Managers located in the Cloud Domain.

   Each Federation Manager is held responsible for keeping the current federation state per platform and for ensuring that all concerned components within its platform space are notified of the relevant changes as well. For tracing purposes and analysis of actions requested by other components, the Federation Manager also maintains a history log.

Figure 7 and Figure 8 depict a detailed workflow of federations management with each component interaction and message is depicted and show all involved components in the Core (blue) and Cloud (orange) Domain.

First, Figure 7 outlines the components and message flow in the Core Domain for the federation management workflow and highlights its interactions for federation creation, update and platform joining or leaving the federation. As depicted in the figure below, message 12 propagates all changes made to the respective platforms.

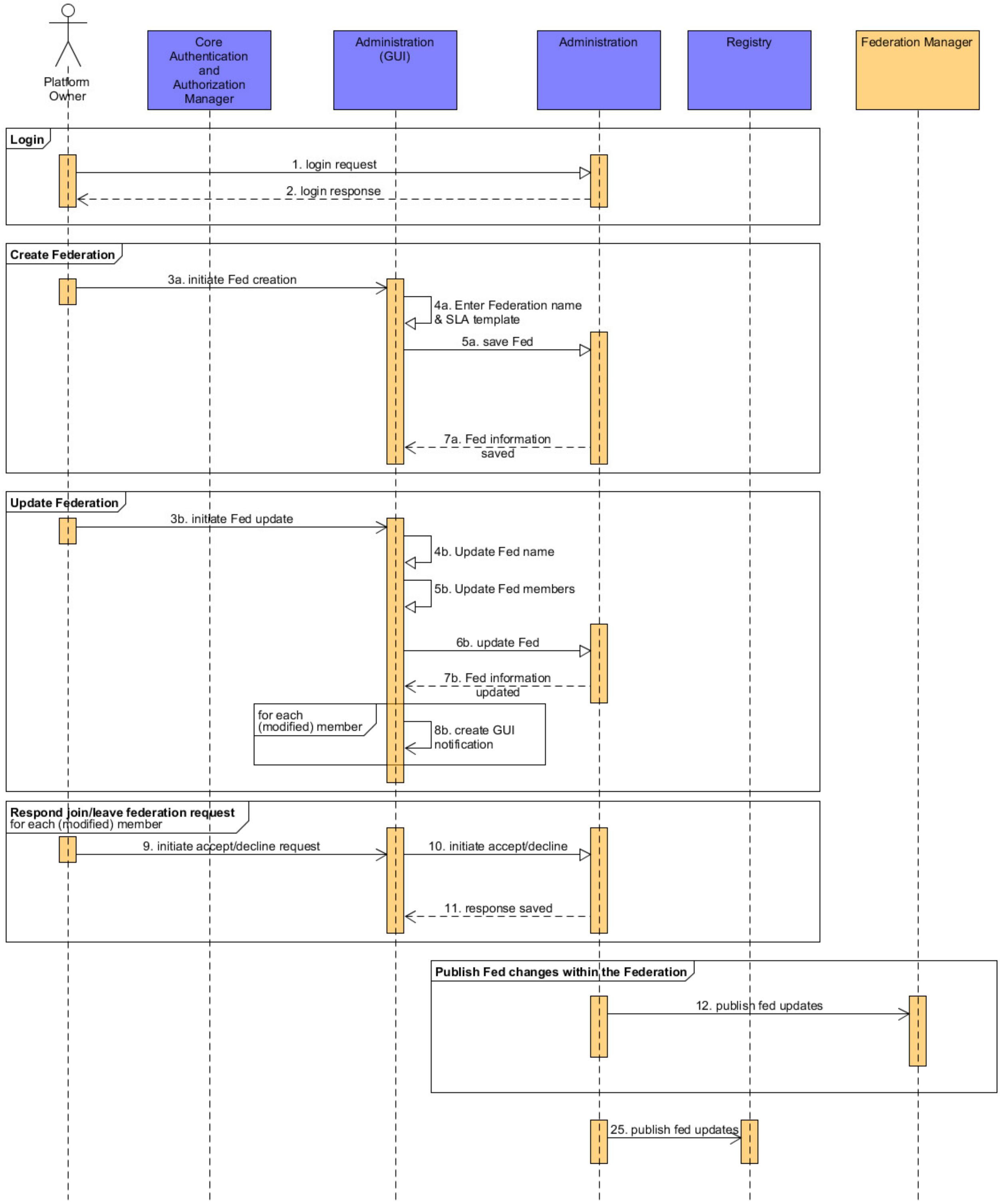
Platform Owner
Core Authentication and Authorization Manager
Administration (GUI)
Administration
Registry
Federation Manager
Login
1. login request
2. login response
Create Federation
3a. initiate Fed creation
4a. Enter Federation name & SLA template
5a. save Fed
7a. Fed information saved
Update Federation
3b. initiate Fed update
4b. Update Fed name
5b. Update Fed members
6b. update Fed
7b. Fed information updated
for each (modified) member
8b. create GUI notification
Respond join/leave federation request
for each (modified) member
9. initiate accept/decline request
10. initiate accept/decline
11. response saved
Publish Fed changes within the Federation
12. publish fed updates
25. publish fed updates

Figure 7. Manage (Create, Join and Leave) federations – Core Domain interaction

In Figure 8, a detailed process focusing on interactions between Federation Manager with other components within the Cloud Domain is depicted. The interaction is triggered by message 12 received from the Core Domain, as previously shown in Figure 7.

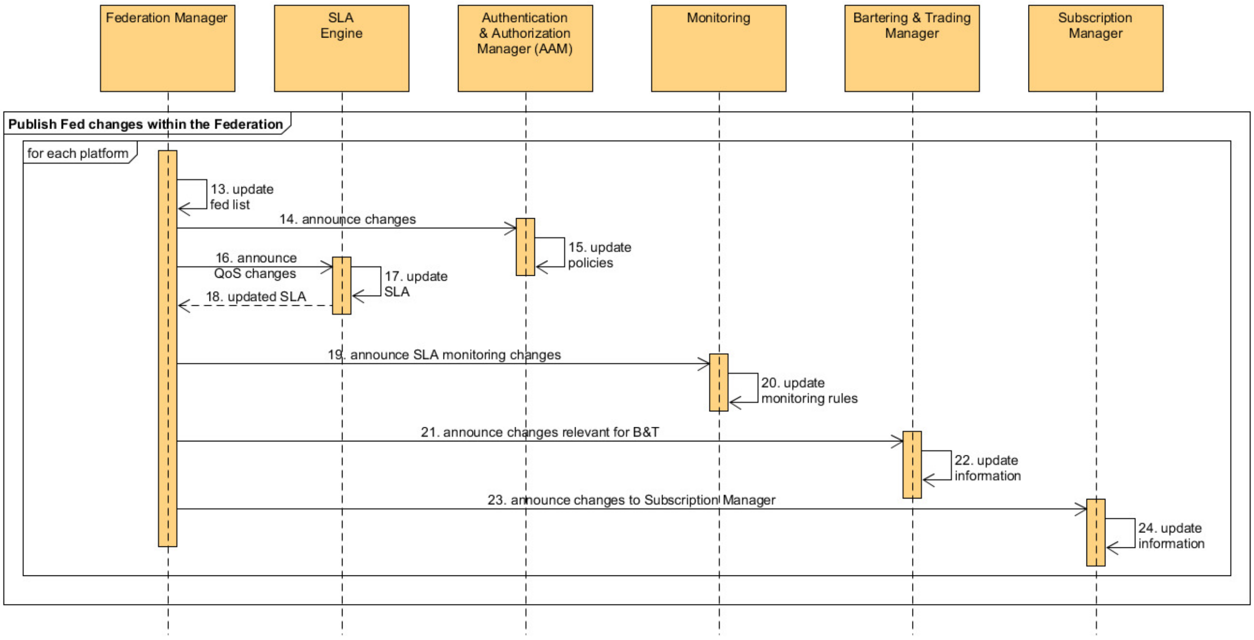
Federation Manager
SLA Engine
Authentication & Authorization Manager (AAM)
Monitoring
Bartering & Trading Manager
Subscription Manager
Publish Fed changes within the Federation
for each platform
13. update fed list
14. announce changes
15. update policies
16. announce QoS changes
17. update SLA
18. updated SLA
19. announce SLA monitoring changes
20. update monitoring rules
21. announce changes relevant for B&T
22. update information
23. announce changes to Subscription Manager
24. update information

Figure 8 Manage (Create, Join and Leave) federations – Cloud Domain interaction

Additionally, each sequence, message and procedure of the sequence diagram depicted in both figures above will be explained in detail in the following paragraph below.

Platform owner login sequence:

- *Message 1*: Generated by the platform owner and sent to the Administration. It is used to request access to the Administration (GUI).
- *Message 2*: Generated by the Administration and sent to the platform owner.

Create a federation sequence:

- *Message 3a*: Generated by the platform owner and sent to the Administration (GUI). The Administrator can now create a new federation.
- *Procedure 4a*: The platform owner creates a new federation by inserting the information (name, QoS constraints) regarding the federation via the Administration GUI.
- *Message 5a*: Generated by the Administration GUI and sent to the Administration to save the federation object.
- *Message 6a*: Generated by the Administration and sent to the Administration GUI to inform of successful creation.

Update a federation sequence:

- *Message 3b*: Generated by the platform owner and sent to the Administration (GUI). The Administrator can now update already joined federations.
- *Procedure 4b*: The platform owner updates the federation name via the Administration GUI.

- *Procedure 5b*: The platform owner updates the members of federation (adds and/or removes platforms).
- *Message 6b*: Generated by the Administration GUI and sent to the Administration to save the federation object.
- *Message 7b*: Generated by the Administration and sent to the Administration GUI. It is used to request/inform the other platform owners about the changes regarding the federation.
- *Procedure 8b*: The Administration GUI creates a notification to inform the platform owners about a pending request.

Respond to join or leave a federation request (for each added or removed platform) sequence:

- *Message 9*: Platform owner accepts or declines the federation join or leave request in the Administration GUI which sends this message to the Administration.
- *Message 10*: The administration GUI sends this information to the Administration to update the federation membership information. The Administration will then inform about these changes to each affected platform (see message 15 on).
- *Message 11*: Generated by the Administration and sent to the Administration GUI. It is used to acknowledge the previously received feedback.

Publish federation changes within federation sequence:

- *Message 12*: Generated by the Administration and sent to the Federation Manager. It is used to update the federation affiliation of the platforms at the platform level.
- *Procedure 13*: The Federation Manager updates the federation information with respect to the received data.
- *Message 14*: Generated by the Federation Manager and sent to the Authentication & Authorization Manager. It is used to request the federation policy modification.
- *Procedure 15*: The Authentication & Authorization Manager performs the policy update.
- *Message 16*: The Federation Manager informs the SLA Manager about changes in its status of membership with a federation, passing the QoS constraints in case the platform is joining the aforementioned federation.
- *Message 17*: the SLA Manager will act upon this notification. In case of joining, it will sign a new SLA and send it to the Federation Manager. In case of leaving, it will remove the previously enforced SLA.
- *Message 18*: generated by the SLA Manager to the Federation manager, it will return the signed or unsigned SLA depending if the interaction is for joining or leaving a federation.
- *Message 19*: Generated by the Federation Manager and sent to the Monitoring to update the monitoring rules.
- *Procedure 20*: The Monitoring component updates the respective resource monitoring rules.

- *Message 21*: Generated by the Federation Manager and sent to Bartering & Trading to update the trading rules.
- *Procedure 22*: The Bartering & Trading component updates the respective trading rules based on the federation information.
- *Message 23*: Generated by the Federation Manager and sent to the Subscription Manager to notify that a new platform is available in the federation.
- Procedure 24: The Subscription Manager updates the list of potential platforms it has to notify to of resource availability within the federation.
- *Message 25*: Generated by the Administration and sent to the Registry. It is used to inform the Registry about the update/creation of a federation within the symbIoTe ecosystem.

4.3 Resource Management in Federations

In comparison to the L1 resource registration and sharing implementation [3], L2 facilitates a peer-to-peer approach for resource metadata distribution among interested participating federation members. Thus, L2 introduces a distributed resource management concept, where updates are transmitted using the Publish/Subscribe Pattern.

The resource registration, update and delete operations within a federation are triggered by the platform owner in a way similar to the workflow implemented for L1. However, the specification of a particular set of federations where the resource is going to be offered, triggers an alternative workflow where the local Platform Registry will receive and manage all resource metadata registration and updates instead of the Core Registry. Thus, the Core Domain does not hold any information with respect to offered or consumed resources in the federation, which creates an additional layer of security and privacy as well as eliminates a single point of failure. Next, the Platform Registry notifies the Subscription Manager of changes in the availability of resources within the federation, which initiates the distribution of this updates to the federated IoT platforms.

In detail, each Subscription Manager component maintains an up-to-date list of all platforms within a federation (i.e. their Subscription Manager components) and their subscriptions specifying interest of other platforms specific resources. This approach enables an efficient and consistent way of notification distribution between federated platforms as only interested parties receive resource updates and unnecessary message broadcasts to all platforms are obviated.

Another point worth mentioning is that we do not enforce the overhead of using semantic mapping inside federations. To this end, at creation time of a federation an optional Best-Practice Information Model (BIM) can be selected which has to be used within this federation. Additionally, any other common information model can be agreed that is going to facilitate any data and metadata transactions within the federation. The used federation information model should be a valid extension of the Core Information Model, as defined in Deliverable D2.4 [2]. However, neither the validation against a particular instance of the federation model nor mapping will be supported by the generic federation components. If such functionalities are deemed as mandatory, the participating platforms should use adequate Core Services and develop their own plugins to support semantic mapping.

The process of search and access to resources within the federation is similar to the L1 approach where an application searches the symbIoTe Core to find the desired resources, and accesses the resources by using the RAP of the corresponding platform. Since in L2 approach symbIoTe Core is not used, an application searches for adequate resources shared within a federation using its own Platform Registry. Access to the resources from other platforms is, as in the L1 approach, enabled through the RAP of the corresponding, in this case, federated platform.

Subsection 4.3.1 shows the workflow for sharing resource metadata within a federation, i.e., how the metadata of the shared resources of a home platform is distributed to interested federated platforms. Subsection 4.3.2 shows how an application registered with a home platform can access resources from a foreign, federated platform.

4.3.1 Resource Sharing Workflow

The detailed workflow for resource sharing with interactions between components of home (Platform A) and foreign platforms (Platform B) is shown in Figure 9.

We describe two workflows. In the first one, one platform owner shows interest in receiving updates about some kind of resources which might be shared within the federation. Then, another platform owner of a home platform adds descriptions of new resources, or updates or deletes the existing ones within a federation. This information is stored in the Platform Registry of the home platform that shares its resources and is distributed to other interested platforms by the home and foreign Subscription Managers who forward received updates to be stored by the foreign Platform Registries.

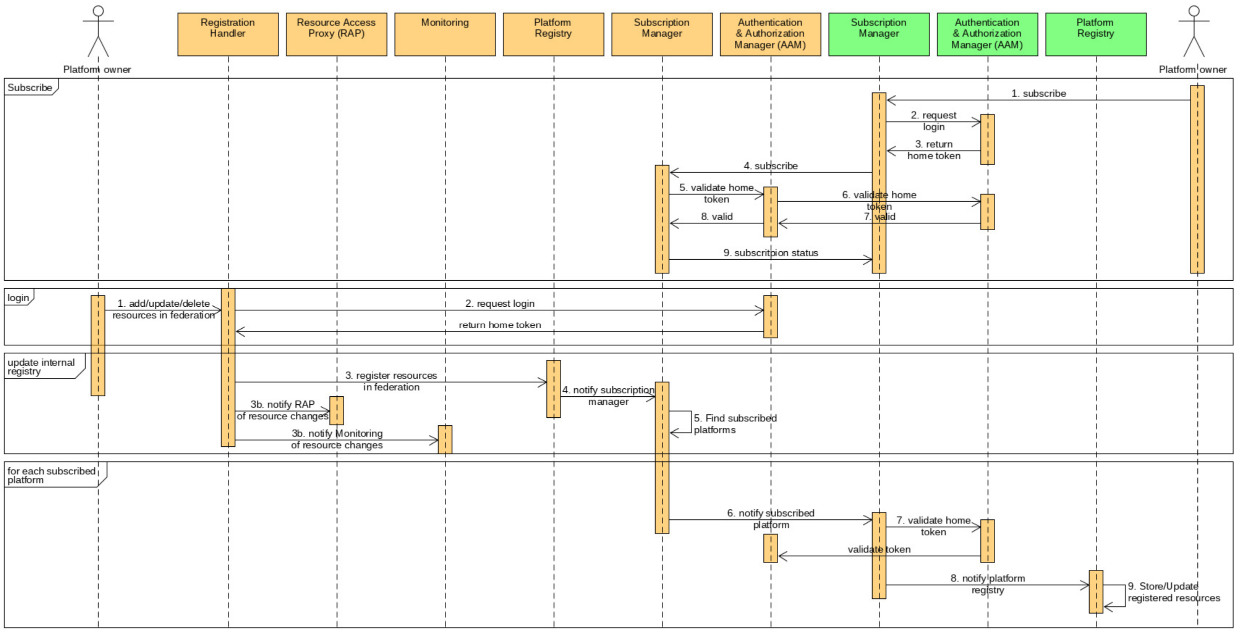
Platform owner
Registration Handler
Resource Access Proxy (RAP)
Monitoring
Platform Registry
Subscription Manager
Authentication & Authorization Manager (AAM)
Subscription Manager
Authentication & Authorization Manager (AAM)
Platform Registry
Platform owner
Subscribe
1. subscribe
2. request login
3. return home token
4. subscribe
5. validate home token
6. validate home token
8. valid
7. valid
9. subscription status
login
1. add/update/delete resources in federation
2. request login
return home token
update internal registry
3. register resources in federation
4. notify subscription manager
3b. notify RAP of resource changes
5. Find subscribed platforms
3b. notify Monitoring of resource changes
for each subscribed platform
6. notify subscribed platform
7. validate home token
validate token
8. notify platform registry
9. Store/Update registered resources

Figure 9. Manage resources within the federations

Subscription Workflow

- *Message 1*: Platform owner B sends a request to Subscription Manager to subscribe for updates about some kind or resources.
- *Message 2*: If Subscription Manager doesn't have a Home Token yet, it asks for one to its home AAM.

- *Message 3*: Home AAM sends the Home Token to Subscription Manager.
- *Message 4*: Subscription Manager in Platform B sends the subscription request to Subscription Managers in the rest of the platforms of the federation. In this case, we show the interaction with Platform A.
- *Message 5*: Subscription Manager in Platform A validates the Home Token sent with Message 4 to its home AAM
- *Message 6*: AAM in Platform A communicates with AAM in Platform B to validate the Home Token
- *Message 7*: AAM in Platform B validates the Home Token and sends the validity status to AAM in Platform A
- *Message 8*: AAM in Platform A sends the validity status of the Home Token to Subscription Manager in Platform A.
- *Message 9*: Subscription Manager in Platform A registers the interest of Platform B in some kind of resources and returns the subscription status to Subscription Manager in Platform B

Resource Sharing Workflow

- *Message 1*: Platform owner A sends a request to Registration Handler in its own platform to add new resources to a federation or to update or delete the existing ones.
- *Message 2*: Registration Handler sends login request to home AAM and receives home token. If Registration Handler already has Home Tokens, this interaction does not occur.
- *Message 3*: Generated by Platform A’s Registration Handler and sent to the its own Platform Registry. Its main purpose is to provide the metadata describing a resource or a set of resources, which the platform exposes to the Federation.
- *Message 4*: Generated by Platform A’s Platform Registry and sent to its own Subscription Manager, which will notify other platforms within the federation of new, updated or removed resources.
- *Procedure 5*: Find platforms interested in the updated resources. This is done during a matching process which compares subscriptions from federated platforms with metadata specifying new or cancelling existing resources.
- *Message 6*: Generated by the Platform A’s Subscription Manager and sent to the federated platform Subscription Manager in Platform B. The message contains information about the new, updated or removed resources. This information is forwarded only to platforms with a matching subscription specifying interest in new resources (platforms are identified in Procedure 5).
- *Message 7*: generated by Platform B’s Subscription Manager and sent to its own AAM to request the Token validation performed between foreign and home AAM.
- *Message 8*: generated Platform B’s Subscription Manager and sent to its Platform Registry.

- *Procedure 9*: Platform Registry in Platform B stores the metadata of new or updated resources, or removes the metadata of deleted resources.

4.3.2 Search and Access to Resources within a Federation

This sequence diagram in Figure 10 shows the interactions to access a shared resource within a federation. Here, two scenarios may be distinguished:

1. *Native App adapted to cope with symbIoTe security principle and mechanisms or symbIoTe-enabled app developed from scratch.*

   In this scenario, the app logs directly in with the AAM and receives valid tokens. This allows direct access to the offered resource from the federated platforms. Here we have to keep in mind that the application itself has to convert/translate the foreign data format into its native data structure.

2. *Native App only supports the native AuthZ of the platform*

   Here, the native application does not need to be touched nor made symbIoTe aware. To enable the same security level as with all other symbIoTe applications the Application and App Security Handler (shown below) act as a symbIoTe client wrapper for native applications. This wrapper, which can be deployed in the platform Cloud Domain, ensures the consistent security approach and may be used for data transformation.

The diagram depicts scenario two, which includes symbIoTe client wrapper, which can also have the functionality of the aforementioned plugin performing mappings between different information models of federated platforms.

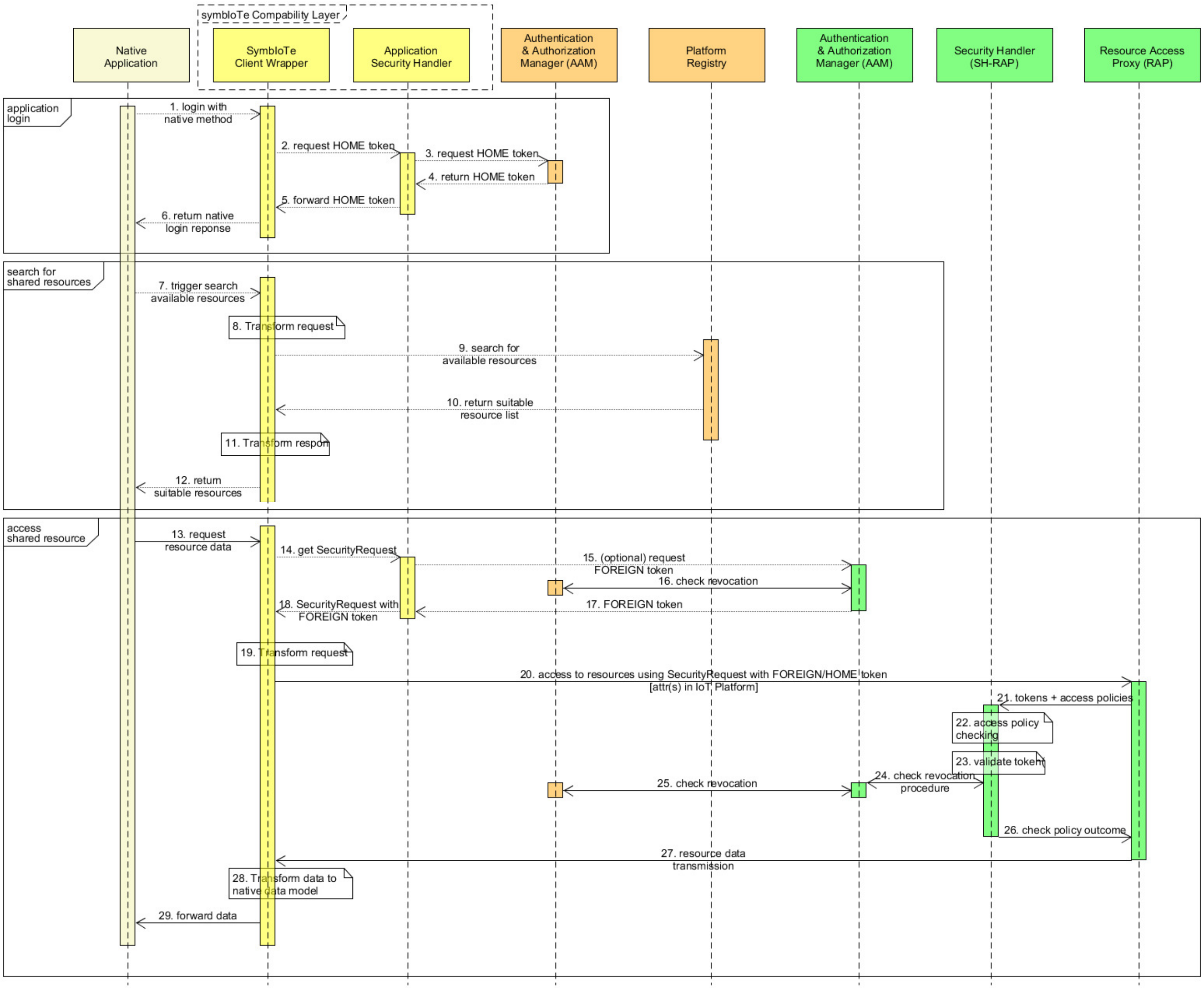
symbIoTe Compatibility Layer
Native Application
SymbIoTe Client Wrapper
Application Security Handler
Authentication & Authorization Manager (AAM)
Platform Registry
Authentication & Authorization Manager (AAM)
Security Handler (SH-RAP)
Resource Access Proxy (RAP)
application login
1. login with native method
2. request HOME token
3. request HOME token
4. return HOME token
5. forward HOME token
6. return native login reponse
search for shared resources
7. trigger search available resources
8. Transform request
9. search for available resources
10. return suitable resource list
11. Transform respon
12. return suitable resources
access shared resource
13. request resource data
14. get SecurityRequest
15. (optional) request FOREIGN token
16. check revocation
17. FOREIGN token
18. SecurityRequest with FOREIGN token
19. Transform request
20. access to resources using SecurityRequest with FOREIGN/HOME token [attr(s) in IoT Platform]
21. tokens + access policies
22. access policy checking
23. validate token
24. check revocation procedure
25. check revocation
26. check policy outcome
27. resource data transmission
28. Transform data to native data model
29. forward data

Figure 10 Search and access resources within the federation

- Message 1 (optional): Generated by the Native Application and sent to the symbIoTe Client Wrapper. It is used to trigger the recovery of the HOME token from the Platform A. If the Application is already logged in Platform A, it is not necessary.
- Message 2 (optional): Generated by the symbIoTe Client Wrapper and sent to the Application Security Handler. It is used to trigger the recovery of the HOME token from the Platform A. If the Application is already logged in Platform A, it is not necessary.
- Message 3 (optional): Generated by the Security Handler and sent to the home AAM. It is used to trigger the recovery of the HOME token from the Platform A. If the Application is already logged in Platform A, it is not necessary.
- Message 4 (optional): Generated by the home AAM in the platform A and sent to the Application Security Handler. It is used to provide the HOME token with attributes included. If the Application is already logged in Platform A, it is not necessary.
- Message 5 (optional): Generated by the Application Security Handler in the platform A and sent to the symbIoTe Client Wrapper. It is used to provide the HOME token

with attributes included. If the Application is already logged in Platform A, it is not necessary.

- Message 6 (optional): Generated by the symbIoTe Client Wrapper in the platform A and sent to the Native Application. It is used to provide the HOME token with attributes included. If the Application is already logged in Platform A, it is not necessary.
- Message 7: Generated by the Native Application and sent to the symbIoTe Client Wrapper to find the available resources.
- Procedure 8: Transforms the request to the format understandable to Platform Registry
- Message 9: Generated by the symbIoTe Client Wrapper and sent to the Platform Registry to find the available resources.
- Message 10: Generated by the Platform registry and sent to the symbIoTe Client Wrapper. It contains the list of available resources.
- Procedure 11: Transforms the response to the format understandable to the Native Application
- Message 12: Forwards the list of available resources from symbIoTe Client Wrapper to the Native Application
- Message 13: Request for resource data generated by the Native Application, sent to symbIoTe Client Wrapper
- Message 14 (optional): Request for FOREIGN tokens, which enable access to resources in Platform B. It is generated by symbIoTe Client Wrapper and sent to the Security Handler. If the Client Wrapper already has foreign tokens, the request is not necessary.
- Message 15 (optional): Request for FOREIGN tokens, which enable access to resources in Platform B. It is forwarded by the Security Handler and sent to the AAM of the Platform B.
- Message 16 (optional): Check revocation procedure between foreign and home AAMs.
- Message 17 (optional): Foreign AAM generates foreign tokens and sends them to Security Handler.
- Message 18 (optional): Security Handler forwards foreign tokens to symbIoTe Client Wrapper.
- Procedure 19 (optional): Transforms the response, generates new request for the foreign RAP to access resources.
- Message 20: Request to access the resources with foreign token. Generated by the symbIoTe Client Wrapper and sent to the foreign RAP.
- Message 21: Checking tokens and access policies, forwarded by RAP to RAP Security Handler.
- Procedure 22: Access policy checking (at SH-RAP).

- Procedure 23: Token validation (at SH-RAP).
- Message 24: SH-RAP initiates check revocation procedure, sends request to foreign RAP.
- Message 25: Check revocation procedure between foreign and home AAMs.
- Message 26: SH-RAP forwards check policy outcome to RAP.
- Message 27: Requested data is found by foreign RAP and forwarded to symbIoTe Client Wrapper.
- Procedure 28: symbIoTe Client Wrapper transforms data to the model understandable by the Native Application.
- Message 29: symbIoTe Client Wrapper forwards data to the Native Application.-

4.4 SLA and QoS Enforcement within Federations

Resources shared within a federation must comply with a series of Quality of Service parameters, mainly availability and load, to guarantee a certain quality level for applications accessing them and also to assist in the calculation of resource and platform trust score. To do so, each federation defines a threshold for these parameters that each resource must comply with. When a platform joins a federation, it signs a Service Level Agreement (SLA) based on these parameters and associated constraints. The monitoring frameworks will then gather periodic reports and measurements relevant to defined parameters and metrics to check and assess that the agreements are respected. If at least one of those parameters is not respected, the platforms in the federation will be notified. This might affect the trust calculation for the misbehaving platform.

Two main components are in charge of these interactions: The SLA Manager and the monitoring framework.

4.4.1 SLA

The Service Level Agreement Manager (SLAM) runs in the Cloud Domain of every platform in a federation and it is in charge of storing templates and agreements as well as of assessing periodically that these agreements are upheld. If they are not, then it will raise violation notifications.

This component is composed of several sub-components (Figure 11). Each one takes care of concrete functionalities.

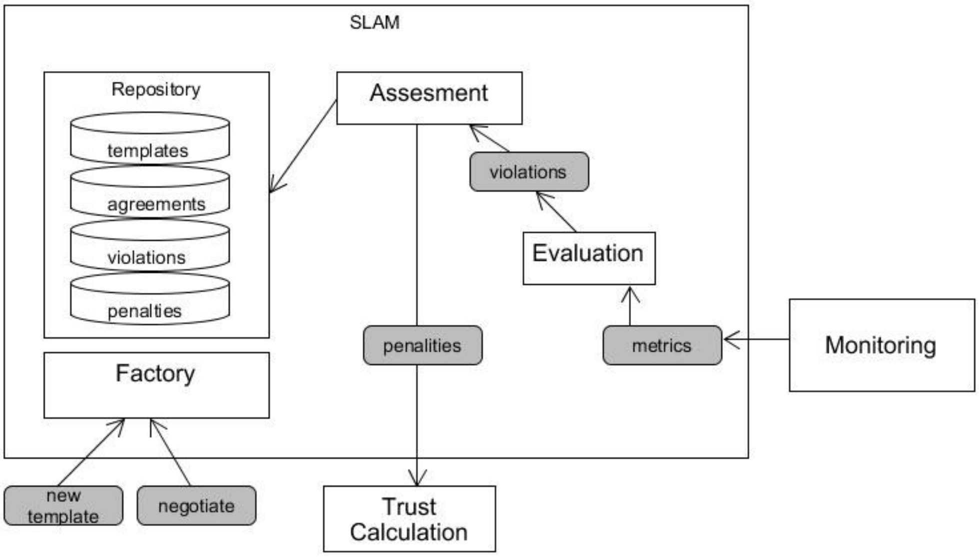

Figure 11. Component diagram of SLA Manager (SLAM)

- **SLA Factory**: This is the main entry point of the engine. It provides the API to external entities and it takes care of the steps of the negotiation, interacting with SLA Repository in order to generate and retrieve SLAs templates and agreements. It also activates the SLA enforcement once an agreement is “signed”.
- **SLA Repository**: This sub-component facilitates data storage, update and retrieval from the database. It manages the storage of SLA templates, SLA agreements and events related to SLA violation actions.
- **SLA Assessment**: This sub-component manages some SLA rules to determine the way to proceed when SLA is activated. It detects SLA violations and performs some notification actions according to the defined rules. It interacts with the SLA personalization.
- **SLA Evaluation**: It provides access to the monitoring information related to the agreed QoS aspects. It is possible to determine whether SLAs are being fulfilled or not.

These sub-components interact with some external entities such as the Administration component, Federation Manager, Trust Manager and Monitoring, as it is detailed below.

- Message 1: Join message generated by the Administration component, it is sent to the Federation Manager passing the QoS constraints defined for the federation.
- Message 2: Generated by the Federation Manager, an SLA is created with these constraints.
- Message 3: Facade added to the SLA Factory that transforms these constraints to an SLA Template.
- Message 4: Template saved in the SLA Repository. With this template, the facade signs an agreement that will be stored in the repository.
- Message 5: Agreement returns to the Federation Manager.

- Message 6: The Federation Manager sends this SLA to the Monitoring component to notify it about the metrics that it needs to monitor.
- Message 7: Monitoring component adds necessary metrics suggested by the Federation Manager.
- Messages 8-10: Periodically, the SLA Assessment checks the active SLAs and asks the SLA Evaluation to evaluate them. This sub component talks to the Monitoring component, asking for relevant metrics to check that the QoS is maintained in all shared resources.
- Message 11: generated by the SLA Manager, if a violation occurs, it informs the SLA Assessment, which will pass this information to the Federation Manager.

These interactions are shown in Figure 12.

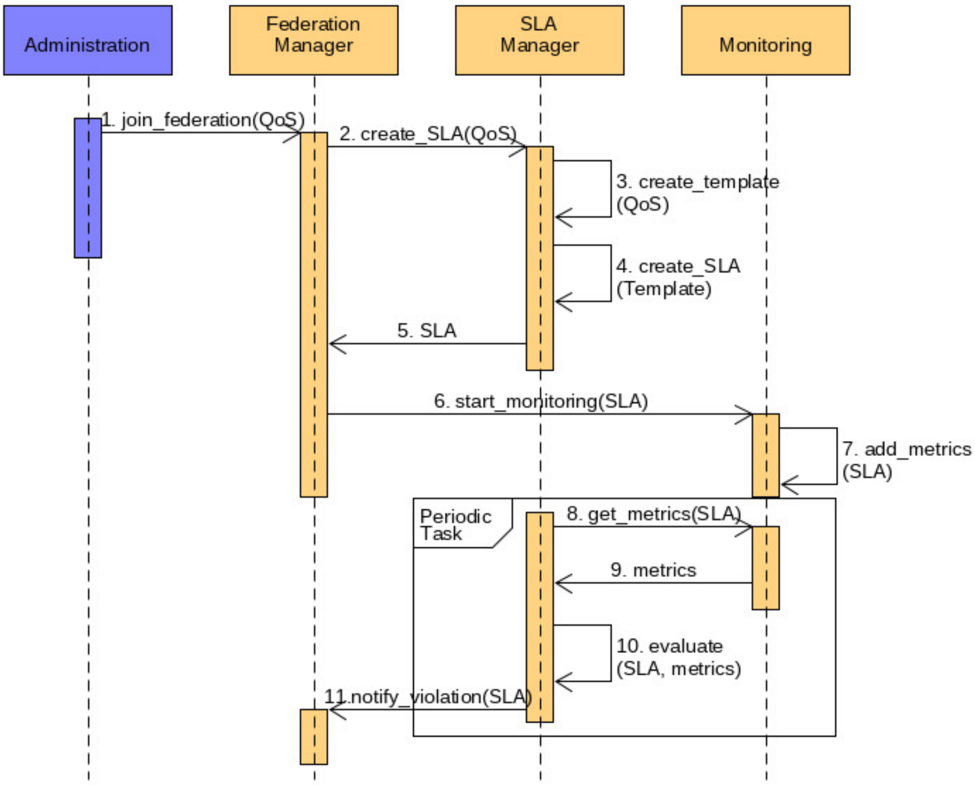

Figure 12 Sequence diagram of the SLA lifecycle

4.4.1.1 SLA Definitions

The SLA Engine works internally with documents following the WS-Agreement standard [5]. This standard defines a format based on XML with different parts, the main relevant ones for us being:

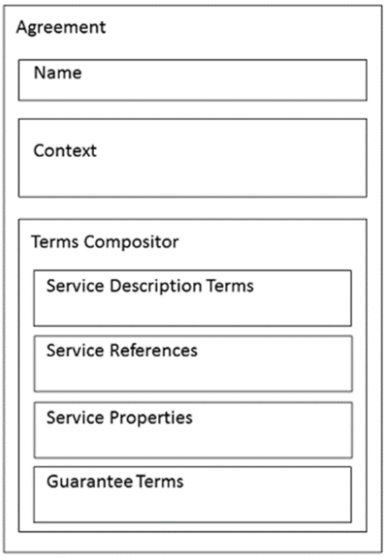

Figure 13. WS-Agreement data format

- **Context**: information about agreement parties, the agreement's lifetime, and the template reference.
- **Service Description Terms**: functional description of the service to provide (domain-specific description).
- **Service References**: refer to the existing services (domain-specific description).
- **Service Properties**: define variables in the context of an agreement.
- **Guarantee Terms**: define how guarantees are assessed and which compensation methods apply in case of meeting or violating the service guarantees:
    - Service Level Objective (QoS rules),
    - Penalties, Rewards.

Since we are not using all of the protocol interactions and options, the SLA Engine provides a REST interface that allows to manage SLA templates and agreements in a custom JSON format. Hence, it transforms them to and from the XML SLA-Agreement format needed by the repository and enforcement sub-components.

WS-Agreement is quite generic and allows the definition of almost any term that might constitute a QoS constraint. We have identified two groups of QoS aspects, which may affect the performance of SymbIoTe resources and platforms:

- Availability,
- Load.

In order to formulate Key Performance Indicators based on these aspects, the Monitoring component saves and performs statistical calculations across different periods of time such as:

- percentage of availability of resources,
- average load of resources.

All these metrics are sent to the SLA Engine upon request for assessment.

4.4.2 QoS Enforcement: Monitoring

To guarantee a certain level of QoS, platforms need to monitor their resources, sending metrics periodically to the Monitoring component. This is done by the Monitoring framework, which is shown in Figure 14.

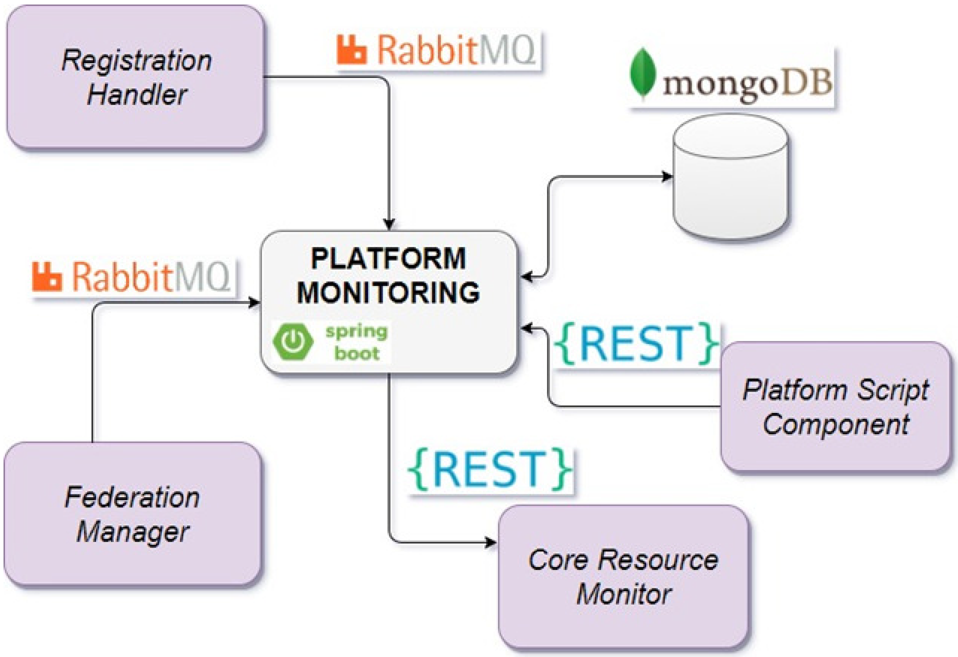

Figure 14. Monitoring framework

The central component present at every platform of the federation is the Platform Monitoring component. This component gathers metrics from a series of resources, which are shared to the rest of the platforms in the federation. The metrics and devices to monitor are provided by symbIoTe components:

- **Registration Handler**: It sends notifications about resource registration and unregistration. When a resource is registered to be shared within a federation, the Monitoring component, upon a notification reception, waits for metrics to be monitored for that particular resource.
- **Federation Manager**: When a platform joins a federation, it receives a set of QoS constraints that every resource shared in that particular federation must comply with. These QoS constraints will be based on metrics that must be monitored per shared resource. The Federation Manager informs the Monitoring component about those constraints and from that moment on, it waits for metrics associated to constraints on every shared resource.

The Platform Monitoring component offers a REST interface for data input that must be sent periodically. Given that the monitoring process is specific for every platform and even for every device, the Platform Owner must develop a system to gather these metrics and send them to the aforementioned REST service. We provide a sample script that can be used as a template or a starting point for monitoring. The Platform Owner may personalize it and execute it periodically, for example in a task scheduler like Unix *cron*, to send periodically data about load and availability of resources, which have been shared, to the Core or to any federation.

The Platform Monitoring listens to these metrics and sends this data periodically to the Core Resource Monitor in the core. This data includes raw metrics, such as availability and load for every device shared with the core at one moment in time.

4.4.2.1 KPIs Metrics convention

The monitoring system is based around a simple and quite universal concept for metrics. A metric is composed of a tag, a value and a timestamp. An SLA will define the tags that it will check but for simplicity, we are going to consider two KPIs only as a basis (Table 1).

Tag KPI	Description	Measure
availability	Accumulated calculation based on device availability	Percentage
load	Accumulated calculation based on device load	Average

Table 1. Core Authentication and Authorization Manager

Note that this list can be easily expanded at any time with minimal modifications on the Monitoring Platform.

Once the simple metrics are defined, that is, those containing the device's values, we will implement a "KPI tag" using accumulated metrics. These tags are named using the following convention:

KPI Tag = [device_type].[metric].[period],

Where:

- *[device_type]* = Type of device. This is optional and for individual devices it is omitted;
- *[metric]* = Metric tag;
- *[period]* = Time period in which the calculation is done. It might be a number of days or the special tag "all" that will perform the aggregation with all the values available for the device since it was shared.

For example:

"*g1.avai.all*" is the % of availability for all metrics registered of devices of type "g1".

"*load.30*" is the average load of a particular device during the last 30 days.

With these tags, we can compose QoS rules like:

- The availability of a every resource must be greater than X,
- The load average of devices of type T must not be greater than X,
- The availability of resources of type T must be greater than X in the last 10 days,
- And so on....

4.5 Trust and Reputation Approach

The symbIoTe framework enables efficient and interoperable resource sharing and access between multiple L2 compliant IoT platforms. Trust and reputation of the entire system plays a key role to ensure the acceptance of the proposed solution and fairness in the symbIoTe ecosystem which engages invloved IoT platforms to follow certain rules and comply with established standards or best practices.

First, we need to discuss and define the actual meaning and context of trust and reputation within symbIoTe to ensure a consistent terminology in the following sections. Similarly to the approach proposed in the FP7 project COMPOSE, we distinguish between the following two terms:

- **Trust** reflects the probability that an unknown party acts as planned and provides the requested action as promised. Thus, the trust calculation does not include any historical data or previous behaviour but bases the trust estimation on future interactions only which strongly creates a subjective and error-prone assessment.
- **Reputation**, however, uses different metrics from the past interactions to calculate the appropriate value and therefore allows a more concise and broader estimation of future behaviours based on historical information.

In symbIoTe, we are using these definitions but applying them on different levels of granularity to gain a best-of-both approach on an extended scope. Thus, trust will be relevant for offered and shared resources only, while reputation applies to platforms as a whole. Thus, it allows the calculation and estimation of a multi-faceted trust and reputation relationship within symbIoTe using all relevant information but still keeps in mind that IoT resources may increase their quality and therefore also raise the resource trust value.

Besides the classic trust and reputation approach, we also follow an adapted Web-of-Trust (WoT) principle [4] established in the Pretty Good Privacy (PGP) community. In the symbIoTe context, each platform offering a resource within a federation will assign a pre-calculated resource trust value to it. This information will be shared among all other federation members and will ease the way for trust calculation in the foreign platforms per resource. However, as this value is communicated by an unknown provider, another mechanism should be established to prove the trustworthiness of the received trust value and adapt it to other metrics as well. Here, the calculated platform reputation comes into play as an important asset to internally assess the resource trust from the consumer perspective. Based on the calculated resource trust and platform reputation values, the adaptive resource trust value is calculated which now reflects the real subjective probability taking into account the announced offered resource trust plus the internal reputation of the provider itself. Thus, the calculation of adaptive resource trust for shared resources provides a basis for taking better decisions and risk assessment for accessing foreign resources within the federation

Using this multi-level trust approach in symbIoTe ensures that all involved IoT platforms will benefit from uniform and common trust computation and representation scheme within the symbIoTe ecosystem. This enables a convenient and automated approach to enrich the resource selection procedure with additional properties representing the trust in resources and their offering platforms as depicted in Figure 15.

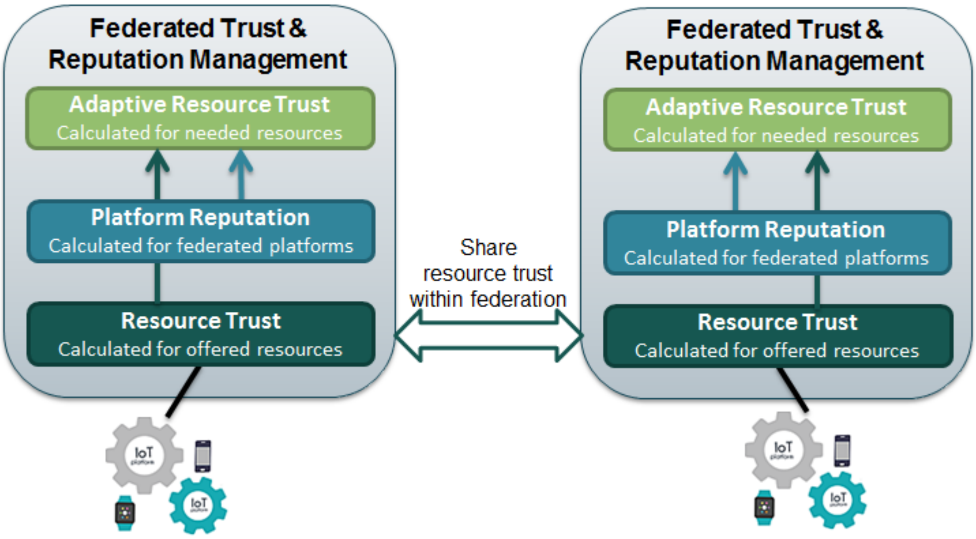
Federated Trust & Reputation Management
Adaptive Resource Trust
Calculated for needed resources
Platform Reputation
Calculated for federated platforms
Resource Trust
Calculated for offered resources
Share resource trust within federation
Federated Trust & Reputation Management
Adaptive Resource Trust
Calculated for needed resources
Platform Reputation
Calculated for federated platforms
Resource Trust
Calculated for offered resources
IoT
IoT
IoT
IoT

Figure 15. Multi-level trust management.

Thus, the multi-level trust management addresses the confidence in offered resources and the reputation of other federated IoT platforms in the symbIoTe ecosystem. Amongst other criteria, these trust values serve as a basis for calculation of an adaptive resource trust rating reflecting a more realistic and platform-centric indicators, which may impact a subjective decision for using specific resources, shared within the federation or influence the decision for accepting bartering coupons of other platforms.

The following sequence diagram describes the needed interactions initiated by the Trust Manager to calculate the three levels of trust.

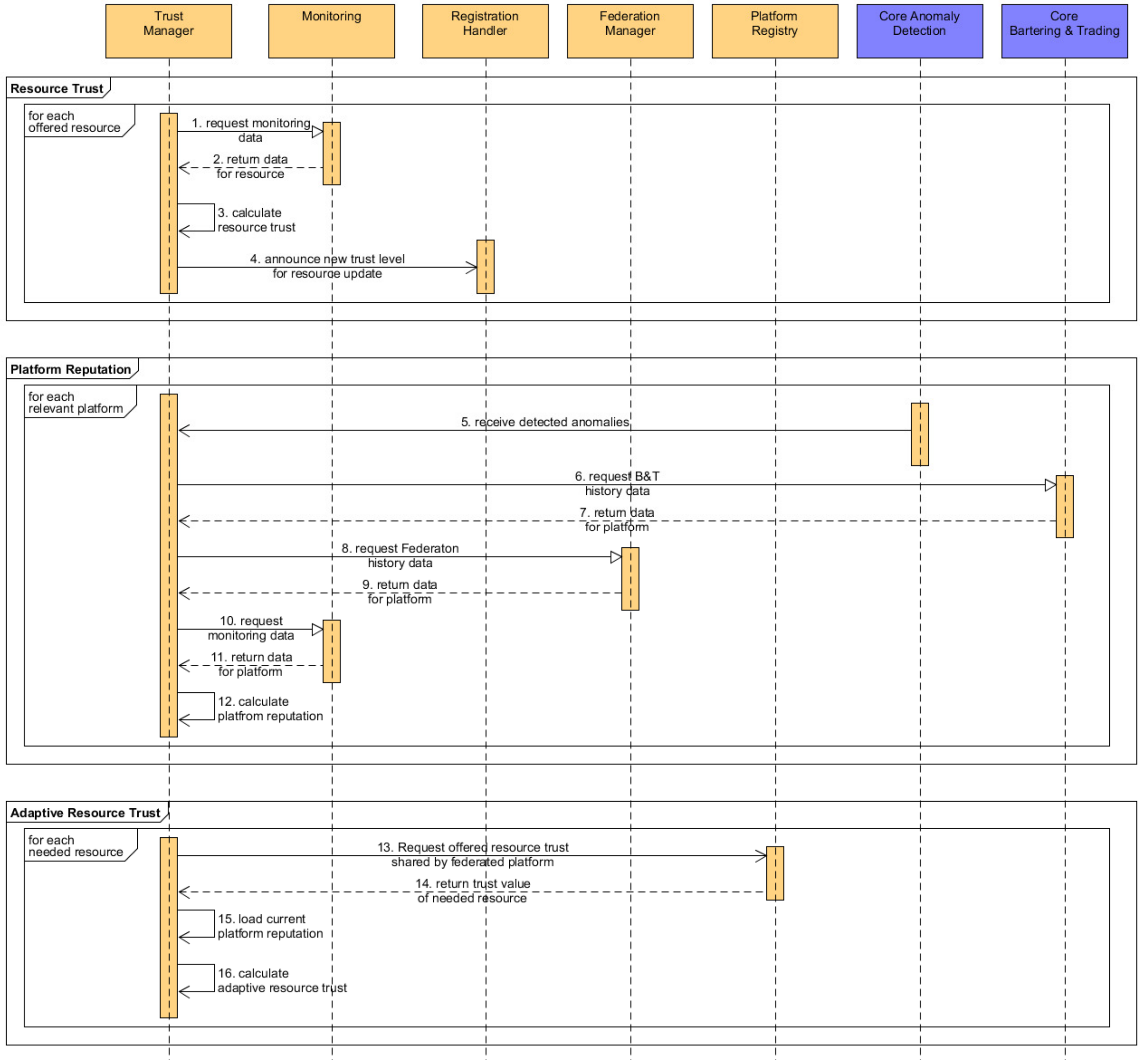
Trust Manager
Monitoring
Registration Handler
Federation Manager
Platform Registry
Core Anomaly Detection
Core Bartering & Trading
Resource Trust
for each offered resource
1. request monitoring data
2. return data for resource
3. calculate resource trust
4. announce new trust level for resource update
Platform Reputation
for each relevant platform
5. receive detected anomalies
6. request B&T history data
7. return data for platform
8. request Federaton history data
9. return data for platform
10. request monitoring data
11. return data for platform
12. calculate platfrom reputation
Adaptive Resource Trust
for each needed resource
13. Request offered resource trust shared by federated platform
14. return trust value of needed resource
15. load current platform reputation
16. calculate adaptive resource trust

Figure 16. Sequence diagram of trust management

Additionally, each sequence, message and procedure of the sequence diagram depicted in Figure 16 are explained in detail in the following paragraph.

Resource Trust:

- *Message 1*: Generated by the Trust Manager and sent to the Monitoring component. It is used to request monitoring data associated with the platform's own resources.
- *Message 2*: Generated by the Monitoring and sent to the Trust Manager. It is used to provide the previously requested data.
- *Procedure 3*: The Trust Manager calculates the resource trust.

- *Message 4*: Generated by the Trust Manager and sent to the Registration Handler. It is used to update the trust level of the resources registered within a federation.

Platform Reputation:

- *Message 5*: The Core Anomaly Detection component notifies the Trust Manager of detected anomalies relevant for platform reputation update.
- *Message 6*: Generated by the Trust Manager and sent to the Core Bartering. It is used to request the B&T history data associated with a foreign platform.
- *Message 7*: Generated by the Core Bartering & Trading and sent to the Trust Manager. It is used to provide the previously requested B&T history data.
- *Message 8*: Generated by the Trust Manager and sent to the Federation Manager. It is used to request federation history data associated with a foreign platform.
- *Message 9*: Generated by the Federation Manager and sent to the Trust Manager. It is used to provide the previously requested federation history data.
- *Message 10*: Generated by the Trust Manager and sent to the Monitoring. It is used to request monitoring data associated with a foreign platform's resource.
- *Message 11*: Generated by the Monitoring and sent to the Trust Manager. It is used to provide the previously requested monitoring data.
- *Procedure 12*: The Trust Manager calculates the foreign platform's reputation level.

Adaptive Resource Trust:

- *Message 13*: The Trust Manager requests the shared resource trust value by the federated platform from the Platform Registry.
- *Message 14*: The Platform Registry returns the current resource trust value.
- *Procedure 15*: The actual platform reputation value from the federated platform is loaded.
- *Procedure 16*: The Trust Manager calculates the adaptive resource trust used by other platform components.

The next subsections will define the approach for calculating the individual trust values. The calculation methods, described below, will take input data and (usage) statistics from other components for their algorithms to compute these ratings.

4.5.1 Resource Trust

Resource trust is a calculation done by every platform over the resources shared from other platforms in the federation. This calculation gives indication about how well the resource is behaving and how trustable it is to continue to work correctly. Many aspects can be taken into consideration here to provide information about the different aspects of the resource, e.g., security, battery life, dependability, behaviour and data stability. All these aspects depend on different metrics to get a value with a final step to normalize them giving a score between 0 (not trustable at all) and 100 (completely trustable).

So given a series of scores for aspects $A_i \in [0{,}100]$, where each aspect is a rational number associated with a weight $W_i \in [0{,}1]$ each weight being a rational number too so that $\sum_{i=0}^{n} W_i = 1$, the resource trust value will be calculated as $T_r = \sum_{i=0}^{n} A_i x W_i$, where T_r is the trust value for a particular resource that will be a rational number between 0 (not trustable at all) and 100 (100% trustable).

With the gathered metrics, we can implement a simple algorithm based on two aspects:

- ***Dependability***: Based on the availability of the resource. Monitoring will gather metrics about whether a resource is available (1) or not (0) at a moment in time. The dependability value will be calculated as the percentage of metrics in which availability is one over the total values of this metric. This information is obtained directly by querying the Monitoring component.
- ***Access***: Based on statistics about successful and wrongly rejected attempts of access the resource. On every access, the Resource Access Proxy will inform about the success or failure of such operation. From these notifications, the Trust Manager will calculate the percentage of successful accesses over the total number of them. A wrongly rejected access is considered an attempt to get data from a resource in which the user had the correct token and bartering coupon (see Section 5) but the resource itself rejected the request.

The final value of resource trust will be calculated as a normalized value of both aspects, with a weight of 50% for each of the two aforementioned aspects.

4.5.2 Platform Reputation

Each platform calculates its individual platform reputation value for other IoT platforms it communicates or will interact with. Thus, this metric represents a platform-centric indication per platform and therefore reflects a subjective confidence in other platforms that they comply with the rules as agreed when a federation is formed. This perspective facilitates an individual but a broader and long lasting view on surrounding peers and their historic behaviour and actions. Key of the reputation calculation is the enforcement of a fair and active symbIoTe ecosystem, which favours positive activities, active resource offerings, and durable federation membership relationships. The next diagram visually depicts the key categories and inputs used for platform reputation estimation.

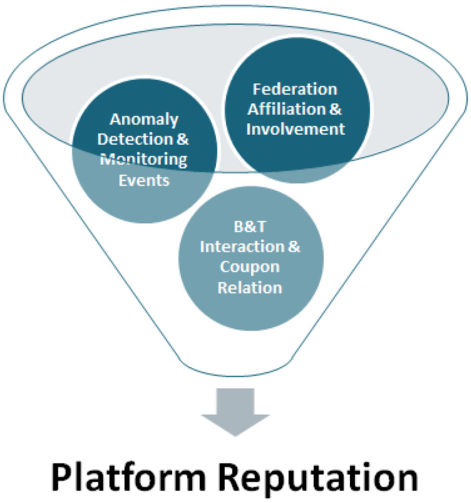
Federation Affiliation & Involvement
Anomaly Detection & Monitoring Events
B&T Interaction & Coupon Relation
Platform Reputation

Figure 17. Platform reputation

The platform reputation calculation is broken down into several sub functions taking into account each of these categories listed in the illustration above.

Similar to the resource trust, the reputation value will also range from 0 (= no reputation) to 100 (= full reputation) which will represent 100% reputation for a foreign platform.

To periodically calculate each platform reputation, the following information is used:

- ***Federation***: One crucial source of information is captured from the federation management itself. It is based on the given historic and current memberships in common federations plus the actual stability period during the membership. Also, the number of shared resources within these federations enriches the overall picture with respect to the possible platform reputation.
- ***B&T***: Another important input channel for reputation calculation is B&T activities in the past and the ratio of generated to consumed coupons as well as any negative actions noticed by the Core Bartering & Trading component such as cancelled or only one way confirmed coupon consumption.
- ***Anomaly Detection***: The third source used for platform reputation estimation is the central Anomaly Detection in the symbIoTe Core domain. Any abnormal or suspicious behaviour reported by any platform will be processed and corresponding events will be sent to concerned trust managers. Thus, these violation events will also impact platform reputation.

Overall, the entire platform reputation calculation integrates three of the major information sources currently available in the symbIoTe ecosystem. The generic and flexible calculation approach, however, allows an easy extension to further parameters.

4.5.3 Adaptive Resource Trust

Based on the previously calculated resource trust and platform reputation information, each platform is able to produce and adapt its individual trust estimation for shared resources. Thus, the adaptive resource trust states only an internal, subjective value but better reflects the real resource trust value from a platform-centric perspective.

As illustrated below, only these two factors enhanced with own monitoring information of used foreign resources – if available - are influencing the adaptive resource trust value.

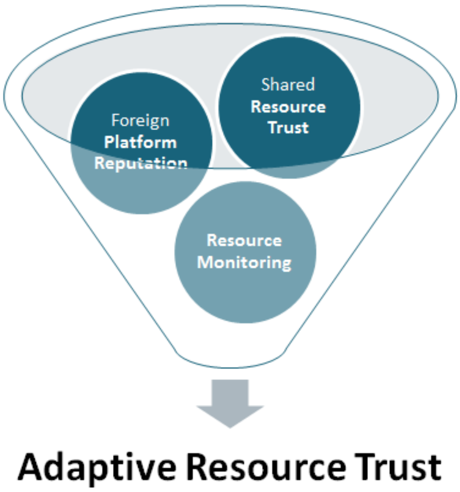

Figure 18. Adaptive Resource Trust

In general, the shared resource trust values are not questioned by the platform and therefore directly input to the adaptive resource trust calculation. In case the trustworthiness of these values need to be verified, the own monitoring component could be adapted for foreign resource monitoring as well. Based on this additional information the respective resource trust could be calculated for foreign resources and compared against the shared value.

Further, the entire trust management does not rely on the reliability of external information or a central trust and reputation system. Therefore, using such process ensures that long-lived involvement and a good relationship with other peer platforms increase the general confidence that the shared resource trust value is indeed accurate.

Also combining all trust and reputation levels from external as well as internal calculations avoids negative impacts from misbehaving third parties or bad-mouthing effects.

4.6 Federated Resource Recommendations

The goal of this component is to detect equivalent resources (from other platforms) within the same federation and, with this information, suggest to the platforms to only use one of them. The main objective of taking such an action is to optimize power/energy consumption (by not using needlessly the same resource multiple times in the same place), to apply load balancing of equivalent resources and to enable a general cost reduction within the Smart Space.

The following diagram (Figure 19) demonstrates how such a component would work.

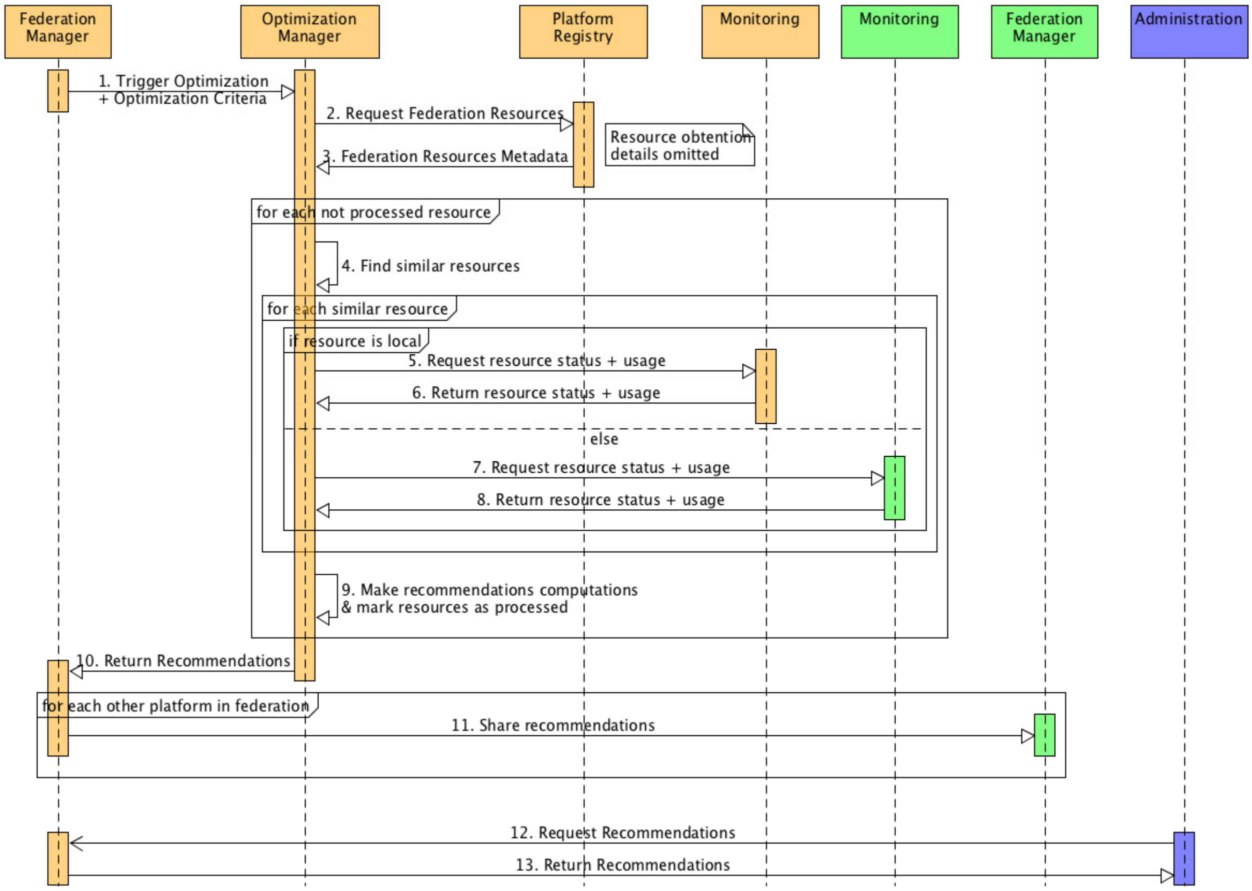
Federation Manager
Optimization Manager
Platform Registry
Monitoring
Monitoring
Federation Manager
Administration
1. Trigger Optimization + Optimization Criteria
2. Request Federation Resources
3. Federation Resources Metadata
Resource obtention details omitted
for each not processed resource
4. Find similar resources
for each similar resource
if resource is local
5. Request resource status + usage
6. Return resource status + usage
else
7. Request resource status + usage
8. Return resource status + usage
9. Make recommendations computations & mark resources as processed
10. Return Recommendations
for each other platform in federation
11. Share recommendations
12. Request Recommendations
13. Return Recommendations

Figure 19. Sequence diagram of Federated Resource Recommendations

- *Message 1*: Generated by Federation Manager and sent to the Optimisation Manager. It is used to trigger optimisation procedures. This trigger is activated on defined fixed time intervals (i.e. scheduled tasks). Some criteria for the optimisation can be passed by the Federation Manager (e.g., closeness of the sensors, current load of the sensors).
- *Message 2*: Generated by the Optimisation Manager and sent to Platform Registry. It is used to request the metadata of the resources within the federation.
- *Message 3*: Generated by the Platform Registry and sent to the Optimisation Manager. It is used to provide the Optimisation Manager with the metadata of the resources belonging to the platforms within the federation.
- *Procedure 4*: Procedure by the Optimisation Manager used to find resources similar to the selected one.
- *Message 5*: Generated by the Optimisation Manager and sent to the local Monitoring. It is used to request the current status and usage of the resource.
- *Message 6*: Generated by the local Monitoring and sent to the Optimisation Manager. It is used to return the status and usage of the given resource.
- *Message 7*: Generated by the Optimisation Manager and sent to the foreign Monitoring. It is used to request the current status and usage of the resource.
- *Message 8*: Generated by the foreign platform's Monitoring and sent to the Optimisation Manager. It is used to return the status and usage of the given resource.
- *Procedure 9*: Procedure by the Optimisation Manager. It is used to choose generate recommendations regarding the usage of the resources based on their status and current workload. Additionally, the resources considered are marked as processed, disabling them from being considered in the future during this process (optimising the entire process).
- *Message 10*: Generated by the Optimisation Manager and sent to the Federation Manager. It is used to provide the federation manager with the recommendations of the best resources to be used.
- *Message 11* (*optional*): Generated by the Federation Manager and sent to all the federation managers of platforms belonging to the federation. It is used to share the recommendations with the other platforms in the federation, since the information is equally valuable and useful for the other platforms.
- *Message 12* (*async*): Generated by the Administration and sent to the Federation Manager. It is used to request the computed recommendations.
- *Message 13*: Generated by the Federation Manager and sent to the Administration. It returns the calculated recommendations, to which the administrator can use to take actions to optimize the usage of the platform's resources.

4.7 Conclusions and Future Work

The work discussed in this section highlights the architectural and component design for L2 functionalities available in symbIoTe. In the current version, the detailed workflows and interactions between the involved Core and Cloud components were presented from a holistic system design perspective to envision the big picture of the federation management concept used in symbIoTe and all underlying processes and aspects such as resource sharing, SLA monitoring and enforcement, trust management or recommendations.

Based on the detailed design and component workflows described in this section, future work and activities will focus on the development, test and delivery of these components and functionalities. Additionally, we will concentrate on the actual implementation and documentation of the previously designed algorithms. To increase the impact and attraction of external stakeholders by explicitly addressing the most urgent existing gaps in current IoT environments, we will focus our efforts and work done in L2 on specific challenges like distributed SLA handling, trust and reputation mechanisms and peer-to-peer resource sharing within a highly distributed and heterogeneous ecosystem.

5 Bartering and Trading

5.1 Overview

In this chapter, we report on the current state of the activities concerning resource bartering and trading mechanisms in the context of IoT platform federations. The current section presents the relevant concepts of bartering and trading and describes the questionnaire designed to understand the IoT platforms involved in the symbIoTe project and what aspects of cooperation they value. The results of the survey are then presented, along with the conclusions that can be taken from it. These were used to design bartering and trading solutions that are relevant and can bring value to platform owners. Finally, a scenario for the first implementation of Bartering is presented, along with its architecture design.

5.2 Fundamentals

The basic economic concept of bartering refers to a market situation where two or more market participants exchange their respective goods or services directly for other goods or services, without monetary implications. While the concept itself is a rather old one, it has been repeatedly criticized for its alleged inefficiency, for instance with respect to difficulties in matching suitable partners, issues with determining common value metrics, and problems arising from the fact that certain goods may be indivisible and hence impossible to precisely match in terms of their value. Eventually, the main justification for employing a bartering mechanism originates from the fact that it allows two parties achieving a joint win-win situation without the need of resorting to the explicit exchange of money.

In the context of an IoT middleware like symbIoTe, most of the aforementioned problems disappear by definition: matching suitable partners is relatively easy, as all platforms participating in symbIoTe are assumed to be prosumers, i.e. are interested in offering services to other platforms (as producers) and using services from other platforms (as consumers) at the same time. Hereby, a service typically consists of allowing or making use of access to IoT resources, e.g., sensors and their corresponding data, which circumvents the problem of indivisibility: we can easily define small units of service and thus provide a mutually acceptable metrical unit for comparing the value/worth of an offer or a request.

However, in order to increase the efficiency of the mechanism, we will not employ bartering in its purest form, but instead consider the commonly accepted vouchers as a means to subsume all important properties referring to a service offer or service request. Hence, a voucher typically includes:

- Access Token;
- QoS Constraints;
- Details on requested service (wanted);
- Details on expected value (price);
- Time constraints (e.g., time-out conditions).

An alternative to vouchers (although very similar) is the concept of coupons. Contrary to vouchers, these do not establish an agreement of exchange of resources between federated platforms, but simply offer another platform access to a resource from the platform that issued the coupon. The following example illustrates better, how they would work:

1. Platform 1 (P1) wants to access a resource in Platform 2 (P2).
2. P1 issues a coupon that grants the holder access to one of its resources.
3. P1 request access to P2, offering the generated coupon.
4. P2 can, in the future, use the coupon to access P1 resources.

A more concrete use of the coupons can be found in Section 5.5.

Of course, symbIoTe will also offer a way to access resources from other platforms without an immediate material counteroffer, i.e. by *trading*. Here, three basic scenarios have to be considered:

- *Direct Buy*: a platform sells access to its own resources to an application/enabler or another platform for a fixed price;
- *Forward Trading*: a platform is offering access to its own resources and asks for corresponding requests (bids) from other platforms;
- *Backward (reverse) Trading*: a platform is looking for access to resources offered by foreign platform(s).

Here, an agreement on monetary compensation is fundamental for closing a deal. In microeconomics, such situations are usually treated within the framework of auction theory, i.e., forward auctions (access to resources is offered, and requests are submitted in the form of bids) and reverse auctions (access to resources is requested, and corresponding offers, including access conditions, are received by the platform).

5.3 Survey about Practice and Concepts of Cooperation

In order to further model a concept of Bartering and Trading between symbIoTe platforms, a questionnaire was presented to Platform Owners. In this questionnaire, platform owners were requested to present their perspective of their platforms in general and related to aspects of cooperation as well as the concept of a federation of IoT platforms as it would be meaningful and valuable to them.

The answers from symbIoTe partners will then serve to model bartering and trading scenarios that are relevant and of value to platform owners. Below it is possible to observe the questions presented to the partners. Their answers are included in Annex A – Answers of .

A) Platform Properties

1. What constitutes **your platform**: How do **customers interact**? Which **resources** are you dealing with, and where do they come from? Why do **customers participate**? How is **revenue generated**?
2. Generally, from your perspective: **What constitutes** a symbIoTe-compatible IoT platform in general?

B) Federation Properties

3. Generally, from your perspective: **What constitutes a federation**: Agreements? Mutual trust? Easy access to foreign resources? Please be as specific as possible.
4. What could be **incentives or necessary conditions for your platform to enter a federation**? Be as specific as possible.
   a. Can you describe the **relevance of creating added values as an incentive** for your IoT platform to enter a federation?
   b. Can you describe the **relevance of the size of the other partners** as a potential condition for your IoT platform to enter a federation? Would you rather prefer a **federation amongst** (roughly equal) **peers, or a more heterogeneous** (in size etc.) composition of partners in a federation?

C) Cooperating Platforms within a Federation

5. Generally: What could be **incentives** or **necessary conditions to cooperate** with other IoT platforms?
6. Generally, from your perspective: Are **cooperations** among IoT platforms **rather orthogonal or complementary** in nature? Why?
7. What **value** do **you** see in **cooperating with other IoT platforms** in symbIoTe?
8. How would **you** like to **cooperate** with other platform owners? Be as specific as possible.
9. What partners are valuable especially **to you, with whom would you like to cooperate**? Be as specific as possible.
10. Please describe **your ideal platform cooperation scenario, including partners**, which you would like to cooperate with **in order to create value**, either for you or your customers.
11. How could **resource bartering support you** in this cooperation? Be as specific as possible.
12. How could **resource trading support you** in this cooperation? Be as specific as possible.

5.4 Fundamental Bartering and Trading Scenarios

In this section, the conclusions that can be taken from the answers to the questionnaire are presented. Through these conclusions, it is possible to define bartering and trading scenarios that are relevant and bring value to the platform owners.

The analysis of the questionnaire has led to the classification of platform owners into three clusters with different characteristics, as can be seen in the following table (Table 2).

	Lightweight	Standardization-related	Trade
Main Characteristics	Volunteers	Product	Platform as a Federation
Incentives	Exclusive Access	Operative interoperability, seamless interaction	Network effect
Own Perspective	Bartering Only	-	Bartering as trust and reputation building tool;

			Eventually leading to trading
Partners	n:m; improve offer	Customer of customers; exchange data	1:n; compliance
Goods	Sensor access/raw data	Filtered data	Data/SLA
Interaction	Prosumers	Customer of customers	Consumers
Example Platform	OpenIoT, NAssist, MoBaaS	Navigo	NAssist, MoBaaS, Kiola
symbIoTe mechanism	**Bartering**		Trading

Table 2. Classification of platform owners

The various types of classification are the following:

- **Lightweight**: consists of platforms, which generate data as well as consume high quality data and are willing to exchange data/services exclusively by means of bartering.
- **Standardization-oriented**: the main focus relies on the interoperability provided by symbIoTe itself and sees the B&T functionality just as a nice feature to have.
- **Trade**: the platform's main focus is to exchange data/services by means of trading, thus the monetary gain can be considered the main goal. Before committing to a trade agreement, bartering can be used as a prospecting tool, which can aid a platform in assessing the trustworthiness of a potential partner.

We can also infer from the survey that, within the consortium, OpenIoT can be considered the paradigmatic platform for a Bartering scenario, while NAssist is the best example for a Trading scenario.

Using all this information, we designed three fundamental scenarios:

Bartering outside of a federation (Phase 1: "Dating"):

Assume platforms P1...Pn form an L2-compliant federation F within symbIoTe, while platform P0 is not yet registered in symbIoTe. Because either platform P0 or federation F (or both) have an incentive that P0 should join federation F, a bartering mechanism should allow the potential partners to meet and get to know each other. In order to achieve this goal, P0 is invited by the Core Bartering & Trading Manager to register with symbIoTe (L1) and become L2-compliant. Then, the Core Bartering & Trading Manager asks P0 about which of the platforms P1...Pn is interested in getting data from, and asks each of the platforms P1...Pn whether they are interested in getting data from P0. Based on the replies, the Core Bartering & Trading Manager sends P0 one Bartering Voucher (B-Voucher, which gives access to platform data) for each platform it is interested in and sends a "reverse" B-Voucher (giving access to P0 data) to each of the platforms P0 has shown interest in. Similarly, all platforms P1...Pn which are interested in P0 data receive one B-Voucher each (enabling this access), and at the same time P0 receives corresponding "reverse" B-Vouchers allowing access to platform data within F.

Bartering within a federation (Phase 2: "Engagement"):

Suppose P0 has joined federation F. In order to build up further trust relationships, P0 and one/several of the platforms P1...Pn decide to start bilateral data exchange for free. To this end, the cloud (L2) Bartering & Trading managers of the two platforms willing to cooperate bilaterally produce corresponding B-Vouchers, which give access to the data of the respective partner platform. In contrast to the B-Vouchers of Phase 1, these B-Vouchers now include some rough Quality of Service (QoS) specification (e.g. in terms of quality classes like best effort etc.).

Trading within a federation (Phase 3: "Marriage"):

Finally, assume P0 and some other platforms within federation F have managed to build up sufficient trust with each other that they would like to negotiate data/resource exchange on a more fine-grained and binding level. To this end, their cloud (L2) Bartering & Trading managers will have to agree on a detailed QoS agreement as part of a Trading-Voucher (T-Voucher). Note that in this case, it is also possible that data/resource access is receiving a monetary compensation - in this case, the T-Voucher fulfills merely the function of a contract plus receipt. Such bilateral relations are non-exclusive ("polygamy" is possible). The payment functionality, however, is considered out of scope for symbIoTe and could be realized by some appropriate third-party service.

Note that, in any of these cases, voucher resolution is triggered by the cloud (L2) Bartering & Trading component, which has to initiate an access policy update in the platform components, i.e. AAM and/or RAP, respectively.

5.5 Bartering Sequence Diagram

In order to provide to the symbIoTe ecosystem a Bartering concept that can easily be produced and deployed, it was decided that, for a first approach, a simpler Bartering model should be designed and developed. This approach is similar to the one described in **Phase 2: “Engagement”**. It opts to use coupons instead of the previously presented vouchers. The difference between these two concepts is that vouchers imply an agreement between two platforms on what is going to be bartered between them, while a coupon is a simpler concept, where the holder of the coupon has access to resources from a platform that generated it.

As such, a coupon contains the following information:

- **Issuer (platformId)**: Who issued the coupon.
- **Beneficiary (platformId) (optional):** Who is the beneficiary of the coupon. This is an optional field and, if left as such, it can be passed around through several platforms.
- **Federation Identifier (federationId)**: The federation this coupon belongs to.
- **Resource Type**: Type of resources being bartered.
- **Expiration**: Expiry date of the coupon.
- **single Use**: Boolean indicating if the coupon can be used only once, or several times.

The following diagram represents the flow of actions when a platform wants to access another federated platform’s data under a bartering scenario. The basic idea is that a platform (P1), to access another platform’s (P2) resources, must provide a coupon that grants such access. If P1 does not have such a coupon, it will generate its own coupon and offer it in exchange for a coupon that grants access to the desired resource in P2. P2 can later use the received coupon to access resources on P1. By keeping track of the coupons that are generated and used, the federation has an idea of which platforms are not contributing to the federation (i.e. are generating a lot of coupons that go unused by other platforms) and can take appropriate action.

The platforms are free to define the rules for bartering their own resources. This means that platforms can specify, for example, if they are only willing to barter for certain types of resources, or with platforms above a given trust level. These rules can be defined through a JSON config file, and can be as simple or as complex as the platform owner desires.

It is also important to note the function of the Core B&T component. It keeps track of all the movements regarding coupons (creation, usage, consumption). This allows it to have a general idea of how the Bartering is working within a given federation, allowing the detection of platforms that are not contributing or seeing if certain coupons have expired. These sorts of statistics can be provided later to the Trust Manager.

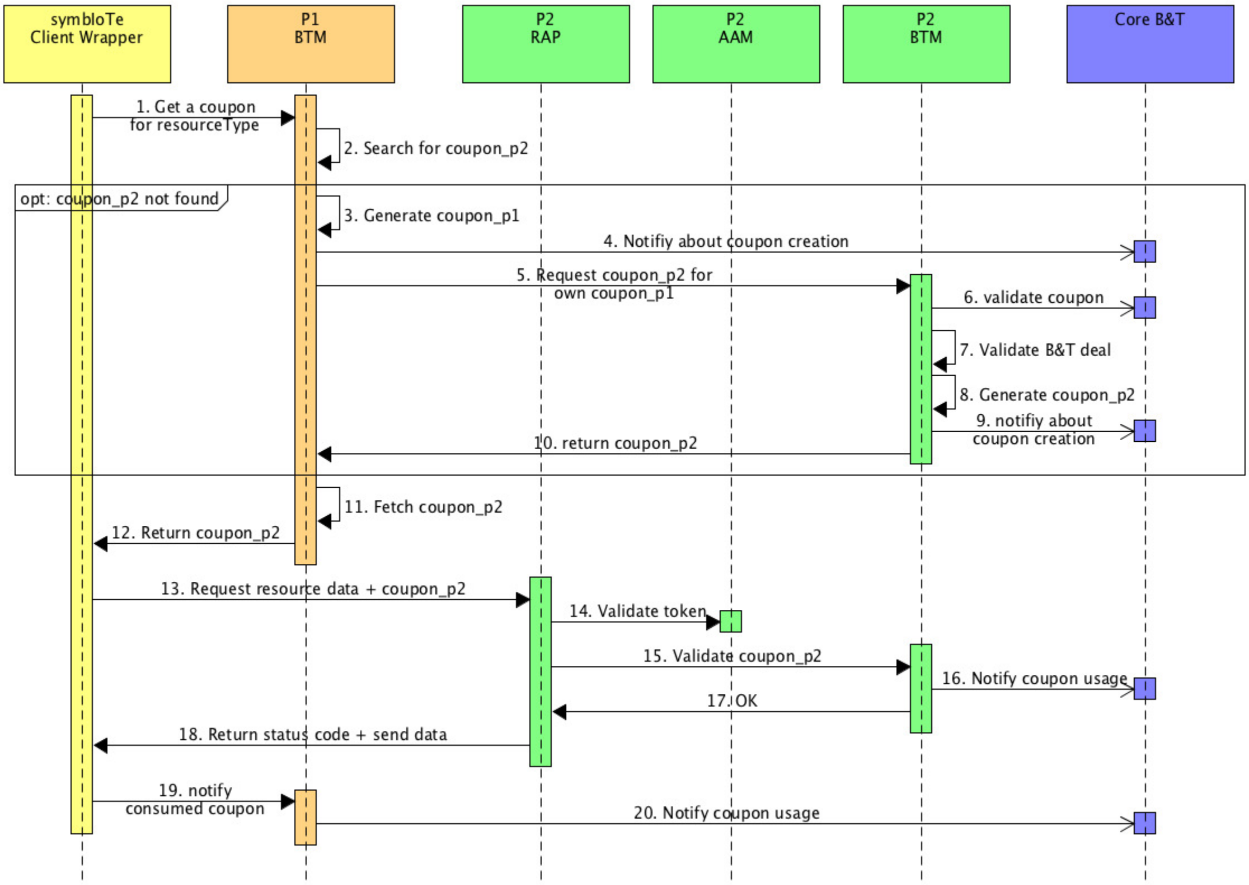

Figure 20. Sequence diagram of Bartering

- *Message 1*: The symbIoTe Client requests from P1 BTM a coupon to access resources in P2.
- *Process 2*: P1 BTM searches for coupons that allow access to resource in P2.
- **If no coupon to access P2**:
    - *Process 3*: P1 BTM generates a coupon that allows the platform holding the coupon to access P1 resources.

  - *Message 4*: Notify Core B&T about the creation of this coupon.
  - *Message 5*: Request a coupon from P2 BTM, sending generated coupon in return.
  - *Message 6*: P2 BTM validates with Core B&T if coupon is valid.
  - *Process 7*: P2 BTM validates if the deal is acceptable.
  - *Process 8*: P2 BTM generated a coupon to be sent to P1 to allow access to its resources.
  - *Message 9*: Communicate to Core B&T about the creation of the coupon.
  - *Message 10*: Return coupon to P1, allowing it to access resources in P2.
  - *Process 11*: Fetch coupon from storage.
- *Message 12*: Return coupon to the client.
- *Message 13*: Request resource data, offering the coupon in exchange.
- *Message 14*: Validate federation token.
- *Message 15*: Validate received coupon.
- *Message 16*: Validate with Core B&T that received coupon has been consumed/used.
- *Message 17*: Acknowledge that everything is ok with coupon.
- *Message 18*: Return the requested data.
- *Message 19*: Notify P1 BTM that it has consumed/used the coupon and successfully received the data.
- *Message 20*: Notify Core B&T that coupon has been used.

5.6 Conclusion and Future Work for B&T

symbIoTe's architecture has to be sufficiently versatile to be able to support different platforms, domains and technologies. As such, a system that can cope with such a rich environment will be complex by nature. As a module of the symbIoTe architecture, Bartering and Trading might not necessarily be a core module needed to make the system function, but it is very important for whoever will use the system in the future. It allows users to buy, sell and trade access to their resources, making the system very dynamic. This is of great interest to prosumers, which are envisioned to be symbIoTe's main users.

Through a questionnaire to the platform owners of the consortium, it was possible to design Bartering & Trading scenarios that are relevant and of value to them. Three B&T scenarios were envisioned that created a flow of trust creation between the platform owners, where, through the stages of “Dating”, “Engagement” and “Marriage”, platform owners could create a more connected partnership.

With a simpler bartering scenario designed, future work will focus on developing, deploying and testing the bartering components, so that platforms can take value from this symbIoTe component.

6 Federated Security

Security in symbIoTe was always treated with utmost care and the developed framework was from the beginning designed to offer unreachable authentication and authorization layer.

In order to achieve this task symbIoTe security architecture comprises multiple layers, each with a different purpose, which are combined into a complete solution. As such, we can define the following security layers:

1. Baseline Security.
2. Authentication.
3. Authorization.
4. Adverse Events Detection.

We will present how each of these layers contributes to secure federations with B&T mechanisms.

6.1 Baseline

Baseline security is a set of dev-ops practices and requirements imposed on the system required to provide a secure deployment environment, e.g., by enforcing Transport Layer Security (TLS) on all communication channels (HTTP, AMQP) as well as providing audits of software developed in the project. This way we make sure that the interactions defined in the next sections can assume a secure channel between the actors and build upon this assumption.

6.2 Authentication

Federations make full use of the symbIoTe Public Key Infrastructure (PKI) to manage identities of actors and services participating in distributed scenarios. symbIoTe Core services offer a root Certification Authority (CA) and the registered platforms in their Cloud services are given intermediate CA powers to issue credentials for their users.

SymbIoTe-defined PKI provides, therefore, a vital support for authentication of actors and services in federation scenarios by building certification chains and verifying if communicating parties originate from the same root CA (symbIoTe Core) which proves their authenticity.

Finally, the cryptographic material (elliptic curve keys) generated at this level is utilized to provide signatures for payloads in the authorization layer. This way we are able to provide a stateless mutual authentication mechanism where both the client and service can sign the business payloads and be authenticated by the communication receiving party

6.2.1 Credentials Validation

Validation of tokens, keys stored in them and the signature is one of the most important security tools in symbIoTe.

A unified API is exposed for the application developers to validate tokens, available in the Security Helper library. To check the validation, symbIoTe security layer firstly verifies the token string content to determine whether it is malformed during transmission or whether it has a correct signature and has not yet expired. Afterwards, the AAM authority checks the token.

Core or Platform AAMs check if the token issuer exists in the symbIoTe ecosystem, if the issuer's or subject's public key was revoked, or if the issuer's or subject's certificate has expired. What's more, AAMs check in their databases, if the token was not revoked due to a security breach.

In case of Foreign Token, validation of participation in the federation is made. If a Platform has been removed from at least one federation after Foregin Token generation, the token is considered invalid and client has to acquire a new one.

Furthermore, if an AAM of a federated platform is presented with a Foreign token issued by itself, it reaches out to the platform that issued the Home token to check if the origin credentials (aforementioned home token and clients public keys) were not revoked which would trigger revocation of the Foreign token used in attempt for authorization.

6.2.2 Revocation

Due to the requirements in federations and components – actual usage of Foreign Tokens - the revocation APIs presented in D1.4 [1] were improved. Now, symbIoTe additionally allows the following actors to revoke the following:

- *symbIoTe administrator*:
  - Core AAM components certificates;
- *AAM administrator*:
  - its components certificates;
- *all of the actors in all AAMs*:
  - foreign tokens.

All actors will need to present their credentials (username and password) to authorize their actions. In case of foreign token revocation, only corresponding remote home token is needed.

6.2.3 Mutual authentication

There were no changes made to the challenge-response procedure described in section 5.4.4 of D1.4 [1].

6.3 Authorization

At the authorization level, since symbIoTe strives to achieve interoperability in a complex ecosystem with many resources and actors, we have decided to provide a very generic authorization scheme framework, namely the Attribute-Based Access Control (ABAC) paradigm. This allows platform owners to map their native authorization schemes, e.g., Identity Based Access Control schemes on top of ABAC which is available out of the box in symbIoTe.

As a standardized vector for these attributes, symbIoTe makes use of JSON Web Token concept and its trusted JSON Web Signature implementation. All authorization credentials (attributes) are released to the actors from dedicated Authentication and Authorization Managers (acting as the aforementioned CAs) and signed with their certificates. This way the ABAC protected services (resources) can trust the contents of received attributes set from the clients. A client can prove its rights to use a particular JWS Authorization Token by providing to the service a hashed challenge signed with the private key matching the one included in the Authorization token, serving as the supplement for the aforementioned service authentication.

It is worth noting that the Authorization layer also supports two other concepts:

1) the Multi-Domain Access Right Composition - gaining access to a particular resource by presenting authorization credentials acquired from different parties;
2) an Attribute Evaluation Engine that adds semantics to the attributes themselves, i.e., not only that it allows to check if the authorization payload contains a required attribute, but it also can evaluate whether the value(s) associated with attribute(s) match specified constraints (e.g. numeric value greater than a threshold or a string value containing a specific substring).

6.3.1 Authorization token

At Level 2 we use the same definition of tokens as defined in D1.4 [1], hence there are no changes here. To log into a service, an actor has to generate and send Login Request to the AAM. Login Request is a JWS with relevant claims. To identify an actor trying to get access into the system, a Login Request contains actor's unique identifier and the actor's client identifier signed by the actor’s client itself. The procedure is described in detail in Section 10.2 (Annex B of this document).

6.3.2 Tokens supporting federations

A Foreign Token is a token generated by AAMs of platforms participating in federation. It is created in exchange for an actor’s Home Token. After its verification and validation, a Foreign Token is generated containing new attributes defining affiliation to the appropriate federation(s).

A Foreign Token preserves information about a Home Token issuer and its unique identifier. This information is needed to check whether in the meantime the Home Token, based on which the Foreign Token was issued, was not revoked due to any reasons, e.g., a security breach.

6.3.3 Access Policies

Section 5.4 of D1.4 [1] has already briefly presented the Access Policies to resources used in symbIoTe. The mechanism has in the meantime evolved.

We have already supported Single Domain Access Policies, which fully satisfy the needs of Federations. However, as symbIoTe security-related solution is a flexible and reusable, T3.2 has been working on **Multi-Domain Access Rights Composition**.

The Multi-Domain Access Rights Composition (MDARC) represents an innovative paradigm conceived in the context of H2020 symbIoTe project, which envisages the opportunity to share heterogeneous resources in a federated ecosystem to applications with properties and access rights obtained from different domains. This paradigm is an evolution of what has already been developed and goes beyond the objectives of the project itself. Furthermore, it could be used in symbIoTe for L3 and L4 Compliance.

Details of MDARC are available in Annex B (Section10). It include the following:

- Overview and its possible applications in symbIoTe.
- Possible approaches to implementing MDARC using eXtensible Access Control Markup Language - including a review of few existing frameworks out in the market.
- Discussion showing that symbIoTe is designed in accordance with the XACML architecture.
- Description of symbIoTe custom JSON-based Access Policies Domain Specific Language.

6.4 Adverse Events Detection

Finally, at the highest level, symbIoTe Core Services can receive alerts about suspicious authentication and authorization operations across the ecosystem and process them using the Anomaly Detection Module in order to identify possible threats (e.g. multiple login attempts, databases crawling or denial-of-service attacks on the core and platforms components).

The WorkInProgress (WIP) Anomaly Detection Module (ADM) will:

- gather information about suspicious authentication and authorization operations:
  - from AAMs;
  - from components;
- analyze Zipkin traces.

For receiving information about detected anomalies and security requests containing suggested actions, a plugin to existing modules is being developed: Anomaly Listener Extension. It will offer an Anomaly Listener Service and be integrated with the Security Handler component.

Although this layer is not strictly related only to L2, it should be possible to integrate it with the L2 Trust Manager to penalize misbehaving platforms (and their clients).

6.4.1 Zipkin

Zipkin is a distributed tracing system. It helps gather timing data needed to troubleshoot latency problems in microservice architectures. It manages both the collection and lookup of this data. Zipkin’s design is based on the Google Dapper paper.

SymbIoTe modules are instrumented to report timing data to Zipkin. In case of troubleshooting latency problems or errors, traces can be sorted based on an application, length of trace, annotation or timestamp. This information can be processed by the Anomaly Detection Module to discover strange network traffic, which indicates an attack.

6.4.2 Anomaly Listener Extension

Anomaly Listener Extension (ALE) is a dedicated library that depends on the Authentication Authorization Manager component and provides an interface (IAnomaliesHelper) for managing blocked actions repository for a specific AAM.

It can be used by codes provided in SymbIoTeSecurity library to memorize malicious actions and check if a user is blocked for his behavior.

Internally it consists of the following:

- Blocked Actions Repository that contains entries specifying blocked username, blocked event type, and timeout for blockade;
- A service that implements IAnomaliesHelper;
- A RESTful controller, that handles requests related to detected anomaly (requests from Anomaly Detection Module - ADM).

The communication between the modules is shown below:

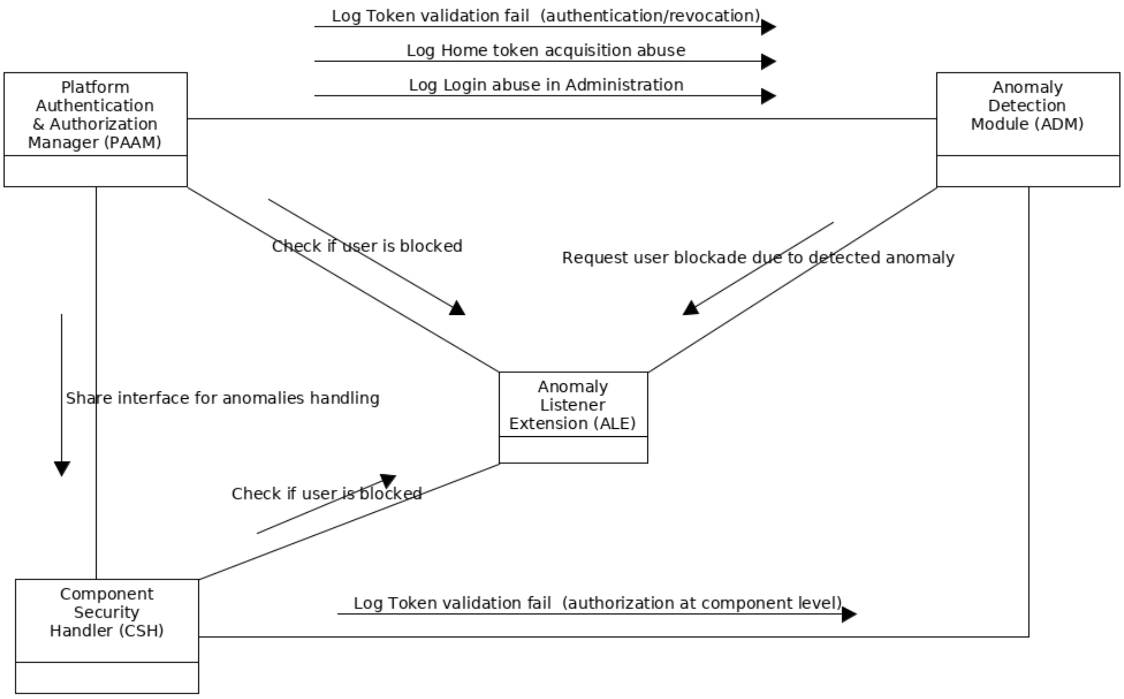
Log Token validation fail (authentication/revocation)
Log Home token acquisition abuse
Log Login abuse in Administration
Platform Authentication & Authorization Manager (PAAM)
Anomaly Detection Module (ADM)
Check if user is blocked
Request user blockade due to detected anomaly
Share interface for anomalies handling
Anomaly Listener Extension (ALE)
Check if user is blocked
Component Security Handler (CSH)
Log Token validation fail (authorization at component level)

Figure 21. Interactions for anomaly detection

6.5 Security Levels

With consideration of the security layer's communication impact on the SymbIoTe performance we propose two modes of security in symbIoTe:

- **Offline Mode:**

On this level, a component presented with AuthZ tokens is contacting its local AAM for their validation. therein this case, the revocation verification is performed using tokens that have been issued by this AAM. If the tokens were issued by a different AAM, only the Certificates-Trust-Chain validation is performed (for details see D1.4 [1] Section 5.4.1).

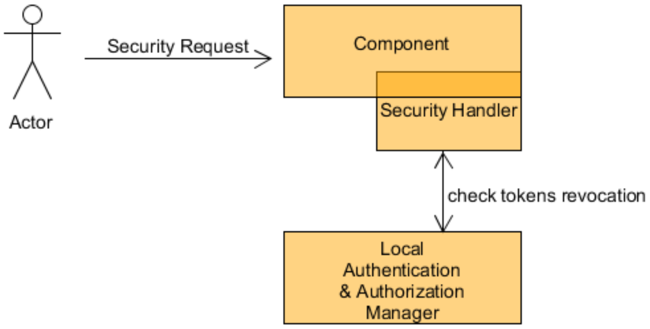

Figure 22. Offline Mode

- **Online Mode**:

The highest level of security is offered when communication with all available AAMs can be guaranteed. A local AAM connects to the issuers of tokens and certificates to check whether they are still valid and have not been revoked in the meantime

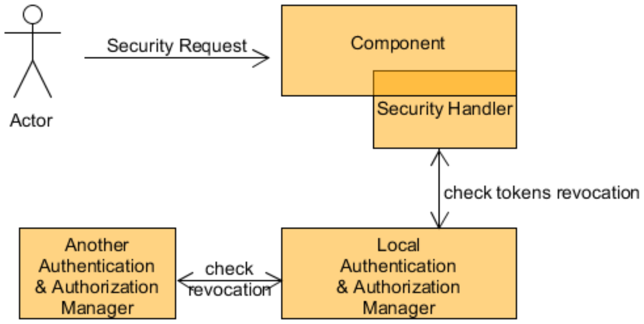
Actor
Security Request
Component
Security Handler
check tokens revocation
Another Authentication & Authorization Manager
check revocation
Local Authentication & Authorization Manager

Figure 23. Online Mode

It is worth mentioning that since security-related communication may create a high overhead in terms of generated messages and interactions between all the involved AAMs, the AAMs in symbIoTe offer a configurable cache to store validation results for remote tokens (if a tokens was found to be valid) and automatically persist revoked tokens to reduce unneeded cross platform communication. Of course, this cache might contain some stall entries and should associate a validity period to each cached entry.

6.6 Security Interfaces and Services supporting federations

6.6.1 AAMs: Interfaces and Services

Changes in federations have forced some updates in the existing interfaces and services. The new APIs has been created to manage federations, acquire foreign tokens and revoke them.

Management of the federations can be performed only by the Administration. Admin credentials, federation identifier and set of platform identifiers participating in federation are needed.

Foreign token acquisition can be performed using REST clients. A POST request containing client's Home Token and optionally a Privacy Enhanced Mail (PEM) Certificate String matching SPK from token is sent to the AAM.

Similarly to foreign token acquisition, foreign token revocation can be performed: a REST client sends a POST request containing its Foreign Token and the corresponding Home Token.

6.6.2 Security Handler (SH): Interfaces and Services

Security handler is a thin Java client providing methods allowing clients to acquire authorization and authentication credentials required to gain access to symbIoTe resources. It also contains method helping to acquire federated tokens from not local AAMs:

- *Map<AAM, Token> login(List<AAM> foreignAAMs, String homeToken)* - allows one to acquire FOREIGN tokens from AAMs in which one does not have an account in exchange for a selected home token.

Foreign tokens are also cached in the Security Request wallet with the rest of the credentials.

6.7 Plans for Security

The symbIoTe security team will collaborate closely with bartering and trading team to reuse the developed concepts of handling Auth(Z) credentials and apply them to the Bartering coupons (issuing, validation, revocation).

Furthermore, the MDARC and attribute level access policies definition mechanisms are being worked on.

Then, existing AAM code will be modularized to allow reuse of its features within the WP4 monolithic SSP deployments.

Finally, T3.2 team will work on improving the Client Security Handler to offer higher level interface to the symbIoTe Security layer, SH will also receive a boost to its independence from the symbIoTe Core. T3.2 will also focus the efforts on Anomaly Detection Engine development.

7 Conclusion and Next Steps

This document provides a description of the concepts and mechanisms that allow platforms to form federations in symbIoTe, taking into account the different paradigms for resource sharing such as Quality of Service based or bartering and trading, with a special emphasis on the security and trust issues and solutions that we have found like Service Level Agreements for QoS assessment, ABAC, certificates trust chains and JWT for security and the use of secure signed coupons for bartering.

Future work, which has been described in detail in each section, will complete the implementation of the different components of the architecture as well as the algorithms described in this deliverable, applying the different security mechanisms to all of them and providing a final set of components that will enable IoT platforms to collaborate in an autonomous, reliable and secure way.

References

[1] symbIoTe project Deliverable D1.4 - Final Report on System Requirements and Architecture; July 2017.

[2] symbIoTe project Deliverable D2.4 – Revised Semantics for IoT and Cloud Resources; July 2017

[3] symbIoTe project Deliverable D2.5 - Final symbIoTe Virtual IoT Environment Implementation; July 2017

[4] Caronni G., Walking the Web of Trust, 2010

[5] WS-Agreement standard – https://www.ogf.org/documents/GFD.107.pdf

[6] Oasis, Advancing open standards for the information society: https://www.oasis-open.org/org

[7] XACML4J: https://mvnrepository.com/artifact/org.xacml4j

[8] SpEL + Spring Security: https://www.researchgate.net/publication/266659391_Attribute_based_access_control_for_APIs_in_Spring_security

[9] AuthZForce - https://authzforce.ow2.org/bin/view/Main/

[10] OpenAZ - https://github.com/apache/incubator-openaz

[11] Axiomatics - https://www.axiomatics.com/attribute-based-access-control/

[12] Casbin/jCasbin - https://github.com/casbin/casbin

[13] AT&T XACML - https://github.com/att/XACML

[14] node-abac - https://github.com/simon-barton/node-abac

8 Abbreviations

AAL	Ambient Assisted Living
AAM	Authentication and Authorization Manager
ABAC	Attribute Based Access Control
ADM	Anomaly Detection Module
AMQP	Advanced Message Queuing Protocol
ALE	Anomaly Listener Extension
API	Application Programming Interface
ARR	Access Resource Request
Auth(N)	Authentication
Auth(Z)	Authorization
CRUD	Create, Read, Update and Delete
CSR	Certificate Signing Request
DoW	Description of Work
GA	Grant Agreement
HCI	Human-Computer-Interaction
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technology
IoE	Internet of Everything
IoT	Internet of Things
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MDARC	Multi-Domain Access Rights Composition
PAP	Policy Administration Point
PDP	Policy Decision Point
PGP	Pretty Good Privacy
PEM	Privacy-enhanced Electronic Mail
PEP	Policy Enforcement Point
PIP	Policy Information Point
POPD	Protection of Personal Data
QoE	Quality of Experience
QoS	Quality of Service
RAP	Resource Access Proxy
REST	Representational State Transfer

RAP	Resource Access Proxy
SH	Security Handler
SLA	Service Level Agreement
SPK	Subject Public Key
TLS	Transport Layer Security
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language
WoT	Web of Trust
WTP	Willingness-To-Pay

9 Annex A – Answers of questionnaires

	1. Your Platform	2. Platforms Generally	3. Federations Generally	4. Your Incentives for federating	4a. Relevance of creating added values for federating
MoBaaS	Basically, MoBaaS accepts various types of sensor data from cities (traffic, parking, air quality) and processes the data to offer services to said cities (municipalities and citizens). Through web and mobile apps, users can use the various services (e.g. ecological routing). The resources are sensors deployed throughout the city, provided by the cities themselves. The platform provides its clients with smart city services. Revenue is generated by the city paying for the platforms services.	A platform which makes its data and services available through interoperable interfaces. A platform that complies with the various defined compliance levels (don't know if this is the answer you were looking for?).	Generally, it enables a better cooperation between platforms. As such, through mutual trust, it should provide better ways for platforms to interoperate. It allows platforms the enrich their services with more data and to enlarge their geographical reach. Additionally, it enables the access to foreign resources.	Firstly, when federating, security of data and quality of services must guaranteed. It is also necessary to know if the federation candidate has valuable data that can enrich our own platform. Then, as long there is a benefit to federate with another platform, the only deterrent to federate is the amount of work involved in setting up the services needed.	The added value would be the main incentive. It would have to be enough to justify the effort.
Navigo Digitale	The Navigo Digitale IoT Platform (ND) consists of an integrated IoT system for the management of touristic ports' services. Development of the platform has been funded by Navigo. ND is currently part of Navigo's commercial offer to touristic ports: this offer is still in development and hasn't generated revenues so far (we expect to formally launch it at the Yare conference in April 2017).	Interoperability is the key aspect: we are working on the Smart Yachting (SY) use case, which, in order to work, needs that the platforms of the Yacht and the Port can seamlessly interact.	Access to resources (eg data from sensors in the Yacht) is paramount to make the SY use case work. For federation, security and reliability (e.g. the integration "just works") looks like key issues.	From the SY perspective, federation is an enabling condition, since two different platforms (on the port and on the yacht) must interoperate to make the use case work. In our case, it is really a "chicken & egg" scenario: ports will install SY compliant IoT platforms if enough Yachts can interoperate; Yacht owners will install symbIoTe, SY compliant, IoT platforms if the SY use case is available in enough ports.	For SY is a bit like "the other way round". We should offer incentives to allow yachting manufacturers to install symbIoTe, SY compliant, IoT platforms on board. These incentives in perspective are related to the quality of application services that we will be able to add to the SY use case (eg more port software applications to integrate through enablers; other smart problems that we can solve through M2M interactions, etc.).
KIOLA (not in DoA)	Platform accepts various types of health data usually generated by medical devices, e.g. a blood pressure meter, a glucose meter or a fitness tracker. This data is stored within the platform and made accessible through webapps for medical	Interoperability is the key aspect. A lot of sensor vendors use their own platforms, so symbIoTe could seamlessly tie them all together	Easy access to foreign resources	A key issue is security as KIOLA is dealing with highly-sensitive person-related information	

	staff. Revenue is generated through insurance companies paying for the platform services				
OpenIoT	Platform consists of volunteers who are willing to contribute data about their environment (i.e. air pollution). Except those volunteers, we have only a platform operator/owner as other stakeholder, which provides an infrastructure (computing power and sensor units). Resources are user's (volunteer's) mobile phones + sensors provided by the platform operator. We do not have a specific communication with users and we do not generate any revenue, nor users are paid to carry sensors around. They do it to contribute to their community.	I see a basically three types of users: 1. platform owners that provide an infrastructure and computing power; 2. users that already use some specific platform (they have some domain-specific app on their phone, tablet, pc...) and 3. app developers which aim for new, potential users of their (innovative) apps - they drain those new users from the existing user base which use **some**symbiote-compatible platform. Who will generate revenue? - Not clear to me, users are definitely willing to pay for good apps, but how that money will be divided by all stakeholders (platform owner(s), symbIoTe provider(s), users that actively contribute data) is not clear to me!	In my opinion federation represents an agreement between two platform providers, and that agreement enablers better service (including the better quality or some novel services) to their users. I believe that a federation to exists, both platforms needs to see benefits from it.	Those benefits can be various: exclusively access to the data (but really exclusively, meaning nobody who is not in the federation can not access it), accepting and serving users from different platforms, selling data to users from different platforms.	What is added value in this context? I believe we need to present strong case here, i.e. clearly communicate what is added value for a platform to establish a federation. But as I said earlier, I fully agree **some kind** of added value will be a trigger to create/join a federation.
UniversAAL	UniversAAL is an IoT platform for home automation and health services with a specific focus on elderly people; As UniversAAL is also used within the health area, the answers given in the KIOLA section hold true here as well				
BETaaS	BETaaS will not be a platform that will be integrated within Symbiote neither used in any use cases. BETaaS is a software that can be used in built a platfrom, but we don't have an existing running instance from the platform.				
Symphony	Symphony can monitor, supervise and control many different building systems, devices,	A platform which makes its data and services available through the	Generally speaking, federating platforms allows the resources exposed	One key aspect is the access to federated platform resources: how it is	The use cases are the principal entry point to show the value added of

	controllers and networks available from third-party suppliers. By intelligently correlating cross-system information, a flexible and highly efficient platform is delivered to the stakeholders. The Symphony Insight management station in the cloud allows operations, administration and management of the BMS from any authorized remote terminal. The innovative BM-as-a-service paradigm provides a scalable service architecture, data security and privacy, customized dashboards and business intelligence.	symbIoTe interoperable interfaces	by one to be used by another and vice-versa. This implies many key aspects to be considered (i.e. trading, security, ecc..).	controlled and where rules themselves are specified. The regulation of the access to the resources is key point for federating platform.	entering the federation. Smart Residence and Smart Yachting are scenarios where federation is a sort of enabling condition.
Worldline (not in DoA)	Smart Stadium converts visitors' smartphones and retailers' smartphones/tablets to IoT devices that can be discovered and accessed to satisfy their needs in sport events: selling, buying, getting informed of news and updates. All devices and stadium supplies can be located by proximity using iBeacons throughout the stadium.	Platforms register their services and related devices to make them accessible via standardized APIs.	Federations stablish the rules and benefits to make resources and services from a platform accessible to a second platform.	Federation is required to combine, to make collaborate different platforms to provide further features to end users. Devices (smart devices and beacons) known by a platform are offered to unknown platforms to enrich their experience and increase their benefits.	As said before, the added value is the cooperation of different platforms to enrich their features to end users.
nAssist	nAssist is a software platform that enables the creation of different services on top of it. We have customised the platform to offer smart home, security, energy efficiency and health applications. Depending on the applications, our customers are telecoms, energy service providers, healthcare providers, building construction and management, hardware and appliance manufacturers, retailers, insurance companies and end-users. Customers interact with the platform through a web-based application or a mobile app. They can monitor energy consumption at home, control some appliances, receive notifications and alerts in real time, receive pictures / video	IoT platforms are compatible with SymbIoTe if they can maintain the integrity of their policies (organisational, technical and security aspects) between the different stakeholders, e.g. that users authenticate with their local credentials, data integrity, etc. This means that SymbIoTe needs to guarantee trust between entities, reputation and branding.	A federation should include organisational and business agreements that guarantee autonomy, cooperation and flexibility. From a technical point of view, it should include trust and security.	Collaboration opportunities, network security, reputation, access to new business models, new services without building new infrastructure by connecting solutions, easier access to new customers and partners.	S&C is a private company that enhances competitive differentiation by increasing the added value to its services and products. A federation is an effective way to deploy new service-based business models, and to access customers and collaboration opportunities.

	in real time and obtain predictions about energy consumption. End-users participate because they can achieve a number of benefits from these applications: controlling of their home, awareness of energy efficiency and, therefore, saving money, improving understanding and awareness of their health status, improving wellbeing. The rest of customers achieve insights about end-users behaviours that enable them to improve and customise their services. We deal with sensors and actuators located at home that monitor contextual information that enable monitor user behaviours, adapt environment to their needs and forecasting.				
Spine	Spine is an hardware platform. It's just a board exposing different IoT and wireless protocol on an USB connection.				

	4b. Relevance of partner size for federating	5. Incentives and conditions for cooperating among platforms	6. Cooperations rather orthogonal or complementary	7. Value of platform cooperation for you	8. Your cooperation preferences
MoBaaS	Very indifferent, as long as there is value in the federation and the security and quality of service are guaranteed.	Ease of cooperation between federated platforms; selling/exchange of resources/services. Additionally, from a business perspective, it can be a requirement by the platform buying/selling services/data or a public tender.	Can be both. Using the Smart Mobility use case as an example, for the interpolation workflow, the cooperation between Open IoT and Uwedat is orthogonal, where they both cooperate to rate the air quality levels of streets. On the other hand, MoBaaS can use this data to feed its routing service, where the service offered by the other two platforms is more complementary.	Ease of development of cross-domain applications through the interoperability provided by the symbIoTe ecosystem.	Within the Smart Mobility use case, MoBaaS will use the data generated by OpenIoT and Uwedat in the Interpolation workflow to power its routing engine. This can be extended to any platforms (or combination of) of any city, allowing MoBaaS to provide the routing service anywhere where such data is provided.
Navigo Digitale	Yachting industry is a very peculiar one: there isn't a single large technological provider but a variety of small companies offering services to yacht manufacturers and touristic ports. We are facilitated for the SY use case by the fact that the largest yachting manufacturers are in Italy and already collaborates with symbIote partners Navigo and Nextworks. Navigo also organizes important event for this industry like Yare: we expect that this can facilitate other technological providers to embrace symbIoTe and the SY use case.	Incentives: the availability of the specific IoT platform in the Yachts produced by important manufacturers.	For implementing the SY use case it seems to me that cooperation is complementary (if I correctly get the question…). Platforms in the boat and in the port must interoperate to fulfil the use case.	Mostly related to marketing. We offer a solution for ports: the more Yachts can interoperate with our platform, the more it will be appealing to our potential customers.	Technological providers for the Yachting industry in general: Nextworks to start with.
KIOLA (not in DoA)	It is not really the size of the partner, as the reputation within the healthcare area as well as other factors, such as: does the partner platform comply with regulatory aspects in the health domain ?	In the health domain linkage of different health sensors (which may be in different IoT domains) help to track certain health aspects of a person; e.g. indoor localization + fitness tracker + blood pressure meter sensors help to determine the daily activities of an	Both	Simple integration of various (health-related) sensors in different platforms	e.g. bringing home automation services and health monitoring closer together

		eldery person, which in return can be seen as incentive.			
OpenIoT	Size definitely matters (smile). I see two scenarios: 1st (in my opinion more likely): a small (new) player in the market will try to join a federation with a big player (i.e. platform operator with large user base) to accelerate its penetration to the market share. Big player will not have to do much (since a small player(s) will have to adapt to the big player, and big player will collect its cut in all revenues). In this scenario I don't see a lot of cooperation between players. Examples of big playes: ATOS, IBM offering exclusive rights to use their platform which is used in various domains and have a high number of users. Small players: SME(s) that have developed their own domain-specific platform and (mobile) app for it. Now they want to extend their user base and enable that all 'big player' users can also use their platform/data. 2nd scenario: two roughly equals partners will make a federation (or even some kind of a merger) to increase their cumulative market share (and influence). In this scenario, the cooperation between partners is expected, both sides need to invest some effort.	I see two main incentives: 1st (obviously) money. 2nd user base which becomes accessible to both platforms. Necessary condition: both need to see some interest	Can be both in my opinion... see answer 4b	Ideally through symbIoTe. symbIoTe can offer means to find suitable platform and facilitate the federation agreement (especially from a technical point of view)	So far, OpenIoT did not envision any type of cooperation (except of cooperation between multiple instances which basically means sharing the data).
UniversAAL					
BETaaS					
Symphony	The size matters from a commercial point of view, this is obvious, and with size we refer to the dimensions of the company, more than to the IoT platform which we are federating with. The main relevant thing is the services provided: this is the first thing to consider when joining a federation.	The ease of the federation process is the first incentive to proceed: of course a difficult integration will discourage any possible partner. The second point is the possibility to create an added value to the services provided by the platform, and certainly this is	Mainly complementary. Even collaborating orthogonal platforms can be considered complementary, if we see orthogonal collaboration from a holistic point of view	Relatively simple development of complex services over multiple collaborating platforms.	Smart Yachting: collaborating with Navigo Digitale to implement the use case. Smart Residence: collaborating with KIOLA, if needed, for home automation & health care features.

		strictly related to money.			
Worldline (not in DoA)	Its size is not as relevant as the quality of the information and services it provides.	Incentives: money and/or a big enhancement of my platforms. Conditions: ease of usage and development as well as shared effort in the integration.	Can be both depending on the use case.	Easily integrate features from third-parties into your application via the simplified API from symbIoTe.	Retailer platforms, sports events, calendars and real-time updates.
nAssist	It will depend on the functionalities and services offered by the other partners and their trust and reputation, more than their size. It could be interesting to have trust and reputation models for partner selection when there are more than one partner offering the same services or functionalities.	The main condition to cooperate with other IoT platforms will be the possibility to create new services without building and investing in new infrastructure.	It will depend on the services to be built. For example, sharing outdoor tracking data with indoor tracking data can enable to create more robust user behaviour patterns that will lead to more precise health applications. This would be a complementary cooperation. However, an orthogonal cooperation would be necessary to create smart grids services.	To create innovative solutions without spending time to build new infrastructure. Cooperation will bring together the range of expertise and abilities to design and deploy these solutions.	Sharing data/knowledge and creating new services together.
Spine					

	9. Platform partners valuable to you for cooperating	10. Your ideal platform cooperation scenario	11. Support by bartering in this cooperation?	12. Support by trading in this cooperation?
MoBaaS	The partners within the Smart Mobility Use Case (UNIZG-FER and AIT). In the future, it could be extended to other platforms who provide similar services in different cities (as stated in the previous answer). It could be extended to partners who can use the solution in their local regions or that provide contact with possible customers (counties, environmental entities, etc.)	MoBaaS provides services for cities. It needs sensor data to provide this services. Anything that allows the finding and usage of these resources is valuable. The most desirable cooperation would be with a partner with an excellent data coverage over a city. The main difficulty for the kind of services MoBaaS provides is obtaining high quality data and the lack of investment to create a high quality coverage.	Exchange of the platform's services with the data from providers. Still there needs to be some profit (transforming raw "material" to a product). So maybe trading is more adequate for this case or the profit form selling the services can be divided by the data and service providers somehow. In case some entity doesn't want to make their data available, it can be an incentive for them to do so by trading their data with our services.	Through backward trading, obtain access to resources to feed the services, which are later sold to municipalities. As such, our platform would be in the middle of the value chain Data Source -> Service -> End user, paying the Data Providers for their data and being paid for the offered services by the End Users.
Navigo Digitale	See point 8.	The cooperation scenario is clearly defined in the SY use case. For partners, see point 8.	No bartering scenario foreseen at present for our platform.	No trading scenario foreseen at present for our platform.
KIOLA (not in DoA)	Home automation platforms (e.g. capable of indoor localization)	A home automation platform capable of doing indoor localization	Resource batering and trading is a difficult topic in the health area, as resources are natually very sensitive;	See 11
OpenIoT	Partners from symbIoTe? If yes, then those involved in use case (AIT and UW).	Ideal scenario: when I'm missing something (either a physical device or data item) I would go and look and ask my 'partners' if they have it! With this (when I find it) I can offer some new service. This is valid for platform operators/owners and app developers.	For start, we need to define what is a resource? A physical thing or data? To barter a physical devices we need to develop technical preconditions to enable such functionality! To barter data - how to assess when data is exchanged and how much this is worth? But overall, yes bartering of resources could help our use case, in a way that users would get better quality information.	Same answer as previous question...
UniversAAL				
BETaaS				
Symphony	Navigo for SY and AIT for SR	The collaborating scenario are the SR and the SY use case	No bartering scenario foreseen at present for our platform	No trading scenario foreseen at present for our platform
Worldline (not in DoA)	From symbIoTe context, it would be interesting to collaborate with our colleagues from EduCampus use case. Our platforms involving visitors, retailers and information points can coexist in universities,	A platform offering decentralized services to citizens. We offer almost real-time notifications to visitors, so any partner could	Yes, once the cooperation is defined in terms of what's a resource, how valuable is the provided service, value of the data.	Yes, same as 11.

	plus those having important sport teams and events.	take advantage of this if their features fit our business.		
nAssist	In SymbIoTe context, it could be interesting to cooperate with smart home use case, in particular with health use case. For example, if these partners can measure health indicators such as vital signs, we could combine this information with the monitoring of activities of daily living (ADLs), creating robust healthcare services at home.	Our ideal platform cooperation scenario should include an easy and secure access to the information provided by the different IoT platforms by supporting networking requirements, privacy data trust, data anonymization, context in which the data are embedded (for data analytics purposes).	Data/services bartering will facilitate our inclusion in the marketplace by putting our solutions on the spot. A bartering system will enable to extend our solutions to include shared data/services and to find new opportunities while the brand reputation is built. A bartering system leads to immediate benefits and investment in the future.	Once the reputation and the corresponding trust have been achieved, data/services trading will facilitate our role as providers by improving our business outcomes and finding cost-efficiencies.
Spine				

Table 3 Answers to questioners

10 Annex B – Updated implementation of L1

In this annex, it is presented novelties and updates to security related content described in D1.4 [1] and D2.5 [3] as the work of T3.2 continued and improved some of the security layer mechanism presented there.

The most notable changes is providing two layer authentication:

- using actors credentials to create accounts and acquire their client's certificates:
  - including full Certificates Signing Requests support;
- using the certificates and private keys to request HOME tokens and generate mutual-authentication payloads.

We describe below new and updated procedures necessary to acquire SymbIoTe Auth(N) and Auth(Z) payloads.

10.1 Certificates acquisition

In order to acquire relevant SymbIoTe certificates through Administration GUI or directly using the AAMs' REST endpoint, the actor (user/platform owner) needs to provide their credentials and a CSR with the following specifics:

Actor	AAM type	Input's format (CSR)	Input's format in REGEX	Result
Common (either ordinary user (app) or platform owner)	Core & Platform	CN=username@clientId@platformId (or SymbIoTe_Core_AAMfor core user)	^(CN=)(([\w-])+)(@)(([\w-])+)(@)(([\w-])+)$	User client's certificate for acquiring HOME tokens
Home AAM Administrator credentials		CN=componentId@platformId	^(CN=)(([\w-])+)(@)(([\w-])+)$	components' certificate (e.g. SymbIoTe_Core_AAM for the core)
Platform Owner	Core	CN=platformId	^(CN=)(([\w-])+)$	Platform AAM's certificate

Table 4 Certificates acquisition

10.1.1 Clients' / Components' Certificates acquisition

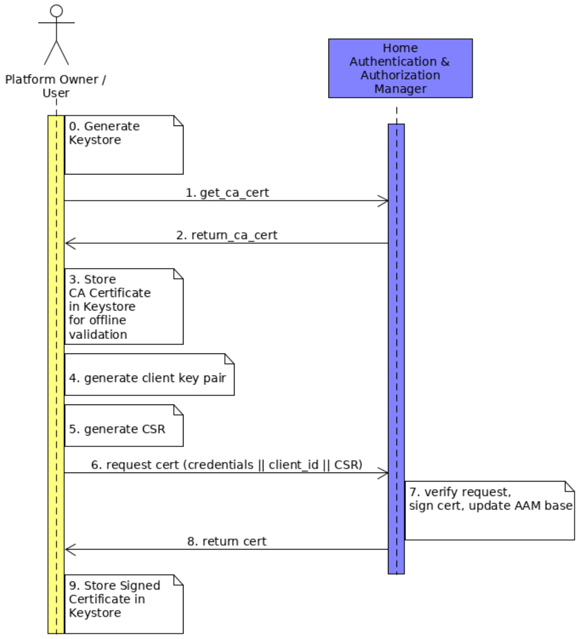
Platform Owner / User
Home Authentication & Authorization Manager
0. Generate Keystore
1. get_ca_cert
2. return_ca_cert
3. Store CA Certificate in Keystore for offline validation
4. generate client key pair
5. generate CSR
6. request cert (credentials || client_id || CSR)
7. verify request, sign cert, update AAM base
8. return cert
9. Store Signed Certificate in Keystore

Figure 24 Sequence Diagram of certificates acquisition

- *Operation 0*: User has generated a Keystore.
- *Message 1*: Generated by the User and delivered to their Home AAM to fetch the certificate fields and fill in the CSR properly, it is a request for the (root or intermediate CA (AAM's) certificate.
- *Message 2*: Home AAM delivers CA certificate to the User.
- *Operation 3*: User stores received CA Certificate in his Keystore for offline Trust-chain Validation.
- *Operation 4*: the User generates a client key pair (public and private) by applying ECDSA_256 algorithm.
- *Operation 5*: the User generates a certificate signing request (CSR) with a CN matching the aforementioned scheme that matches the Home AAM certificate data.
- *Message 6*: the User sends a request for a certificate to the Home AAM (e.g. for the core through the Administration module) that has the following elements:
    - username,
    - password,
    - client_id,
    - CSR (Certificate Signing Request)
- *Operation 7*: Home AAM verifies the request and signs the certificate that was requested in the CSR; newly signed certificate is written in Home AAM's database for that particular user and its client_id.

- *Message 8*: generated by Home AAM and delivered to the client, it returns either a valid certificate or an error message.
- *Operation 9*: User stores received, signed Certificate in his Keystore.

From now on, the User can log in to his Home AAM and acquire home tokens from it.

10.1.2 Platform AAM intermediate CA certificate acquisition from Core AAM

Prerequisites:

- PO is registered in the CoreAAM
- Platform is registered in the CoreAAM

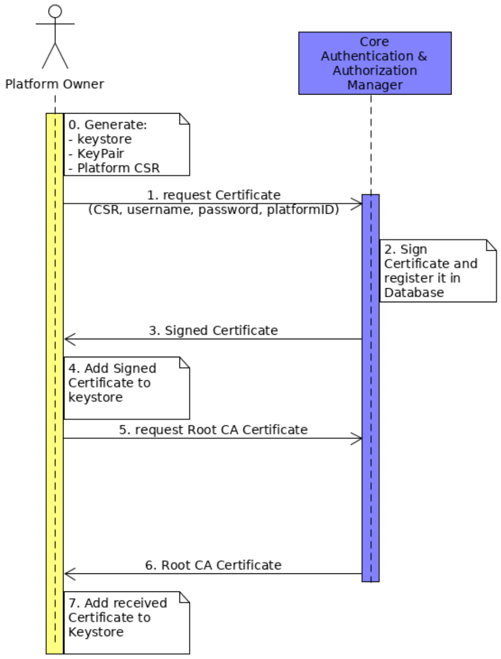
Platform Owner
Core Authentication & Authorization Manager
0. Generate:
- keystore
- KeyPair
- Platform CSR
1. request Certificate
(CSR, username, password, platformID)
2. Sign Certificate and register it in Database
3. Signed Certificate
4. Add Signed Certificate to keystore
5. request Root CA Certificate
6. Root CA Certificate
7. Add received Certificate to Keystore

Figure 25 Sequence Diagram of intermediate CA certificate acquisition

- *Operation 0*: Platform Owner has generated all of the following: Keystore, KeyPair and Platform CSR.
- *Message 1*: Platform Owner requests signing his certificate from the Core Authentication and Authorization Manager. Request should contain: CSR, PO's username, PO's password, Platform's ID.
- *Operation 2*: Core Authentication and Authorization Manager signs received certificate and registers it in Database.
- *Message 3*: Core Authentication and Authorization Manager returns Signed Certificate to Platform Owner.
- *Operation 4*: Platform Owner adds received, signed Certificate from Core Authentication and Authorization Manager to previously generated Keystore.

- *Message 5*: Platform Owner sends a request to the Core Authentication and Authorization Manager to receive Root CA Certificate.
- *Message 6*: Core Authentication and Authorization Manager responds to request with Root CA Certificate.
- *Operation 7*: Platform Owner adds Root CA Certificate to Keystore. At this point Platform Owner has java keystore with the certificates and a private key needed to run a platform AAM instance.

10.2 Token acquisition

Having initialized their client (with AuthN material), the actors (users) can proceed to acquire AuthZ payloads - tokens

10.2.1 Home token acquisition

Prerequisites:

- user has issued a certificate for the client he wants to log from

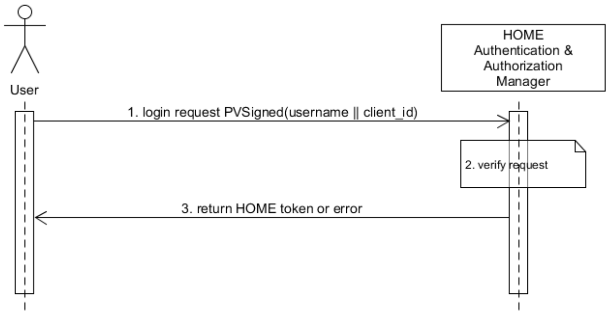
User
HOME Authentication & Authorization Manager
1. login request PVSigned(username || client_id)
2. verify request
3. return HOME token or error

Figure 26. Full online

- *Message 1*: generated by the user and sent to HOME AAM, it is a login request that includes username and client_id and signed with the private key that matches this tuple
- *Message 2*: AAM verifies the data sent by the user by comparing it with its database records
- *Message 3*: generated by HOME AAM and sent to the client; it contains a HOME token with a public key that matches the private key used for signing the login request

10.2.2 Foreign token acquisition

In exchange for HOME tokens, the clients can try to acquire FOREIGN tokens.

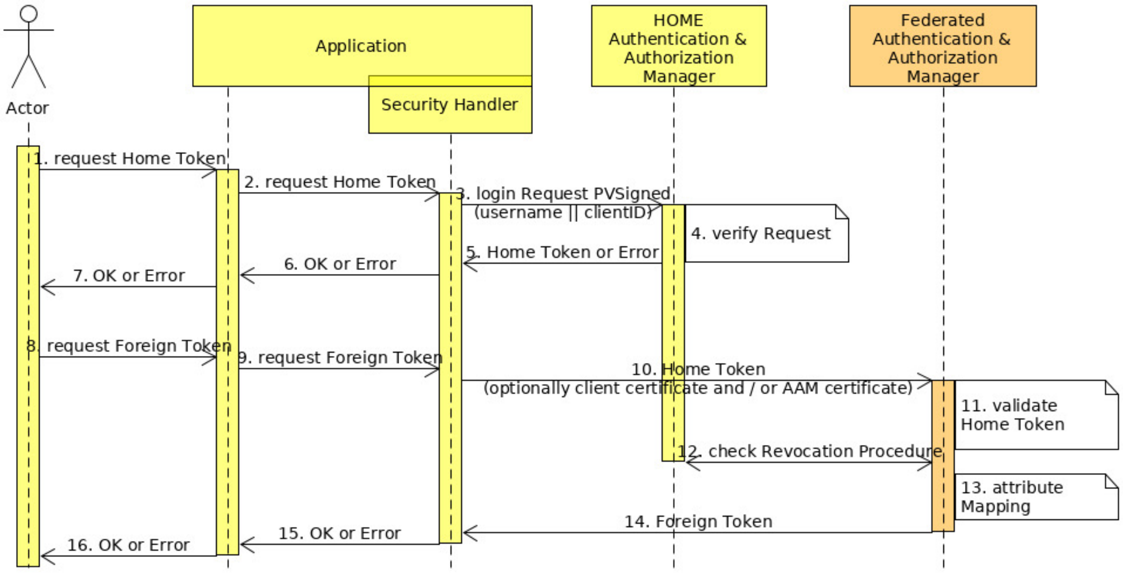

Figure 27. Sequence Diagram of foreign token acquisition

- *Procedure 1*: User requests HOME token.
- *Message 2*: Application forwards request to Security Handler
- *Message 3*: Security handler sends Login Request to Home Authentication and Authorization Manager to obtain Home Token
- *Procedure 4*: Home Authentication and Authorization Manager verifies received request
- *Message 5*: If verification is successful Home Authentication and Authorization Manager returns Home Token to Application's Security Handler
- *Message 6*: Security Handler informs Application whether obtaining Home Token was successful
- *Message 7*: User is informed by Application if Obtaining Home Token succeeded
- *Procedure 8*: User Requests Foreign token
- *Message 9*: Application forwards request to Security Handler
- *Message 10*: Application's Security Handler sends Home Token optionally with client certificate and / or AAM certificate to Federated Authentication and Authorization Manager
- *Procedure 11*: Federated Authentication and Authorization Manager validates received Home Token
- *Procedure 12*: Verification of any asynchronous revocation of the remaining tokens (i.e., if token has been revoked by the Home AAM before the expiration time indicated within the token itself).
- *Procedure 13*: Federated AAM checks with Attribute Mapping Function if the HOME token presented in the request is allowed to obtain a Foreign token

- *Message 14*: Upon successful validation and Attribute Mapping Federated Authentication and Authorization Manager provides Foreign token to Application's Security Handler
- *Message 15*: Application's Security Handler informs Application about outcome of obtaining Foreign Token
- *Message 16*: Application forwards obtained information to User

10.2.3 Guest Token Acquisition

The guest token grants access to public resources registered in the Core and/or Platforms Registries.

Prerequisites:

- client has network access to whatever symbiote AAM

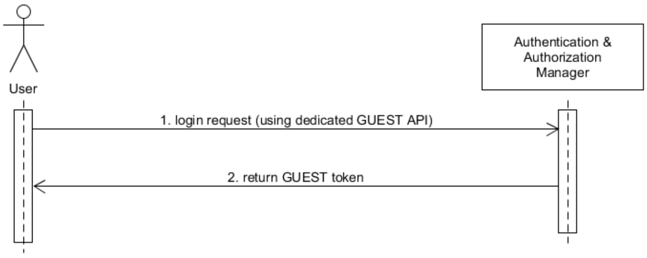
User
Authentication & Authorization Manager
1. login request (using dedicated GUEST API)
2. return GUEST token

Figure 28. Sequence Diagram of guest token acquisition

- *Message 1*: generated by the user and sent to arbitrary AAM in symbIoTe; the user logs in to GUEST AAM with dedicated GUEST API
- *Message 2*: generated by the AAM and sent to the user; it is a response that contains a GUEST token (and possibly, still under discussion a private key and a certificate of that token)

10.3 Access to resource

In this diagram, we present the full interaction in between Core and Cloud domain components for a client using application core account and exchanged federated token to access a resource.

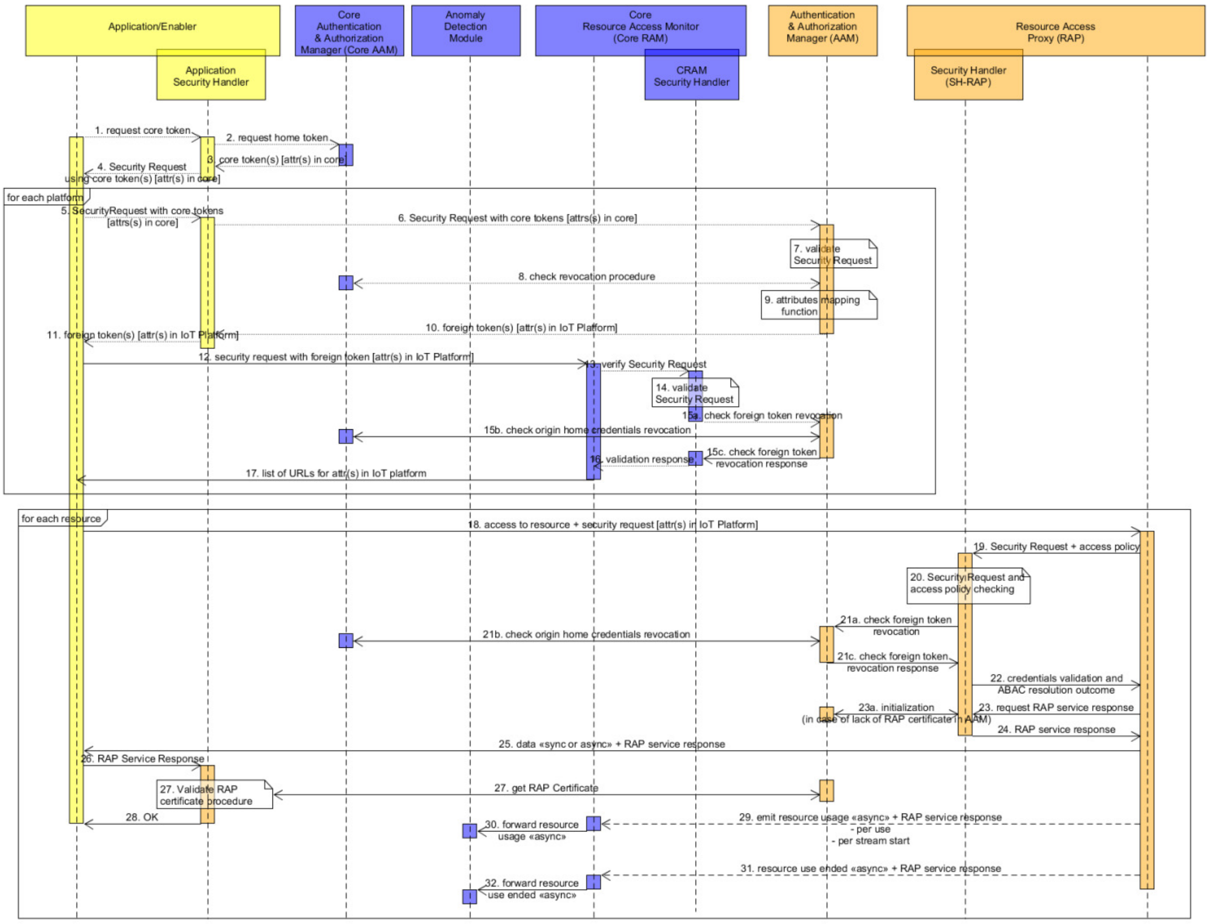
Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager (Core AAM)
Anomaly Detection Module
Core Resource Access Monitor (Core RAM)
CRAM Security Handler
Authentication & Authorization Manager (AAM)
Resource Access Proxy (RAP)
Security Handler (SH-RAP)
1. request core token
2. request home token
3. core token(s) [attr(s) in core]
4. Security Request using core token(s) [attr(s) in core]
for each platform
5. SecurityRequest with core tokens [attrs(s) in core]
6. Security Request with core tokens [attrs(s) in core]
7. validate Security Request
8. check revocation procedure
9. attributes mapping function
10. foreign token(s) [attr(s) in IoT Platform]
11. foreign token(s) [attr(s) in IoT Platform]
12. security request with foreign token [attr(s) in IoT Platform]
13. verify Security Request
14. validate Security Request
15a. check foreign token revocation
15b. check origin home credentials revocation
15c. check foreign token revocation response
16. validation response
17. list of URLs for attr(s) in IoT platform
for each resource
18. access to resource + security request [attr(s) in IoT Platform]
19. Security Request + access policy
20. Security Request and access policy checking
21a. check foreign token revocation
21b. check origin home credentials revocation
21c. check foreign token revocation response
22. credentials validation and ABAC resolution outcome
23a. initialization (in case of lack of RAP certificate in AAM)
23. request RAP service response
24. RAP service response
25. data «sync or async» + RAP service response
26. RAP Service Response
27. Validate RAP certificate procedure
27. get RAP Certificate
28. OK
29. emit resource usage «async» + RAP service response
- per use
- per stream start
30. forward resource usage «async»
31. resource use ended «async» + RAP service response
32. forward resource use ended «async»

Figure 29. Sequence Diagram of access to resource

For apps registered in the:

Core AAM

- *Message 1 (optional)*: generated by the Application/Enabler and sent to the Application Security Handler. It is used to issue home token from the Core AAM. If the Application/Enabler is already logged in, it is not necessary.
- *Message 2 (optional) (AppAAInterface)*: generated by the Application Security Handler and sent to the Core AAM in which the Application/Enabler is registered (the application must perform the "sign in" with the AAM before, so we assume the application is already registered). It is used to authenticate the Application/Enabler. If the Application/Enabler is already logged in, it is not necessary.
- *Message 3 (optional)*: generated by the Core AAM in the IoT platform and sent to the Application Security Handler. It is used to provide the home token(s) with attributes included. If the Application/Enabler is already logged in, it is not necessary.
- *Message 4 (optional)*: generated by the Application Security Handler and sent to the Application/Enabler. It is used to deliver the core token.

Platform/Enabler - interaction is as above but with the relevant (be it Platform/Enabler) home AAM to the use. Acquiring foreign tokens is useful only in FEDERATED scenario, in normal usage of owned resources, a HOME token is enough.

- *Message 5 (optional)*: generated by the Application/Enabler and sent to Application Security Handler. It is used to trigger the operations for obtaining the foreign token from IoT platform. If the Application/Enabler already has valid foreign token, it is not necessary.
- *Message 6 (optional) (AAInterface)*: generated by the Application Security Handler and sent to the foreign AAM in IoT platform. It is used to trigger the operations for obtaining the foreign token(s). If the Application/Enabler already has valid foreign token, it is not necessary.
- *Procedure 7 (optional)*: verification of the time validity, authenticity and integrity of the provided token. If the Application/Enabler already has valid foreign token, it is not necessary.
- *Procedure 8 (optional) (PlatformAAInterface)*: verification of any asynchronous revocation of the token (i.e., if token has been revoked by the home AAM before the expiration time indicated within the token itself). If the Application/Enabler already has valid foreign token, it is not necessary.
- *Procedure 9 (optional)*: procedure that, in case it is needed, translates attributes that the Application/Enabler has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Application/Enabler already has valid foreign token, it is not necessary.
- *Message 10 (optional):* generated by the foreign AAM and sent to the Application Security Handler. It is used to deliver foreign token with new attribute(s). If the Application/Enabler already has valid foreign token, it is not necessary.
- *Message 11 (optional)*: generated by the Application Security Handler and sent to the Application/Enabler. It is used to forward the foreign token generated at the previous step.
- *Message 12 (ChooseResInterface)*: Application/Enabler sends request access to selected resources to Core Resource Access Monitor. Message includes tokens used for authorization
- *Message 13*: It is used to ask to the security handler to verify the complete validity of the Security Request
- *Procedure 14*:verification of the time validity, authenticity and integrity of the provided token
- *Message 15*: verification of any asynchronous revocation of the token (i.e., if token has been revoked by the home AAM before the expiration time indicated within the token itself).
- *Message 16*: It is used to communicate the outcome of the Security Request validation procedures performed by the CRAM Security Handler.
- *Message 17*: Core Recoure Access Monitor returns list of URLs for selected resources in IoT platform
- *Message 18 (AccessResourceInterface)*: generated by the Application/Enabler and sent to the Resource Access Proxy in the resource responsible IoT platform. It is used to access resources, while providing the authorization tokens previously obtained.

- *Message 19*: generated by the Resource Access Proxy and sent to the Security Handler in the resource responsible IoT platform. It is used to ask to the security handler to verify the complete validity of the token.
- *Procedure 20*: verification of the time validity, authenticity and integrity of the provided token and check if the attributes included in the core token satisfy the access policy associated to the requested resource.
- *Procedure 21*: verification of any asynchronous revocation of the token (i.e., if token has been revoked by the home AAM before the expiration time indicated within the token itself).
- *Message 22*: generated by the Security Handler in the resource responsible IoT platform and sent to the Resource Access Proxy. It is used to communicate the outcome of the token validation procedures performed by the RAP Security Handler.
- *Message 23*: Resource Access Proxy requests a RAP Service Response from it's Security Handler.
- *Message 24*: RAP receives RAP Service Response from Security Handler
- *Message 25*: (ReportUsageInterface) asynchronously emit resource usage per use/per stream start along with Resource Access Proxy Service Response
- *Message 26*: Application/Enabler forwards RAP Response to Security Handler for validation.
- *Procedure 27*: Security Handler validates RAP Response using suitable RAP certificate fetchable from AAM.
- *Message 28*: Security Handler returns status of validation.
- *Message 29*: Forward the usage report per use/per stream start along with Resource Access Proxy Service Response to Core Resource Access Monitor
- *Message 30*: Core Resource Access Monitor asynchronously sends resource usage report to Anomaly Detection Module.
- *Message 31*: (ReportUsageInterface) this message informs Core Resource Access Monitor when the stream is ended
- *Message 32*: Forward the usage end report to Anomaly Detection Module

11 Annex C - Multi-Domain Access Rights Composition

The MDARC paradigm throws its basis on the Attribute-Based Access Control (ABAC) mechanisms. ABAC is the baseline approach adopted by symbIoTe to protect the access to resources. With ABAC, the MDARC paradigm still protects resources by allowing remote applications to access to a given resource only if it is able to be in possession of a set of attributes that matches the access policy assigned to the resource itself.

The key interesting aspect that characterizes the MDARC paradigm refers to the nature of these attributes. The main idea is that the application can be registered in more than one domain (or, to simplify, IoT platforms). Therefore, it is able to collect for each domain a set of attributes that locally encodes its properties and its access capabilities. Indeed, the application can combine these attributes coming from different domains in order to ask the access to a resource exposed elsewhere (see Figure 30).

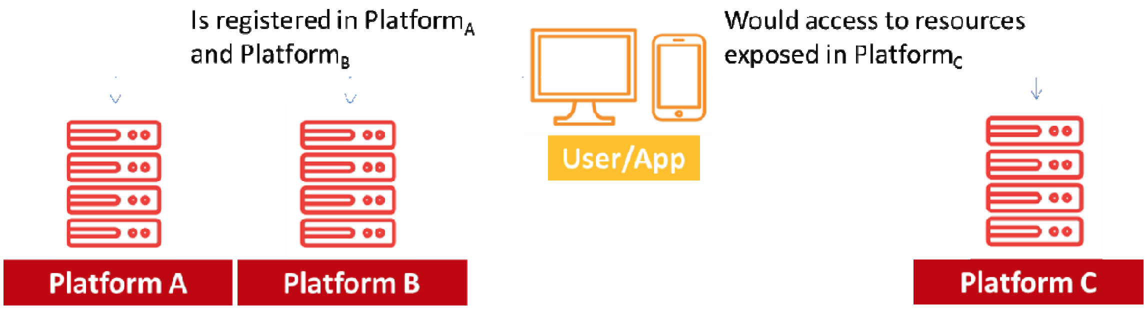
Is registered in Platform$_A$ and Platform$_B$
Would access to resources exposed in Platform$_C$
User/App
Platform A
Platform B
Platform C

Figure 30. Multi-Domain Access Rights Composition

Without loss of generality, the proposed example assumes that there are three IoT platforms: Platform$_A$, Platform$_B$ and Platform$_C$. The application is registered in platforms Platform$_A$ and Platform$_B$. The application would like to access to the resource in Platform$_C$. The policy says that the application must have ATTRIBUTE$_A$ and ATTRIBUTE$_B$ to access the resource. The application can retrieve ATTRIBUTE$_A$ from Platform$_A$ and ATTRIBUTE$_B$ from Platform$_B$. The application combines these attributes and contacts the Resource Access Proxy (RAP) in Platform$_C$.

Thanks to the decoupled native of authentication and authorization mechanisms already developed in symbIoTe for both L1 and L2 levels, the MDARC paradigm can be easily implemented.

Authentication and authorization procedures still embrace three main steps:

- **Home authentications**: At the beginning the application performs the login in the platforms where it is registered to (i.e., Platform$_A$ and Platform$_B$). To this end, it contacts the Authentication and Authorization Manager (AAM) component of each platform for retrieving HOME tokens.
- **Foreign authentication**: The application sends all its HOME tokens to the AAM component of the foreign platform Platform$_C$. The AAM component of the foreign

platform initiates the challenge-response mechanism to verify that the application is the real owner of the tokens, thus preventing both replay and impersonation attacks. In the case the challenge-response mechanism is successfully completed, the AAM component of the foreign platform validates the tokens, verifies that they have not been revoked by contacting AAM components of the HOME platforms (i.e., Platform$_A$ and Platform$_B$) and performs the attribute mapping function. Then, it generates a FOREIGN token that stores the set of attributes mapped from those contained in HOME tokens.

- **Resource access authorization**: The application contacts the RAP and delivers the FOREIGN token retrieved at the previous step. The RAP initiates the challenge-response mechanism to verify that the application is the real owner of the token. In the case the challenge-response mechanism is successfully completed, the RAP verifies that the tokens are valid and that they have not been revoked by contacting its reference platform AAM component. Then it checks the provided attributes against the access policy associated with the requested resource: if the attributes supplied by the applications are sufficient to satisfy the access policy associated to the resource (according to the ABAC logic) the RAP grants access to the resource. Otherwise, the access is denied.

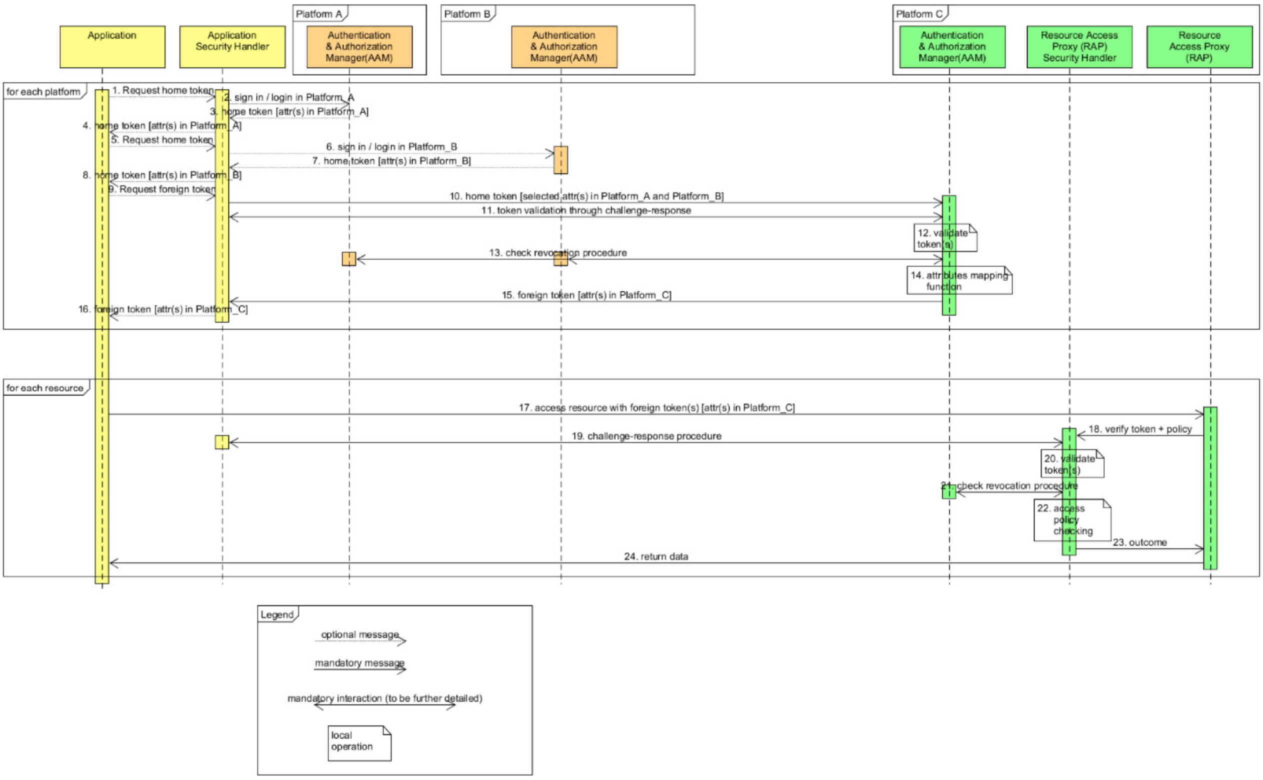
Platform A
Platform B
Platform C
Application
Application Security Handler
Authentication & Authorization Manager(AAM)
Authentication & Authorization Manager(AAM)
Authentication & Authorization Manager(AAM)
Resource Access Proxy (RAP) Security Handler
Resource Access Proxy (RAP)
for each platform
1. Request home token
2. sign in / login in Platform_A
3. home token [attr(s) in Platform_A]
4. home token [attr(s) in Platform_A]
5. Request home token
6. sign in / login in Platform_B
7. home token [attr(s) in Platform_B]
8. home token [attr(s) in Platform_B]
9. Request foreign token
10. home token [selected attr(s) in Platform_A and Platform_B]
11. token validation through challenge-response
12. validate token(s)
13. check revocation procedure
14. attributes mapping function
15. foreign token [attr(s) in Platform_C]
16. foreign token [attr(s) in Platform_C]
for each resource
17. access resource with foreign token(s) [attr(s) in Platform_C]
18. verify token + policy
19. challenge-response procedure
20. validate token(s)
21. check revocation procedure
22. access policy checking
23. outcome
24. return data
Legend
optional message
mandatory message
mandatory interaction (to be further detailed)
local operation

Figure 31. Sequence Diagram of Multi-Domain Access Rights Composition

- *Message 1 (optional)*: generated by the Application and sent to the Application Security Handler. It is used to trigger the recovery of the HOME token from the Platform A. If the Application is already logged in Platform A, it is not necessary.

- *Message 2 (optional) (HomeAAInterface)*: generated by the Application Security Handler and sent to the home AAM in Platform A, in which the Application is registered. It is used to authenticate the Application. If the Application is already logged in, it is not necessary.
- *Message 3 (optional)*: generated by the home AAM in the platform A and sent to the Application Security Handler. It is used to provide the HOME token with attributes included. If the Application is already logged in Platform A, it is not necessary.
- *Message 4(optional)*: generated by the Application Security Handler and sent to the Application. It is used to deliver the HOME token from the Platform A.
- *Message 5 (optional)*: generated by the Application and sent to the Application Security Handler. It is used to trigger the recovery of the HOME token from the Platform B. If the Application is already logged in Platform B, it is not necessary.
- *Message 6 (optional) (HomeAAInterface)*: generated by the Application Security Handler and sent to the Home AAM in Platform B, in which the Application is registered. It is used to authenticate the Application. If the Application is already logged in, it is not necessary.
- *Message 7 (optional)*: generated by the home AAM in the Platform B and sent to the Application Security Handler. It is used to provide the HOME token with attributes included. If the Application is already logged in Platform B, it is not necessary.
- *Message 8 (optional)*: generated by the Application Security Handler and sent to the Application. It is used to deliver the HOME token from the Platform B.
- *Message 9 (optional)*: generated by the Application and sent to Application Security Handler. It is used to trigger the operations for obtaining the foreign token(s) from IoT platform C. If the Application already has valid foreign token(s) for IoT Platform C, it is not necessary.
- *Message 10 (optional) (ForeignAAInterface)*: generated by the Application Security Handler and sent to the foreign AAM in Platform C. It is used to trigger the operations for obtaining the FOREIGN token from Platform C. If the Application already has valid FOREIGN token from Platform C, it is not necessary.
- *Procedure 11 (optional) (SecurityInterface)*: procedure that allows the Application Security Handler that is acting on behalf of the Application to demonstrate that it is the real owner of the token(s). If the Application already has valid foreign token(s) from Platform C, it is not necessary.
- *Procedure 12 (optional)*: verification of the time validity, authenticity and integrity of the provided token(s). If the Application already has valid foreign token(s) from Platform C, it is not necessary.
- *Procedure 13 (optional) (ForeignAAInterface)*: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the respective home AAM before the expiration time indicated within the token itself). If the Application already has valid foreign token(s), it is not necessary.
- *Procedure 14(optional)*: procedure that, in case it is needed, translates attributes that the Application has in the HOME Platform A and Platform B in a new set of

attributes that it has in the Platform C. If attributes are the same or the Application already has valid FOREIGN token from Platform C, it is not necessary.

- *Message 15 (optional)*: generated by the foreign AAM and sent to the Application Security Handler. It is used to deliver the FOREIGN token with the new attribute(s). If the Application already has valid FOREIGN token from Platform C, it is not necessary.
- *Message 16 (optional)*: generated by the Application Security Handler and sent to the Application. It is used to forward the foreign token generated at the previous step.
- *Message 17 (AccessResourceInterface) (mandatory)*: generated by the Application and sent to the Resource Access Proxy in the FOREIGN Platform C. It is used to access resources, while providing the foreign token previously obtained.
- *Message 18 (mandatory)*: generated by the Resource Access Proxy and sent to the RAP Security Handler in the FOREIGN Platform C. It is used to ask to the security handler to verify the complete validity of the token and the related access policy.
- *Procedure 19 (AppSecurityInterface) (mandatory)*: procedure that allows the Application Security Handler that is acting on behalf of the Application to demonstrate that it is the real owner of the token(s).
- *Procedure 20 (mandatory)*: verification of the time validity, authenticity and integrity of the provided token(s).
- *Procedure 21 (mandatory)*: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the foreign AAM before the expiration time indicated within the token itself).
- *Procedure 22 (mandatory)*: it is used to check if the attributes included in the FOREIGN token satisfy the access policy associated to the requested resource.
- *Message 23 (mandatory):* generated by the RAP Security Handler and sent to the Resource Access Proxy. It is used to deliver the result of the operation executed at the previous step.
- *Message 24(mandatory)*: generated by the RAP and delivered to the Application. It is used to communicate the required data or to provide some error codes in case the access is denied.

11.1 eXtensible Access Control Markup Language (XACML) Engines

The concept of Attribute Based Access Control (ABAC) represents a logical authorization model, which provides a dynamic and context-aware access control mechanism. It serves to protect resources (i.e. data, devices, services and other) from unauthorized operations like reading, writing, editing, deleting, copying, modifying and executing. The owner of a resource establishes a policy that describes with attributes whom and what operations can be performed on this object. If a subject has the attributes that satisfies the access control policy established by the resource owner, then the subject is authorized to perform the desired operation on that object.

eXtensible Access Control Markup Language is a standard to define the access control policies to resources in XML format, maintained by OASIS [6]. The latest version is

XACML 3.0, standardized in January 2013. Furthermore, the standard defines an architecture that specifies four main actors to handle access decisions: Policy Administration Point (PAP), Policy Enforcement Point (PEP), Policy Decision Point (PDP), and Policy Information Point (PIP).

- **Policy Administration Point (PAP)**: PAP is the repository where the policies are defined and managed. It provides the policies to the PDP.
- **Policy Enforcement Point (PEP)**: PEP is the interface that protect a resource. It receives the access requests and makes a decision request to the PDP to obtain the access decision.
- **Policy Decision Point (PDP)**: PDP is the main point for the access requests. It collects all the necessary information (policy, subject, resource, access type, context) and concludes if permits or denies the access to the resource.
- **Policy Information Point (PIP)**: PIP is the point where the source attributes values are defined.

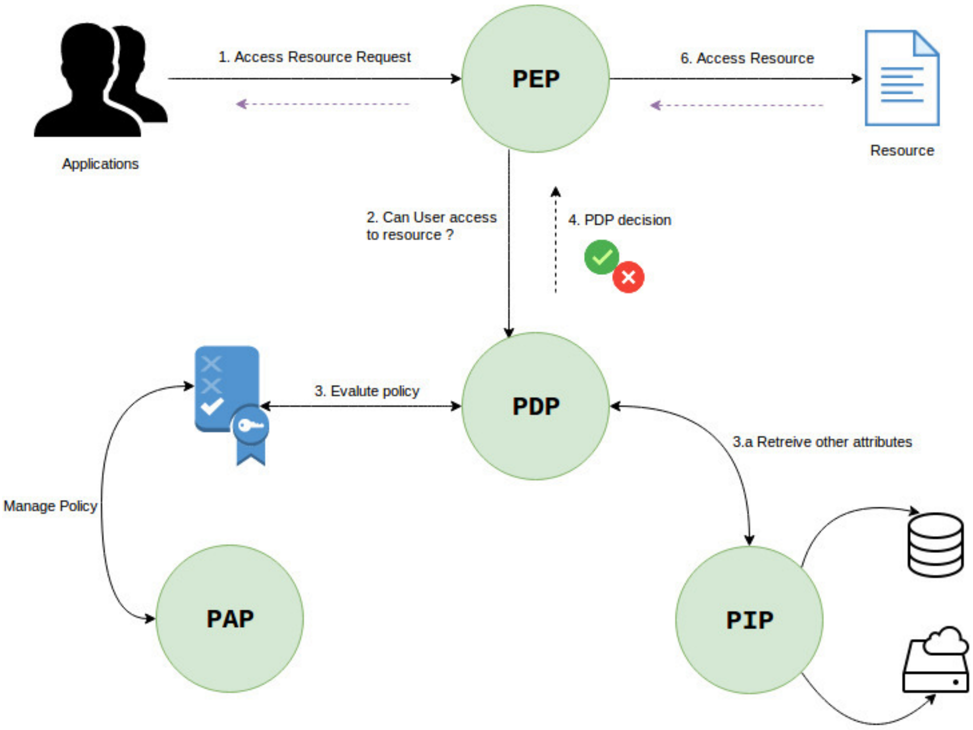

Figure 32. XACML Architecture

Workflow, an Application sends an Access Resource Request (ARR) which is intercepted by the PEP.

1. The PEP converts the ARR in XACML format and forwards it to the PDP.
2. The PDP evaluates the ARR against the policy managed by PAP.
3. If needed it also retrieves attribute values from PIP.

4. The PDP makes a decision (Permit/Deny) and send it to the PEP.
5. The Application can/cannot access to the requested resource.

11.1.1 ABAC Framework Comparison

In this section, an ABAC framework comparison table is presented by highlighting the pros and cons for each solution that could be adopted in symbIoTe.

Name	XACML v.	Tech.	License	Pros	Cons
XACML4J [7]	XACML 2.0, 3.0	Java	LGPL 3.0	• Native implementation of XACML architecture • Spring Module support	• Last commit 2014
SpEL + Spring Security [8]	none	Java	Open Source	• Fully customizable • Possible Integration with XACML	• Implementation from scratch • Poor documentation
AuthZForce [9]	XACML 3.0	Java	Apache 2	• Project maintained. • Fully documented. • Standard compliant.	• Use external PEP proxy • Configuration effort
OpenAZ [10]	XACML 3.0	Java	Apache 2	• Provide PEP Decision API • Standard compliant	• Project retired • Poor documentation
Axiomatics [11]	XACML 3.0	Java	Proprietary	• Perfect integration with Spring Security • Supports multiple environments • Streamlines policy editing	• Commercial nature: • This is not suitable for research purposes • Currently the white papers are not available
Casbin/jCasbin [12]	none	Golang/Java	Apache 2	• Fully customizable • Support ACL, RBAC, ABAC • RESTful • PERM metamodel (Policy, Effect, Request, Matchers) • Many adopters • Provides an XACML translator	• Not manage authentication • Not manage the list of users or roles
AT&T XACML [13]	XACML 3.0	Java	MIT	• Provides PEP, PDP, PAP, PAP Admin, • REST support • Contains sample code with test • PAP contains XACML 3.0 Policy Editor, attribute dictionary support	• XACML JSON Profile v1.0 WD 14 (Last update is Oct 2017)
node-abac [14]	none	JavaScript	MIT	• JSON/YAML Policy • Simple usage, simple configuration	• Poor documentation, examples and test • Basic data type operations

Table 5 ABAC Framework Comparison Table

The advantages of XACML technology compared to using a custom approach include:

- It is a standard language, so it's revised daily by a large expert community. It is not necessary to implement your customized system from scratch.
- It is interoperable, therefore it guarantees the system capability to cooperate and exchange information with other components in a complete and bug-free manner, with resources' reliability and optimization using the same standard language.
- It is generic, so you can enable Access Control in any environment, which allows you to write access policies that fit into different types of applications.
- It is powerful, in fact, it provides support for various features, data types, and extensions.

The goal is to find an effective solution for defining access policies, processing user's attributes, and making decisions on the access request. In order to use a single format to enable data representation (i.e. JSON) in symbIoTe, it is necessary to analyze the frameworks' properties in the literature.

In fact, the introduction of a framework that uses XML as a policy implementation language would be a further effort for integration.

A detailed study of solutions available in the State of the Art, it might be efficient to adopt a not-standardized solution (proprietary), in order to permit:

- an easier integration;
- save time and effort for the implementation;
- avoid the Access Policy engine implementation and the integration from scratch;
- inline with JSON message format exchanged between components.

Although this approach is efficient respect to the adoption for a framework already defined in the literature, this implies a lack of interoperability compared to XACML.

11.2 XACML in symbIoTe

The follow sequence diagram proposed, describes how could be adopted XACML architecture in symbIoTe, in order to provide the Attribute Based Access Control functionalities.

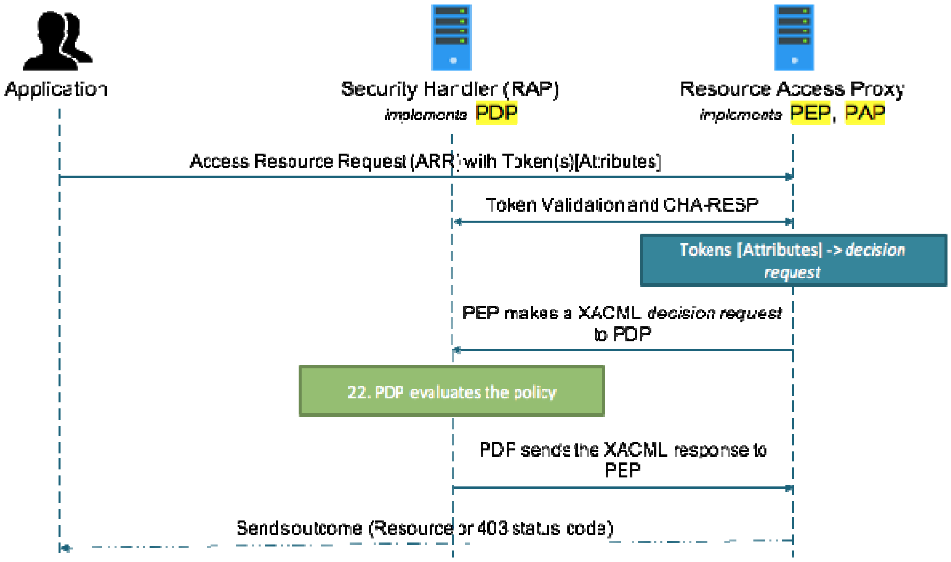
Application
Security Handler (RAP)
implements PDP
Resource Access Proxy
implements PEP, PAP
Access Resource Request (ARR) with Token(s)[Attributes]
Token Validation and CHA-RESP
Tokens [Attributes] -> decision request
PEP makes a XACML decision request to PDP
22. PDP evaluates the policy
PDF sends the XACML response to PEP
Sendsoutcome (Resource or 403 status code)

Figure 33. XACML Architecture proposal in symbIoTe

The solution proposes:

- PDP component should be integrated in the RAP Security Handler;
- PEP and PAP components in the RAP.

Regarding this solution, an application that wants to access resources, should send an Access Resource Request (ARR) by including the Authorization Token(s) that contain attributes. Subsequently the Token Validation operation and the Challenge Response procedure, the PEP will make a XACML decision request to PDP (parsing firstly the received attributes). The PDP defined the SH will evaluate if the attributes sent by the application match the policy by sending the XACML response to the PEP component. Finally, the RAP will send the outcome (the resource requested or an error message status code) to the Application.

11.3 Custom JSON Resolver for Access Policies

Following the requirements of the symbIoTe with regard to fine-grained access control from multiple domains, a DSL (Domain Specific Language) for definition of composite access policies has been designed. First of all, it is vital to identify key entities and keyword for access policies DSL. Following the composite pattern, we have marked access policy with its characteristics as main building block for DSL.

Access policy in symbIoTe can be built in a composite or simple manner. Composite access policy is described using other simple or composite access policies and the relations between them.

For definition of Access Policies in symbIoTe, following keywords should be used (and those are therefore reserved keywords):

- operator - Operator for validating simple access policies or denoting relations between policies in composite access policy;
- policy - Set of simple access policies in composite access policy;
- policyType - Type of access policy: {composite, boolean, numeric, string};
- tokenFieldName - Field name in token that should be used for access policy validation;
- valueType - Type of value against access policy should be validated (Optional) : {numeric, bool, string, enum};
- value - Value against which token field is validated: anySimpleType or array(for string policy operators).

Relations between access policies are denoted using:

- **Composite Policy Operators**
  - AND
  - OR
  - NAND
  - NOR

  Due to the reason that composite access policies in symbIoTe can be built using nested single access policies, it is essential to define rules for building simple access policies. The main difference between simple and composite access policies is that simple access policies cannot carry other access policies needed to satisfy logical clauses during validation of access rights.
- **Main building blocks for simple access policies are**:
  - Comparable type - data type which should be compared against value provided by user in order to validate access policy.

    - Operator - mathematical or logical operation again which the value is compared
        - Definition of operator is dependent on the comparable type in the access policy.
- **Currently defined comparable types in symbIoTe access policy DSL are**:
    - Numeric (integer, float, double);
    - String;
    - Boolean.
- **Furthermore, operators used within simple access policies(depending on the comparable types) are** :
    - Boolean Policy Operators
        - isTrue
        - isFalse
    - Numeric Policy Operators
        - equals (EQ)
        - notEquals (NOT)
        - greatherThan (GT)
        - lessThan (LT)
        - greatherOrEqualThen (GE)
        - lessOrEqualThen(LE)
    - String Policy Operators
        - equalsIgnoreCase
        - IN
        - IN-IgnoreCase
        - NOT IN
        - NOT IN - IgnoreCase
        - regexp

Using this XACML-alike DSL future integrators of the platforms within symbIoTe will be provided with a capability to define fine-grained ABAC based access policies, matching the needs of their native platforms and applications.