Symbiosis of smart objects across IoT environments

688156 - symbIoTe - H2020-ICT-2015

Complete Federation Environment

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: D3.3 – Complete Federation Environment

Type: Public

Editor(s): Christoph Ruggenthaler

E-mail: christoph.ruggenthaler@ait.ac.at

Author(s): Christoph Ruggenthaler (AIT), Jose Antonio Sanchez (Atos), Pietro Tedeschi (CNIT), Giuseppe Piro (CNIT), Gennaro Boggia (CNIT), Giuseppe Bianchi (CNIT), Mikołaj Dobski (PSNC), Jakub Toczek (PSNC), Vasilis Glykantzis (ICOM), Ilia Pietri (ICOM), Pavle Skocir (UniZG-FER), Petar Krivic (UniZG-FER), João Garcia (UW)

Doc ID: D3.3 - Complete Federation Environment_v1.0.docx

Amendment History

Version	Date	Author	Description/Comments
v0.1	24/04/2018	Christoph Ruggenthaler	Initial version and ToC
v0.2	28/05/2018	Jose Antonio Sanchez, Pietro Tedeschi, Giuseppe Piro, Gennaro Boggia, Giuseppe Bianchi, Pavle Skocir, Petar Krivic, João Garcia	Added component descriptions and sequence diagrams
v0.3	29/05/2018	Vasilis Glykantzis	Added additional inputs
v0.4	05/06/2018	Christoph Ruggenthaler	Merged inputs and updated introduction and summary
v0.5	11/06/2018	Christoph Ruggenthaler	Updated section 3.
v0.6	11/06/2018	Pietro Tedeschi, Giuseppe Piro	Added privacy section.
v0.7	13/06/2018	Jakub Toczek, Mikołaj Dobski	Drafting security related sections.
v0.8	15/06/2018	João Garcia, Christoph Ruggenthaler	Updated interfaces and aligned content + structure
v0.9	18/06/2018	Vasilis Glykantzis, Ilia Pietri, Jakub Toczek, Mikołaj Dobski	Updated Platform Registry interfaces; Added BTM content & security updates
V0.10	20/06/2018	Christoph Ruggenthaler	Streamlined layout and structure; Added Conclusion
V1.0	27/06/2018	Christoph Ruggenthaler	Incorporated review comments from Jose Antonio Sanchez, Zvonimir Zelenika (VIP) and Raquel Ventura Miravet (S&C)

Table of Contents

Table of Figures

Table of Tables

1 Executive Summary

The deliverable D3.3 documents the final version of all supported features, sequence diagrams, component descriptions and implementation needed for IoT platforms to become symbIoTe Level 2 compliant. Additionally, the deliverable addresses the requirements in D1.4 [1] which must be fulfilled by symbIoTe and each platform to be Level 2 compliant, the benefits and the challenges which emerge within a decentralized domain as well as the proposed solutions.

Apart from the syntactic and semantic interoperability of Level 1, Level 2 delivers components and workflows to facilitate enterprise interoperability by forming dedicated federations for resource sharing and access between compliant IoT platforms.

This document substantiates the design and framework discussed in the previous deliverables D2.4 [2], D2.5 [3] and D3.2 [4] with the needed Level 2 specific implementation and interaction perspective of each involved component. Thus, the described functionality in this document builds upon and uses already implemented features from Level 1 specified in D2.5 [3] and therefore does not repeat these implementation or design details again. However, to reflect the final state of the L2 implementation, the following sequence diagrams and workflows are illustrated which cover the following main features:

1. Manage platforms within federations
2. Manage resources within federations
3. Search and access resources within federations
4. Calculate resource trust and platform reputation
5. Practical bartering within federations
6. Service Level Agreement (SLA) Management and Quality of Service (QoS) enforcement

Based on these sequence diagrams, the required Core and Cloud services are explained in detail. Each component paragraph documents the adequate links to the source code repository and Java documentation which contain the actual implementation of the highlighted features. Additionally, all internal and external interfaces which are provided by the component are listed with the appropriate endpoints, payloads and consuming components.

Finally, the Level 2 specific security principles and mechanisms are discussed to enable a secure environment for platform federations and resource sharing between multiple Level 2 compliant platforms within the symbIoTe ecosystem.

2 Introduction

2.1 Purpose of the Document

This deliverable extends the specification and design of components as described in D3.2 [4] and highlights the final description of the components and their interactions to enable L2 compliance which comprise a well-functioning federation of platforms within the symbIoTe ecosystem. The document will provide a description of every component implementation and the features they provide.

2.2 Structure and Overview

Section 3 provides an overview of the architecture which encompasses the required structure for a platform to be Level 2 compliant thus allowing it to fully benefit of the added value provided by a federation and therefore enables direct resource sharing and access between existing platform applications by using defined Information Models, Service Level Agreements and notification mechanisms.

The next section specifies all the designed features provided by the components at Level 2 as well as their interactions by means of sequence diagrams which provide a detailed representation of the workflow. The main features cover the management of platforms and resources within a federation.

In section 5, the detailed component descriptions as well as all internal and external interfaces are listed. These interfaces provide the necessary data for Level 2 compliant platform within the platform but also between federated platforms or the Core.

The designed and implemented security approach and principles, relevant for L2-compliance, is presented in section 6. The section contains the matured security mechanism deployed within a federation which is responsible for the authentication and authorization of the distinct stakeholders as well as the detection of anomalies which could be the result of a technical malfunction or a bad intentioned actor.

Finally, section 7 concludes the deliverable by providing a summary of the designed architecture and the implementation of the components which fulfil the expectations and meet the requirements of a well-functioning federation of platforms within the symbIoTe ecosystem.

3 Architecture for IoT platform federations

This section will highlight the final symbIoTe architecture which supports IoT platform federations based on the principles and components provided by symbIoTe and detailed in the previous documents D2.4 [2], D2.5 [3] and D3.2 [4].

The focus of the described building blocks and components below lays on the extension of functionality provided by Level 1 implementation to support and deliver the designed features and workflows to gain Level 2 compliance which are described in Section 4.

The presented design and implementation allows L2-compliant platforms to share resources on federation level in a reliable, controlled and secure way without any needed registration on the Core level or exposal of information on the centralized symbIoTe Core instance. Further, advanced features such as enforced Service Level Agreements or trust and reputation metrics ensure more advanced selection and informed decision making of the optimal resource allocation and consumption within each federation. The L2 ecosystem also introduces the resource bartering concept and implementation for costly or limited resources which cannot be offered without any compensation. The implemented architecture itself also provides extensive but still generic REST API interfaces which ensure that platform owners are capable to exploit the provided symbIoTe L2 features by their existing platforms and native applications and therefore facilitate a broader range of available resources they can access in a transparent way.

The high-level interaction and architectural communication flow between Core Services and Cloud Services as well as inter-cloud data exchange within the federation is depicted in Figure 1 and consists of following steps:

- The first step consists in initiating the establishment of a federation with assigned Quality-of-Service QoS constraints which is maintained in the Administration at Core level and distributed to all associated L2 compliant platforms (1).
- Following communication is mainly handled within each platform or between federated platforms establishing a peer-to-peer publish/subscribe approach for receiving relevant resource announcements and updates (2).
- In contrast to L1, the offered resources will not be registered on Core Level but only on platform level to reduce the dependency on Core Services (3).
- Thus, resource updates are directly published to interested federated platforms only, determined by previous subscription request and filtering properties (4).
- Finally, existing apps are capable of searching and looking up local as well as shared foreign resources and consuming them without any further modification. In case the offered data format or information model is not understood by the app, the platforms within a federation may agree on a commonly agreed format or individual data transformations may be implemented (5).
- To gain additional insights into transactions and suspicious behaviour of all involved parties, each platform is calculating and processing internal statistics e.g. anomaly detection events or bartering interactions which will be saved locally and may be reported to the Core instance in an aggregated summary allowing a better impact assessment (6). Further, also validation and violation of defined Quality-of-Service (QoS) constraints agreed within the federation will be added for internal statistics.

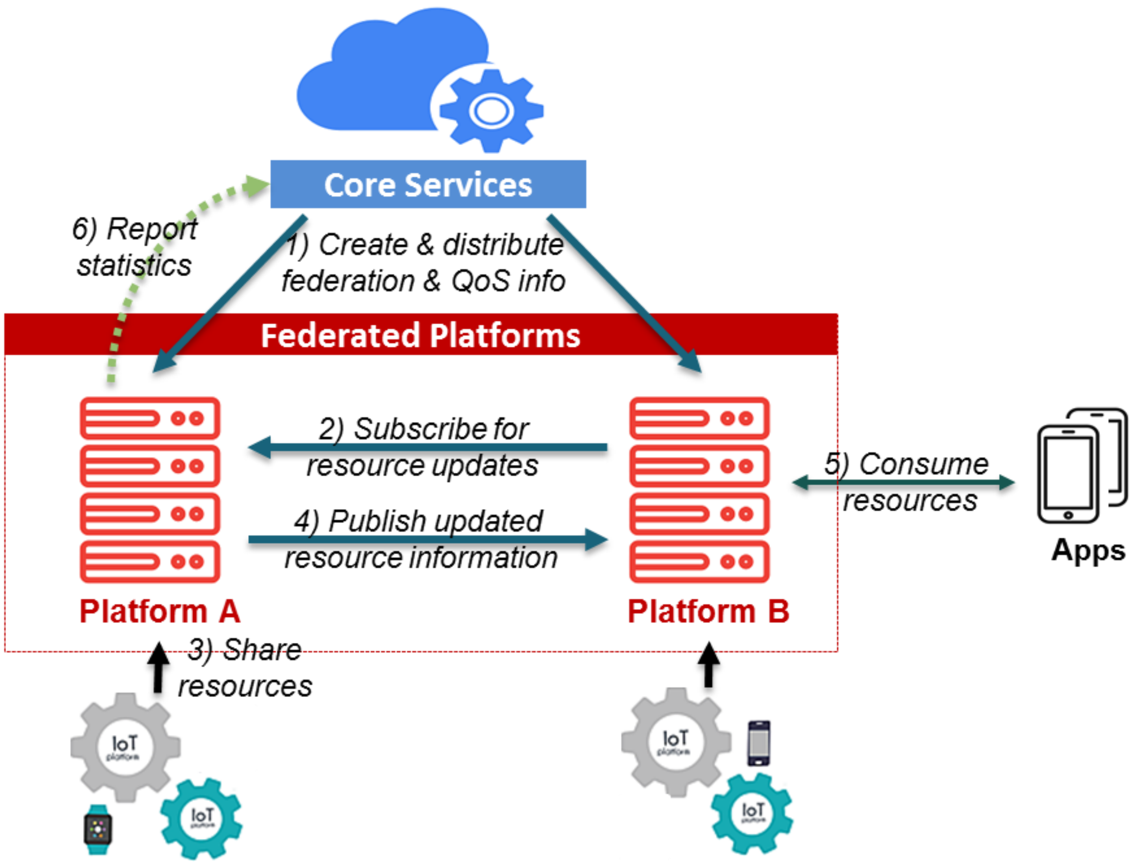

Figure 1. Enable access to shared resources from federated platforms

Delivering the necessary features above, implies the implementation of several enhancements in existing L1-compliant components but also creating new services to support the L2 specific workflows. Figure 2 outlines all symbIoTe components involved and needed for L2-compliance.

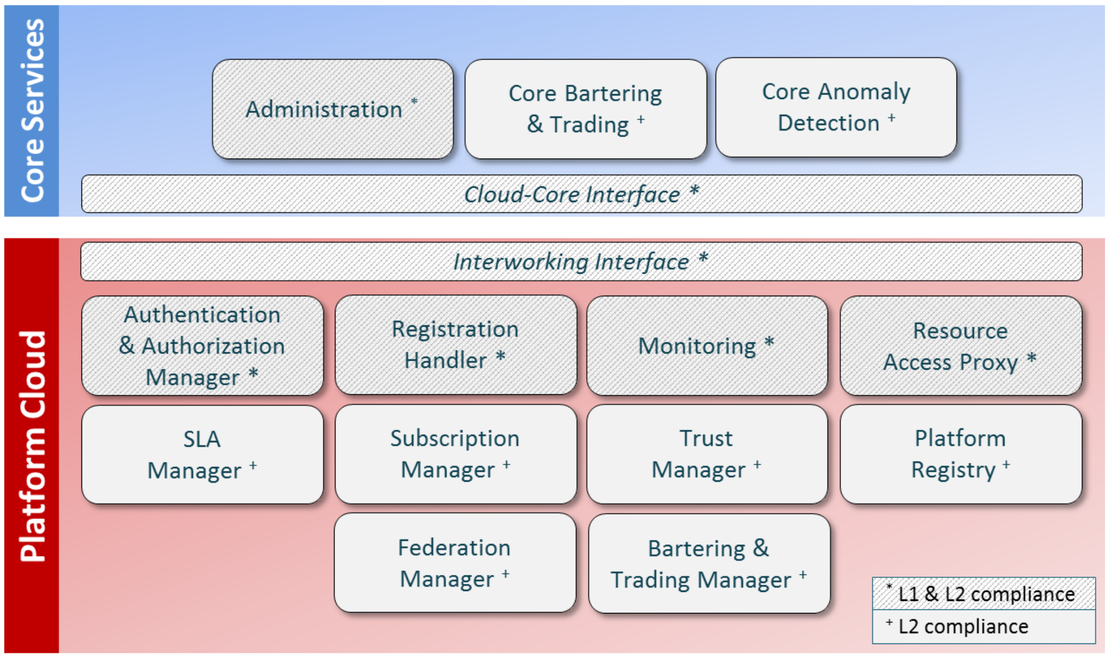

Figure 2. symbIoTe components forming L2 compliance

As shown above, five existing components on Core and Cloud Domain were enhanced to support L2 specific features (L1 & L2 compliance). Also, eight new components were designed and implemented running on the Cloud Domain providing the advanced functionalities ensuring platform interoperability (L2 compliance).

Following components are needed for L2 compliance and will be explained in the later sections:

Core Services:

- **Administration:** Central GUI to administrate federations and QoS constraints;
- **Core Bartering & Trading (B&T):** Stores all issued and consumed coupons and provides according statistics
- **Core Anomaly Detector (CAD):** Detection and storage of all events relevant for anomaly detection

Cloud Services:

- **Authentication and Authorization Manager (AAM):** Supports authentication and authorization within federations
- **Registration Handler (RH):** Provides features for resource registration and deregistration within federations
- **Monitoring:** Gathers metrics from resources managed by the platform and provides statistics of resource consumption and access
- **Resource Access Proxy (RAP):** Enables access to resources shared within the federations
- **SLA Manager (SLAM):** Manages SLA management and validation
- **Subscription Manager (SM):** Forwards and manages resources notifications for own but also shared resources within the federation
- **Trust Manager (TM):** Stores and calculates resource trust and platform reputation values
- **Platform Registry (PR):** Facilitates local resource search and lookup of shared resources within the federations
- **Federation Manager (FM):** Manages all current federation memberships and stored Quality-of-Service (QoS) constraints
- **Bartering & Trading Manager (BTM):** Handles the coupon creation and verification on Cloud domain

Each of the extended and created component above and all considered design and implementation details will be discussed in the following sections.

4 L2 Features and Sequence Diagrams

The following sections describe and highlight implemented L2 features and the interaction between the different components of the Core and Cloud domain to deliver the needed functionality.

The following sequence diagrams are documenting the final version of all designed and implemented features which were introduced and highlighted in D3.2 [4]. Additionally, most workflows were slightly modified to reflect the symbIoTe overall architecture and taken design decisions. Where applicable, the necessary changes and alignments in sequence flows are also discussed in this chapter.

Following coloring scheme is applied for all sequence diagrams below:

- Core Services are coloured in violet
- All Cloud Services are marked orange or green
- To show interactions between two platforms on Cloud Domain, the respective platforms are coloured orange for platform 1 and green for platform 2

4.1 Manage Platforms within Federations

The federation management workflow covers the following sequences and flows:

- Manage federation object properties such as id, name, visibility or assigned information model
- Administrate the list of involved L2-compliant federation members (platforms)
- Assign and maintain Quality of Service (QoS) constraints applicable for all shared resources within the federation context (The QoS enforcement and monitoring is described in section 4.6)

The setup and administration of each federation is done within the Core in the Administration GUI. This ensures a central entry point for platform owners to work with the federation.

Even though the federation administration is done within the Core Domain, all changes applied to the federation properties, associated platforms or assigned QoS constraints are populated to all affected platforms using the defined REST interfaces of the Federation Manager component and therefore the current federation states are also available on the Cloud level.

Thus, the entire federation management sequence diagram may be split in two major parts.

The first interaction will involve the respective Core components for login and federation object management and forwarding of all changes to the Core Anomaly Detection and the affected federated platforms as depicted in Figure 3.

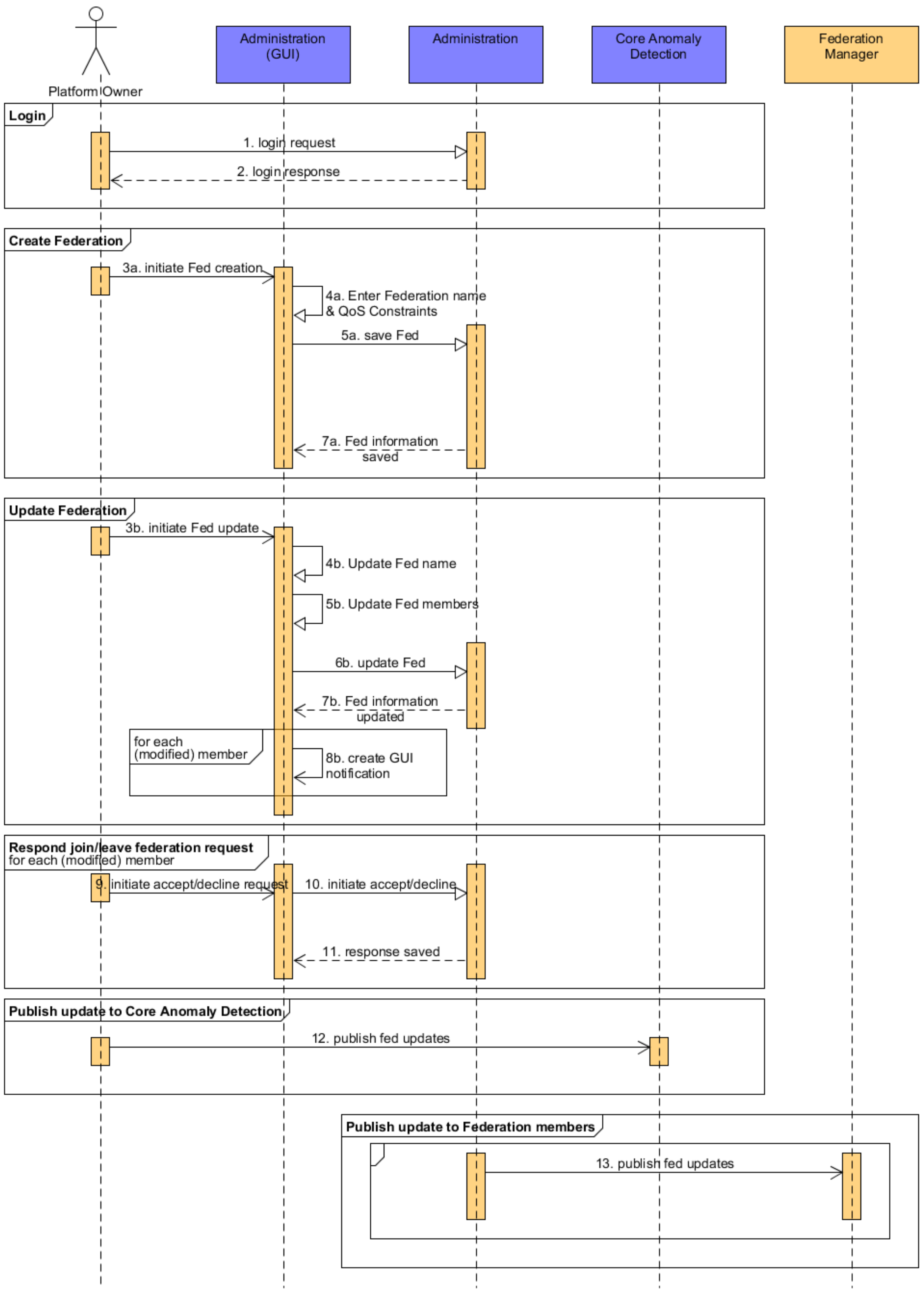
Platform Owner
Administration (GUI)
Administration
Core Anomaly Detection
Federation Manager
Login
1. login request
2. login response
Create Federation
3a. initiate Fed creation
4a. Enter Federation name & QoS Constraints
5a. save Fed
7a. Fed information saved
Update Federation
3b. initiate Fed update
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 update to Core Anomaly Detection
12. publish fed updates
Publish update to Federation members
13. publish fed updates

Figure 3. Federation management workflow on Core Domain

The second step, which is shown in Figure 4, deals with the receipt of the federation updates, validation and verification of the data entries and distribution of the information to other internal symbIoTe components in Cloud Domain, which need such details for further processing and triggering of dependent workflows or actions such as SLA management or federated resource notifications.

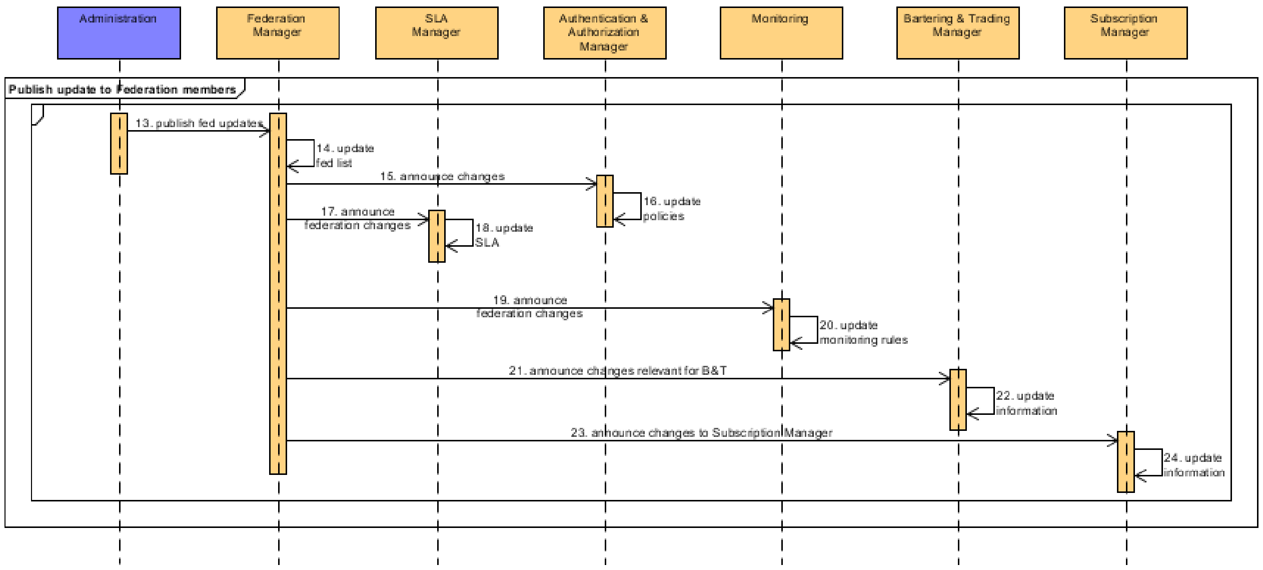
Administration
Federation Manager
SLA Manager
Authentication & Authorization Manager
Monitoring
Bartering & Trading Manager
Subscription Manager
Publish update to Federation members
13. publish fed updates
14. update fed list
15. announce changes
16. update policies
17. announce federation changes
18. update SLA
19. announce federation 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 4. Federation management workflow on Cloud Domain

Additionally, each sequence, message and procedure of the sequence diagram depicted in Figure 3 and Figure 4 are explained below.

Sequence: Platform owner login

- *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.

Sequence: Create federation

- *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.

Sequence: Update federation

- *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 federation members (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 federation join or leave request.

Sequence: Respond to join or leave federation request (for each modified platform)

- *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.
- *Message 11*: Generated by the Administration and sent to the Administration GUI. It is used to acknowledge the previously received feedback.

Sequence: Publish federation updates to Core Anomaly Detection

- Message 12: Generated by the Administration and sent to the Core Anomaly Detection module. It is used to update the federation affiliation of the platforms.

Sequence: Publish federation updates to all affected platform members

- *Message 13*: Generated by the Administration and sent to the Federation Manager. It is used to update the federation affiliation of the platforms at Cloud Domain.
- *Procedure 14*: The Federation Manager updates the federation information with respect to the received data.
- *Message 15*: Generated by the Federation Manager and sent to the Authentication & Authorization Manager. It is used to request the federation policy modification.
- *Procedure 16*: The Authentication & Authorization Manager performs the policy update.
- *Message 17*: 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.
- *Procedure 18*: the SLA Manager will act upon this notification. In case of joining, it will sign a new SLA. In case of leaving, it will remove the previously enforced SLA.
- *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 Manager (BTM) to update the trading rules.
- *Procedure 22*: The BTM 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.

4.2 Manage Resources within Federations

After the federation is established, platforms are able to share and access resources from federated platforms. To ensure a proper management of these resources, following workflow is established:

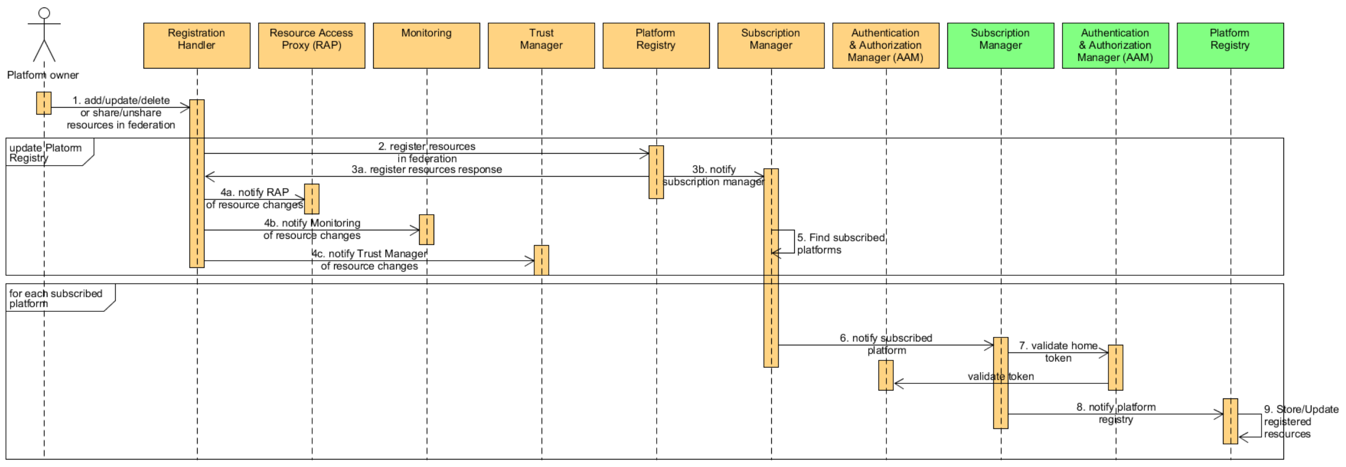
Platform owner
Registration Handler
Resource Access Proxy (RAP)
Monitoring
Trust Manager
Platform Registry
Subscription Manager
Authentication & Authorization Manager (AAM)
Subscription Manager
Authentication & Authorization Manager (AAM)
Platform Registry
1. add/update/delete or share/unshare resources in federation
update Platorm Registry
2. register resources in federation
3a. register resources response
3b. notify subscription manager
4a. notify RAP of resource changes
4b. notify Monitoring of resource changes
4c. notify Trust Manager of resource changes
5. Find subscribed platforms
for each subscribed platform
6. notify subscribed platform
7. validate home token
validate token
8. notify platform registry
9. Store/Update registered resources

Figure 5. Manage resources within federations

Sequence: Update Platform Registry

- *Message 1*: Platform owner sends a request to Registration Handler to add new resources to a federation, or to update or delete the existing ones.
- *Message 2*: Generated by the home platform Registration Handler and sent to the home 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 3a*: Generated by home Platform Registry and sent to home platform Registration Handler to notify it about the completion of the action.
- *Message 3b*: Generated by home Platform Registry and sent to home platform Subscription Manager which will notify other platforms within the federation of the new, updated or removed resources.
- *Messages 4a, 4b, 4c*: Generated by home platform Registration Handler and sent to Resource Access Proxy, Monitoring and Trust Manager about the resource event.

- *Procedure 5*: Find platforms interested in the updated resources.

Sequence: Notify Participating Platforms

- *Message 6*: Generated by the home platform Subscription Manager and sent to the federated platform Subscription Manager. The message contains information about the new, updated or removed resources.
- *Message 7*: Generated by federated platform Subscription Manager and sent to federated platform AAM to request the token validation performed between foreign and home AAM.
- *Message 8*: Generated by federated platform Subscription Manager and sent to the Platform Registry.
- *Procedure 9*: Platform Registry stores metadata of new or updated resources, or removes the metadata of deleted resources.

4.3 Search and Access Resources within Federations

All resource updates are communicated to subscribed platforms within the federation. Thus, each native application is able to perform search and lookup of relevant resources with its Platform registry component. To facilitate the advance security mechanisms symbIoTe provides, the native application could use the symbIoTe combability layer to interact with the newly implemented components. There, the Client Wrapper and Security Handler (SH) ensure the valid authentication and authorization using the native interfaces and security concepts.

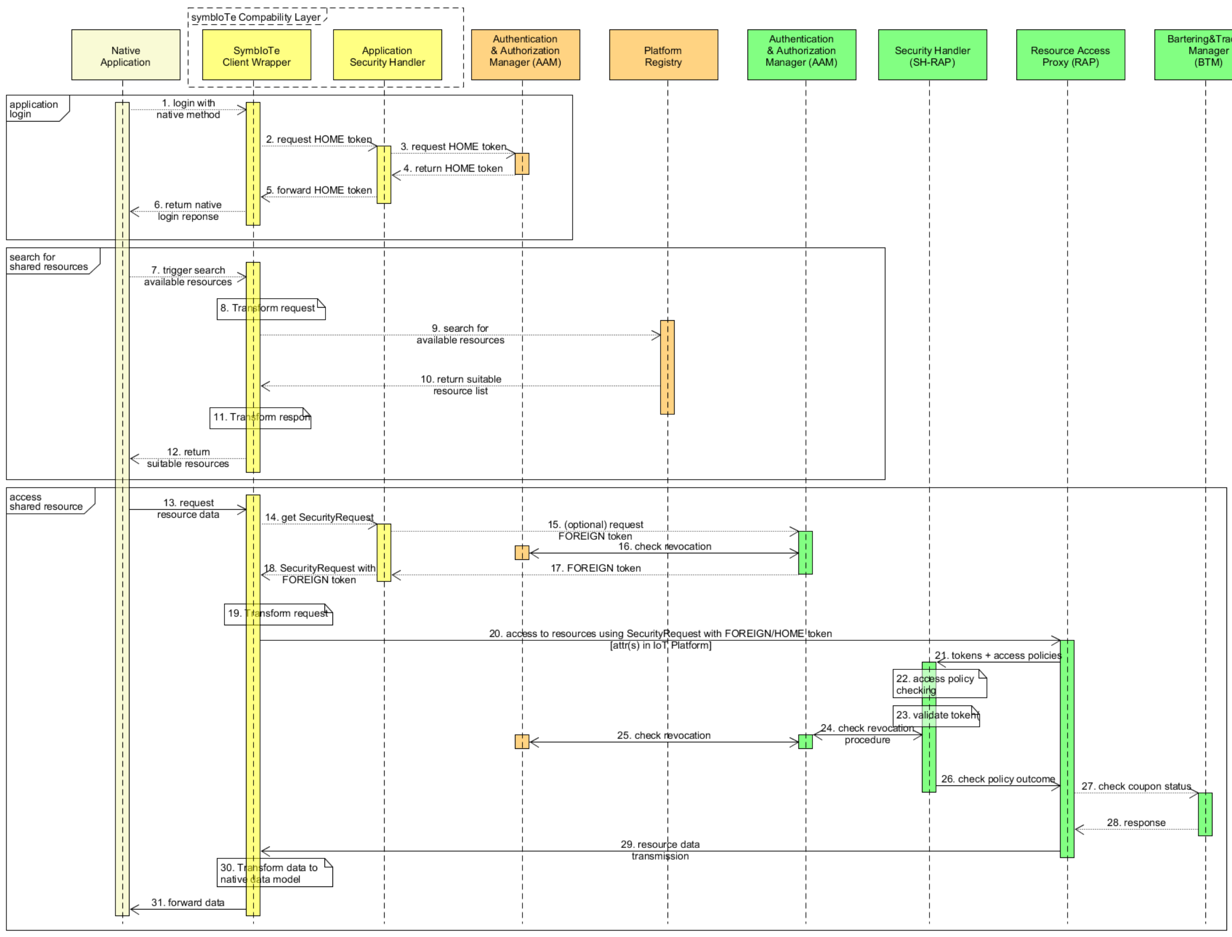
symbIoTe Compability Layer
Native Application
SymbIoTe Client Wrapper
Application Security Handler
Authentication & Authorization Manager (AAM)
Platform Registry
Authentication & Authorization Manager (AAM)
Security Handler (SH-RAP)
Resource Access Proxy (RAP)
Bartering&Trading Manager (BTM)
application 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. check coupon status
28. response
29. resource data transmission
30. Transform data to native data model
31. forward data

Figure 6. Search and access resources within federation

Additionally, each message and procedure interaction in the sequence diagram above will be detailed below:

- *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):* the symbIoTe Client Wrapper calls the Application Security Handler 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 Authentication and Authorization Manager (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 Resource Access Proxy (RAP).
- *Message 21:* checking tokens and access policies, forwarded by RAP to RAP Security Handler
- *Procedure 22:* Check against access policy with the Security Handler functionality integrated in the RAP (SH-RAP)
- *Procedure 23:* SH-RAP validates the token

- *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 (optional):* for bartered resources RAP checks at BTM if the application has rights to access resources
- *Message 28 (optional):* response from B&T Manager
- *Message 29:* requested data is found by foreign RAP and forwarded to symbIoTe Client Wrapper
- *Procedure 30:* symbIoTe Client Wrapper may transform the data to the model understandable by the application if needed
- *Message 31:* symbIoTe Client Wrapper forwards data to the native application

4.3.1 Collect and report failed access attempts

To enable detection and reporting of any anomalies during resource access, failed requests for accessing resources are logged and reported to the Core Anomaly Detection (CAD) component for further aggregation and validation as shown in Figure 7 below.

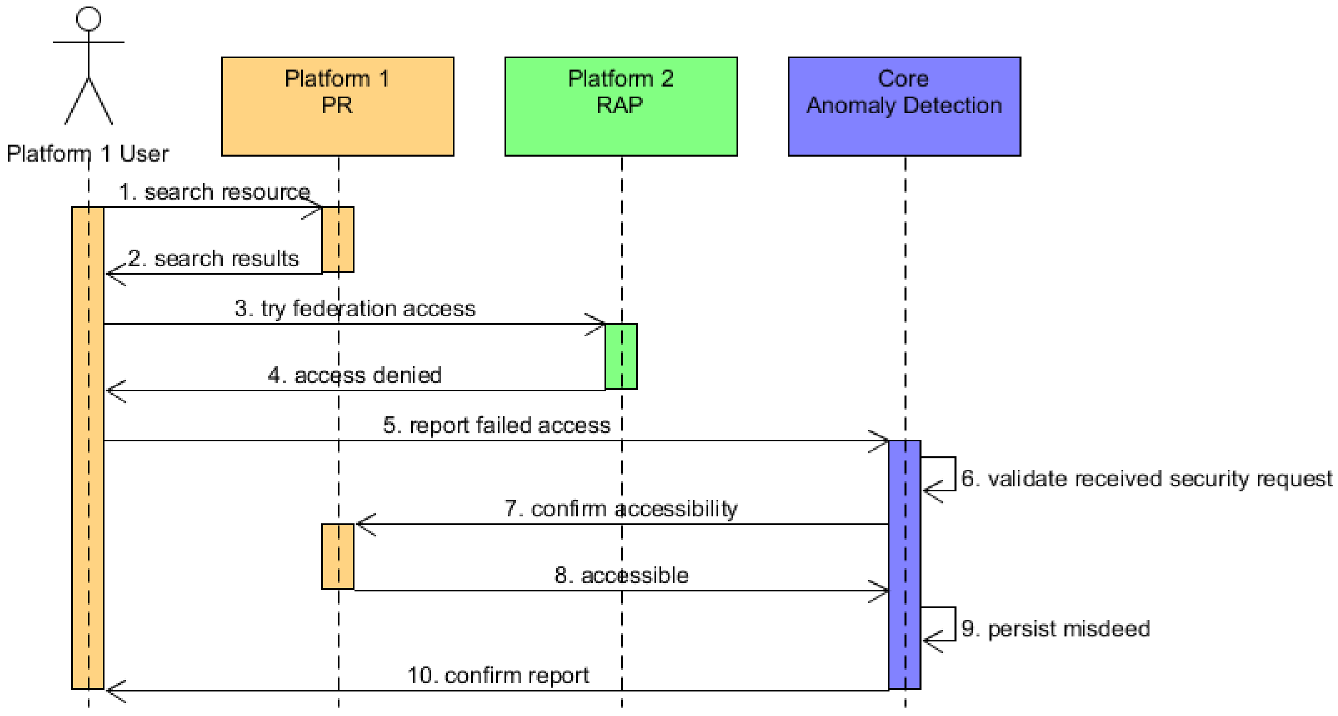

Figure 7. Failed federated access report

Sequence: Request access for resource

- *Message 1:* User search for desired federated resource in Platform Registry.
- *Message 2:* Platform Registry returns requested search results.
- *Message 3:* User tries wants to access to the resource from platform 2.

- *Message 4:* RAP in platform 2 denies access the valid request.

Sequence: Report failed federated access to Core Anomaly Detection

- *Message 5:* User reports failed access of this federated resource by sending used security request, federation Id and information about platforms and resource.
- *Procedure 6:* Core Anomaly Detection validates if the received security request should have access to the federated resource.
- *Message 7:* Core Anomaly Detection confirms that the resource should be accessible.
- *Message 8:* RAP in platform 1 sends confirmation.
- *Procedure 9:* Core Anomaly Detection persists the misdeed.
- *Message 10:* Core Anomaly Detection returns status to the user.

4.4 Calculate Resource Trust and Platform Reputation

The resource trust and platform reputation calculation workflow is structured in three main sequences which build on one another.

- The **resource trust** is calculated for each individual resource offered within the federation. That value is added to the resource metadata and will be shared with other federated platforms using the sequence described in Section 4.2.
- The (foreign) **platform reputation** will be calculated on a periodic basis. Only platforms which are currently in the same federation will be considered. This value is only used internally (e.g. by B&T Manager) and will not be shared with other platforms or external services.
- Based on the shared (foreign) resource trust and (foreign) platform reputation values, an internal trust metric – **adaptive resource trust** – can be calculated. This value represents the internal, subjective trust in resources offered by foreign platforms within the federation. As the adaptive trust value also reflects individual platform specific factors and historic events (e.g. past B&T interactions) it will define the overall trust in that resource more accurately.

The sequence diagram in Figure 8 will illustrate the individual procedures to calculate these three values. Also, all required interactions with other components which provide the necessary data will be shown below.

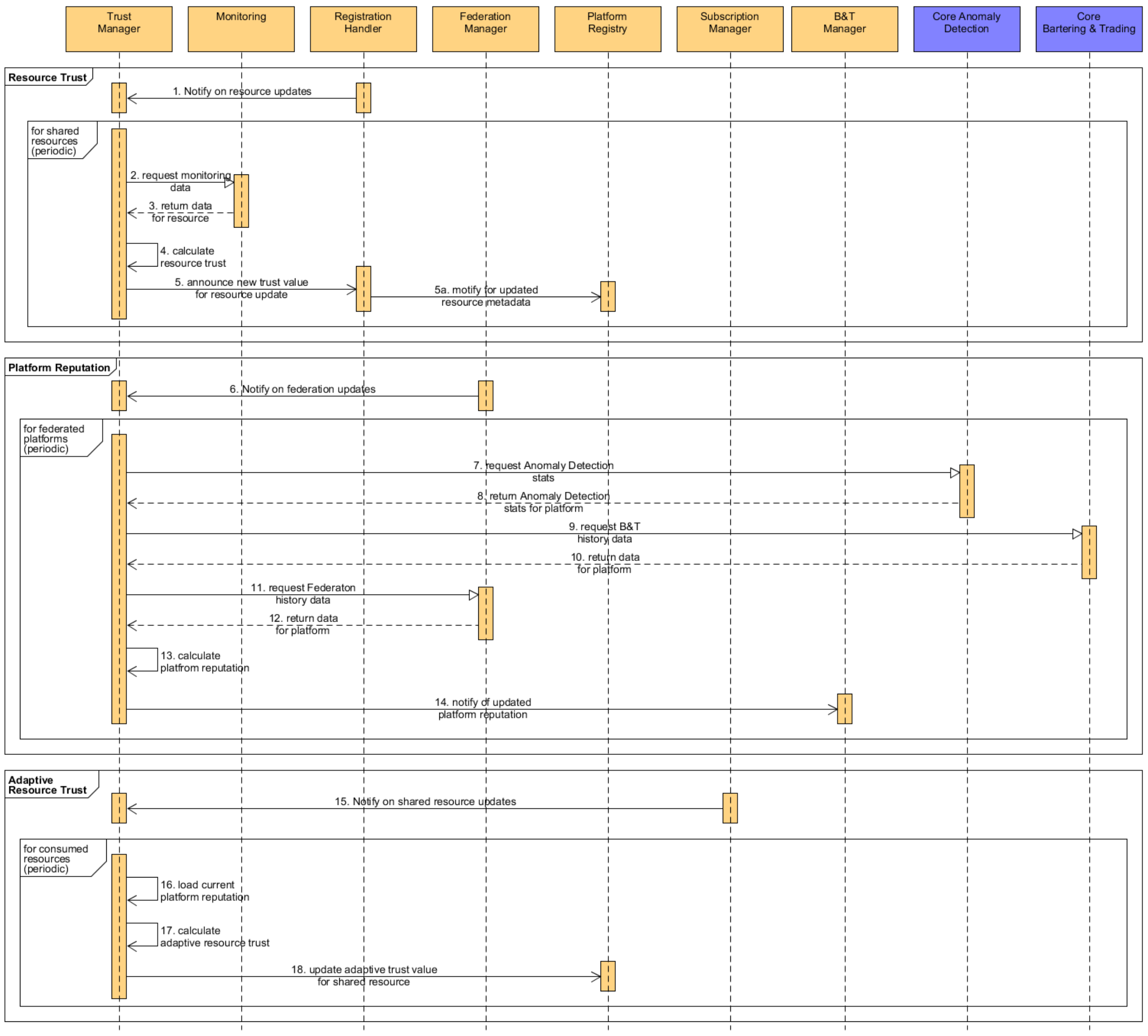
Trust Manager
Monitoring
Registration Handler
Federation Manager
Platform Registry
Subscription Manager
B&T Manager
Core Anomaly Detection
Core Bartering & Trading
Resource Trust
1. Notify on resource updates
for shared resources (periodic)
2. request monitoring data
3. return data for resource
4. calculate resource trust
5. announce new trust value for resource update
5a. motify for updated resource metadata
Platform Reputation
6. Notify on federation updates
for federated platforms (periodic)
7. request Anomaly Detection stats
8. return Anomaly Detection stats for platform
9. request B&T history data
10. return data for platform
11. request Federaton history data
12. return data for platform
13. calculate platfrom reputation
14. notify of updated platform reputation
Adaptive Resource Trust
15. Notify on shared resource updates
for consumed resources (periodic)
16. load current platform reputation
17. calculate adaptive resource trust
18. update adaptive trust value for shared resource

Figure 8. Resource trust and platform reputation calculation

Each interaction and procedure listed in the sequence diagram above, will be listed and discussed in the following paragraph.

Sequence: Resource Trust

- *Message 1:* Notification of own shared resource updates (added, deleted) is send from Registration Handler to Trust Manager.
- *Message 2:* Trust Manager requests periodically monitoring data for all shared resources.
- *Message 3:* The Monitoring component returns the appropriate availability data to the Trust Manager.
- *Procedure 4:* The Trust Manager calculates and updates the resource trust values.

- *Message 5:* The Trust Manager notifies the Registration Handler of new trust value for the own shared resource.
- *Message 5a:* The Registration Handler notifies Platform Registry of updated resource metadata.

Sequence: Platform Reputation

- *Message 6:* Notification of federation updates is send from Federation Manager to Trust Manager.
- *Message 7:* Trust Manager requests periodically statistics data for federated platforms from Anomaly Detection.
- *Message 8:* Anomaly Detection returns aggregated event statistics (e.g. granted / rejected access to platform) to the Trust Manager.
- *Message 9:* Trust Manager requests bartering history data for federated platforms from Core Bartering & Trading component.
- *Message 10:* B&T component returns aggregated history (e.g. issued / consumed coupons) to the Trust Manager.
- *Message 11:* Trust Manager requests federation history data for federated platforms from Federation Manager.
- *Message 12:* Federation Manager returns event history (e.g. time platform joined / left) to the Trust Manager.
- *Procedure 13:* The Trust Manager calculates and updates the platform reputation values.
- *Message 14:* The Trust Manager notifies the B&T Manager of an updated platform reputation value.

Sequence: Adaptive Resource Trust

- *Message 15:* Notification of foreign shared resources (from other federated platforms) is send from Subscription Manager to Trust Manager.
- *Procedure 16:* The actual platform reputation value of the federated platform is loaded
- *Procedure 17:* The Trust Manager calculates the adaptive resource trust using the foreign resource trust and its own platform reputation value
- *Message 18:* The Trust Manager notifies the Platform Registry of new adaptive resource trust value for the foreign shared resource.

4.5 Practical Bartering within Federations

symbIoTe provides the possibility for federations to share resources under bartering rules. It uses the concept of coupons (described in section 6.2) to grant the holders access to other platforms' resources.

By keeping track of the generated and used coupons, it is possible to understand which platforms are contributing to the federation, and allow or deny access according to this. The bartering system was initially designed based on a questionnaire answered by the project partners and documented in D3.1 [7]. This questionnaire contained items that related to concepts of interest to the project (e.g. cooperation), with the goal of obtaining the view of each members on how the bartering should work.

Figure 9 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), in order 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 just consuming resources and not actively contributing to the federation (i.e. issued coupons to access foreign resources are never consumed by participating platforms) which may impact their platform reputation rating and could further lead to rejected resource access or bartering requests.

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. Each platform owner is able to define and adjust these rules for on demand for their platforms individually.

It is also important to note that the function of the Core B&T component keeps track of all activities regarding coupons (creation, consumption) and therefore enables overall interaction statistics for all L2 compliant platforms. This information also allows better platform trust and reputation calculation as well as further insights of the bartering process.

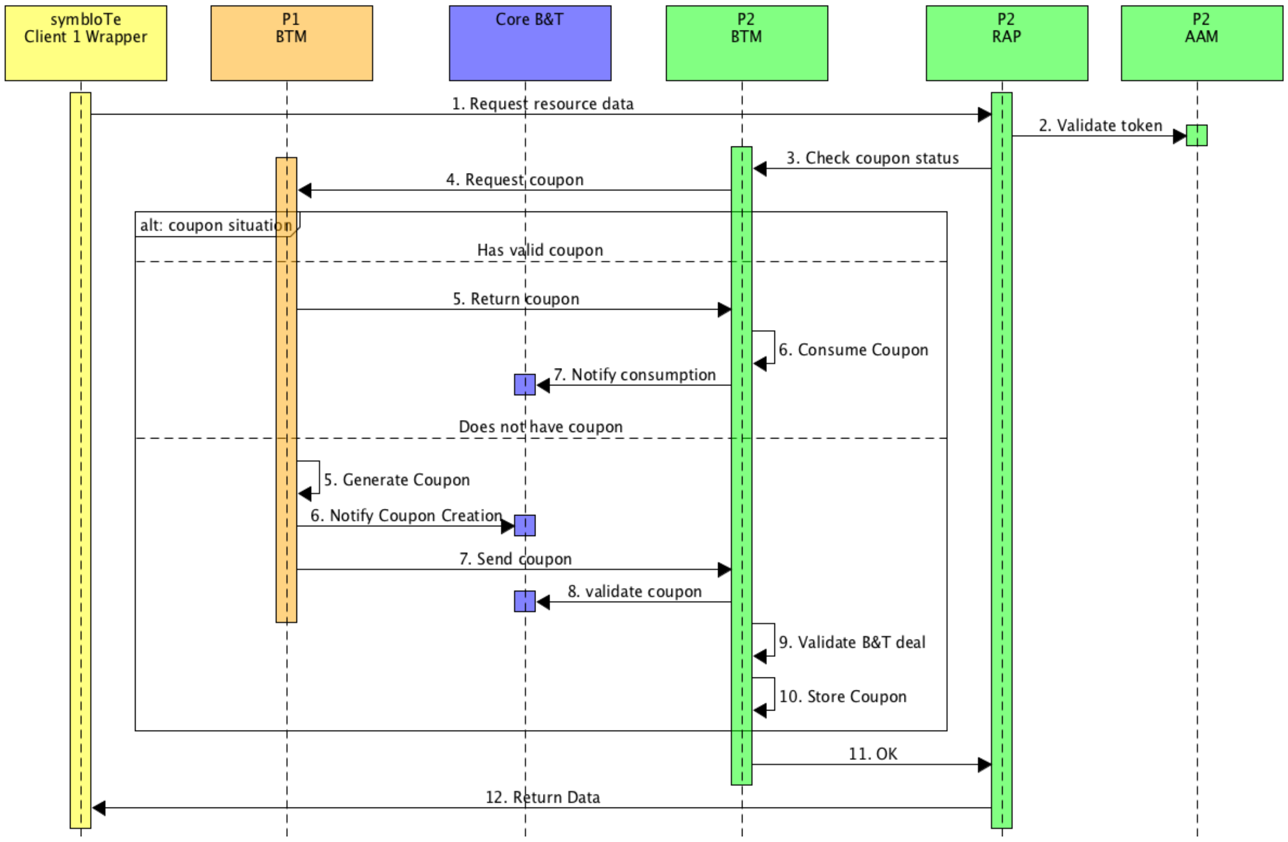

Figure 9. Practical Bartering workflow in symbIoTe

Additionally, each message and procedure interaction in the sequence diagram above will be detailed below:

- *Message 1*: Generated by the symbIoTe Client from P1 and sent to the P2's RAP. It is a request to access a resource in P2.
- *Message 2*: Generated by P2's RAP and sent to P2's AAM. It is used to validate the received token.
- *Message 3*: Generated by P2's RAP and sent to P2'd BTM. It is used to check the status of the bartering between P1 and P2.
- *Message 4*: Generated by P2's BTM and sent to P1's BTM. It is used to request a valid coupon.

Sequence: P1 has a valid token

- *Message 5*: Generated by P1's BTM and sent to P2's BTM. A valid coupon is returned.
- *Procedure 6*: P2's BTM consumes P1's coupon
- *Message 7*: Generated by P2's BTM and sent to Core B&T. It is used to notify of the coupon consumption.

Sequence: P1 does not have a valid token

- *Procedure 5*: P1's BTM generated a coupon to access resources in P1.
- *Message 6*: Generated by P1's BTM and sent to Core B&T. The core is notified of a coupon creation.
- *Message 7*: Generated by P1's BTM and sent to P2's BTM. The generated coupon is sent to P2.
- *Message 8*: Generated by P2's BTM and sent to Core B&T. It is used to validate the received coupon.
- *Procedure 9*: P2's BTM validates if the bartering deal is to its liking.
- *Procedure 10*: P2's BTM stores the received coupon to be used later.
- *Message 11*: Generated by P2's BTM and sent to P2's RAP. Notifying that the bartering was successful.
- *Message 12: Generated by P2's RAP and sent to the symbIoTe Client. The requested data is returned.*

4.6 SLA Management and QoS Enforcement

Federations might define some Quality of Service constraints that all shared resources must comply with. This determines the creation of different types of federations and rules where, for example, a federation might impose that only highly available resources should be shared with it. To enforce these requirements, each time that a platform joins a federation it signs a Service Level Agreement with the federation itself, where is declared that only those resources that accomplish the QoS constraints will be shared. This SLAs are managed by symbIoTe's SLA Manager component with the help of Monitoring and Federation Manager. A complete description of how the architecture and design of the SLA Manager looks like can be found in D3.2 [4] as well as its interaction with other components.

As described in Figure 10, SLAs are created and updated based on interactions with the Federation Manager. The following sequence diagrams describes the three main functionalities in detail.

Sequence: SLA Creation/Update

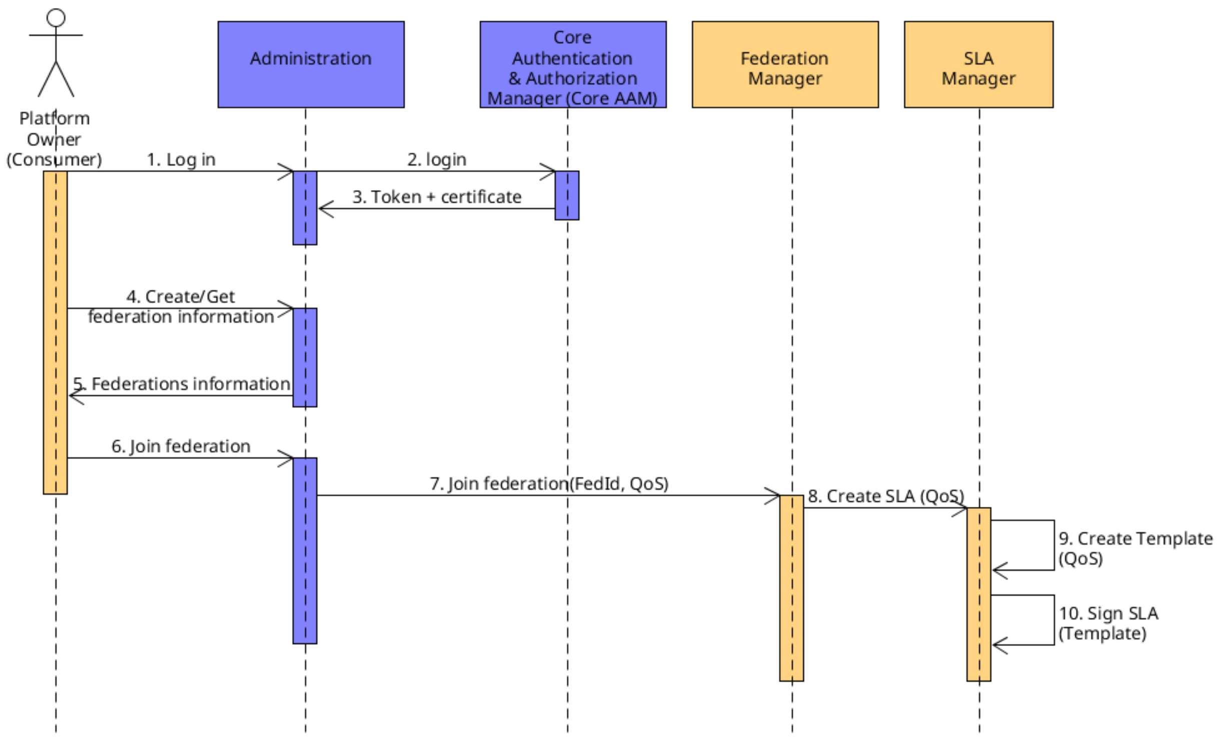
Platform Owner (Consumer)
Administration
Core Authentication & Authorization Manager (Core AAM)
Federation Manager
SLA Manager
1. Log in
2. login
3. Token + certificate
4. Create/Get federation information
5. Federations information
6. Join federation
7. Join federation(FedId, QoS)
8. Create SLA (QoS)
9. Create Template (QoS)
10. Sign SLA (Template)

Figure 10. SLA creation and update

Additionally, each message and procedure interaction in the sequence diagram above will be detailed below:

- *Message 1*: Platform owner logs in to the system through the Administration interface.
- *Message 2*: Administration component logs in to the Core AAM
- *Message 3*: Core AAM sends the token + certificate to the Administration component
- *Message 4:* Once logged in, Platform owner creates or gets information about a federation
- *Message 5:* Administration returns such information
- *Message 6:* Platform owner decides to join a federation and informs the Administration component
- *Message 7:* Administration sends a message to the Platform's Federation manager with the federation information that it just joined. It contains the QoS constraints that it must comply with.
- *Message 8:* Federation Manager passes these QoS constraints to the SLA Manager in order to create a new SLA for the federation.
- *Message 9:* SLA Manager creates a new template for the QoS constraints.

- *Message 10:* SLA Manager signs a new SLA with the previous template and from that moment on, it will check that it's upheld by all shared resources.

Sequence: SLA validation and violation notification

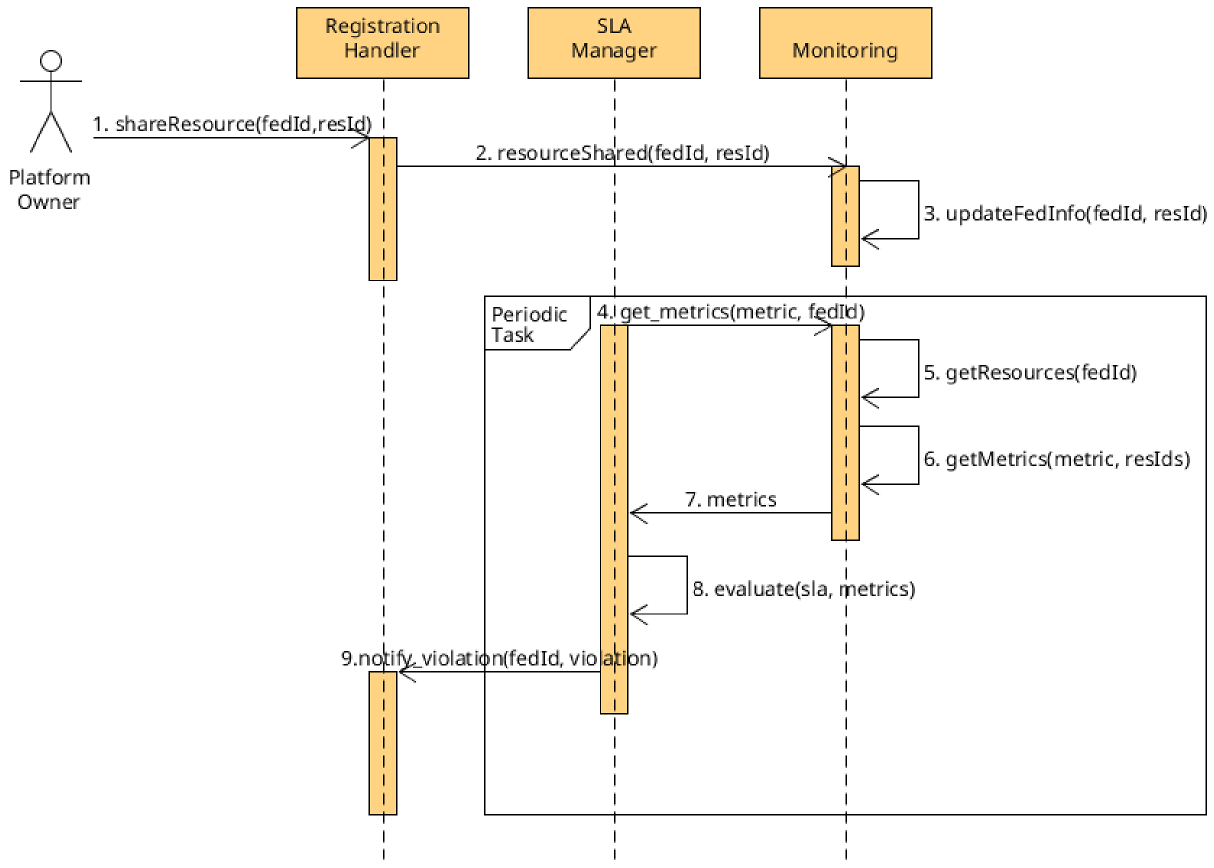
Registration Handler
SLA Manager
Monitoring
Platform Owner
1. shareResource(fedId,resId)
2. resourceShared(fedId, resId)
3. updateFedInfo(fedId, resId)
Periodic Task
4. get_metrics(metric, fedId)
5. getResources(fedId)
6. getMetrics(metric, resIds)
7. metrics
8. evaluate(sla, metrics)
9.notify_violation(fedId, violation)

Figure 11. SLA validation workflow

Additionally, each message and procedure interaction in the sequence diagram above will be detailed below:

- *Message 1:* Platform Owner shares a resource with a federation through the Registration Handler.
- *Message 2:* Registration Handler informs Monitoring that a new resource has been shared in a federation, passing the resource and federation identifiers.
- *Message 3:* Monitoring updates the list of resources shared with that federation.
- *Message 4:* Periodically, SLA Manager will request Monitoring metrics values needed to evaluate the QoS constraints of a federation.
- *Message 5:* Monitoring will get a list of resources associated with the federation identifier.
- *Message 6:* Monitoring will get the historical metric values gathered for those resources.

- *Message 7:* Monitoring will return the requested metric values to SLA Manager
- *Message 8:* With this metrics data, SLA Manager will evaluate the SLA associated with the federation.
- *Message 9:* If a QoS rule evaluation fails to comply, SLA Manager will inform Federation Manager with the federation Id and information about the violation such as which rule was violated and which value broke it.

Sequence: SLA deletion

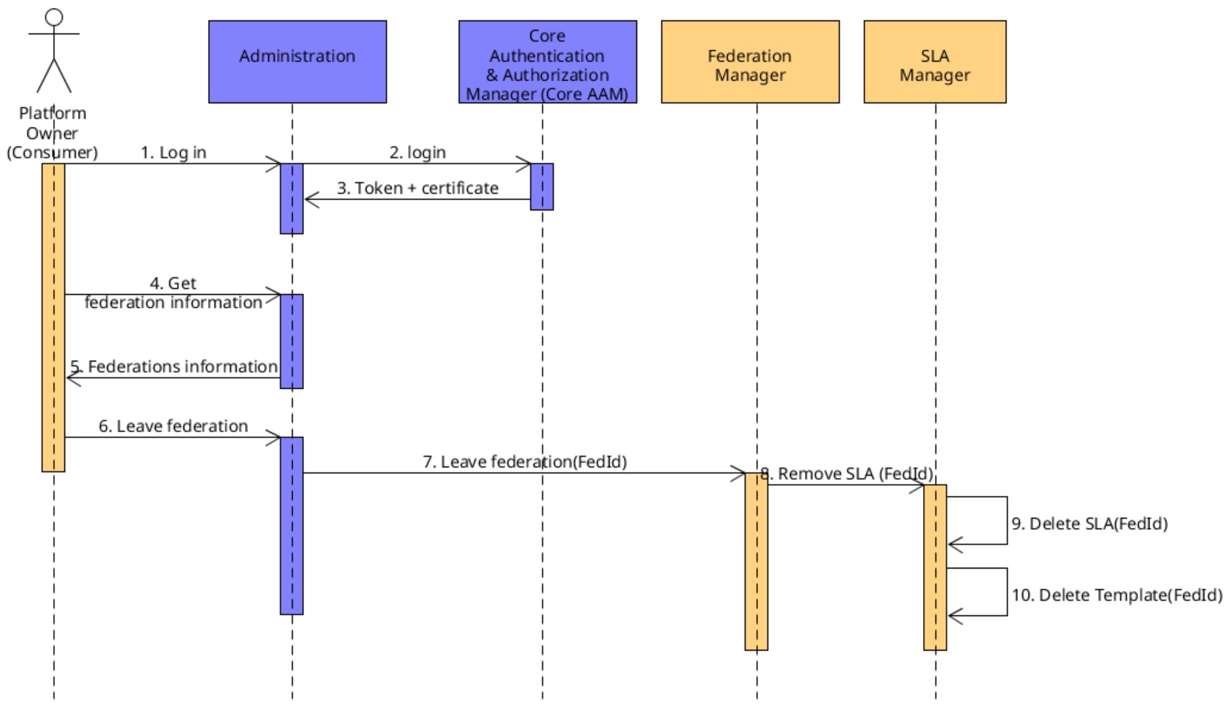
Platform Owner (Consumer)
Administration
Core Authentication & Authorization Manager (Core AAM)
Federation Manager
SLA Manager
1. Log in
2. login
3. Token + certificate
4. Get federation information
5. Federations information
6. Leave federation
7. Leave federation(FedId)
8. Remove SLA (FedId)
9. Delete SLA(FedId)
10. Delete Template(FedId)

Figure 12. SLA deletion sequence

Additionally, each message and procedure interaction in the sequence diagram above will be detailed below:

- *Message 1:* Platform owner logs in to the system through the Administration interface.
- *Message 2:* Administration component logs in to the Core AAM
- *Message 3:* Core AAM sends the token + certificate to the Administration
- *Message 4:* Once logged in, Platform owner gets information about a federation
- *Message 5:* Administration returns such information
- *Message 6:* Platform owner decides to leave a federation and informs the Administration component
- *Message 7:* Administration sends a message to the Platform's Federation manager with the federation identifier to leave.

- *Message 8:* Federation Manager asks SLA Manager to remove the SLA associated to such federation identifier
- *Message 9:* SLA Manager deletes the SLA associated to the input federation identifier.
- *Message 10:* SLA Manager deletes the template used to create such SLA.

5 Final design of components for Level 2 compliance

5.1 Core Services

5.1.1 Component: Administration

Component/service name	**Administration**
URL of source codes	https://github.com/symbiote-h2020/Administration
URL of Javadoc documentation	https://symbiote-h2020.github.io/Administration/doxygen
List of features included	• Federation registration • Leave federation • Delete federation

5.1.1.1 Administration component description

This component facilitates the control and administration of the symbIoTe Core Services by providing a web-based GUI. symbIoTe administrators have access to a control panel that allows them to perform management actions and update parameters related to the symbIoTe Core Services, such as removing specific platforms from the registry or changing the values of search engine variables to improve ranking. Furthermore, administrators will have access through the web-based interface to internal information, e.g. logs, IoT service popularity data, platform usage/status data or unauthorized user access attempts.

This component will also provide features to non-administrator users. It will enable IoT platforms and applications to register with symbIoTe and to receive credentials that are required for the subsequent usage of symbIoTe services. Particularly for L2, platform owners will be able to create, join and leave platform federations.

The current Administration release provides the following L2-specific features:

- federation management, thus providing a user-friendly GUI for creating, leaving and deleting federations with notification support for participating platforms about federation relevant events.
- Management and distribution of assigned QoS constraints and used information model
- Notification of Core Anomaly Detection component concerning federation updates

5.1.1.2 Administration component interface

Administration is the only core service layer component that exposes a GUI element for managing platforms, SSPs and federations. In the L2 concept, Administration stores the federation-related information in its own database, and notify all the interested core components about federation events e.g. federation creation, update or deletion. These interfaces are described in the Table 1 below. Furthermore, on such events, Administration also contacts the Federation Managers of the participating platforms via the Federation Manager API described in section 5.2.9.

#	Interface	Name	From	Consumer	Address/Queue	Payload[1]	Response	Description
1	RabbitMQ TOPIC	Federation (affiliation) added	Administration	CAD	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.created	Federation	-	Notify internal consumers about a federation creation event.
2	RabbitMQ TOPIC	Federation (affiliation) updated	Administration	CAD	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.changed	Federation	-	Notify internal consumers about a federation update event.
3	RabbitMQ TOPIC	Federation (affiliation) removed	Administration	CAD	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.deleted	String federationID	-	Notify internal consumers about a federation deletion event.

Table 1. Administration component interfaces

5.1.2 Component: Core Bartering & Trading

Component/service name	**Core Bartering & Trading**
URL of source codes	https://github.com/symbiote-h2020/BarteringAndTrading
URL of Javadoc documentation	https://symbiote-h2020.github.io/BarteringAndTrading/doxygen
List of features included	• Bartering management ○ Store bartering activity ○ Validate created coupons ○ Provide information to Trust Manager

5.1.2.1 Core Bartering & Trading component description

The Core Bartering & Trading (B&T) component acts a centralized manager of the bartering within symbIoTe. What this implies is that all activity related to coupon creation, exchange and usage is communicated to the Core B&T. This allows it to have a general view of what is happening within the various federations, allowing the understanding which platforms contribute to a given federation. This information can be provided to the Trust Manager as another way of evaluating the trust of a platform. Furthermore, it allows platforms to validate the coupons obtained on an exchange.

5.1.2.1 Core Bartering & Trading component interface

The interfaces listed in Table 2, document the available REST endpoints exposed by the Core B&T component for managing coupons and accessing usage statistics. Each endpoint is secured using symbIoTe security request headers and verified applying service response headers for mutual authentication.

#	Interface	Name	From	Consumer	Address/Queue	Payload	Response	Description
1	REST	isCouponValid	BTM	Core B&T	POST /is_coupon_valid	Header *x-auth-coupon* containing coupon string	CouponValidity	Request to the Core B&T checking if provided coupon is valid.
2	REST	registerCoupon	BTM	Core B&T	POST /register_coupon	Header *x-auth-coupon* containing coupon string	HTTP Status 200, 400, 401	Request to the Core B&T to register new Coupon and make it possible to validate it.
3	REST	consumeCoupon	BTM	Core B&T	POST /consume_coupon	Header *x-auth-coupon* containing coupon string	HTTP Status 200, 400, 401	Request to the Core B&T to notify coupon consumption
4	REST	cleanupConsumedCoupons	BTM	Core B&T	POST /cleanup_coupons	long timestamp	HTTP Status	Request to the Core B&T to cleanup DB from all consumed coupons before provided timestamp.
5	REST	listCouponUsage	TM	Core B&T	/couponusage	FilterRequest	FilterResponse	Request to the Core B&T the usage of coupons of certain platforms or federations to support the Trust Calculation

Table 2. Core Bartering & Trading component interfaces

5.1.3 Component: Core Anomaly Detection

Component/service name	**Core Anomaly Detection Module**
URL of source codes	https://github.com/symbiote-h2020/AnomalyDetectionModule
URL of Javadoc documentation	https://symbiote-h2020.github.io/AnomalyDetectionModule/doxygen
List of features included	• Accept failed federated access authorization reports • Provide aggregated reports about federated platforms' misdeeds ○ Grouped by search origin platform ○ Grouped by federation id

5.1.3.1 Core Anomaly Detection component description

The Core Anomaly Detection component (CAD) acts a centralized recipient and manager of all abnormalities detected in the system.

Any failed authorization to the federated resources that should be accessible thanks to a cross platform federation agreement has to be reported by the actors who experience such a platform misdeed.

When they issue such a report (including the federated resource id and used SecurityRequest) to the CAD, it checks if the report passes the standardized Federation Access Policy relevant to that resource. Resolving the policy is possible as the CAD module listens for federation notifications from the Administration module and gathers information about all the federations in the system. Hence it knows how the policy should be built on the RAP that offers that resource.

If the client's SecurityRequest was indeed valid and should've granted access to given federated resource, CAD reaches out to the actor's Platform Registry module to check, if the requested resource was truly registered in it (by subscription rules) for federated accessibility.

If both checks pass, the report is persisted in the local misdeeds database. Entries there allow to build an overview of the interactions between the platforms in the federation. This information is then provided to the platforms Trust Manager (according to diagram Figure 8) as a way of evaluating the trust of a platform in the system.

5.1.3.2 Core Anomaly Detection component interface

The Table 3 below lists all exposed and consumed interfaces used for inter-component communication. The REST services are secured with security request and response headers to ensure mutual authentication of the interacting services.

#	Interface	Name	From	Consumer	Address/Queue	Payload	Response	Description
1	REST	handleFailFederationAuthorizationReport	End-user (application) clients	Core ADM	POST /log_failed_federation_authorization	FailedFederationAuthorizationReport	HttpStatus	Report to the Core ADM about failed federated access authorization, that should be possible according to user's knowledge (based on PlatformRegistry search).
2	REST	getMisdeedsGroupedByPlatform	Trust Manager	Core ADM	GET /federated_misdeeds /bySearchOriginPlatform	platformId searchOriginPlatformId	Map<String, OriginPlatformGroupedPlatformMisdeedsReport>	Request to the Core ADM to get map of federated platforms containing number of misdeeds grouped by search origin platforms.
3	REST	getMisdeedsGroupedByFederations	Trust Manager	Core ADM	GET /federated_misdeeds /byFederation	platformId federationId	Map<String, FederationGroupedPlatformMisdeedsReport>	Request to the Core ADM to get map of federated platforms containing number of misdeeds grouped by federation ids.
4	RabbitMQ TOPIC	Federation created	Administration	Core ADM	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.created	Federation	-	Creates a new federation in DB, used to validate federation membership.
5	RabbitMQ TOPIC	Federation updated	Administration	Core ADM	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.changed	Federation	-	Updates the federation in DB, used to validate federation membership.
6	RabbitMQ TOPIC	Federation deleted	Administration	Core ADM	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.deleted	Federation	-	Deletes the federation in DB, used to validate federation membership.

Table 3. Core Anomaly Detection component interfaces

5.1.3.3 Malicious client's detection

The current functionality provided by the Core Anomaly Detection component could be extended in the future to allow processing information about suspicious authentication and authorization operations (e.g. clients certificate issuing, login requests and authorization attempts):

- from AAMs;
- from components' security handlers;

which may identify possible threats (e.g. multiple login attempts, databases crawling or denial-of-service attacks on the core and platforms components). However, proper testing of detecting those attacks is not yet achievable due to the lack of true production server with plethora of interactions and coding just to simulated streams can lead to overfitting of the proposed classifiers.

A dedicated library that depends on the Authentication Authorization Manager component was developed. Anomaly Listener Extension (ALE) 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 appropriate implementation details and developer documentation is available in the respective Github repositories for each component [6].

5.2 Cloud Services

5.2.1 Component: SLA Manager

Component/service name	SLA Manager
URL of source codes	https://github.com/symbiote-h2020/SLAManager
URL of Javadoc documentation	https://symbiote-h2020.github.io/SLAManager/doxygen
List of features included	• SLA management ○ Creation ○ Update ○ Deletion • SLA enforcement and violation notification ○ SLA Evaluation ○ Violation notification

5.2.1.1 SLA Manager component description

The SLA Manager evaluates, with the help of the Monitoring component that all resources shared in a federation comply with the QoS constraints defined in the federation properties.

Each time a federation is joined, a new SLA for the platform itself and the entire federation is created. Once created, the SLA Manager requests metrics data from the Monitoring components about the resources that have been shared in the federation and evaluate the SLA rules according to such data. If a rule is violated, a notification will be send to the Federation Manager about such violation.

5.2.1.2 SLA Manager component interface

The SLA Manager only consumes and produces platform internal messages. Thus, the provided symbIoTe specific interfaces using RabbitMQ topics for communication as listed below.

#	Interface	Name	From	Consumer	Address/Queue	Payload[2]	Response	Description
1	RabbitMQ TOPIC	Federation created	Federation Manager	SLA Manager	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.created	Federation	-	Creates a new SLA with the QoS constraints found in the Federation payload
2	RabbitMQ TOPIC	Federation updated	Federation Manager	SLA Manager	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.changed	Federation	-	Updates the federation's SLA with new constraints (if applicable) and removes the SLA altogether if the platform has left the federation
3	RabbitMQ TOPIC	Federation deleted	Federation Manager	SLA Manager	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.deleted	Federation	-	Deletes the SLA (if applicable) associated to the federation that has been deleted
4	RabbitMQ TOPIC	Violation Notification	SLA Manager	Federation Manager	Exchange: symbIoTe.slam Routing key: symbIoTe.sla.violation	ViolationNotification	-	Notifies about a violation of a federation's SLA

Table 4. SLA Manager component interfaces

5.2.2 Component: Monitoring

Component/service name	Monitoring
URL of source codes	https://github.com/symbiote-h2020/Monitoring
URL of Javadoc documentation	https://symbiote-h2020.github.io/Monitoring/doxygen
List of features included	• Metrics gathering • Metrics retrieval ○ Raw ○ Aggregated ○ Summary

5.2.2.1 Monitoring component description

Monitoring is in charge of gathering metrics related to IoT devices shared either at L1 or 2. To do so, it offers a REST interface which accepts a list of metrics that will be saved in the component database. A metric is composed of:

- Tag: The metric name
- Value: The value at a moment in time
- Date: The date in which the value was taken

The same REST interface offers a series of methods to retrieve such metrics both in raw form or aggregated by device and tag and applying different operations to data to get averages, maximum, minimum, etc.

5.2.2.2 Monitoring component interface

The table below lists all consumed and provided interfaces of this component. For internal communication RabbitMQ is used whereas external communication is done via REST interfaces.

#	Interface	Name	From	Consumer	Address/Queue	Payload[3]	Response	Description
1	RabbitMQ TOPIC	Resource Registered	Registration Handler	Monitoring	Exchange: symbIoTe.registrationHandler Routing key: symbIoTe.rh.resource.core.register	List<CloudResource>	-	New resources have been registered at the core. Monitoring will upload its metrics to CRM.
2	RabbitMQ TOPIC	Resource Unregistered	Registration Handler	Monitoring	Exchange: symbIoTe.registrationHandler Routing key: symbIoTe.rh.resource.core.delete	List<String>	-	Some resources have been deleted from the core. Monitoring will stop sending its metrics to CRM.
3	RabbitMQ TOPIC	Resource Shared	Registration Handler	Monitoring	Exchange: symbIoTe.registrationHandler Routing key: symbIoTe.rh.resource.shared	Map<String, List<CloudResource>>	-	Some resources have been shared in some federations. The map key is the federation identifier while the value are resources shared on that federation. Monitoring will use this information when SLA Manager asks for metrics of resources in a federation.
4	RabbitMQ TOPIC	Resource Unshared	Registration Handler	Monitoring	Exchange: symbIoTe.registrationHandler Routing key: symbIoTe.rh.resource.unshared	Map<String, List<CloudResource>>	-	Some resources have been unshared from some federations. The map key is the federation identifier while the value are resources unshared on that federation
5	RabbitMQ	Resource	Registration	Monitoring	Exchange:	List<String>	-	The metadata of a

	TOPIC	Deleted	Handler		symbIoTe.registrationHandler Routing key: symbIoTe.rh.resource.deleted			resource has been deleted from the local registry. The input is the list of resource identifiers deleted.
6	RabbitMQ TOPIC	Resource Accessed	Resource Access Proxy	Monitoring	Exchange: symbIoTe.resourceAccessProxy Routing key: symbIoTe.rap.resource.access	NotificationMessage		RAP provides statistics about resources that have been accessed. Monitoring will use this information to inference availability values.
7	REST	Save Metrics	Platform Owner	Monitoring	POST /monitoring/metrics/raw	List<DeviceMetric>	List<DeviceMetric>	Save some metrics in the monitoring component
8	REST	Get Raw Metrics	Platform Owner	Monitoring	GET /monitoring/metrics/raw	* device * metric * startDate * endDate	List<DeviceMetric>	Get a list of metrics in raw format. Metrics can be filtered by any combination of the input parameters by using them in the request query but all of them are optional. If none is provided, it will return the whole set of metrics gathered so far.
	REST	Get Aggregated Metrics	Platform Owner	Monitoring	GET /monitoring/metrics/aggregated	* device * metric * startDate * endDate * operation * count	List<AggregatedMetrics>	Get some metrics grouping them by resource and tag and providing also aggregated values provided in the operation and count parameter lists. Query parameters are the same as in raw query plus a list of operations to aggregate data that can be: • sum • avg • max • min

								• stdDevPop • stdDevSamp Count parameter will return the elements whose value match the ones in the count query parameter.
	REST	Get Summary Metrics	SLA Manager	Monitoring	GET /monitoring/metrics/summary	* federation * metric	Map<String, Double>	Gets the average value of a metric among all the resources shared in a federation during a period. Metric query parameter should be in the format <tag>.<type>.<duration> Where • tag is a metric tag to get the average from • type is an optional element to count only those devices of a certain type • duration is an optional element telling the number of days, counting back from the present moment, to get metric values from. If not present it's implicit that all metrics from the moment the resource

								was shared up to now will be considered.

Table 5. Monitoring component interfaces

5.2.3 Component: Registration Handler

Component/service name	**Registration Handler**
URL of source codes	https://github.com/symbiote-h2020/RegistrationHandler
URL of Javadoc documentation	https://symbiote-h2020.github.io/RegistrationHandler/doxygen
List of features included	• Resources Metadata Management at L1 and 2 ○ Registration ○ Update ○ Delete • Resource Sharing and Unsharing at L2 • Synchronization with the Core

5.2.3.1 Registration Handler component description

The Registration Handler acts as an intermediary between Platform Owners and SymbIoTe Core (for L1 resources) and Local Registry (for L2 resources), allowing the easy registration and deregistration of resources both at Level 1 with the core and Level 2 with federations. To do so, it offers a REST interface that can be used by Platform Owners.

5.2.3.2 Registration Handler component interface

The table below lists all consumed and provided interfaces of this component. For internal communication RabbitMQ is used whereas external communication is done via REST interfaces.

#	Interface	Name	From	Consumer	Address/Queue	Payload[4]	Response	Description
1	RabbitMQ TOPIC	Resource registered	Registration Handler	Monitoring RAP	Exchange: symbIoTe.registrationHandler Routing key: symbIoTe.rh.resource.core.regis ter	List<CloudRes ource>	-	Some resources have been registered at the core
2	RabbitMQ TOPIC	Resource updated	Registration Handler	Monitoring RAP	Exchange: symbIoTe.registrationHandler Routing key: symbIoTe.rh.resource.core.upda te	List<CloudRes ource>	-	Some resource metadata has been updated at the core
3	RabbitMQ TOPIC	Resource deleted	Registration Handler	Monitoring RAP	Exchange: symbIoTe.registrationHandler Routing key: symbIoTe.rh.resource.core.delet e	List<String>	-	Some resources have been deleted from the core
4	RabbitMQ TOPIC	Resource registered local	Registration Handler	Monitoring RAP TM	Exchange: symbIoTe.registrationHandler Routing key: symbIoTe.rh.resource.updated	List<CloudRes ource>	-	Some resources have been registered or updated in the local registry
5	RabbitMQ TOPIC	Resource deleted local	Registration Handler	Monitoring RAP TM	Exchange: symbIoTe.registrationHandler Routing key: symbIoTe.rh.resource.deleted	List<String>	-	Some resources have been deleted from the local registry
6	RabbitMQ TOPIC	Resource shared	Registration Handler	Monitoring RAP	Exchange: symbIoTe.registrationHandler Routing key: symbIoTe.rh.resource.shared	Map<String, List<CloudRes ource>>	-	Some resources have been shared in some federations. The map key is the federation identifier while the value are resources shared on that federation.
7	RabbitMQ TOPIC	Resource unshared	Registration Handler	Monitoring RAP	Exchange: symbIoTe.registrationHandler Routing key:	Map<String, List<String>>		Some resources have been unshared from some federations.

					symbIoTe.rh.resource.unshared			The map key is the federation identifier while the value are resources unshared on that federation.
8	REST	Resources information	Platform Owner	Registration Handler	GET /resources	-	List<Cloud Resource>	Gets the list of all resources shared by L1 or L2
9	REST	Resource information	Platform Owner	Registration Handler	GET /resource	resourceInternalId	CloudResource	Get the metadata description of a registered resource on L1 or L2
10	REST	Register resource	Platform Owner	Registration Handler	POST /resource	CloudResource	CloudResource	Register a resource in the core
11	REST	Register resources	Platform Owner	Registration Handler	POST /resources	List<CloudResource>	List<Cloud Resource>	Register a list of resources into the core
12	REST	Register RDF resources	Platform Owner	Registration Handler	POST /rdf-resources	RdfCloudResourceList	List<Cloud Resource>	Register a list of resources described in RDF format
13	REST	Update resource	Platform Owner	Registration Handler	PUT /resource	CloudResource	CloudResource	Update the information of a previously registered resource If the resource wasn't previously registered, it will be registered with this call as well.
14	REST	Update resources	Platform Owner	Registration Handler	PUT /resources	List<CloudResource>	List<Cloud Resource>	Update the information of some previously registered resources If any resource wasn't previously registered, it will be registered with this call as well.
15	REST	Delete resource	Platform Owner	Registration Handler	DELETE /resource	resourceInternalId	CloudResource	Deletes from the core the resource with internal id which appears in the query parameter
16	REST	Delete resources	Platform	Registration	DELETE /resources	resourceIntern	List<Cloud	Deletes from the core

			Owner	Handler		allds	Resource>	the resources with internal IDs which appear in the query parameter as a comma-separated list
17	REST	Clear resources	Platform Owner	Registration Handler	DELETE /clear	-	List<Cloud Resource>	Removes all resources registered at the core and clears the local database Returns a list of the resources that have been deleted.
18	REST	Synchronize	Platform Owner	Registration Handler	PUT /sync			Deletes all resources from the core and re-registers the ones at the local database
20	REST	Update local resources	Platform Owner	Registration Handler	POST /local/resources	List<CloudResource>	List<Cloud Resource>	Registers or updates a list of resources in the Platform Registry If the resource was previously registered, it will be updated.
21	REST	Delete local resources	Platform Owner	Registration Handler	DELETE /local/resources	resourceIds	List<Cloud Resource>	Removes some resources from the Platform Registry Receives a comma-separated list of resource IDs to delete as query parameter. If some resources were shared in some federations, they will be automatically unshared.
22	REST	Share resources	Platform Owner	Registration Handler	PUT /local/resources/share	Map<String, Map<String, Boolean>>	Map<String, List<CloudResource>>	Share resources with a federation Resources need to be registered previously Input is a JSON object

								whose keys are the federation IDs. As value, there's another object with the resource internal ID and if the resource should be shared by bartering for that federation. Output is a JSON object whose key is the federation ID and the value is a list of the resource's metadata which have been shared to that federation.
23	REST	Unshare resources	Platform Owner	Registration Handler	DELETE /local/resources/share	Map<String, List<String>>	Map<String, List<CloudResource>>	Removes a list of resources from some federations Input is a JSON object whose keys are the federation IDs and each value is a list of resource internal ID's to remove from that federation. Output is a JSON object whose keys are federation IDs and each value is a list of resource metadata from the resources that were removed from that federation.

Table 6. Registration Handler component interfaces

5.2.4 Component: Resource Access Proxy

5.2.4.1 Resource Access Proxy component description

The Resource Access Proxy (RAP) component is used by applications to access the resources from the underlying platform. The RAP implementation from L1 is reused and updated to achieve L2 compliance. New features include receiving information from Registration Handler when registering L2 resources and checking federation access policies when accessing L2 resources. Resource access data is now forwarded to the Monitoring component, instead of the Core Resource Access Monitor, as in previous releases. When accessing L2 resources, the RAP needs to check the federation access policy, i.e. if the resource being accessed has already been shared within the same federation as the application trying to access it. The updated and the new interfaces from the L2 implementation are listed in Section 5.2.7.2

5.2.4.2 Resource Access Proxy component interface

The Resource Access Proxy exposes both REST / OData and RabbitMQ interfaces. It communicates with the Registration Handler, Monitoring, Federation Manager and the B&T components via RabbitMQ, and exposes REST / OData-like interfaces for direct resources' access as listed below. An additional interface is used as a push mechanism, which makes use of WebSockets to link the notifications to the client applications. These interfaces have been implemented in previous releases and have not been altered during the implementation of federation environment.

The RAP interfaces listed below, document the final version of all provided and consumed endpoints.

#	Interface	Name	Message Type	From	Msg Consumers	Address/Queue	Payload	Description
1	Resource (L1) registration	symbIoTe.rap.registrationHandler.register_resources	RabbitMQ	RH	RAP	Exchange: symbIoTe.registrationHandler Routing key: routing-direct	Resource information	Event requesting registration of new L1 resource
2	Resource (L1) registration	symbIoTe.rap.registrationHandler.unregister_resources	RabbitMQ	RH	RAP	Exchange: symbIoTe.registrationHandler Routing key: routing-direct	Resource id	Event requesting unregistration of existing L1 resource
3	Resource (L1) registration	symbIoTe.rap.registrationHandler.update_resources	RabbitMQ	RH	RAP	Exchange: symbIoTe.registrationHandler Routing key: routing-direct	Resource information	Event requesting update of existing L1 resource
4	Resource (L2) registration and update	symbIoTe.rap.registrationHandler.l2.update_resources	RabbitMQ	RH	RAP	Exchange: symbIoTe.registrationHandler Routing key: routing-direct	Resource information	Event requesting registration of new or update of existing L2 resource
5	Resource (L2) unregistration	symbIoTe.rap.registrationHandler.l2.unregister_resources	RabbitMQ	RH	RAP	Exchange: symbIoTe.registrationHandler Routing key: routing-direct	Resource id	Event requesting unregistration of existing L2 resource
6	Resource (L2) share	symbIoTe.rap.registrationHandler.l2.share_resources	RabbitMQ	RH	RAP	Exchange: symbIoTe.registrationHandler Routing key: routing-direct	List of shared resources	Event requesting sharing of existing L2 resource within a certain federation(s)
7	Resource (L2) unshare	symbIoTe.rap.registrationHandler.l2.unshare_resources	RabbitMQ	RH	RAP	Exchange: symbIoTe.registrationHandler Routing key: routing-direct	List of unshared resources	Event requesting unsharing of existing L2 resource within a certain federation(s)
8	Forwarding access data		Rabbit MQ	RAP	Monitoring	Exchange: symbIoTe.resourceAccessProxy Routing key: routing-direct	Information about successful and failed access attempts	Event sharing information about the number and type of access attempts to resources
9	Checking bartering & trading rights		Rabbit MQ	RAP	B&T Monitor		Information about the resource and platform which tries to access the resource	For bartered resource the access rights should be checked with Bartering & Trading Monitor
10	Federation creation	symbIoTe.federation.queue.created	Rabbit MQ	Federation Manager	RAP	Exchange: symbIoTe.federation Routing key: topic-exchange	Information about federations and its members	Event sharing information about new federation. Needed for creation of federation access policies

11	Federation change	symbIoTe.federation.queue.changed	Rabbit MQ	Federation Manager	RAP	Exchange: symbIoTe.federation Routing key: topic-exchange	Information about federations and its members	Event sharing information about federation changes. Needed for creation of federation access policies
12	Federation deleted	symbIoTe.federation.queue.deleted	Rabbit MQ	Federation Manager	RAP	Exchange: symbIoTe.federation Routing key: topic-exchange	Information about federations and its members	Event sharing information about federation deletion. Needed for creation of federation access policies

Table 7. Resource Access Proxy component interfaces

5.2.5 Component: Subscription Manager

Component / service name	Subscription Manager
URL of source code	https://github.com/symbiote-h2020/SubscriptionManager
URL of Javadoc documentation	https://symbiote-h2020.github.io/SubscriptionManager/doxygen
List of features included	• Platform subscription definition and updates • Platform subscription distribution among federated platforms • Storing subscriptions of federated platforms • Distribution of shared resources to all subscribed federated platforms • Informing subscribed federated platforms about removed shared resources

5.2.5.1 Subscription Manager component description

The Subscription Manager (SM) implements two major features needed for resource sharing and access within a federation:

- Notification of resource updates to all federated platforms while taking into account their subscription settings (resource criteria the platforms are interested in)
- Subscription management and distribution of available and updated information received from other federated platforms

Every platform's initial subscription object is set to demand all resources from the federated platforms. The subscription can be altered in accordance with a platform's own preferences. There are several parameters that can be specified in the subscription definition:

- resource type (mandatory)
- location (optional)
- observed properties (optional for sensor resources)
- capabilities (optional for actuator resources)

Based on the acquired values the Subscription Manager can determine if a federated platform is interested in a specific resource.

When a platform joins a federation, the SM will broadcast its subscription settings to inform others about the interest in certain resource types. Also, each time a subscription is updated by a platform owner, the new subscription is delivered to federated platforms, so they are able to update their notification settings for future resource announcements.

The actual filtering for resources announcements is done in the Subscription Manager as soon as the Platform Registry sends the newly registered resource notification.

The SM iterates over the received resources, and forms a different message payload for each federated platform which contains only a relevant subset of resources that match their subscriptions.

After receiving information about newly shared federated resources, other federated platforms can use them as long as they don't receive notification about their deletion, or one of the resource providing platforms leaves the federation.

5.2.5.2 Subscription Manager component interface

The Subscription Manager component uses a HTTP REST interface to receive subscription definitions from the platform owner, which is also used to communicate with the SMs of other federated platforms and is secured by embedded security headers of appropriate access policies. For internal communication with the Federation Manager and the Platform Registry components, the SM uses the RabbitMQ message broker.

An overview of the SM interfaces is shown in Table 8. Subscription Manager component interfaces. Interfaces 1 – 3 receive internal messages from the Federation Manager about the federation environment, interfaces 4 – 7 handle the bidirectional internal communication between the Platform Registry and the SM regarding the federated resources. The platform owner specifies the subscription using the REST endpoint, which is shown in interface 8. Finally interfaces 9 – 11 represent the HTTP REST communication between the SMs of the federated platforms.

#	Interface	Name	From	Consumer	Address/Queue	Payload[5]	Response	Description
1	RabbitMQ TOPIC	Federation (affiliation) added	FM	AAM SLAM Monitoring B&T SM	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.created	Federation	-	New Federation created.
2	RabbitMQ TOPIC	Federation (affiliation) updated	FM	AAM SLAM Monitoring B&T SM	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.changed	Federation	-	Update Federation info.
3	RabbitMQ TOPIC	Federation (affiliation) removed	FM	AAM SLAM Monitoring B&T SM	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.deleted	Federation	-	Federation deleted.
4	RabbitMQ TOPIC	Resources added/updated	PR	SM	Exchange: symbIoTe.subscriptionManager Routing key: symbiote.subscriptionManager.addOrUpdateFederatedResources	ResourcesAddedOrUpdatedMessage	-	New resources are added/updated to federation and will be disseminated to all subscribed federated platforms.
5	RabbitMQ TOPIC	Resources deleted	PR	SM	Exchange: symbIoTe.subscriptionManager Routing key: symbiote.subscriptionManager.removeFederatedResources	ResourcesDeletedMessage	-	Resources are deleted from federation and all subscribed federation members will be notified about it.
6	RabbitMQ DIRECT	Resources added/updated	SM	PR	Exchange: symbIoTe.platformRegistry Routing key: symbiote.platformRegistry.addOrUpdateFederatedResources	ResourcesAddedOrUpdatedMessage	-	New resources information received from federated platform.
7	RabbitMQ DIRECT	Resources deleted	SM	PR	Exchange: symbIoTe.platformRegistry Routing key:	ResourcesDeletedMessage	-	Deleted resources information received from federated platform.

					symbiote.platformRegistry.remove FederatedResources			
8	HTTP	Subscription definition	Platform owner	SM	POST /subscriptionManager/subscribe	Subscription	HTTP Status code	Subscription definition which represent an interest in some specific subset of resources. It is then forwarded to the corresponding federated platforms.
9	HTTP	Subscription definition	SM	SM (platforms in federation)	POST /subscriptionManager/subscription	Subscription	HTTP Status code	Broadcast of subscription to all federated platforms. It occurs when platforms are federated for the first time, and every time when subscription is updated.
10	HTTP	Resources added/updated	SM	SM (subscribed platforms in federation)	POST /subscriptionManager/addOrUpdate	ResourcesAddedOrUpdatedMessage	HTTP Status code	Broadcast of added or changed resources to subscribed federated platforms.
11	HTTP	Resources deleted	SM	SM (subscribed platforms in federation)	POST /subscriptionManager/delete	ResourcesDeletedMessage	HTTP Status code	Broadcast of deleted resources to subscribed federated platforms

Table 8. Subscription Manager component interfaces

5.2.6 Component: Authentication and Authorization Manager

Component/service name	Authentication and Authorization Manager
URL of source codes	https://github.com/symbiote-h2020/AuthenticationAuthorizationManager
URL of Javadoc documentation	https://symbiote-h2020.github.io/AuthenticationAuthorizationManager/doxygen
List of features included	• user registration • local attribute management • user certificates management • credentials validation

5.2.6.1 Authentication and Authorization Manager component description

The main enhancements in the Authentication and Authorization Manager (AAM) relevant for L2, are dealing with improved performance and internal enhancements as well as the final alignment of necessary interfaces and payloads listed in Table 9.

Additionally, the internal functionalities were enhanced to address and ensure the General Data Protection Regulation which is discussed in Section 6.3.

5.2.6.2 Authentication and Authorization Manager component interface

The table below lists all consumed and provided interfaces of this component. For internal communication RabbitMQ is used whereas external communication is done via REST interfaces.

#	Interface	Name	From	Consumer	Address/Queue	Payload[6]	Response	Description
1	HTTP	Get available AAMs	REST client	AAM	GET /get_available_aams	empty	AvailableAAMsCollection, HTTP status: 200 on ok and 500 on error	Returns information about all available AAMs in the system
2	HTTP	Get available AAMs internally	REST client	AAM	GET /get_internally_aams	empty	AvailableAAMsCollection, HTTP status: 200 on ok and 500 on error	Returns information about all available AAMs in the system containing internal URLs
3	HTTP	Get component certificate	REST client	AAM	GET /get_component_certificate/platform/{platformIdentifier}/component/{componentIdentifier}	platformIdentifier componentIdentifier	component certificate in PEM format, HTTP status 200 or 404 on missing, 500 on error	Returns component certificate in PEM format
4	HTTP	Get user details	REST client	AAM	POST /get_user_details	Credentials	UserDetails, HTTP status code (200, 400 missing user, 401 bad user password)	Return registered user details
5	AMQP	Get user details	Administration	AAM	Exchange: symbIoTe.AuthenticationAuthorizationManager Queue: symbIoTe-AuthenticationAuthorizationManager-get_user_details_request Exchange: symbIoTe-AuthenticationAuthorizationManager.get_user_details_request	UserManagementRequest	UserDetailsResponse	Return registered user details
6	AMQP	Manage user (application / platform owner)	Administration	AAM	Exchange: symbIoTe.AuthenticationAuthorizationManager Queue: symbIoTe-AuthenticationAuthorizationManager	UserManagementRequest	ManagementStatus	Used to manage users (create, update, delete…)

					-manage_user_request Exchange: symbIoTe.AuthenticationAuthorizationManager.manage_user_request			
7	HTTP	Manage user (application / platform owner)	REST client	AAM	POST /manage_users	UserManagementRequest	ManagementStatus, HTTP status code	Used to manage users (create, update, delete…)
8	HTTP	Sign certificate request	REST client	AAM	POST /sign_certificate_request	CertificateRequest	certificate in PEM format, HTTP status	Used to issue new certificate for client/component/platform
9	HTTP	Home login	REST client	AAM	POST /get_home_token	Headers with X-Auth-token containing Signed LoginRequest	Headers with X-Auth-token containing token String for that client	Returns HOME token used to access restricted resources offered in SymbIoTe
10	HTTP	Roamed / federated / foreign login	REST client	AAM	POST /get_foreign_token	Headers with: X-Auth-token containing HOME Authorization Token String for that client; (opt) X-Auth-Client-Cert containing PEM Certificate String matching SPK from token (opt) X-Auth-AAM-Cert containing PEM Certificate String used to sign the client certificate	Headers with X-Auth-token containing FOREIGN/ROAMED/FEDERATED token String for that client	Returns FOREIGN token used to access restricted resources offered in SymbIoTe federations
11	HTTP	Guest login	REST client	AAM	POST /get_guest_token	empty	Headers with X-Auth-token containing GUEST token String	Returns GUEST token used to access public resources offered in SymbIoTe
12	AMQP	Validate credentials	Administration	AAM	Exchange: symbIoTe.AuthenticationAuthorizati	ValidationRequest	ValidationStatus, HTTP status	Verifies, if provided token

					onManager Queue: symbIoTe-AuthenticationAuthorizationManager-validate_requestExchange: symbIoTe.AuthenticationAuthorizationManager.validate_request			is valid
13	HTTP	Validate credentials	REST client	AAM	POST /validate_credentials	Headers with: X-Auth-token containing Authorization Token String for that client; (opt) X-Auth-Client-Cert containing PEM Certificate String matching SPK from token (opt) X-Auth-AAM-Cert containing PEM Certificate String used to sign the client certificate (opt) X-Auth-ISS-Cert containing PEM Certificate String matching the ISS, IPK and signature from the FOREIGN token	ValidationStatus, HTTP status	Verifies, if provided token is valid
14	HTTP	Validate Foreign Token Origin Credentials	AAM	AAM	POST /validate_foreign_token_origin_credentials	Foreign token String in body	ValidationStatus, HTTP status	Allows to confirm that the origin (HOME) credentials (SUB & SPK) used to issue the given FOREIGN token in another AAM have not been revoked
15	AMQP	Revoke credentials	Administration	AAM	Exchange: symbIoTe.AuthenticationAuthorizationManager Queue: symbIoTe-AuthenticationAuthorizationManager-manage_revocation_request Exchange: symbIoTe-AuthenticationAuthorizationManager.manage_revocation_request	RevocationRequest	RevocationResponse	Allows to revoke compromised tokens and certificates

16	HTTP	Revoke credentials	REST client	AAM	POST /revoke_credentials	RevocationRequest	HTTP Response isRevoked boolean in body, httpStatus: 200/400/401/500	Allows to revoke compromised tokens and certificates
17	AMQP	Released attributes provisioning	Administration	AAM	Exchange: symbIoTe.AuthenticationAuthorizationManager Queue: symbIoTe-AuthenticationAuthorizationManager-manage_local_attributes_request Exchange: symbIoTe-AuthenticationAuthorizationManager.manage_local_attributes_request	LocalAttributesManagementRequest	Map<String,String> localAttributes	Allows local user attributes management
18	AMQP	Federation create	Administration	AAM	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.created	Federation	-	Allows to inform AAM about federation creation
19	AMQP	Federation delete	Administration	AAM	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.deleted	Federation	-	Allows to inform AAM about federation delete
20	AMQP	Federation update	Administration	AAM	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.changed	Federation	-	Allows to inform AAM about federation update

Table 9. Authentication and Authorization component interfaces

5.2.7 Component: Trust Manager

Component / service name	Trust Manager
URL of source code	https://github.com/symbiote-h2020/TrustManager
URL of Javadoc documentation	https://symbiote-h2020.github.io/TrustManager/doxygen
List of features included	• Calculate (own) resource trust for offered resources with federated platforms ○ Calculate resource accessibility metric ○ Calculate resource dependability metric • Calculate (foreign) platform reputation for federated platforms ○ Calculate federation affiliation metric ○ Calculate Anomaly Detection metric ○ Calculate B&T interaction metric • Calculate (foreign) adaptive resource trust for shared resources from foreign platforms within the federation ○ Aggregate shared (foreign) resource trust value and internal platform reputation

5.2.7.1 Trust Manager component description

The Trust Manager component is responsible for calculating the three trust and reputation levels detailed in section 4.4 and in [4]. The calculated values lay the ground for more advanced search queries and will also impact the B&T behaviour to allow the rejection of platforms with poor reputation measures.

The calculation of resource trust, platform reputation and adaptive resource trust is periodically calculated only for relevant resources and platforms. Thus, the resource trust is only updated for as long as the resource is shared within the federations. The same applies for platform reputation, which is only calculated for federated platforms. As the adaptive resource trust relies on actual platform reputation and foreign resource trust values, also the third metric will be calculated only if the underlying base values (resource trust and platform reputation) are up-to-date.

The batch processing and update enables a fixed time span where current trust and reputation values are available to all other platforms which consume the respective RabbitMQ topic.

5.2.7.2 Trust Manager component interface

The Trust Manager does not expose any interface to external services or Core components. It mainly consumes internal information and status updates sent by the Federation Manager, the Subscription Manager and the Registration Handler. Also, it requests global event statistics from the Core Bartering & Trading and Anomaly Detection components using the common REST communication approach secured by symbIoTe security headers.

To distribute trust and reputation updates to internal consumers, one dedicated topic with specific routing keys is available as listed in Table 10.

#	Interface	Name	From	Consumer	Address/Queue	Payload[7]	Response	Description
1	RabbitMQ TOPIC	Resource Trust updated	TM	Registration Handler	Exchange: symbIoTe.trust Routing key: symbIoTe.trust.resource_trust.updated	TrustEntry	-	Notify internal consumers about a resource trust update.
2	RabbitMQ TOPIC	Platform Reputation updated	TM	B&T Manager	Exchange: symbIoTe.trust Routing key: symbIoTe.trust.platform_reputation.updated	TrustEntry	-	Notify internal consumers about a platform reputation update.
3	RabbitMQ TOPIC	Adaptive Resource Trust updated	TM	Platform Registry	Exchange: symbIoTe.trust Routing key: symbIoTe.trust.adaptive_resource_trust.updated	TrustEntry	-	Notify internal consumers about an adaptive resource trust update.

Table 10. Trust Manager component interfaces

5.2.8 Component: Platform Registry

Component/service name	**Platform Registry**
URL of source codes	https://github.com/symbiote-h2020/PlatformRegistry
URL of Javadoc documentation	https://symbiote-h2020.github.io/PlatformRegistry/doxygen
List of features included	• Register, update and delete home resource metadata exposed to the federations. • Share and unshare resources to federations. • Search federated resources using ranking and filtering to handle search requests.

5.2.8.1 Platform Registry component description

This component enables the registration of IoT resources (devices and services) offered by IoT platforms to be discoverable by their federated platforms, storing the metadata of the resources available within the symbIoTe federation. Each platform maintains a list of available resources in its own local registry, through which the search function is performed. A platform owner of a home platform can register new resources and update or delete existing resources in a federation. Registered resources can be shared with different federations to which the platform belongs and can be unshared from federations that they have been exposed to. The platform can search resources in its own platform registry filtering and sorting them according to specified resource parameters. Resources from other platforms can then be accessed through the RAP of the corresponding platform.

5.2.8.2 Platform Registry component interface

The supported RabbitMQ and REST endpoints of the platform registry component are listed in Table 11. Platform registry receives resource metadata update or delete requests from the Registration Handler using RabbitMQ communication; it registers, updates or removes the set of resources received and returns a list of the updated resources using the direct exchanges 1 and 2, respectively, listed in the table below.

Federated resources exposed by other platforms within the federation are updated or removed when the appropriate added, updated or deleted messages are received from the Subscription Manager (direct exchanges 3 and 4 in the table).

Also, the platform registry component receives share or unshare requests from the Registration Handler to share resources with federations (by bartering or not) or unshare resources exposed to federations (Direct exchanges 5 and 6 in the table).

Finally, it performs search functionalities using the common REST communication approach secured by symbIoTe security headers (labelled as 7 in the table). It handles the HTTP GET requests on the search endpoint, filtering and sorting the available resources according to specified parameters. For example, the following request:

http://localhost:8203/pr/search/?federationId=fed2 is used to list resources that belong to federation fed2.

Any of the parameters below can be specified:

- name: a list with the resource names

- description: the resource description
- id: the list of the federated resource identifiers
- federationId: the list of federation IDs
- observes_property: the property observed by a sensor
- resource_type: the resource type e.g. stationarySensor
- location_name: the names of the locations
- location_lat: the latitude of a WGS84 location for devices
- location_long: the longitude of a WGS84 location for devices
- max_distance: the radius for geospatial queries
- sort: returned resource list will be sorted by the given resource field. Ascending or descending order can also be specified.

#	Interface	Name	From	Consumer	Address/Queue	Payload[8]	Response	Description
1	RabbitMQ Direct	Update cloud resources	Registration Handler	Platform Registry	Exchange: platformRegistry . Routing key: platformRegistry .update	List<CloudReso urce> cloudResources	List<CloudReso urce> cloudResources	Message to store a set of cloud resources to be exposed to the federation.
2	RabbitMQ Direct	Remove cloud resources	Registration Handler	Platform Registry	Exchange: platformRegistry . Routing key: platformRegistry .delete	List<String> internalIds	List<String> internalIdsRemo ved	Message to remove a set of cloud resources exposed in the federation.
3	RabbitMQ Direct	Add or update federated resources	Subscription Manager	Platform Registry	Exchange: platformRegistry Routing key: platformRegistry . addOrUpdateFe deratedResourc es	ResourcesAdde dOrUpdatedMes sage resourcesAdded OrUpdated	-	Message to store the new federated resources offered by other platforms within the federation
4	RabbitMQ Direct	Remove federated resources	Subscription Manager	Platform Registry	Exchange: platformRegistry Routing key: platformRegistry . removeFederate dResources	ResourcesDelet edMessage resourcesDelete d	-	Message to remove the federated resources of other platforms
5	RabbitMQ Direct	Share cloud resources	Registration Handler	Platform Registry	Exchange: platformRegistry . Routing key: platformRegistry .share	Map<String, Map<String, Boolean>> resourcesToBe Shared	List<CloudReso urce> resourcesShare d	Message to share cloud resources within federations.
6	RabbitMQ Direct	Unshare cloud resources	Registration Handler	Platform Registry	Exchange: platformRegistry . Routing key: platformRegistry .unshare	Map<String, List<String>> resourcesToBe Unshared	List<CloudReso urce> cloudResources Unshared	Message to unshare cloud resources exposed to federations

7	HTTP	Search federated resources	Platform Owner	Platform Registry	GET /pr/search/	List<String> name List<String> description List<String> id List<String> federationId List<String> observes_property String resource_type List<String> location_name Double location_lat Double location_long Double max_distance Double resource_trust Double adaptive_trust String sort	FederationSearchResult response	Search request to list the federated resources that match to the parameters specified

Table 11. Platform Registry component interfaces

5.2.9 Component: Federation Manager

Component / service name	Federation Manager
URL of source code	https://github.com/symbiote-h2020/FederationManager
URL of Javadoc documentation	https://symbiote-h2020.github.io/FederationManager/doxygen
List of features included	• Receive and process federation and QoS updates from Administration component • Validate federation information and access (security & business validation) • Distribute federation updates to relevant components within platform • Maintain and generate federation history events for all federation activities • Aggregate federation history per platform for Trust management

5.2.9.1 Federation Manager component description

The Federation Manager component holds and manages the actual state of all federations the current platform is affiliated with. Additionally, the federation object also reflects the necessary properties to identify all participating foreign platforms and assigned QoS constraints applicable for all federation members.

The component is the central entry point for receiving federation updates (creation, update, deletion) from the Administration component in the Core Domain via the Interworking Interface. The communication between these components is established through REST and secured by enforcing the symbIoTe security header and token approach.

All received information is validated, stored and forwarded to all internal platform components which need to know the updates. This distribution is managed using a RabbitMQ topic with distinct routing keys per operation (created, changed, deleted).

5.2.9.2 Federation Manager component interface

The Federation Manager acts as a central access point on the platform level by receiving federation updates from the Administration component within the Core Domain. This exchange is done using defined synchronous HTTP REST communication which is secured by embedded security request and response headers using the symbIoTe ABAC and applied policy mechanisms.

The received federation information is stored in the Federation Manager and distributed to all relevant platform internal symbIoTe components using a dedicated RabbitMQ topic.

The supported REST and RabbitMQ endpoints and payloads are listed in the Table 12.

#	Interface	Name	From	Consumer	Address/Queue	Payload[9]	Response	Description
1	HTTP	Federation added Federation changed	Administration	FM	POST /fm/federations/	Federation	HTTP Status code	Receive federation add or update requests from Administration
2	HTTP	Federation removed	Administration	FM	DELETE /fm/federations/{federationId}	-	HTTP Status code	Receive federation deletion requests from Administration
3	RabbitMQ TOPIC	Federation (affiliation) added	FM	AAM SLAM Monitoring B&T SM	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.created	Federation	-	Notify internal consumers about a federation creation event.
4	RabbitMQ TOPIC	Federation (affiliation) updated	FM	AAM SLAM Monitoring B&T SM	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.changed	Federation	-	Notify internal consumers about a federation update event.
5	RabbitMQ TOPIC	Federation (affiliation) removed	FM	AAM SLAM Monitoring B&T SM	Exchange: symbIoTe.federation Routing key: symbIoTe.federation.deleted	Federation	-	Notify internal consumers about a federation deletion event.
6	RabbitMQ RPC	Get federation history by platform ID	TM	FM	Queue: symbIoTe.federation.get_federation_history	PlatformID	FederationHistory	Return federation history events (date platform joined, left and federation created, deleted)

Table 12. Federation Manager component interfaces

5.2.10 Component: Bartering & Trading Manager

Component/service name	**Bartering & Trading Manager**
URL of source codes	https://github.com/symbiote-h2020/BarteringAndTrading
URL of Javadoc documentation	https://symbiote-h2020.github.io/BarteringAndTrading/doxygen
List of features included	• Bartering procedures ○ Validating bartering operations ○ Exchange coupons for access ○ Communicate activity to Core B&T

5.2.10.1 Bartering & Trading Manager component description

The Bartering & Trading Manager (BTM) acts as a local operative within the federation (contrary to the already presented Core B&T component). Before granting access to a resource, the RAP of the resource sharing platform contacts the BTM to understand the status of the bartering between the platforms. The BTM validates the request and contacts the client's platform BTM. If it is in possession of the coupon issued by resource sharing platform, it is sent back and consumed as highlighted in Section 4.5. If not, the receiving platform generates its own coupon and offers it, granting future access to its resources. After acceptation of the exchange, the RAP is notified that access can be granted. The various bartering activities are reported to the Core B&T component, such as validation and notification of coupon creation and consumption. The full process is shown in Figure 9.

5.2.10.2 Bartering & Trading Manager component interface

The table below lists all consumed and provided interfaces of this component. For internal communication RabbitMQ is used whereas external communication is done via REST interfaces.

#	Interface	Name	From	Consumer	Address/Queue	Payload	Response	Description
1	RabbitMQ	authorizeBarteredAccess	RAP	BTM	Exchange: symbIoTe.resourceAccessProxy Routing key: symbIoTe.rap.btm.access	BarteralAccess	String success	Request made be the RAP to its local BTM.
2	REST	getCoupon	BTM	Foreign BTM	GET /get_coupon	CouponRequest	String coupon	Request to the Core B&T to register new Coupon and make it possible to validate it.

Table 13. Bartering & Trading Manager component interfaces

6 Applied Security and Privacy Mechanisms

6.1 Access Policies

To restrict access to the federated resources, an additional access policy was set up. It grants access to users of the platform involved in the federation and it can be satisfied in one of the following ways:

- Using a HOME token, which is issued by the resource local Authentication and Authorization Manager for local users/apps that have claims required to access the resource
- Using a FOREIGN token issued by the local AAM in exchange for a HOME token from the federation members and containing the federation identifier claim
- (optional) Using a HOME token issued by one of the federation members

Resources shared within the federation are only identified via the federation ID where the resources belong to. Thus, the federation members need to be fetched as well to allow building the access policy per request (with enhanced caching if suitable) which will guarantee up-to-date data.

```
/**
 * Creates a new access policy object
 *
 * @param federationIdentifier      identifier of the federation
 * @param federationMembers         set containing federation members identifiers
 * @param localPlatformIdentifier   so that local HOME Tokens are properly
identified
 * @param requiredClaims            map of claims that should appear in local Home
Token to pass the policy
 * @param doesRequireAllLocalTokens requires exchange of platform Home Tokens to
FOREIGN Token issued by local AAM with proper claims to pass the policy
 */
public SingleFederatedTokenAccessPolicy(String federationIdentifier,
                                        Set<String> federationMembers,
                                        String localPlatformIdentifier,
                                        Map<String, String> requiredClaims,
                                        boolean doesRequireAllLocalTokens)
```

Example:

Allow access to the federated resources in OpenHAB instance by user from platforms involved in the federation identified by the federationID. Note: In this example users with valid HOME token from any federation member (OpenHAB, SomeCustomPlatform1 and SomeOtherPlatform2) can access the resources offered by OpenHAB based on the federationID rules.

```
{
    "policyType":"SFTAP",
    "requiredClaims": {
      "fed_id":"federationID",
      "fed_h":"OpenHAB",
      "fed_s":"3",
      "req_loc":"false",
      "fed_m_1":"OpenHAB",
      "fed_m_2":"SomeCustomPlatform1",
      "fed_m_3":"SomeCustomPlatform2"
    }
}
```

A stricter federated access policy comes below. It once again shows access to the federated resources in OpenHAB instance by users from platforms involved in the federation identified by the federationID. Note: This example however requires the foreign/remote platforms' users to exchange their token first in the OpenHAB AAM. This is an extension to support situations where the attribute mapping takes place.

```
{
    "policyType":"SFTAP",
    "requiredClaims": {
      "fed_id":"federationID",
      "fed_h":"OpenHAB",
      "fed_s":"2",
      "req_loc":"true",
      "fed_m_1":"OpenHAB",
      "fed_m_2":"FederatedPlatform"
    }
}
```

6.2 *Bartering Coupons structure*

To enable bartering between the federated platforms, a coupon structure was introduced. The coupon is a JSON Web Token (JWT) generated by the BTMs of platforms participating in the federation. It is created in exchange for the access to the other platforms federated resource. It grants the owner access to the issuer's resources in a defined federation. Its structure is quite similar to tokens generated by the Authentication and Authorization Modules as defined in D1.4 [1].

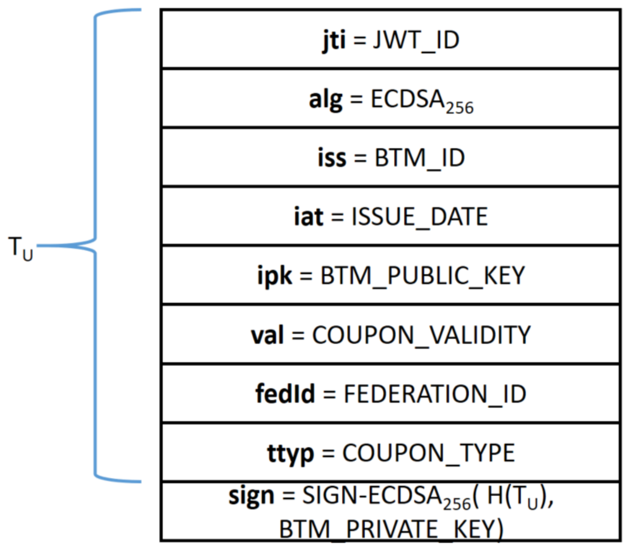

Figure 13. Coupon structure and fields

Each coupon must contain a creation date and federation id, to which they are uniquely associated. The coupon includes its type, i.e., if it is granting access to the resources limited number of usages or for limited period counting from the first use. Moreover, the coupon also contains at the end an element that certifies its authenticity and integrity. Any entity in the system that receives a coupon can easily verify its authenticity by acquiring the public key of the issuer BTM of the coupon (specified in the coupon itself). What's more, such coupon must be registered in the Core BTM, which stores the information about their validity and usages.

The signature field, computed over the whole set of parameters previously mentioned, is included at the end of the coupon. It is generated through the ECDSA-256 algorithm, computed over the whole set of fields included in the coupon.

6.3 Privacy Aspects - Requirements and Solutions

The following section describes and highlights requirements and solutions regarding the privacy aspects in symbIoTe.

Security is defined as the state of a system in which Confidentiality, Integrity and Availability (CIA) of data is granted. Privacy is considered as the ability of people to control the distribution of their personal information. The EU General Data Protection Regulation (GDPR) is set to replace the Data Protection Directive 95/46/EC effective May 25, 2018 [5]. The GDPR is directly applicable in each member state and will lead to a greater degree of data protection harmonization across EU nations and citizens. The *GDPR* is the *most important* change in data privacy regulation in the last two decades and affects almost all software solutions processing personal data. Personal data defined in symbIoTe can be classified as the following:

Account management

- Username
- Password
- E-mail
- Set of Public Keys bound to the user clients
- JWT tokens

User services

- Platforms
- Enablers
- Smart Spaces

Main **components** involved in the personal data storage and processing are:

1) Authentication and Authorization Managers
2) Administration
3) Anomaly Detection (AD)
4) Resource Access Proxies
5) Search

To comply with GDPR and address privacy issues, we defined a new ruleset as following:

1) **Legitimate interest:** Administrator/Platform (Platform, enabler or SSP) Owner must have a valid reason to collect and store the sensitive user input and confidential data.
2) **Consent**: In case an Administrator or a Platform Owner would prefer to use certain types of personal data for individual commercial or marketing purposes, valid approval and explicit permission of each affected individual has to be obtained before.
3) **Revocation**: Individuals may revoke access to their personal data, and the Administrator/Platform Owner is obliged to remove the data in question. Furthermore, it is mandatory to correct errors in personal data upon the individual's request.
4) **Anonymization/Pseudonymization**: Personal data must either be anonymized or go through pseudonymization when stored. This ensures that confidential data is not accessed inappropriately.
5) **Data breach notification**: Violations of personal data that may jeopardize the rights of the subjects involved, must be notified and communicated.

These requirements are adopted in symbIoTe leveraging the following technical solutions:

1) **Separate accepting terms and conditions**
    a. During the user registration phase in the Core/Platform, the user should accept the accepting terms and conditions by checking different checkboxes in each form
    b. The users will be informed regarding the logs save location or IP
    c. Furthermore, the end users must agree accepting terms and conditions, and make sure users read them
    d. The user’s choices will be stored on the AAM and force to the user to accept both
2) **Inform the user if we use certain types of personal data**
    a. During the registration phase in the Core/Platform, a text message will be displayed for each input box that explains if symbIoTe uses the personal data for commercial, analysis or marketing purposes
3) **Send verification email during user registration for activating the account**
4) **Adopt hashing techniques to anonymize DB data and using a pseudonym**
    a. Use a well-documented hashing function as SHA2, SHA3 and apply it to the main personal data as e-mail, username, ID (user, application) in Core and Platform AAM if needed
    b. It is possible for users/maintainers of the software to modify and harden the database by providing data types hashing but the reference implementation doesn’t do that. Current codes were implemented to maximize the visibility of system usage patterns allowing the project team to identify well implemented and faulty elements of the system. Furthermore, to notify the users of their accounts being abused we need their e-mail addresses in plain text. Hence

pseudo-anonymization is not enabled in the AAM code and is an identified future work, in case the project was to be deployed into production. For research deployments the users are notified of their data being available to the platform owners for analytics purposes

5) **Account revocation/deletion**
    a. The user has the capability to remove himself from the platform, by means of a button that facilitates the account deletion from the AAM DB
    b. User public key revocation
6) **Granular permission**
    a. During the registration phase in the Core AAM or Platform AAM, via the user's Privacy Settings Control Panel, the user can give/remove the permissions for each type of processing (e-mail, username, public key) just for marketing and commercial purposes
    b. The user's choices will be stored on the respective AAM
7) **Data breach policy**
    a. A data breach policy will be created in order to notify the user involved in a personal data violation
8) **Log encryption**
    a. All the symbIoTe components owners are requested to encrypt logs and store them in a safe place (i.e. by using logrotate utility and GPG) or by using any other preferred logging utility that supports it.

7 Conclusion

This document describes the final implementation design with all inter-component interactions and workflows to support federation management and resources sharing within federated platforms. Also, more advanced mechanisms for Resource Bartering, SLA Management, Trust Calculation and Anomaly Detection were highlighted by discussing the sequence flows and integration in the overall symbIoTe ecosystem to deliver the full scope of components to facilitate L2-compliant platforms. Further, all provided internal and external component interfaces are documented listing the used communication pattern, offered endpoints and consumed payloads.

The documented work in this deliverable highlights the implementation efforts and shows the evolution of the symbIoTe framework and components from L1 compliance facilitating syntactic and semantic interoperability described in D2.4 [2] and D2.5 [3] towards fully compliant L2 features ensuring enterprise interoperability initially discussed and designed in D3.1 [7] and D3.2 [4].

Thus, this document focussed on the implementation-centric details and referenced code artefacts and augments the developer-specific documentation found in the respective Github Cloud repositories [6] with detailed workflow description and design considerations.

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] symbIoTe project Deliverable D3.2 - Resource Trading, Security and Federation Mechanisms; January 2018

[5] European Commission - 2018 reform of EU data protection rules; https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en ; accessed on 20/06/2018

[6] H-2020 symbIoTe Cloud Github Repository; https://github.com/symbiote-h2020/SymbioteCloud; accessed on 20/06/2018

[7] symbIoTe project Deliverable D3.1 - Basic Resource Trading Mechanisms and Access Scopes; December 2016

Abbreviations

AAM	Authentication & Authorization Manager
ABAC	Attribute Based Access Control
ALE	Anomaly Listener Extension
AMQP	Advanced Message Queuing Protocol
AP	Access Policy
API	Application Programming Interface
ARR	Access Resource Request
BTM	Bartering & Trading Manager
B&T	Bartering & Trading
CAD	Core Anomaly Detection
CI	Core Interface
CCI	Cloud Core Interface
CRUD	Create, Read, Update and Delete
CSR	Certificate Signing Request
FM	Federation Manager
GDPR	General Data Protection Regulation
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technology
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
JWS	JSON Web Signature
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
RDF	Resource Description Framework
REST	Representational State Transfer
RH	Registration Handler
PR	Platform Registry
SH	Security Handler
SLA	Service Level Agreement
SLAM	SLA Manager
SM	Subscription Manager
SPK	Subject Public Key
symbIoTe	Symbiosis of Smart Objects across IoT Environments
TLS	Transport Layer Security
TM	Trust Manager
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language