**Deliverable D2.5**

# Final System Architecture

| | |
|---|---|
| Editor: | Elias Tragos, FORTH |
| Deliverable nature: | Report (R) |
| Dissemination level: (Confidentiality) | Public (PU) |
| Contractual delivery date: | 31 August 2015 |
| Actual delivery date: | 4 September 2015 |
| Suggested readers: | Researchers, Application Developers, IERC members, Network Administrators |
| Version: | 1.0 |
| Total number of pages: | 157 |
| Keywords: | Internet of Things, functional architecture, security by design, privacy by design, network topology, domain model, Smart Cities. |

***Abstract***

This deliverable presents the final overall system architecture of the RERUM project. RERUM builds upon the target to define an architectural framework for the Internet of Things (IoT) based on the concepts of "security, privacy and reliability by design". The foundations of the RERUM architecture lie on the Architecture Reference Model (ARM) of IoT-A. However, concepts of other IoT-related projects have also been utilised and mapped to the project objectives. This deliverable also presents the final RERUM IoT domain model (initially described in D2.2 and D2.3), showing the key elements of the RERUM system and their interactions. Then, building on the initial version of the architecture presented in D2.3, the final high level functional architecture is being described, split into functional entities, giving also details on the interfaces between these entities. Next, the internal components of the functional entities are analysed in detail, together with message sequence charts for explaining some functionalities that the RERUM system provides. Finally, examples of deployment scenarios of a RERUM system are also presented. The key advantage of the RERUM architecture is that it spreads in a cross-layer manner, using both a service-oriented and a device-oriented approach, on the contrary with previous attempts, which had limited focus on the devices. RERUM argues that to design a secure and privacy preserving IoT architecture, the devices play a major role, and thus it defines many functional components that have to run on the devices. The concepts of this deliverable were built on the initial version of the architecture and on the work of the project's technical work packages.

**Disclaimer**



*The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 609094.*

 

**Impressum**

| | |
|---|---|
| Full project title | Reliable, resilient and secure IoT for smart city applications |
| Short project title | RERUM |
| Number and title of work-package | WP2 - The IoT Architectural Framework for Smart Cities |
| Number and title of task | T2.4 – Integrated Architecture and Refinement |
| Document title | Final System Architecture |
| Editor: Name, company | Elias Tragos, FORTH |
| Work-package leader: Name, company | Theodore Mouroutis, CYTA |
| Estimation of person months (PMs) spent on the Deliverable | |

(page intentionally left blank)

 

# Executive summary

A key challenge for IoT deployments in Smart City applications is ensuring their reliability. Here, reliability incorporates the issues of security, privacy, availability, robustness and flexibility to changing environmental conditions. As the IoT devices become more intelligent, autonomous, and seamlessly integrated in the everyday life of Smart Cities, new security and privacy threats arise as described in detail in RERUM [1] Deliverable D2.1 [2]. Without guarantees that the IoT devices are: (i) sensing correctly the environment, (ii) exchanging the information securely, and (iii) safeguarding private information, users are reluctant to adopt this new technology. Therefore, if these concerns are not addressed proactively at the early stages of a Smart City deployment they may act as a barrier to the adoption of this technology by users and businesses.

To address these issues, the FP7 project RERUM aims to enhance the trustworthiness of IoT technologies by designing an IoT architecture that adopts the concepts of "security, privacy and reliability by design". This deliverable provides the final draft of the RERUM architecture, leveraging on the work of three previous deliverables, namely D2.1 (Use Cases Definition and Threat Analysis) [2], D2.2 (System Requirements and Smart Object Model) [3] and D2.3 (System Architecture) [4]. An analysis of the results of these D2.1 and D2.2 was utilised for the definition of the initial versions of the RERUM Domain Model, the RERUM Functional Architecture and the Physical View of the RERUM Architecture as they were presented in D2.3 [4].

The RERUM Domain Model (presented in Section 2) identifies the key concepts that are relevant for the RERUM architecture. It is an updated and refined version of an initial model defined in D2.2 and D2.3. For the definition of the RERUM domain model, an analysis of the IoT-A [5] domain model was performed in D2.3. The IoT-A domain model is considered as the basic model that identifies the key concepts of the Internet of Things. RERUM acknowledges also the concepts of BUTLER [6] and iCore [7] for the use of Context in the overall framework. Thus, RERUM defines the domain model after taking the most relevant parts of IoT-A mapped on the objectives of RERUM, including also the notion of Context from BUTLER. The resulted domain model includes the concepts of Administrator, and Security/Privacy/Access policies, as well as the notion of User Consent. In this version of the RERUM Domain Model, the concept of Trustworthiness/Reputation is also added to show the importance of Trust for the RERUM system. All of these are considered as mandatory for defining a "secure and privacy preserving" IoT architecture.

The functional architecture of RERUM is also based on the Architectural Reference Model (ARM) of IoT-A [5]. However, RERUM follows not only a service-oriented approach like IoT-A and most IERC [8] projects, but also assumes that the Devices play a very important role into ensuring the security and the privacy of the architecture. Thus, RERUM follows also a device-oriented approach, implementing many Security, Privacy and Trust (SPT), as well as networking functional components on the devices (depending also on their technical capabilities). These advanced components will transform the existing devices into "RERUM Devices" (RDs), which is a term used within the RERUM project to identify the devices that include all or parts of the RERUM mechanisms and are able to communicate with the RERUM Middleware.

RERUM adopts the concept of Virtualization, abstracting the real world objects into virtual objects, for concealing their heterogeneity. In this respect, both RDs and Physical Entities are virtualised so that they can be discovered and accessed by applications. The virtualization and all the functionalities with respect to service management are included in the RERUM Middleware (MW). The MW is considered as a group of functional components that interact in order to allow the RDs to be discovered, to expose services, to form Federations in order to provide composed services and to communicate with each other in a virtualised way. The MW includes also functionalities for configuration of its components and monitoring their performance (i.e. creating alerts when an abnormal behaviour of some services, users, and devices is detected). The Federations of RDs can take place only between trusted and authenticated devices and only according to the privacy preferences and access policies that have been

 

set by the users. We have to note here that it is not the ultimate goal of the project to implement from scratch the MW functionalities, so we will try to re-use existing implementations from other projects, adapting them according to the needs of the project and mapping them to the RERUM domain model and the RERUM MW architecture accordingly. For the implementation of the RERUM system, the OpenIoT middleware has been selected.

Security and Privacy are integral parts of the RERUM architecture, influencing the decisions of many MW functionalities. For example, user consent is being considered in order to allow the users to have a "choice" regarding the data to be captured by the devices and the data to be sent to third parties. A privacy dashboard is also considered as a graphical interface for the user to declare his preferences and to control the disclosure of his information. Privacy enhancing technologies for anonymisation and pseudonymisation are considered to be applied when required by the respective policies. However, acknowledging the fact that for some reasons (e.g. accounting) the system could request the mapping of actions with real users, efficient de-pseudonymisation techniques are also considered in RERUM.

Regarding security by design, RERUM includes components for integrity generation/verification, data encryption, policy handling mechanisms (PDP, PEP and PRP), as well as secure storage of data and credentials. Key innovations within RERUM are the mechanisms for device to device authentication (allowing the distributed authentication between devices, without the need for centralised control) and the secure credential bootstrapping, so that whenever there is a need for changing/assigning credentials to devices, this will be done using a secure channel. Considering the specifics of the smart transportation use case that has special interest in terms of privacy, RERUM defines advanced and use case-tailored Privacy Enhancing Techniques for Geo-location so as to not disclose the location information of the user.

RERUM introduces a powerful trust and reputation management framework in the architecture, acknowledging the fact that the reputation of devices, services and users may affect the execution of security and privacy policies. The reputation of the devices and of the services they provide can be computed via the measurements/data that they send, comparing them with previous own data or with data of neighbouring devices, in order to identify abnormal operation (which lowers the reputation). Furthermore, the reputation of users can change in time according to their actions and their behaviour.

For the system reliability and availability, RERUM focuses on ensuring the seamless connectivity of the devices, adding in the architecture networking components that can run both on devices and on centralised physical components, like gateways or cluster heads. Clustering is a technique used for hierarchical management of the network of devices, contributing to the scalability of the overall system. A key innovation of the RERUM architecture regarding the connectivity of the devices is the CR Agent, which utilizes the advantages of the Cognitive Radio technology in order to allow the devices to select by themselves the suitable wireless spectrum bands according to various criteria, setting also customised interface parameters (bandwidth, modulation, and coding). That way, the devices will be always available to send data and this also contributes to mitigating wireless jamming attacks.

After defining and describing the functional view of the RERUM architecture, this deliverable presents an updated example deployment view (compared with the one presented in D2.3), describing the high level networking topology for various indoor and outdoor scenarios, as well as an updated mapping of functionalities on the RERUM Gateway and the RERUM Device. Of special interest is the final hybrid/hierarchical scenario described in Section 4, which shows how RERUM deployments can work in a hybrid indoor/outdoor deployments, maintaining the privacy of the indoor users' data.

The RERUM architecture aims to provide guidelines and solutions for designing IoT architectures that will apply security mechanisms through the whole lifecycle of the information, from data gathering to data exploitation and presentation to the user, as well as protecting the personal user data and avoiding their disclosure in third parties. RERUM aims to continue the established cooperation with other IoT projects within the IERC cluster, in order to provide guidance for adapting the RERUM solutions to the widest possible group of IERC projects for enhancing the security of IoT.

 

# List of authors

| Company | Author | Contribution |
|---|---|---|
| ATOS | Yildirim Huseyin Umut | Refinement of middleware functional components with special focus on stream processor. Development of firmware updater component. |
| | Darío Ruiz | Contribution to the description of the functional components for trust manager, policies management, SW components management, user authentication, Privacy Policy Checker and Attribute Need Reporter (section 3). |
| | Cristo Reyes<br>Javier García | Contributions on the SW Components Manager and Alert Processor Functional Components.<br>Description and design details of the OAP downloader, OAP builder and OAP deployer components. |
| | Michal Mardiak | Contribution to the review and refinement of the Middleware functional components. |
| SAG | Jorge Cuellar<br>Kai Fischer<br>Santiago Suppan<br>Ricarda Webber | Contribution to the Privacy components, message sequence charts and interfaces. Refinement of the security components and message sequence charts related with secure bootstrapping. |
| UNIVBRIS | George Oikonomou | Refinement of the RD Adaptor, descriptions and message charts. |
| LiU | Vangelis Angelakis<br>Johan Eriksson,<br>Tobias Edström | Revision and refinement of the Communication & Network Manager components and message sequence charts. Review and contribution to Chapter 4 regarding the deployment scenarios. |
| UNI PASSAU | Henrich C. Pöhls | Communication with the Integrity Generator / Verifier and message sequence charts |
| FORTH | Elias Tragos<br>George Stamatakis<br>Apostolos Traganitis | Overall editing and refinement of the document. Contribution to the executive summary, introduction and conclusions. Update of the domain model (section 2). Update of the overall functional architecture and the functional entities and description of the interfaces between the entities. Refinement of the RD Adaptor, the Communication & Network Manager and the respective message sequence charts and interfaces. Overall refinement of the text of the rest of the components. Contribution to Chapter 4 and the scenarios. Design of the hybrid deployment scenario. |
| Cyta | Athanasios Lioumpas<br>Theodore Mouroutis<br>Yiannis Stylianou<br>George Konios | Refinement of the Chapter 4 related with the RERUM deployment model and the respective data flows and message sequence charts. |
| SSRL | Septimiu Nechifor<br>George Moldovan<br>Bogdan Tarnauca | Contribution to the service model and the respective ontologies. Refinement of the Middleware components, their interactions and their internal interfaces. |

 

 

# Table of Contents

 

# Table of Figures



 

 

 

# Table Index

# Abbreviations

| | |
|---|---|
| AAA | Authentication, Authorization and Accounting |
| ADSL | Asymmetric digital subscriber line |
| AE | Augmented Entity |
| AMQP | Advanced Message Queuing Protocol |
| AP | Access Point |
| API | Application Programming Interface |
| ARM | Abstract Reference Model |
| AS | Application server |
| BPMN | Business Process Model and Notation |
| C&C | Command and Control |
| CBA | Credential Bootstrapping Authority |
| CBC | Credential Bootstrapping Client |
| CEP | Complex Event Processing |
| CH | Cluster Head |
| COAP | Constrained Application Protocol |
| CPU | Central Processing Unit |
| CR | Cognitive Radio |
| CRC | Cyclic Redundancy Check |
| CS | Compressed Sensing |
| CVO | Composite Virtual Object |
| DSP | Digital Signal Processor |
| DTLS | Datagram Transport Layer Security |
| EMF | Electromagnetic Field |
| EU | European Union |
| EUI | European University Institute |
| FCAPS | Fault, Configuration, Accounting (Administration), Performance and Security |
| FG | Functionality Group |
| FH | Federation Head |
| GE | Generic Enabler |
| GPRS | General Packet Radio Service |
| GPS | Global Positioning System |
| GSM | Global System for Mobile Communications |
| GSN | Global Sensor Networks |
| GUI | Graphical User Interface |

 

| | |
|---|---|
| GVO | Generic Virtual RERUM Object |
| GW | Gateway |
| HTTP | Hypertext Transfer Protocol |
| HU | Human User |
| IBE | identity Based Encryption |
| ICT | Information and Communications Technology |
| ICV | Integrity Check Value |
| IdA | Identity Agent |
| IERC | Internet of Things European Research Cluster |
| IETF | Internet Engineering Task Force |
| IF | Interface |
| IoT | Internet of Things |
| IP | Internet Protocol |
| ISO | International Organization for Standardization |
| IT | Internet Technology |
| JSON | JavaScript Object Notation |
| LAN | Local Area Network |
| LTE | Long-Term Evolution |
| MAC | Medium Access Control |
| MFSK | Multi-Frequency Shift Keying |
| MQAM | M-ary Quadrature Amplitude Modulation |
| MW | Middleware |
| NFC | Near field communication |
| NGSI | Next Generation Services Interface |
| OAP | Over the Air Programming |
| OLSR | Optimized Link State Routing Protocol |
| OMA | Open Mobile Alliance |
| OS | Operating System |
| OSI | Open Systems Interconnection |
| PDP | Policy Decision Point |
| PE | Physical Entity |
| PEP | Policy Enforcement Point |
| PET | Privacy Enhancement Technology |
| PKI | Public Key Infrastructure |
| PPL | Privacy Policy Language |
| PRP | Policy Retrieval Point |

 

| | |
|---|---|
| PRRS | Platform for Real time Reconfiguration of Security |
| PSTN | Public switched telephone network |
| RAM | Random Access Memory |
| RD | RERUM Device |
| RDF | Resource Description Framework |
| REST | Representational State Transfer |
| RF | Radio Frequency |
| RFID | Radio-Frequency Identification |
| RPL | Routing Protocol for Low-Power and Lossy Networks |
| RSS | Rich Site Summary |
| RWK | Real World Knowledge |
| SCE | Service Creation and provision Environment |
| SE | Sensing Engine |
| SLA | Service Level Agreement |
| SOAP | Simple Object Access Protocol |
| SSN | Semantic Sensor Network |
| SW | Software |
| SWRL | Semantic Web Rule Language |
| SWSL | Semantic Web Services Language |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| UC | Use Case |
| UDP | User Datagram Protocol |
| URL | Uniform Resource Locator |
| USB | Universal Serial Bus |
| USDL | Unified Service Description Language |
| VE | Virtual Entity |
| VO | Virtual Object |
| VOC | Volatile Organic Compound |
| VRD | Virtual RERUM Device |
| WAN | Wide-Area Network |
| WAVE | Wireless Access in Vehicular Environments |
| WSDL | Web Services Description Language |
| XACML | eXtensible Access Control Markup Language |
| XML | Extensible Markup Language |
| XPDL | XML Process Definition Language |

 

# 1 Introduction

## 1.1 Scope

This document presents the final version of the high-level architecture of the overall system developed by the EU-FP7-SMARTCITIES-2013 project RERUM [1]. It is a result of the work done in the project until now and presents the updated version of the Deliverable D2.3 [4] that included the initial version of the architecture. Its scope is to identify the core framework that will enable the project to provide IoT applications and the required architectural functional components to support the concepts of **security and privacy by design** with the assistance of a trust and reputation framework. In this respect, this deliverable builds upon the results of many previous project deliverables from all the workpackages of the project. The main differences of this deliverable compared with D2.3 is that it includes an updated version of the RERUM IoT domain model, with some corrections and the inclusion of the trustworthiness/reputation that is a core component of RERUM. Furthermore, it includes an updated version of the functional entities with a revision of the functional components, the inclusion of additional components that have been identified as required by the technical workpackages and the definition of the interfaces between the functional components of the system. Furthermore, examples of message sequence charts for various functionalities of the system are also included in this deliverable. Moreover, an update of the deployment model and the network interfaces is also included together with a revision of the example topologies for indoor, outdoor and hybrid installations. Finally, the deliverable also aims to **position RERUM in the IoT world**, identifying the relationships with the other projects of the IERC cluster [8] and the advances that RERUM brings in the IoT domain.

## 1.2 Intended Audience

This document provides the functional architectural framework, presenting the ideas and concepts of the project partners. The document is intended primarily for the project consortium, to be used as guidance for the implementation and the execution of the project trials. However, since the document presents the overview of the functional architecture of the project that deals with introducing security, privacy and trust in IoT-based Smart City systems, it is also intended for researchers and developers that are involved in IoT-related research or are implementing Smart City applications and want to make their systems more secure. Additionally, system administrators can also benefit from this deliverable utilizing the description of the deployment model for identifying the possible ways of installing a RERUM system for providing secure smart city applications. System designers could exploit the RERUM domain model for making their systems and their applications more secure and trustworthy. Furthermore, the document aims at other IoT related projects and IERC members that are willing to follow the activities of RERUM and are interested into identifying proposed solutions for enhancing the security, privacy and reliability of their systems. The deliverable attempts to map the proposed components to the IoT-A ARM [5], with the goal to complete it with additional concepts and ideas that will support the concept of “security and privacy by design”.

## 1.3 Methodology for deriving the Architecture

The methodology for deriving the RERUM architecture was presented in RERUM Deliverable D2.3, however in terms of easing out the reading of this document it is also briefly described in this deliverable.

The RERUM methodology is an extension of the IoT-A methodology for deriving architectures [5]. The steps of the RERUM methodology are:

- Define in detail the use cases under consideration within RERUM (provided by RERUM Deliverable D2.1 [2])
- Analyse the use cases in terms of security and privacy (provided by D2.1)
- Identify the security and privacy threats of these use cases and generalize them in the context of IoT applications (provided by D2.1)
- Identify the RERUM domain model (provided by RERUM Deliverable D2.2 [3] and D2.3 and finalised in this deliverable in Section 2)
- Define the functional and non-functional requirements for the RERUM system (provided by D2.2)
- Analyse the requirements with regards to project objectives and importance (provided by D2.3)
- Extract the functional components (provided initially by D2.3 and finalised in this deliverable in Section 3)
- Identify the interconnectivity of the components (provided initially by D2.3 and finalised in this deliverable in Section 3)
- Group the components and identify the functional entities of the system (provided initially by D2.3 and finalised in this deliverable in Section 3)
- Identify the place of each functional component at each physical component (provided initially by D2.3 and finalised in this deliverable in Section 4)
- Specify the interfaces between the components (in this deliverable in section 3)
- Gather input from the technical WPs and refine the architecture and the interfaces (this describes the refinement process done within the second year of the project and resulted in this deliverable)

Figure 1 shows the interactions of the various steps presented in the bullets above in order to derive the RERUM architectural views. The figure shows also the feedback loop with WP3, WP4 and WP5 (the WP number is denoted in the rectangular box next to the self-referencing arrow at some of the steps).

RERUM follows a bottom-up approach acknowledging the major role of the Smart City applications in the system design. The process for the architecture design started with a detailed definition of the use cases that have been considered within RERUM in D2.1.

D2.1 also included the analysis of the use cases for identifying the open security and privacy threats, including (i) the identification of the IT assets that have to be protected, (ii) the risk sources, (iii) the data flows and (iv) the attack examples on those assets during the respective data flows at each use case. The threats for Confidentiality, Integrity and Authenticity (C-I-A), as well as for Authentication, Authorization and Accounting (AAA) were also identified. Since the protection of the privacy of the user data is one of the major goals of the project, a detailed analysis of the Privacy threats in each of the use cases was also provided in D2.1.

The RERUM IoT domain model that was initially described in D2.2 and D2.3 and is being revised and updated to present the final RERUM Domain Model in Section 2 of this deliverable.

The functional and non-functional requirements of the RERUM system extracted after the analysis of the use cases were included in D2.2 and were split into six major categories, namely (i) non-functional, (ii) application, (iii) networking, (iv) RERUM Device Hardware/Software/Modelling, (v) Middleware/Virtualization and (vi) Security/Privacy requirements.

The analysis of the RERUM requirements and their mapping to the project objectives for each one of the requirement categories and according to their mapping in architectural layers was done in D2.3.

 

The goal of this analysis was to identify the needs of the RERUM architecture in terms of functional components and to assist the initial definition of those functional components, which was also delivered in D2.3.

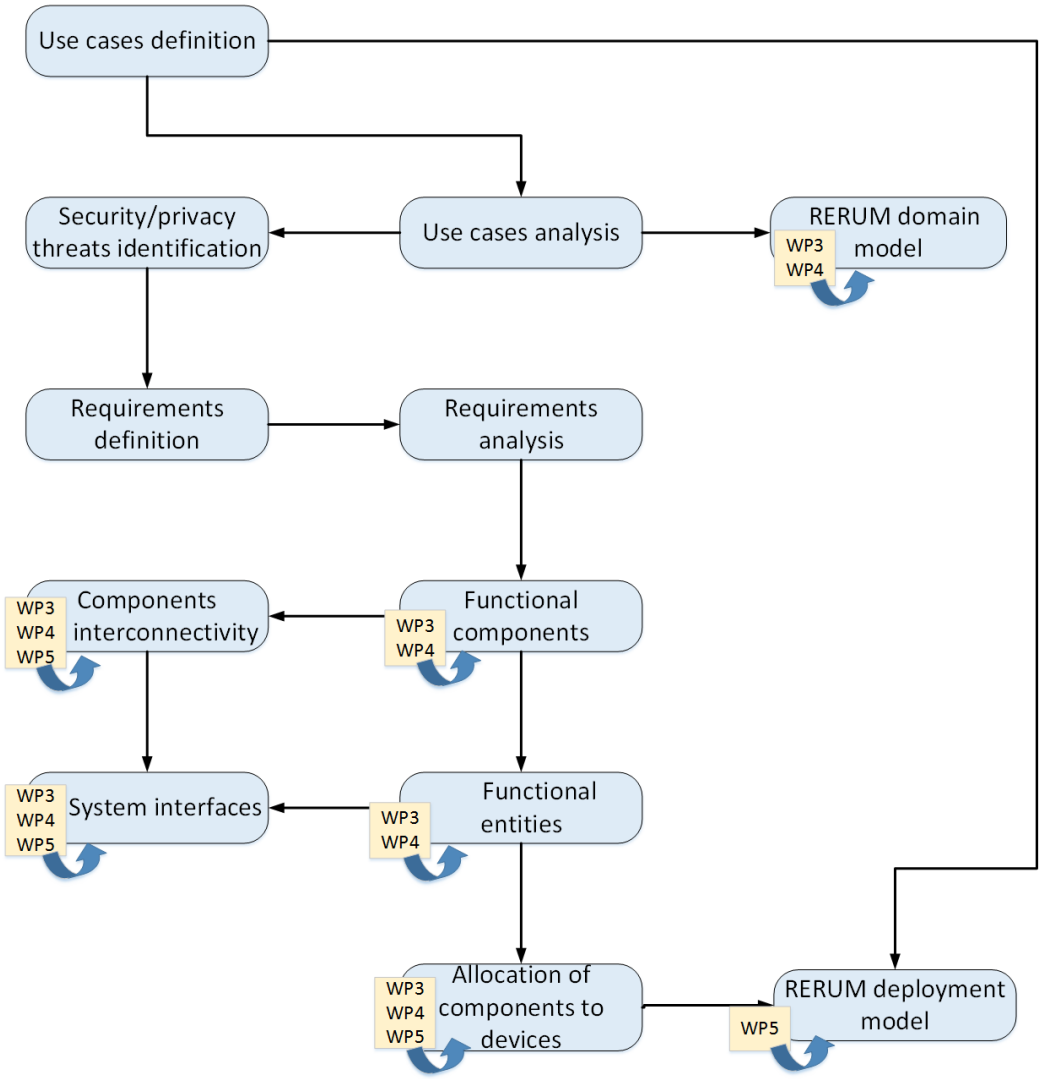


**Figure 1 – RERUM methodology for deriving the architecture**

The grouping of the functional components in functional entities and the definition of the interconnectivities between the components of each functional entity was done in D2.3 and is updated in this document in Section 3. The interfaces for all these interconnections are described in this section, according to the feedback loop from the technical work done in WP3, WP4 and WP5 within the second year of the project.

In parallel to the last steps for the interconnectivity of the various functional components, the physical devices/objects of the RERUM system have also been identified, together with an example description of the location of the functional components at each device/object in this document in Section 4.

## 1.4 Relation to other tasks and WPs

This deliverable is one of the most important outcomes of the RERUM project because in combination with D2.3 it is the one that provides the information about the functional view of the RERUM architecture and describes the details on the various functional components and the interfaces required for the implementation of the architecture. The relationship between this deliverable and the rest of the deliverables of the project can be seen in Figure 2. This deliverable uses as input the results of two previous deliverables of WP2 (namely D2.3 and D2.4) in order to finalise the design of the high level view of the RERUM system architecture. It also gathers input from the technical workpackages (WP3 and WP4), as well as from D5.2 regarding the implementation of the applications and the software of the RERUM devices and the RERUM Middleware regarding the updates of the description of the functional components and the interactions between them.

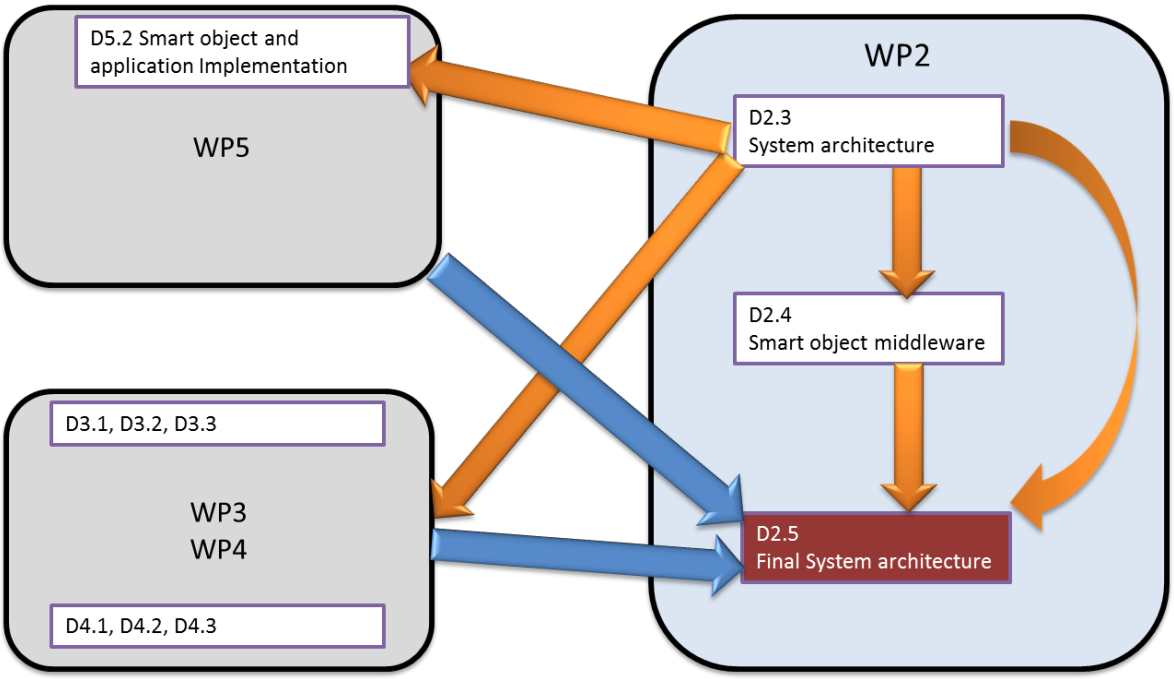


**Figure 2 - Relationship between D2.5 and other tasks/deliverables in RERUM**

## 1.5 Structure

The document is structured as follows:

- Section 2 presents the final version of the RERUM domain model as a follow-up of the detailed description that was given in D2.3. The new version of the domain model includes the addition of the trustworthiness/reputation box to cover the work done within RERUM for trust management, as well as minor corrections.
- Section 3 is the core of this document presenting the final version of the functional view of the RERUM architecture. This version includes several updates regarding the overall view of the functional architecture and of the functional entities. Additionally, here we also include a description of the functional interfaces between the functional entities and internally between the components of each functional entity. Furthermore, examples of message sequence charts for various functionalities are also included to give a more realistic and complete view of how

 

the RERUM functionalities can be implemented. Finally, an overview of the RERUM Information Model and of the RERUM Ontology are also given.

- Section 4 presents the final version of the RERUM physical deployment view, identifying the physical devices that are elements of a RERUM network, together with the network interfaces between them. This section gives also examples of deployments for indoor, outdoor and hybrid scenarios, together with some message sequence charts for various functionalities. It also provides an example of allocation of functional components to a RERUM Device and a RERUM Gateway.
- Section 5 concludes the document, discussing the main findings and key items of the RERUM architecture.

# 2 RERUM IoT domain model

This section will give an update of the IoT domain model of RERUM that were presented in D2.3 and was based on the domain models of IoT-A [9] and BUTLER [11]. This final version of the RERUM domain model includes small changes and additions compared with the initial version that was described in detail in D2.3. Thus, we do not repeat here the whole detailed step-by-step description of the domain model and the reader may refer to D2.3 for more details regarding the components of the model.

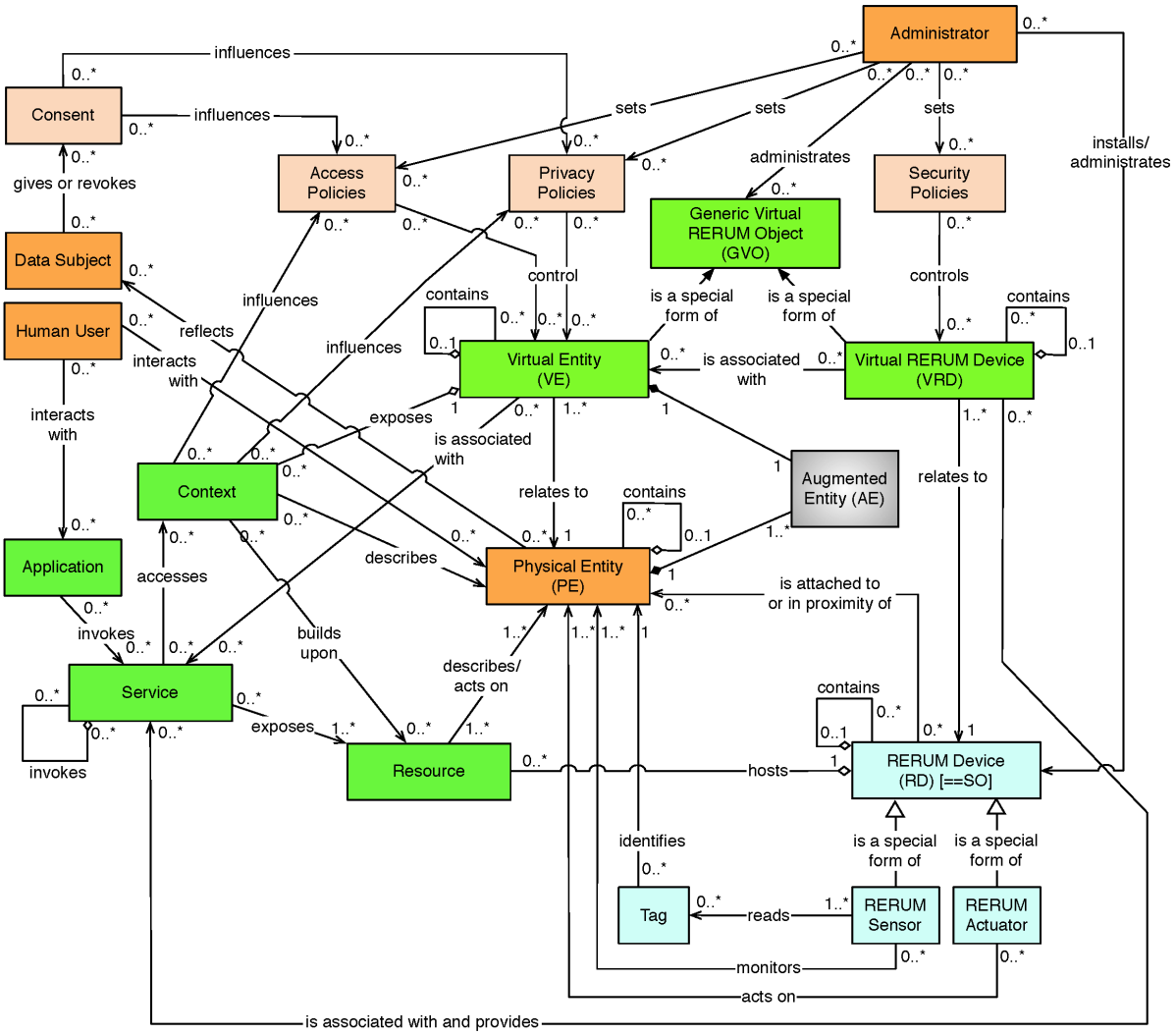


**Figure 3 – RERUM domain model in D2.3.**

The complete RERUM Domain model, as described in D2.3, is seen in Figure 3. As a summary of what was described in that deliverable, to ease the reading of the rest of the text, we can mention the following:

- **RERUM Devices (RDs)** are an enhanced version of ICT devices that have embedded sensors or actuators and have also the RERUM mechanisms on board. The RDs are either attached to or in the proximity of a **Physical Entity (PE)** monitoring or controlling it respectively. RDs can contain other RDs in a similar way that PEs can contain other PEs.
- The RDs are represented in the digital world by **Virtual RDs (VRDs)** that are used as a single point of reference for the services of one RD. Similarly, PEs are represented by **Virtual Entities (VEs)**. The VRDs are associated with VEs. RERUM also defined the term **Generic**

 

**Virtual Object (GVO)** to abstract both the VEs and the VRDs and simplify their representation in software classes.

- The RDs are hosting **Resources** that is software that either gathers data from the sensor(s) of the RD or controls the actuator(s). The Resources are either describing or acting upon PEs.
- The Resources are exposed by **Services**. The Services are associated with VEs and can invoke other Services performing federations or composed Services.
- The **Context** describes the PEs, exposes VEs and builds upon Resources.
- A **Human User** interacts with Applications that invoke Services, which in turn access the Context that identifies the situation regarding the PE/VE.
- There are three types of **policies** defined within RERUM: (i) the **Security Policies** that control the VRDs, (ii) the **Access Policies** and (iii) the **Privacy Policies** that control the access to VEs. The Context can also influence the Access and Privacy Policies i.e. in emergency situations.
- The **Administrator** is a "super user" that installs and manages the RDs and the Services/Resources that run on them (the link between the Administrator and the Resources/Services is not shown in the figure to avoid complicating it even more). The Administrator also sets all the Policies.
- For Privacy, RERUM has also defined the **Consent** that is given or revoked by a **Data Subject** and can also influence the Access and Privacy Policies. The Data Subject is the one that reflects the PE and has some private information that is being requested by an application.

After the work done in the technical workpackages and mainly WP3 during the second year of the project, it was identified that this domain model did not include any element for Trust. Since trustworthiness and reputation play a major role in the RERUM project and influence the decisions of the system it was agreed that they should be also included in the RERUM domain model. In line with this decision, a new element was added in the model, namely the "trustworthiness/reputation" element. The final version of the RERUM IoT domain model is depicted in Figure 4.

Within RERUM and especially in Task 3.4, it has been identified that Trust in the IoT domain should consider at least three different levels: (i) the device, (ii) the services/data and (iii) the user. The reputation of the devices should be considered/calculated in order to be able to identify devices that have malfunctioned or have been compromised and are sending faulty measurements affecting the system decisions or the services. The reputation of the services/data should be considered/calculated in order to avoid offering false measurements/data to the users and in order to avoid composing new services from untrusted simple services. Finally, the reputation of the users should be considered in order to be able to identify untrusted/malicious users and to not allow untrusted users accessing sensitive information or affecting system decisions.

In this respect, and as depicted in Figure 4, the trustworthiness/reputation (the element combines all three different levels described above for simplicity) builds upon the Resources that are hosted on the RERUM Devices (namely the measurements transmitted from the devices) and characterizes the VRDs, the Services and the Users. Finally, the trustworthiness/reputation can influence the Access and Privacy policies allowing (rejecting) access to trusted (untrusted) users/services/devices.

 

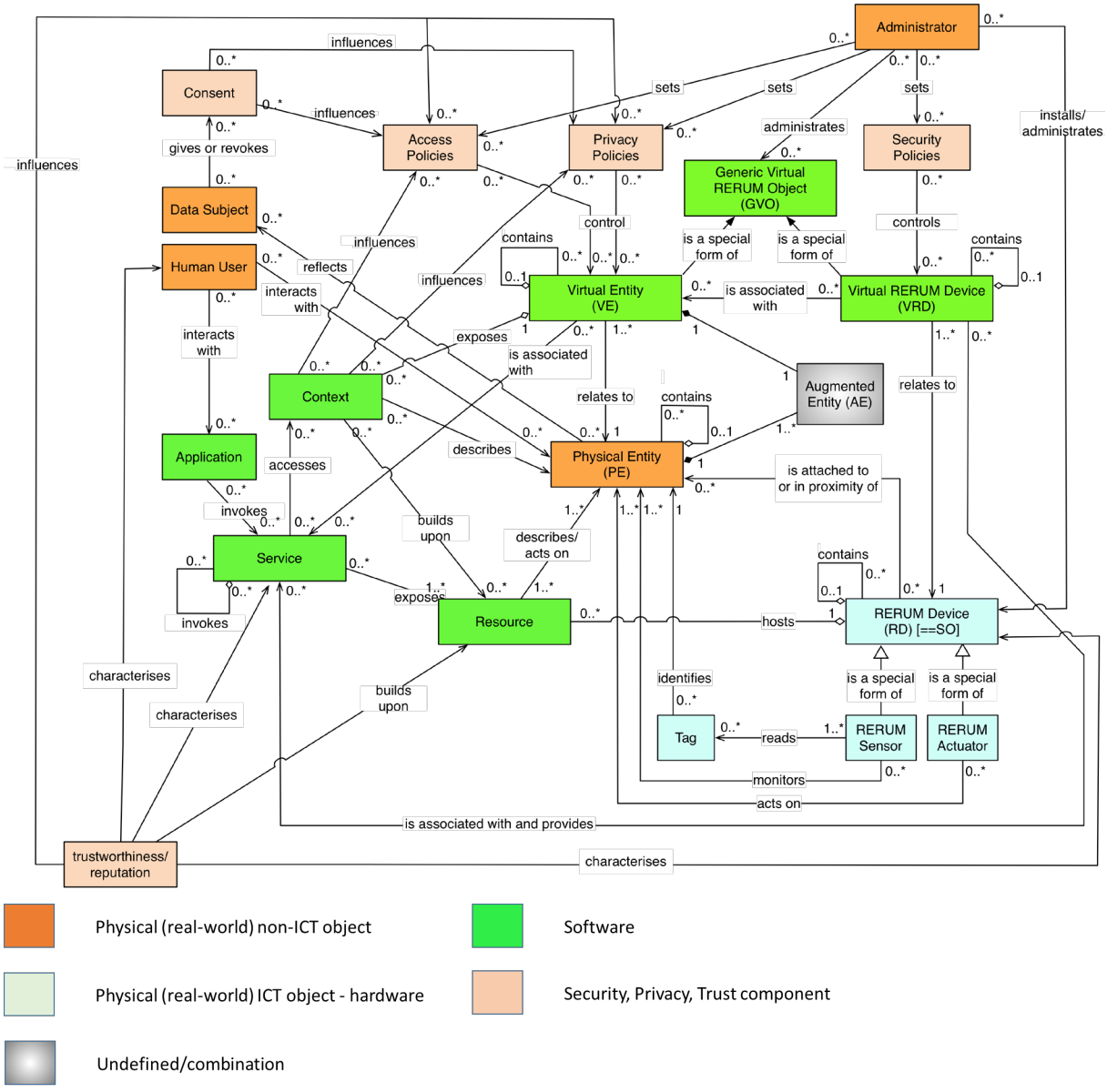


**Figure 4 – Final RERUM domain model**

 

# 3 Final architecture functional view

This section presents the final version of the high level functional view of the RERUM architecture that is a revised version of the initial version of the architecture described in D2.3. In order to have a complete reference for the RERUM architecture, the description of all components and functional entities are going to be included also in this deliverable, with the necessary updates in the text and the figures when necessary.

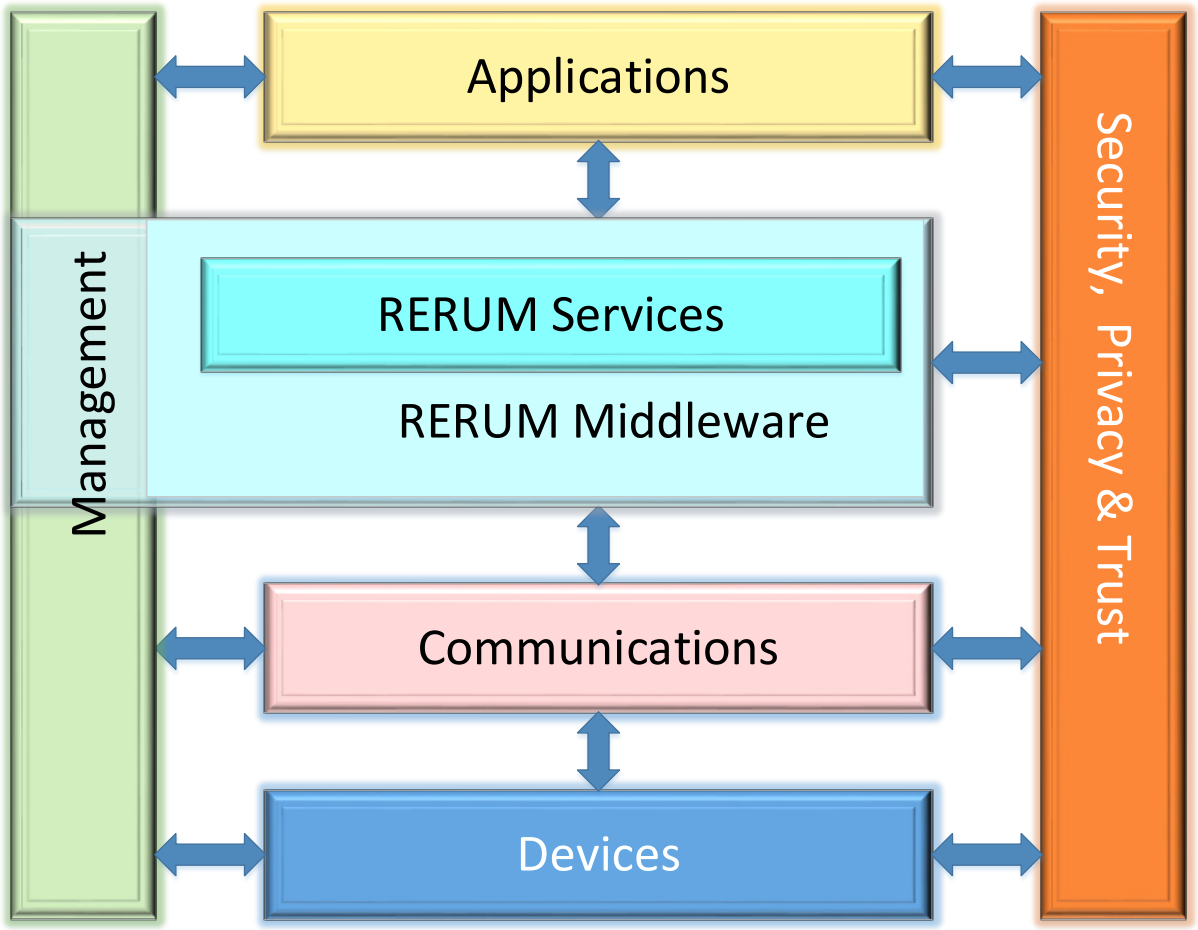


**Figure 5 - Architectural Layers of RERUM**

Considering the IoT-A's Functionality Groups [5], the RERUM architecture is divided in the following layers, (depicted in

Figure 5):

- The **Devices layer** that deals with the software that runs on the devices and includes the RERUM on-device mechanisms for security, privacy, trust and network reliability, as well as for energy efficiency and for handling the Resources that are hosted on the devices.
- The **Communications layer** that handles the end-to-end communications from the devices to the Middleware and to other devices. This layer also deals with the communication and networking of the devices in order to optimise their network connectivity and ensure their high availability and network reliability.
- The **RERUM Middleware (MW) layer** deals with (i) the virtualization of the RERUM Devices (RDs) into Virtual RERUM Devices (VRDs) and the virtualization of the Physical Entities to

Virtual Entities (VEs),, (ii) the interconnectivity of the VRDs from the services point of view, (iii) the registration and discovery of the VRDs and their associations with the Virtual Entities (VEs), and (iv) the grouping of RDs into federations for providing a specific Service. Within RERUM, the MW is also considered to include management mechanisms, mostly for configuration, monitoring and alerting purposes and for ensuring Quality of Service (QoS). These functionalities are very closely related with the rest of the MW components (something that will be evident in the following sections), thus they are included within both the MW and the management layers forcing an intersection of the two as depicted in the figure. A remark here is that the RERUM Middleware also integrates the RERUM Services layer described below.

- The **RERUM Services layer** enables the interactions between applications, services and real world objects (physical entities or devices). It handles the service requests from the applications and forwards them to the appropriate VRDs. According to IoT-A, IoT Services can be (i) Resource-level Services exposing Resources on the Devices, (ii) VE Services that access attributes of the VEs or (iii) Integrated Services that are Service compositions of the previous two types. RERUM uses the term RERUM Services to indicate both the Resource-level Services of the IoT-A, as well as the Services provided by a Federation of VRDs that includes also compositions. RERUM utilizes the same term for VE Services like IoT-A.
- The **Security, Privacy and Trust (SPT) cross-layer functionality group** handles all types of security, privacy and trust related functionalities and can be considered as the core of the RERUM architectural framework. More information will be given in Section 3.10.
- The **Management cross-layer functionality group** handles all types of management and configuration functionalities, either for the devices or the Middleware software components. Management includes basic functionalities for: (i) device management, (ii) identity and certificates management, (iii) configuration and management of the software components of the devices, as well as (iv) for management of the networking components.
- The **Application layer** facilitates the deployment and monitoring of IoT applications, i.e. how to build applications, how to request IoT Services, how to present the results of the applications mainly in terms of graphical presentations, etc. In general, RERUM provides Application independent functionalities and it is not the goal of the project to provide new functionalities for applications. This layer is included in the RERUM architecture only for the sake of completeness. RERUM will work in the application layer only in the context of the trials, where draft smart city applications will be developed to showcase the functionalities and the performance of the most important parts of the RERUM system.

Figure 6 shows the high level conceptual architectural view of RERUM. As we can see, the figure is split horizontally in two parts, the lower of which denotes the **Real world**, namely the physical devices (i.e. sensors, actuators, mobile phones), and the physical entities (i.e. the bedroom, the house, a car, the road), while the upper part denotes the **Virtual world** that includes the virtual objects that are representing the physical objects, their capabilities and their attributes.

The **Devices** are a very important element in the RERUM architecture. In contrast to most IoT projects (i.e. IoT-A, iCore [10], IoT.est [12], etc.) that are abstracting functionalities away from the devices and not considering the latter in their architectures, RERUM aims to include on the Devices as much intelligence and functionality as the capabilities of the devices allow. This will transform the existing devices into **RERUM Devices (RDs)**. Then, these RDs will be abstracted into **Virtual RDs (VRDs)** (their information and capabilities are stored in a GVO Registry) in order to create a uniform way of accessing and discovering them from the applications point of view, concealing their heterogeneity and their networking technology. The VRDs are exposing specified services that use a common protocol (i.e. JSON, WSDL, SOAP) and enable the seamless transfer of data from the devices to the applications. The Physical Entities are being abstracted in the virtual world into Virtual Entities, as per IoT-A's definitions.

 

A group of VRDs can form a **Federation** and cooperate for providing composed services to the applications as needed. The Federations can be: (i) simple service compositions that aim to i.e. calculate the average energy consumption measured by several devices at home and send this to the user, or (ii) more complex service compositions with the inclusion of some business process logic in terms of rules like, e.g. when the indoor temperature is more than 28 degrees and the outdoor temperature is more than 30 degrees, then close the windows/doors and turn the air condition to on to maintain a temperature at 25 degrees. The pink arrows in Figure 6 show the control plane exchanges that are needed for creating a federation of VRDs. Both VRDs and VRD Federations expose services that can be invoked by the applications. Of course, there are also functionalities for security, privacy, trust and management that are applied in both worlds, but these will be described in more detail below.

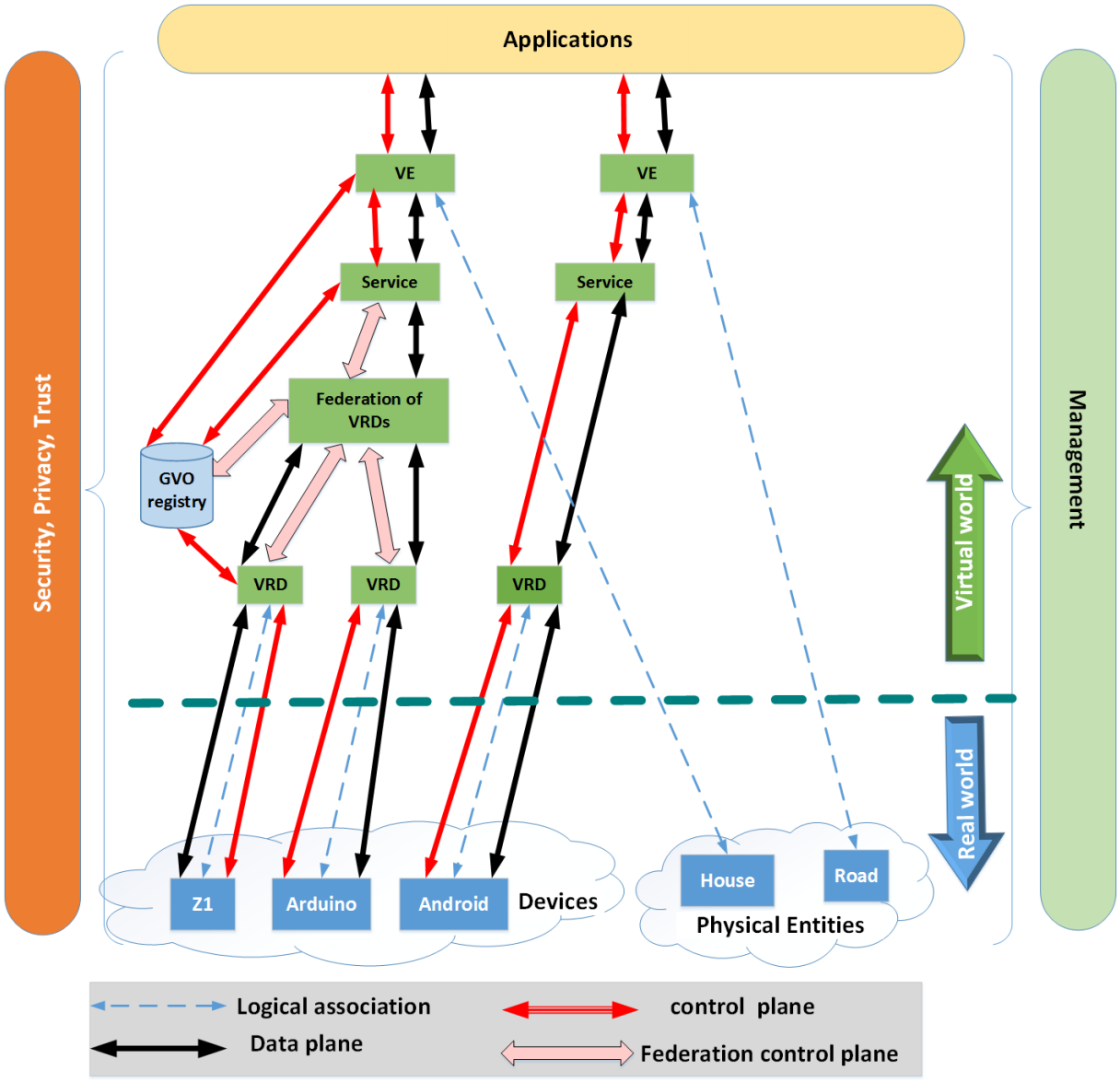


**Figure 6 - Overall RERUM conceptual architecture view**

The overall architecture of RERUM is a revised version of the initial architecture that was described in D2.3 and which was extracted using the results of the requirement analysis, grouping the functional components in Functional Entities and having also the conceptual view in mind.

Figure 7 shows the **final high level architecture of RERUM** and the interconnectivity between the key elements of RERUM, namely the RERUM Devices, the RERUM Gateway, the Virtual RERUM Devices, the Federations, the Services, the overall Middleware and the Applications. The Physical Entities are not shown in this figure to avoid complicating it even more; however, the VEs are included as they are key elements of the RERUM architecture. On the lowest part of the figure, there are the Devices, on which RERUM implements the **RD Adaptor** (see Section 3.3) that is the Functional Entity that allows their abstraction and their representation into Virtual RDs (VRDs).

As it can be seen, RERUM follows the work of IoT-A and acknowledges the existence of two different types of devices in the IoT world, namely the "**constrained devices**" and the "**unconstrained devices**". The first category is the majority of the devices in the IoT world and includes the devices that have significant constraints in terms of processing power, storage, memory and battery. These are basically the standard sensor platforms used currently in a plethora of IoT and sensor network applications, e.g. the Zolertia Z1[1], the Libelium WaspMote[2], the Openmote[3], the Arduino[4], etc. The "unconstrained devices" are devices that are quite powerful and are used for IoT applications, e.g. standard smart phones, laptops, wi-fi cameras, etc. The significant difference between the two categories of the devices is that for interconnecting the constrained devices with the RERUM system the existence of the RERUM Gateway is mandatory, because it is the one entity that can run the necessary networking mechanisms (i.e. protocol translation) and also some security and privacy enhancing functionalities, as well as the necessary mechanisms to connect with the MW. The unconstrained devices can also be connected with a RERUM Gateway when they are static at a specific area but they could be also moving around the city area. In the latter case, the devices can't be connected with the RERUM system through gateways, so they are directly connected to the MW via a Wide-area connection, like 3G or 4G. These devices, since they are unconstrained they can run complex security/privacy/networking mechanisms, as well as the mechanisms to connect with the MW.

The VRDs are handled by the Functional Entity named **GVO Manager** (see Section 3.6). The Federations are handled by the Functional Entity named **Federation Manager** (see Section 3.7). In this figure we can see more details regarding the Federation control plane exchanges. As in the previous figure, the pink arrows in Figure 7 show the control plane exchanges that are needed in order to: (i) allow a Service to request a federation, (ii) allow the Federation manager to identify the VRDs (querying the GVO registry) that will be federated by querying the Registry, (iii) allow the Federation Manager to access the VRDs to federate them and (iv) allow the Federation Manager to send the needed business/composition logic/rules to the **Data and Context manager** (see Section 3.9) to do the respective processing of the data streams coming from the RDs that correspond to the federated VRDs. As mentioned before, both VRDs and Federations expose services (handled by the **Service Manager** described in Section 3.5) that can be invoked by the applications. The Data and Context Manager is the Functional Entity responsible for handling the data sent by the sensors, according to the QoS requirements of the applications and the context that is extracted from both applications and data.

Figure 7 also depicts at a high level the components that comprise the **RERUM Middleware (MW)**. The MW is considered as the layer that groups the Functional Entities that interact for allowing the RERUM Devices to be discovered, to expose services, to form federations and to communicate with each other in a virtualized and seamless way. The MW includes also functionalities for configuration of its components and monitoring their performance (i.e. creating alerts when there is abnormal operation of some Services, users, and RDs). These functionalities are handled by the **Configuration and**

[1] http://zolertia.io/z1

[2] www.libelium.com/products/waspmote

[3] www.openmote.com

[4] https://www.arduino.cc

 

**Monitoring Manager** (see Section 3.8). The MW functionality is usually realized in a distributed way (like in OpenIoT [14]), and there is some degree of flexibility on the decision where to implement the required functionalities and, to some extent, also in which order the MW functionality is to be called.

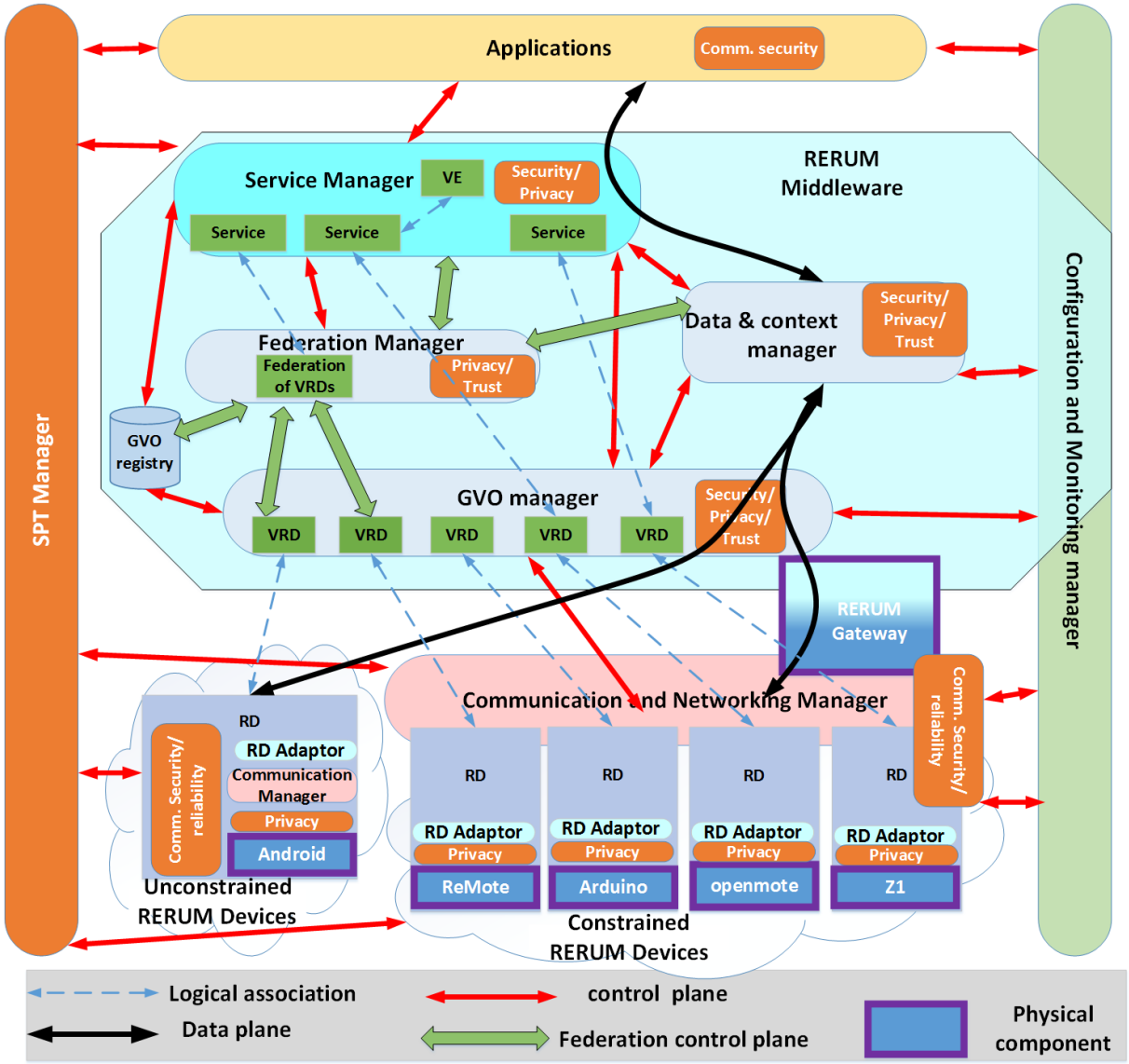


**Figure 7 – RERUM architecture**

Parts of the RERUM functionality to connect the devices with the MW (namely the **RD Adaptor**) are implemented on the RERUM Devices, in order to hide the proprietary interfaces and offer standard interfaces to the applications. Parts of the RERUM MW are implemented on the RERUM Gateway, for providing functionalities to a local network of (mainly constrained) RERUM Devices and allow them to be connected to this particular Gateway and the rest of the RERUM system. This scenario is depicted in Figure 7 with the RERUM Gateway that spans from the Communication and Networking manager until the Middleware. However, the MW functionality may also be implemented in one or several servers "in the Internet" according to the design choice of the Administrator of the RERUM platform. We refer to that latter case by saying the bulk of the MW is implemented in the "cloud"; the precise location does not change our functional model. The RERUM Gateway is also responsible for the communication and the networking of the various RDs that it connects, so it includes the **Communication and Networking manager** (see Section 3.4).

In general, the role of the RERUM MW is: (i) to overcome the heterogeneity of RDs allowing their communication in the IoT world via web services, (ii) to align strongly typed VRDs via a managed repository that has specific VRD templates, (iii) to instantiate each GVO based on a semantic template indicating basic information (service address, call parameters, etc.), (iv) to make each instantiated GVO visible via an indexing/registry mechanism, (v) to use such indexing mechanism for RD/Service discovery and for executing various operations upon RDs/Services (i.e. update, delete, etc.), as well as for generating and executing federations, and (vi) to allow well formatted data collection from RDs in order to deliver well-formed events for application-level stream processing.

RERUM's ultimate goal is to enhance the security, privacy and trust of the IoT architectures and develop a framework under the concept of "security and privacy by design". The Functional Entity responsible for this is the **SPT Manager** (see Section 3.10). In an attempt to depict this added value of RERUM, we include the boxes of privacy, security and trust mechanisms in many layers/entities as depicted in Figure 7. As mentioned above, RERUM aims to include more intelligence on the devices, so that they can decide, based on policies, which level and mechanisms of privacy and security should be applied for each service request. In order to do so, RERUM aims to implement layers of privacy and security on both the constrained and the unconstrained devices as it is shown in Figure 7. The privacy layer will include some key Privacy Enhancing Technologies as a first step towards ensuring Privacy (to be described in Section 3.10.2). Furthermore, a layer of **Communication reliability** is also included in the links through (among others) opportunistic communication mechanisms and Cognitive Radio inspired networking technologies. Communication Security mechanisms are also included both on the RDs and on the Communication layer, i.e. to ensure that even on constrained devices the messages exchanged will be encrypted when needed to protect the data. As it can be seen in Figure 7, the Unconstrained Devices include the Communication Security/Reliability mechanisms, while the constrained devices have only parts of that stack leaving some complex mechanisms to be run on the RERUM Gateway. Several Security, Privacy and Trust mechanisms are also applied on the VRDs (i.e. for Access control, Authorization, Reputation, etc.), as well as on the Data and Context management (i.e. for avoiding transferring private information to the applications, or for identifying the reputation of (V)RDs that are sending false information. Similarly, only trusted VRDs should be federated and the information exchanged between federated VRDs should also be privacy protected.

When the MW is implemented on the cloud, we assume the communication within its components is secure, meaning that the MW is integrity and confidentiality protected and has secure access control mechanisms that implement the RERUM policies. This is of course a strong assumption, but it is not the goal of RERUM to advance the security of the Internet or of the "cloud". On the contrary, **privacy** methods are heavily involved in the MW functionalities. Although the MW itself has **no privacy requirements** on its own, the data that are being processed and stored in the MW cloud should be privacy-protected. In other words, RERUM acknowledges that **privacy methods in the cloud are also needed** and there are currently no state of the art solutions that we can reuse in this area. To avoid any leak of information for any reason (insider attack or espionage), private data should be anonymised or pseudonymized. RERUM will also provide solutions on this.

## 3.1 RERUM Functional entities

In this subsection we describe in more detail the RERUM Functional Entities (depicted in Figure 8). The light blue borders show the functional entities that comprise the overall RERUM Middleware, and the dotted light blue border shows the embedded component that enables the RD to be connected to the Middleware. The RERUM functional entities are (the colour coding of the entity boxes corresponds to the colours of the respective layers of

Figure 5):

 

- The **RD adaptor** functions as a bridge between the physical hardware device and the RERUM system. It includes functionalities for discovering the device Resources in terms of sensing and actuating elements and for exposing services to the external world for accessing the RD's resources.
- The **Communication and Network Manager** is responsible for managing the networking and the communications of the RDs with each other and with the GWs.
- The **Service Manager** is responsible for discovering and handling services related to VEs and RERUM Services (or as IoT-A calls them IoT Services).
- The **GVO Manager** is responsible for registering, managing and accessing GVOs, as well as for discovering VRDs.
- The **Federation Manager** is responsible for discovering, managing and executing federations of RDs in order to perform specific high level services that correspond to VE Services.
- The **Configuration & Monitoring Manager** is responsible for the configuration of the RDs and the MW components, as well as for the monitoring of the system resources (i.e. hardware and network resources), of the provided QoS, the logging of events and the raising of alerts in case of anomalous situations.
- The **Data and Context Manager** is responsible for collecting and managing the data that are gathered by the RDs, as well as for extracting context out of these data.
- The **SPT Manager** is a cross layer entity that spans from the devices to the applications ensuring the overall security, privacy and trust within the system.

In Figure 8 we also see the logical interconnectivity and the logical interfaces between the functional entities of RERUM. The interfaces are briefly defined below and in more detail in Table 1:

- **IF_apps**: this interface connects the RERUM system with the applications and is used for requesting a Service from the RERUM infrastructure. This interface is exposed by the Service Manager since it is the main functional entity that handles incoming service requests.
- **IF_servdata**: this interface is exposed by the Service Manager and is accessed by the Data & Context Manager to forward the data that it gathers from the devices (after the required processing), which then will be sent to the applications.
- **IF_context**: this interface is exposed by the Data & Context Manager for allowing the other functional entities to access the contextual information of a specific VE or service.
- **IF_federation**: this interface is exposed by the Federation Manager and is consumed by the Service Manager for requesting a federation of VRDs when there is not a single VRD that exposes a service that can meet the demands of the user application and a composition of RERUM Services is required.
- **IF_GVO**: this interface is exposed by the GVO manager and is used for searching, discovering, managing and accessing the GVOs.
- **IF_regVRD**: this interface is exposed by the GVO Manager in order to accept incoming registration requests from new RDs that are being installed in the RERUM system.
- **IF_RD:** this interface is exposed by the RD Adaptor and allows the devices to be accessed by the RERUM system and get the measurements or send commands.
- **IF_CN:** this interface is exposed by the Communication & Network Manager for allowing the Configuration & Monitoring Manager to trigger the communication and network management

mechanisms when there is low performance in the network, when a device is not responding correctly or when there is congestion in the system.

- **IF_RCN:** this interface is exposed by the Communication & Network Manager for allowing the RD Adaptor to send information regarding the required transmission, in order to allow the Communication & Networking Manager to make the necessary preparations on the networking components (i.e. configure the network interfaces) for handling the transmission.
- **IF_servQOS**: this interface is exposed by the Configuration & Monitoring Manager for providing the QoS requirements of services that the VRDs are exposing and those that the users are requesting.
- **IF_authoriz:** this interface is exposed by the SPT Manager and is used for accessing the functionalities for authentication, authorization and access control, as well as for accessing and managing the respective policies and certificates.
- **IF_secpriv:** this interface is exposed by the SPT Manager for requesting and executing security and privacy mechanisms.
- **IF_trust:** this interface is exposed by the SPT Manager for managing the trust in the RERUM system for the users, the services and the RDs.

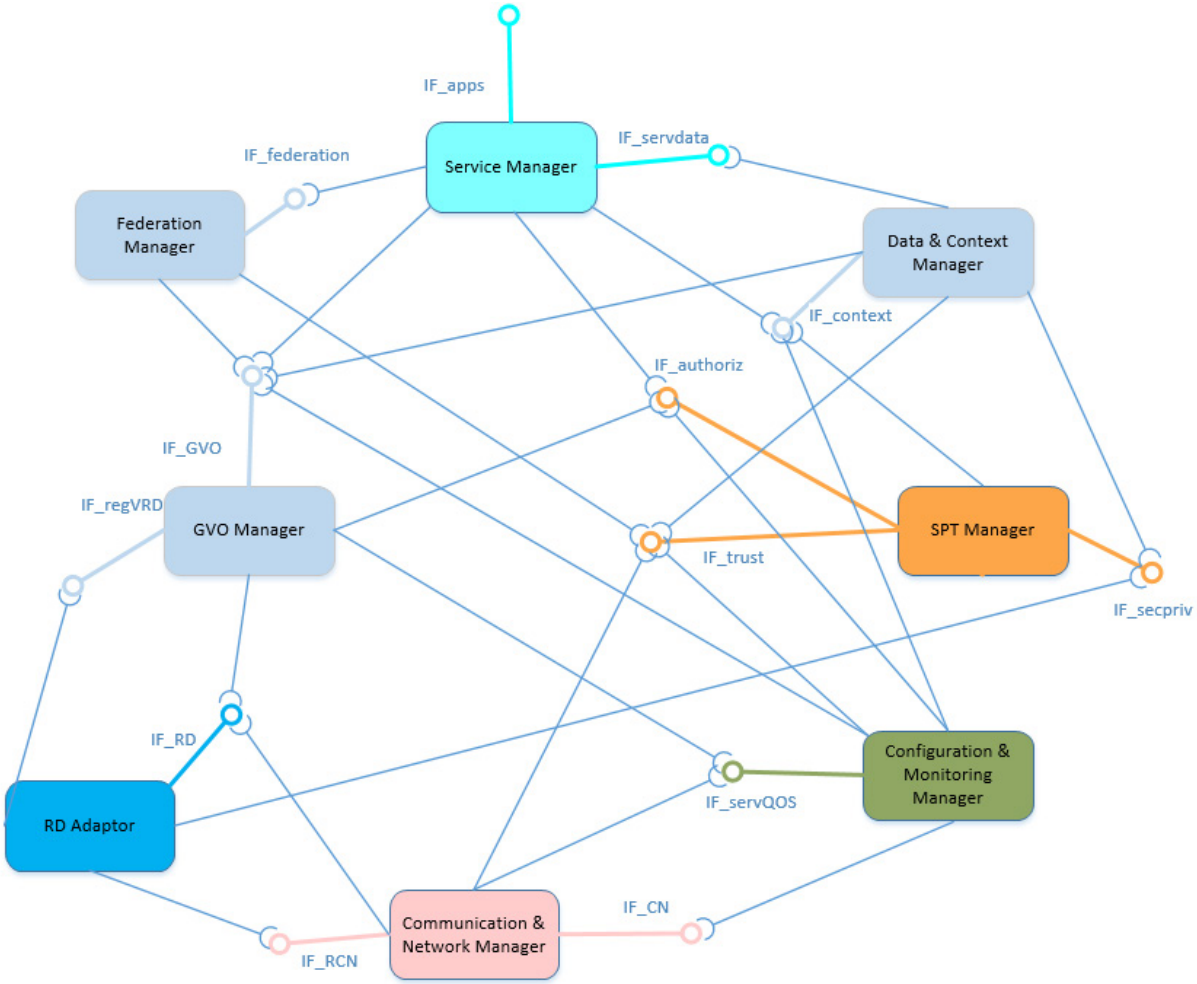


**Figure 8 – RERUM architectural Functional Entities and interfaces**

 

**Table 1: Description of interfaces between the Functional Entities.**

| Interface name | Connected Components | Connection Title | Exchanged data description |
|---|---|---|---|
| IF_apps | Service Manager, User | Service Request and Service Response | REST Service call by the User to the Service Manager, describing the virtual entity and the type of data he needs to receive. The service requests should include all information required, i.e. the type of service, the format of the data, as well as the user credentials for checking the authentication/authorization of the user that uses this application. The service manager responds with (i) the data or (ii) with a "not found" if there is no such service or (iii) with a false authentication message. |
| IF_servdata | Service Manager, Data & Context Manager | Data response | The measurements are passed to the Service Manager in the correct format, after being processed and scheduled according to their priority. |
| IF_context | Data & Context Manager, Service Manager | Send Context and QoS requirements | The Service's contextual information and QoS requirements are passed to the Data & Context Manager, so that it knows how to process the data stream that comes from the devices (e.g. with respect to the priority). |
| | Data & Context Manager, SPT Manager | Request Context of a VE/Service | SPT Manager requests contextual information to identify the appropriate privacy/access policies and make necessary changes according to the context of the VE (i.e. in emergency situations). |
| | Data & Context Manager, Configuration &Monitoring Manager | Send QoS requirements | The Configuration and Monitoring manager gets the QoS requirements of the Service in order to identify the appropriate network resources that are needed and then pass the info to the Communication Manager. |
| IF_federation | Federation Manager, Service Manager | Federation Request | Federation request is passed to the Federation Manager when there is no single RERUM Service that can provide the meet the application requirements for data. |
| IF_GVO | GVO Manager, Service Manager | Discover GVOs | Service Manager sends query to discover the VRDs and the VEs that are related with a specific application request. |

| | | | |
|---|---|---|---|
| | GVO Manager, Federation Manager | Discover VRDs | Federation Manager sends query for discovering the VRDs that need to be federated for providing a composed service. |
| | GVO Manager, Configuration & Monitoring Manager | Discover VRDs | Configuration & Monitoring Manager sends request to discover the VRDs that need to be (re-)configured to address any security-related issues or need to be monitored for their performance, creating alerts when needed. |
| | GVO Manager, Data & Context Manager | Get data | Data & Context Manager uses this interface to get the measurements from the VRDs. |
| IF_regVRD | GVO Manager, RD Adaptor | Register RD | RD Adaptor sends the required registration request (with the required information, e.g. id, services, etc.) to the GVO Manager in order to register a new RD to the system and the GVO Registry. |
| IF_RD | RD Adaptor, GVO Manager | Get data | The GVO Manager requests and gets the measurements from the appropriate RD. |
| | RD Adaptor, Communication & Network Manager | Manage RD | The Communication & Network Manager uses this interface to send networking and communication configuration commands to the RD in case of centralised networking mechanisms. |
| IF_CN | Communication & Network Manager, Configuration & Monitoring Manager | Execute Networking Mechanisms | When the Monitoring Manager identifies low performance in the network, the Communication & Network Manager receives an alert and the required information for executing specific mechanisms to improve the performance. |
| IF_RCN | Communication & Network Manager, RD Adaptor | Prepare transmission | The RD Adaptor sends the required information to the parts of the Communication & Network Manager that run on the RD to prepare the networking components for executing the specific transmission. |
| IF_servQoS | Configuration & Monitoring Manager, Communication & Network Manager | Get QoS requirements | Communication & Network Manager sends requests to get the QoS requirements of the service(s) that are provided by a VRD to ensure the QoS when taking networking or communication decisions, e.g. for spectrum management or routing. |

 

| | | | |
|---|---|---|---|
| | Configuration & Monitoring Manager, GVO Manager | Register QoS capabilities, get QoS requirements | GVO Manager registers the QoS capabilities of the VRD and get the QoS requirements for the services that are requested by the VRDs. |
| IF_authoriz | SPT Manager, Service Manager | Service request authentication and access control | Service Manager sends the service request information to request the authentication of the application/user, as well as perform access control to ensure that the user is authorized to access the service and get the data. |
| | SPT Manager, GVO Manager | Evaluate access/ security policies | GVO Manager requests evaluation of the access and security policies for the specific VRD that is accessed by an application and assesses the appropriate security mechanisms to be applied. |
| | SPT Manager, Configuration & Monitoring Manager | Secure configuration of RDs | Configuration & Monitoring Manager requests the credentials/firmware/etc. for the secure (re-)configuration of the devices and the installation of the most appropriate secure firmware. |
| IF_secpriv | SPT Manager, RD Adaptor | Encrypt/ Decrypt data, apply PETs, Sign/Verify data, etc. | The RD Adaptor requests from the on-device SPT Manager components the execution of appropriate security/privacy mechanisms, e.g. encryption, integrity protection, privacy protection, data minimization, etc. |
| | SPT Manager, Data & Context Manager | Encrypt/ Decrypt data, apply PETs, Sign/Verify data, etc. | The Data & Context Manager requests the execution of advanced security/privacy enhancing mechanism at the MW before forwarding the data to the application. |
| IF_trust | SPT Manager, Federation Manager | Request VRD reputation | Federation Manager requests information regarding the reputation of the VRDs that can be federated in order to ensure that only trusted VRDs can participate to federations that relate with sensitive information. |
| | SPT Manager, Communication & Network Manager | Request VRD reputation | Communication & Network Manager requests the reputation of neighbour VRDs to ensure that only network measurements from trusted VRDs will be considered when taking networking decisions. |
| | SPT Manager, Configuration & Monitoring Manager | Provide statistics | Configuration & Monitoring Manager provides monitoring statistics to the SPT Manager for assisting the evaluation of the reputation of VRDs, Services and Users. |

 

| | | | |
|---|---|---|---|
| | SPT Manager, Data & Context Manager | Send alerts | Data & Context Manager gets alerts (extracted by the measurements of the devices) regarding the reputation of Services/VRDs to assist on the calculation of the reputation by the SPT Manager. |

## 3.2 RERUM relation to IERC

Before we give more details for the internal components of the Functional Entities, we will do a mapping of the RERUM functional entities on the IoT-A ARM (depicted in Figure 9). As mentioned before and can be seen in the figure, RERUM does not conceptually deal with the Application layer. In the IoT Business Process Management we only include the "business logic/rules" part that is needed in complex federations for composing the services exposed by the VRDs. There are several Functional Entities that are Cross-layer, i.e. the Data and Context Manager, the Service Manager, the Federation Manager, the Communication and Network Manager, the QoS Manager and the Configuration and Monitoring Manager. The SPT manager has an important part on the Devices, because it needs to perform various communication security related functionalities (i.e. encryption, privacy enhancing technologies, etc.). Similarly, the Communication and Network Manager also partly runs on the Devices in order to handle their network connectivity and ensure their high availability.

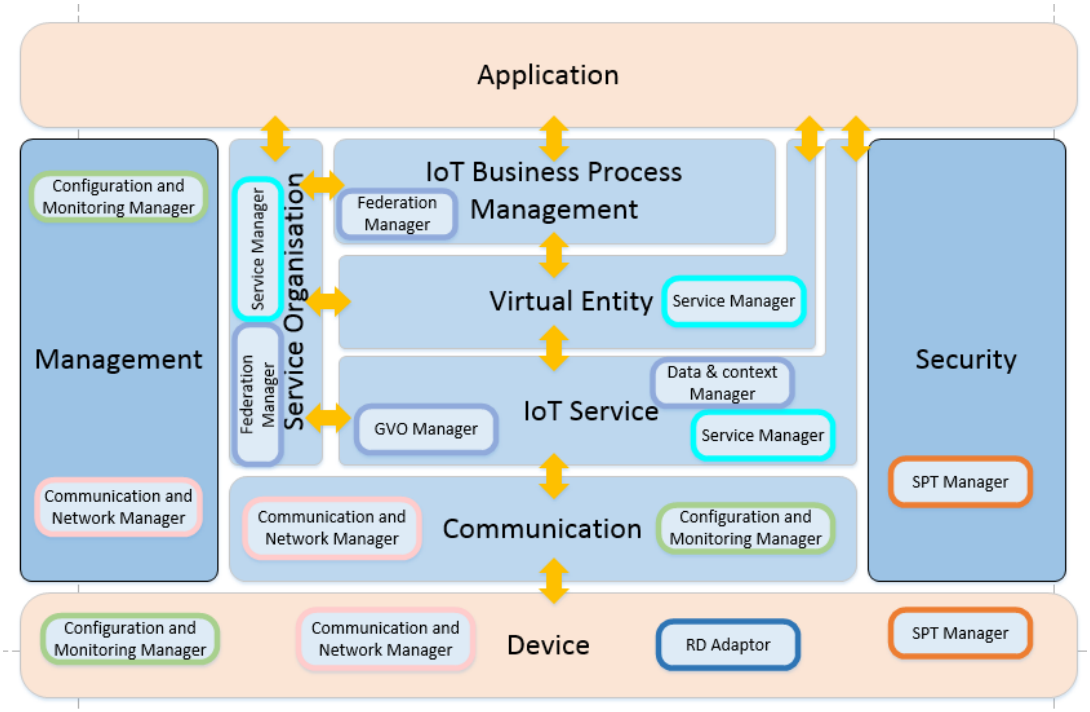


**Figure 9 – Mapping of the RERUM Functional Entities in the IoT-A ARM**

Figure 10 depicts the mapping of the most important RERUM functional components on the Functionality Groups of the IoT-A ARM. Unlike other IoT projects, RERUM has a special focus on security, privacy and trust and, to this purpose, contributes many innovative components depicted in Figure 10. As it is visible from the figure, RERUM adds SPT mechanisms on all IoT-A's FGs with several components on each one of them. The Security FG is of course faded out because the goal of this figure is exactly to map the components of the RERUM Security FG on the other IoT-A FGs. The device-oriented approach of RERUM can be seen by the many components that exist on the devices.

 

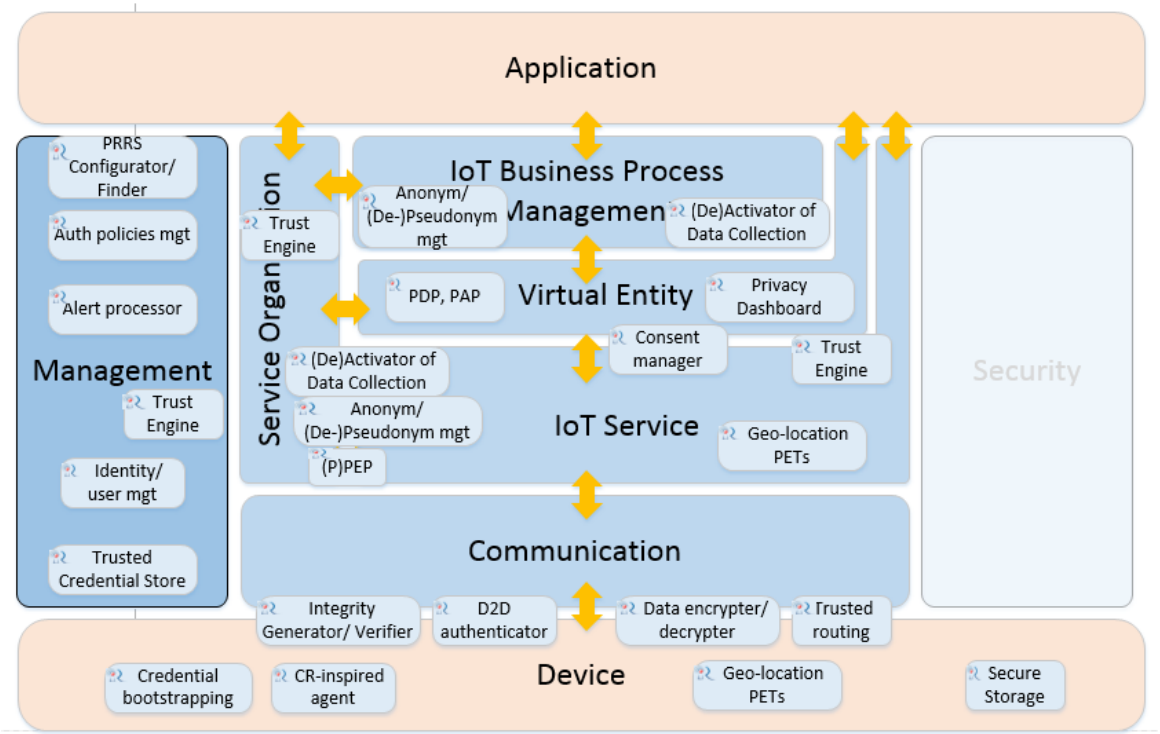


**Figure 10 – RERUM Security, Privacy and Trust components and the IoT-A ARM**

The novelty of RERUM and its focus on security, privacy and trust can also be noticed when Figure 10 is compared against Figure 11 below, which depicts the mapping of the related functional components of other IERC projects, as it was presented in the Whitepaper of the IERC Activity Chain 05 [21] (the colouring shown below the figure denotes the colouring of the borders of the circles). We have to note here that in the original figure that was included in the Whitepaper, a few initial RERUM components were also included, but since the Whitepaper was drafted at a very early in the project these components didn't actually depict the full potential of RERUM; thus we removed them from the figure that we present here (the Whitepaper was submitted in March 2014). Now, comparing Figure 10 against Figure 11 we can see that RERUM pays significant attention to Privacy, while the other projects had very limited work in this area. For example, iCore [7] included only a few privacy aspects on their Usage Control Toolkit, GAMBAS [22] worked only on anonymised data discovery, OpenIoT [14] implemented a Role-Based Authentication and Authorization algorithm, while COMPOSE [23] worked on Sticky Policies, Static Analysis, Declassification and Data Provenance. Furthermore, the most common security aspect that many projects consider is the Access and Usage Control (iCore, IoT@Work, COMPOSE, OpenIoT, GAMBAS). The grey circles denote the areas not covered in IERC projects until now. However, we have to note here that RERUM has a special interest in Trust Negotiation. Furthermore, since the previous IERC projects did not focus so much on devices, RERUM innovates also in the area of D2D authentication and in the area of secure bootstrapping of credentials. Finally, the monitoring of security components and the PRRS are also novel areas not covered in the previous IERC projects so far.

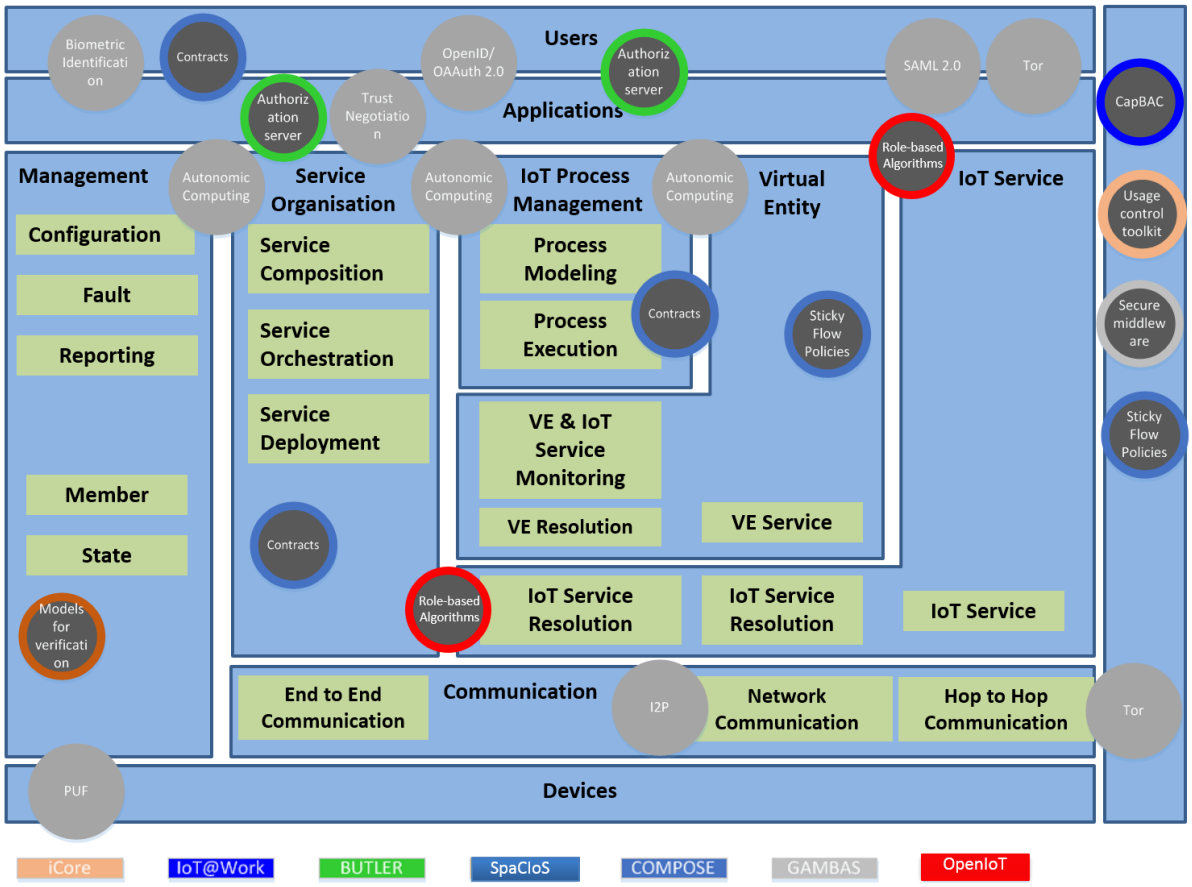


**Figure 11 – Other IERC projects security, privacy and trust functional components.**

## 3.3 RERUM Device and RD Adaptor

### 3.3.1 General

The RERUM Device is a physical object that includes all type of lower layer functionalities. As described in D2.2, the basic block diagram of a RERUM Device is depicted in Figure 12.

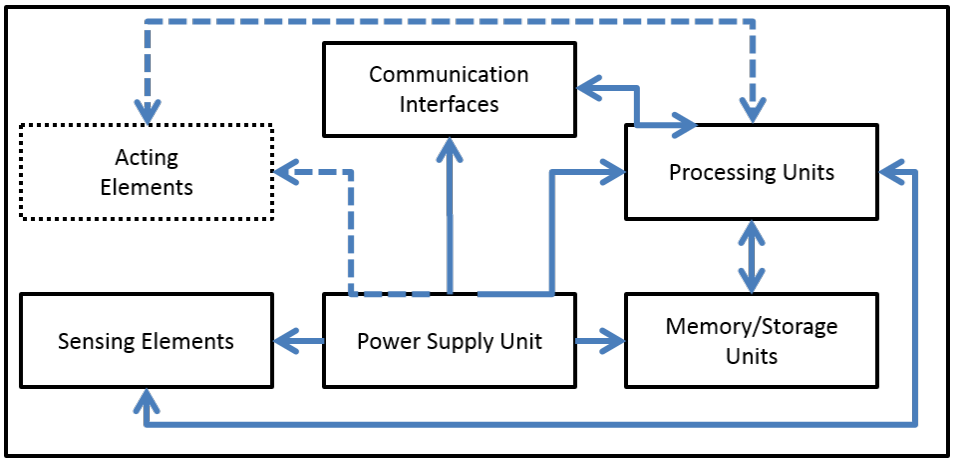


**Figure 12 – RERUM Device low level architecture**

 

Figure 12 shows the basic hardware components of the RD:

- **Power Supply Unit:** consumes power from either small sized batteries or from a USB port and supplies all other components with power.
- **Processing unit:** manages all the computation of the device, being the main unit that executes commands and instructions. It is connected with all other components, as it is the main one that controls the operation of the others.
- **Communication interfaces:** these are the interfaces that allow communication of the device with other devices. Mainly they are wireless (IEEE 802.15.4 or IEEE 802.11), USB and other connectivity interfaces.
- **Sensing elements:** one device can have one or multiple sensing elements that measure properties of one or more Physical Entities of the real world. They also convert the analogue signals to digital, feeding them into the processing unit.
- **Memory/storage unit:** this unit allows the temporary (RAM) or permanent (e.g. flash) storage of data that can be either sensing/actuating data or software code, i.e. OS.
- **Acting elements:** these are optional components, altering the properties of a Physical Entity. They convert digital information from the processing unit into electrical signals to affect the attributes of a Physical Entity.

The current generation of IoT devices has bidirectional wireless communication and sensors that provide real-time measurements. The **RERUM Device (RD)** can be considered as a small computer with communication interfaces, sensors, and (optionally) actuators. This generic hardware specification of the RD allows for RERUM Devices to be implemented on a wide range of physical devices including smartphones and the Zolertia Z1. Most of the platforms used as basis for RERUM Devices are characterized by low computational capacities and small available memory, while battery powered devices also have rather tight power consumption requirements.

Figure 13 below shows the functional model architecture of an RD, split into layers:

- **Hardware**, including all the basic hardware components described in Figure 12 (except from the sensing/acting elements).
- **Sensor/actuator,** including the hardware blocks of the sensing and acting elements.
- **Operating System (OS)**, including the operating system and its functionalities (i.e. Contiki, TinyOS, Android, etc.).
- **Drivers**, including the software that connects the sensing/acting elements with the OS and are part of the OS.
- **RERUM Mechanisms**, including the networking, communication and security/privacy mechanisms that run on the device (and will be defined in Section 4.2). The RERUM mechanisms may also be implemented at “kernel space” as part of the OS of the device. This is represented in the figure with the area on which the two layers overlap.
- **RD Adaptor**, which is the Device part that allows its connectivity with the RERUM Middleware and plays the role of a bridge between the devices and the RERUM system. It includes the functionalities that enable the RD to (i) communicate seamlessly with the other devices, (ii) be able to expose services, (iii) perform federations, etc. Actually, this part is related to functionalities that are needed in order to allow the RD to register to the RERUM MW. This part may include also functionalities that allow the RD to be a Federation Head, if the RD is technically capable of supporting this extra functionality.

 

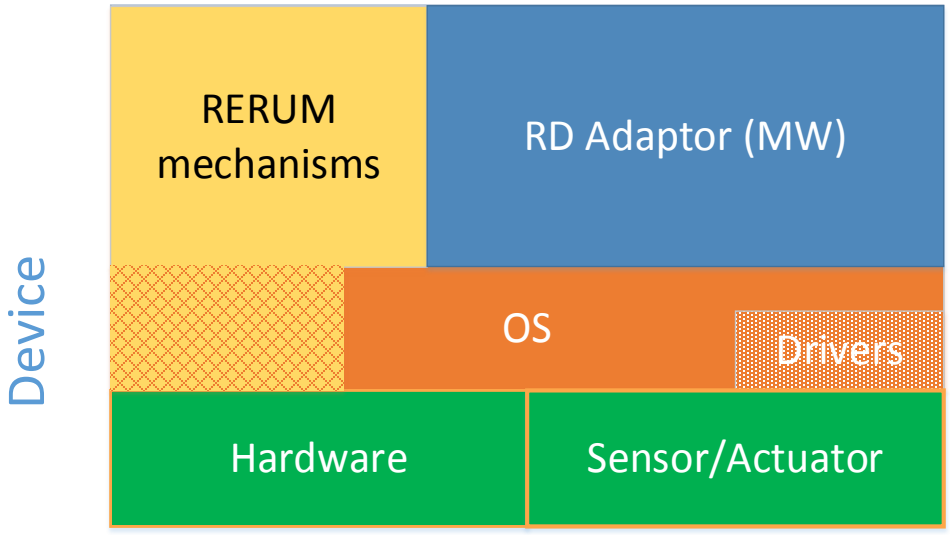


**Figure 13 - RERUM Device functional layers**

As seen in Figure 14, the RD Adaptor mainly consists of four components:

- **Resources Manager**, which manages the RERUM Resources, and has the functionality of handling the software that either gets the sensing elements' measurements or controls the acting element. It is also responsible for adding metadata information to the measurements/commands.
- **RERUM Services Wrapper**, which is the end point that exposes the Services of the devices and also handles and executes the Service requests at the RD level. This module parses the incoming service request and accesses the respective Resources of the Resource Manager. It also sends the QoS requirements of the service request to the Network Monitor (that runs on the device and is described in Section 3.4.2), for managing the network connectivity parameters of the device.
- **Registry Client**, which has all the required information for the RD to be added to the GVO Registry (see section 3.6.2), namely IDs, location, etc. It contains also information about all the Services that the RD can expose, their IDs, URLs and their association with the Resources of the RD, as well as the QoS requirements for each Service. Of course this information is sent to the GVO Manager through the RERUM Services wrapper.
- **Firmware Installer**, which is a specific type of Resource that runs on the device and is responsible for installing and updating the firmware of the RD when there is such a need, i.e. to update the Resources that the device hosts and the corresponding Services or to embed additional security, privacy or networking mechanisms.

 

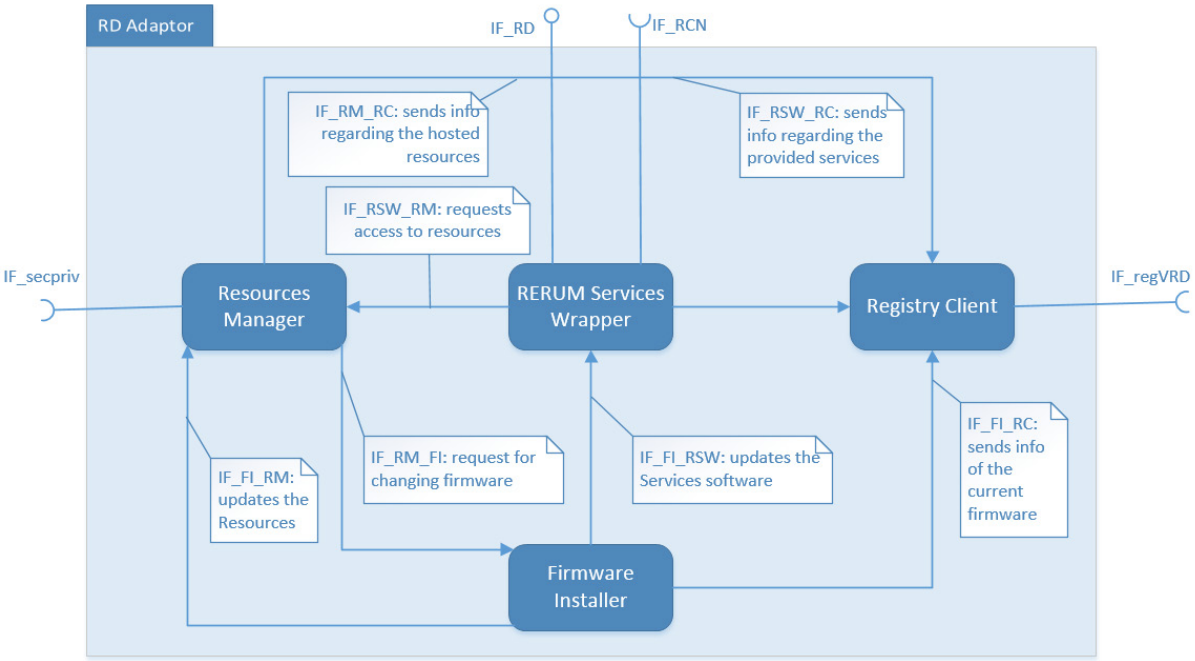


**Figure 14 – RD Adaptor functional components and internal interfaces**

### 3.3.2 RD Adaptor Internal interfaces

Figure 14 shows also the internal interfaces between the components of the RD Adaptor, as well as the external interfaces with the rest of the functional entities. The external interfaces are described in Section 3.1 above. The internal interfaces of the RD Adaptor are defined as below:

- IF_RSW_RM: Resends requests to access specific resources that are exposed by the requested services
- IF_RSW_RC: sends info regarding the provided services
- IF_RM_RC: sends info regarding the hosted resources
- IF_RM_FI: forwards the request for changing the firmware and the respective code
- IF_FI_RSW: changes the software of the Services when the firmware is installed
- IF_FI_RM: changes the software of the resources when the firmware is installed
- IF_FI_RC: sends info regarding the current firmware installed on the RD

The Registry Client gets information from the other two components in order to be able to register the RD at the MW providing the appropriate information. The RERUM Services wrapper is the one that accesses the Resource Manager to wrap the Resources in the appropriate format of the specific Service request.

### 3.3.3 Message sequence charts

In this section, example message sequence charts for three different functionalities of the RD Adaptor are presented. Figure 15 depicts the process for initiating and performing a registration of the device to the RERUM System. When the RD boots, the Registry Client has to initiate the registration procedure. In order to do so, it requests information from the rest of the components regarding (i) the existing firmware on the device, (ii) the Resources it hosts, (iii) the communication and networking capabilities (i.e. number and type of network interfaces), (iv) the Services it can expose. After receiving this information, it invokes the IF_regVRD interface in order to register the RD to the RERUM MW.

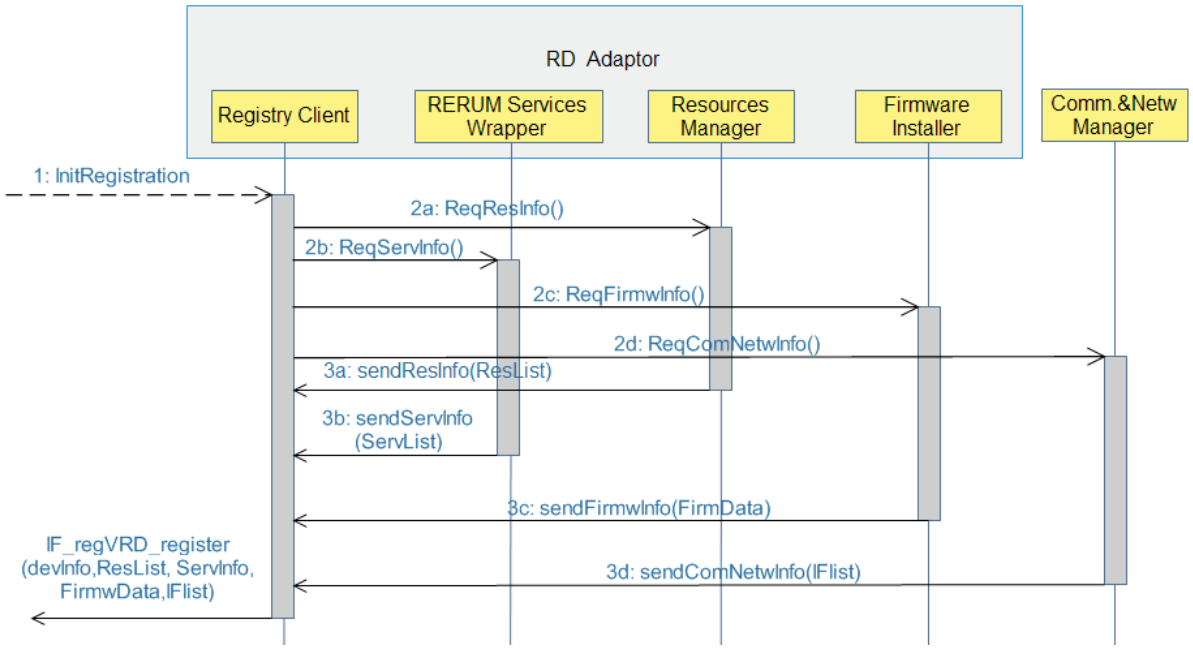


**Figure 15 – RD Adaptor message sequence chart for device registration to the MW**

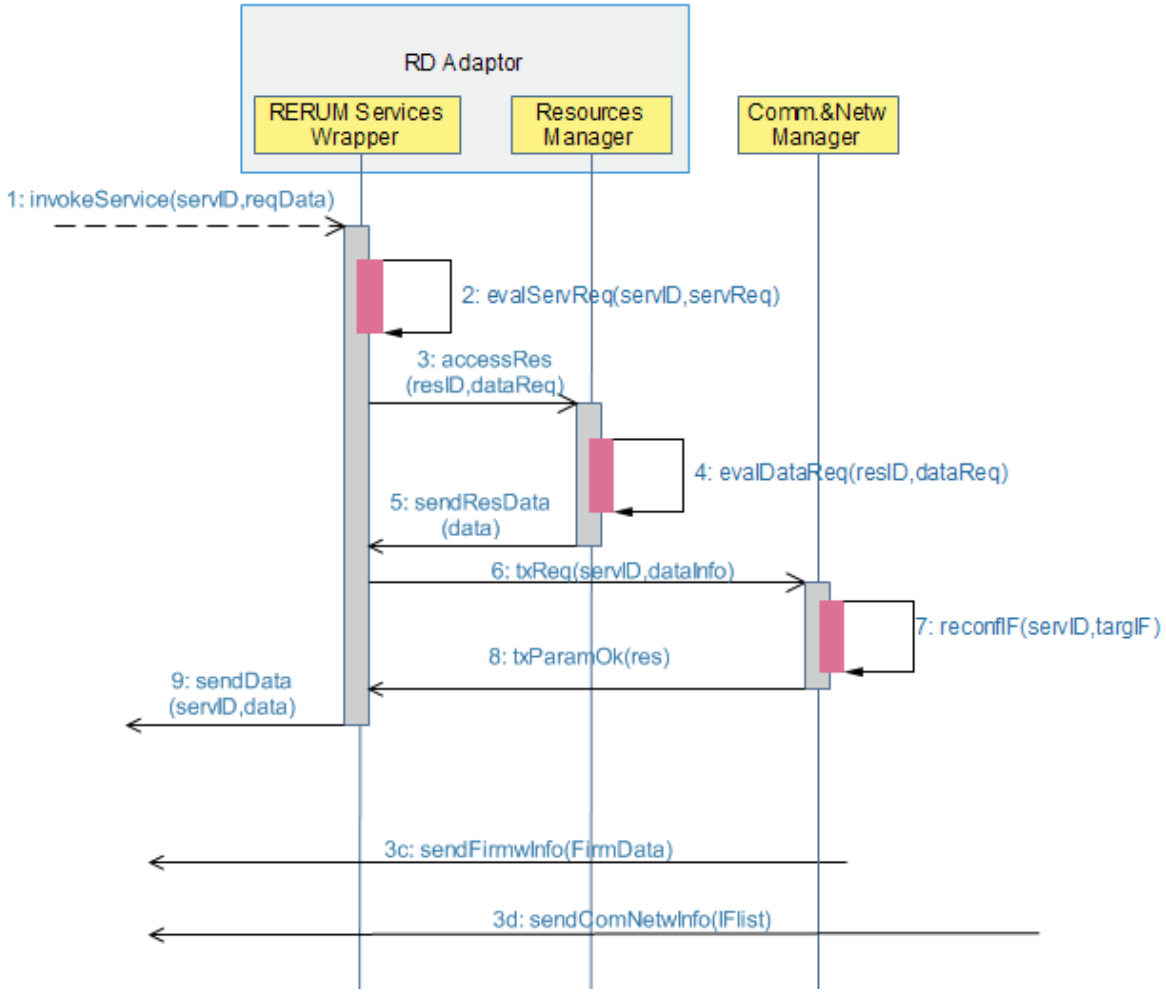


**Figure 16 – RD Adaptor message sequence chart for addressing service request**

 

Figure 16 depicts the process that the RD Adaptor follows for handling a service request. When an incoming request arrives at the RERUM Services Wrapper, it evaluates the request in order to identify if there is a Resource on the device exposing this Service and then forwards the request to access that Resource to the Resources Manager. The latter evaluates the request and identifies the Resource that can provide the requested data, runs the code of the resource and sends back the data to the RERUM Services Wrapper, which sends the response to the part of the Communication & Networking Manager that runs on the device in order to prepare the RD's network interfaces for the transmission. Finally, the RERUM Services Wrapper sends the response with the requested data to the GVO Manager to forward them to the user (following the internal procedures of the RERUM MW).

Figure 17 depicts the process followed by the RD Adaptor to handle the requests for the firmware installation. This functionality is also exposed as a Service from the RD, so any such request is being received by the RERUM Services Wrapper in the same way like any other Service request. As such, the RERUM Services Wrapper evaluates the incoming Service request and identifies the Resource that exposes this Service, forwarding a request to the Resources Manager. The latter evaluates the request and acknowledges that it is a firmware installation request, so it invokes the Firmware Installer component that handles the request, receives the data and performs the update of the firmware. Then, it sends the new firmware info to the Registry Client to be transmitted to the GVO Manager with a registration update. Finally, the OK for the firmware update is sent back to the RERUM MW.

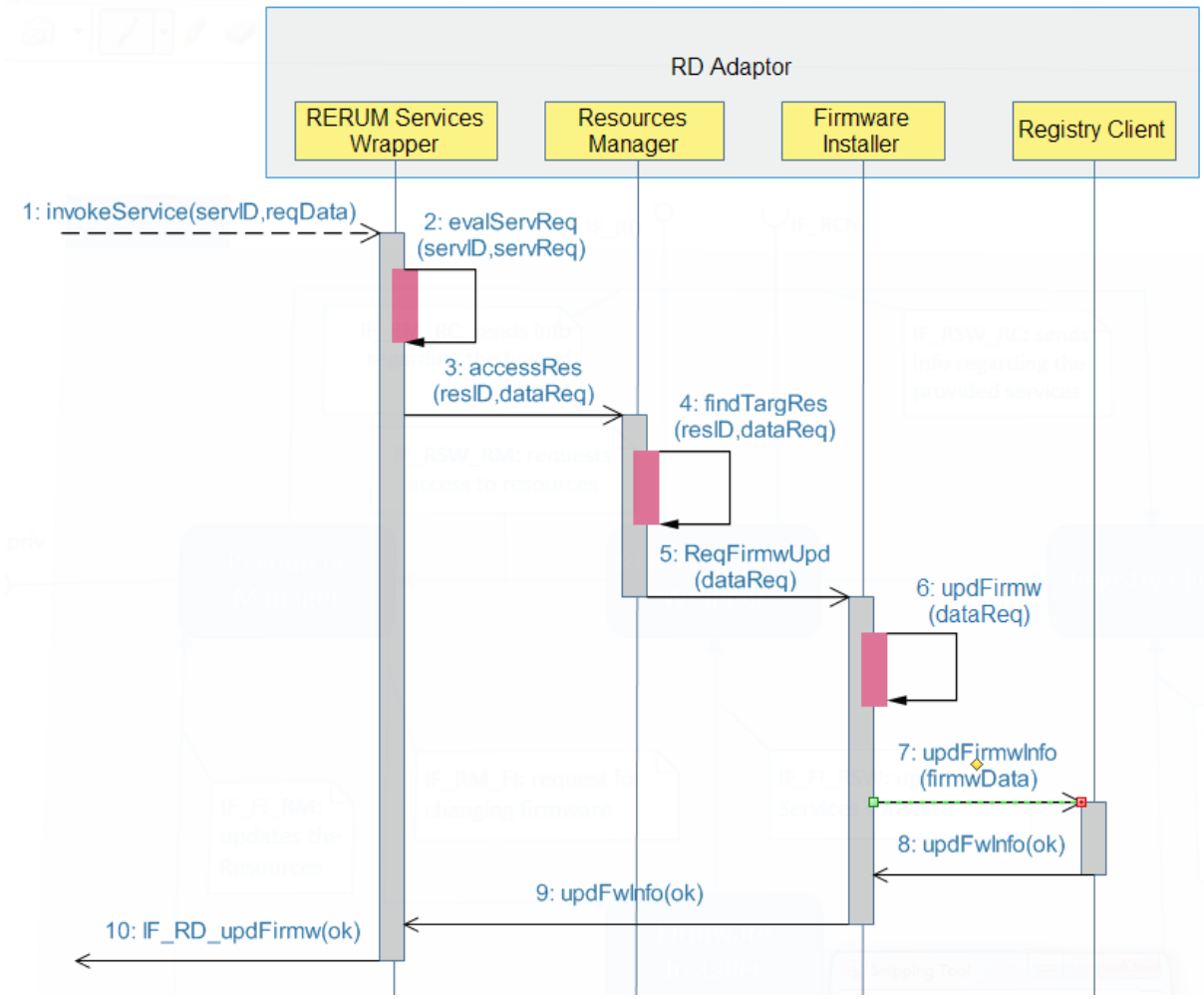


**Figure 17 – RD Adaptor message sequence chart for firmware update**

## 3.4 Communication and Network Manager

The Communication and Network Manager is responsible for handling the communications between RERUM components (i.e. RDs, GW, etc.). It can be split into two entities that are interconnected as seen in Figure 18 below: (i) Communication Manager and (ii) Network Manager. The external interfaces of the entity are described in Section 3.1. There is only one internal interface between the two components:

- IF_NM_CoM: this interface is accessed whenever the Networking Manager needs to send configuration commands to the Communication Manager for the latter to change the parameters of its components according to the results of networking mechanisms, e.g. centralized spectrum management or routing.

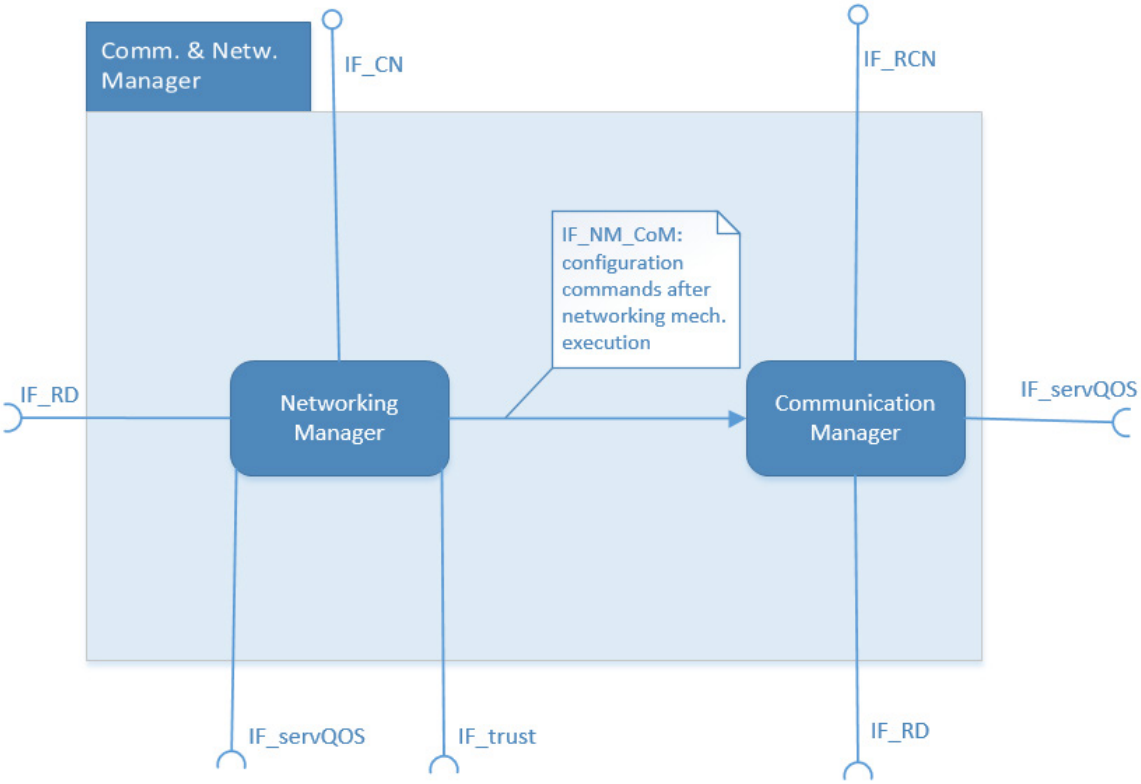


**Figure 18 – Communication and Network Manager internal components – GW**

### 3.4.1 Communication manager

#### *3.4.1.1 Internal description*

The Communication Manager is basically a group of functional components that enable the seamless communication of the RERUM physical devices. The basis for defining the RERUM Communication Manager was the respective Functionality Group (FG) defined by IoT-A [5], which comprises three functional components, (i) the end-to-end communication, (ii) the network communication and (iii) the hop to hop communication.

 

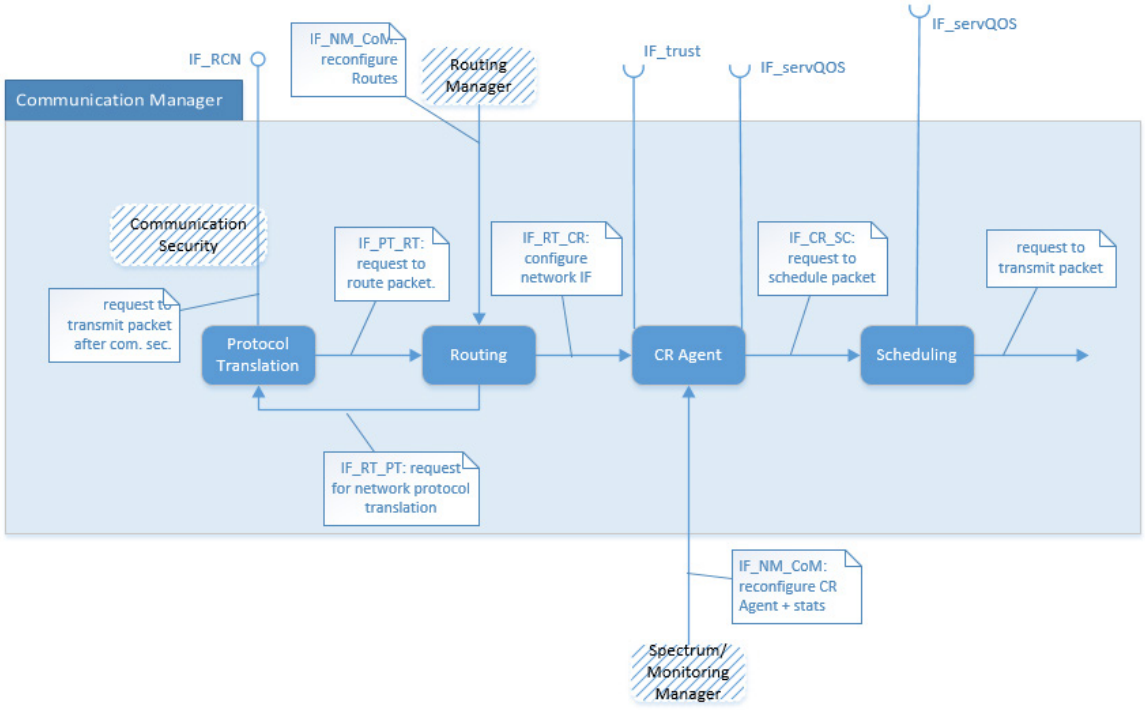


**Figure 19 - RERUM communication manager functional components**

The Communication Manager of RERUM aims to give more attention to the specifics of RERUM, namely the focus on (i) the RDs, (ii) the communication reliability and availability, as well as (iii) the security and privacy. Although security and privacy related mechanisms are part of the SPT Manager, in terms of completeness, some functionalities for communication security are presented in this part.

The functional components of the RERUM Communication Manager are depicted in Figure 19 and detailed below:

- **Protocol translation:** this module is extracted from the IoT-A model and takes care of two types of protocol translations within an IoT communication: (i) end to end protocol translation and (ii) network protocol translation. The first one allows the encapsulation and the translation between end to end protocols, for example when there is a need to translate a CoAP/DTLS[26]/UDP packet/data stream to a HTTP/TLS[27]/TCP packet/data stream. The encapsulation functionality may reside to the RDs if they have the technical capability to support it, otherwise this functionality will be applied on the Gateway. The encapsulation transforms the raw data of the respective service into a meaningful packet for transportation (according to the required protocol by the service). The translation functionality is required to be implemented in intermediate nodes that deal with interconnecting various different networks (i.e. gateways). The network protocol translation [28] enables the encapsulation and translation between different network protocols and mainly between IPv4 and IPv6. The encapsulation functionality can reside in all RDs, but the translation functionality is (similar to the end to end protocol translation) required for intermediate nodes (i.e. gateways) that are connected to networks with different protocols. For example, a router/gateway with two network interfaces, one with 6LowPAN (IPv6) [29] and another with standard WiFi (IPv4) [30] should include this functionality in order to route the packets from one network interface to the other one.

- **Routing:** this module is responsible for routing the packets either within a network or between various networks, as well as for translating the addresses between the various networks. This module is implemented in all devices within the network and deals with identifying which are the available routes to reach a target destination device/user/address. This module makes the translation of the target id/address and **searches the routing table** for all available routes to the destination, extracting the possible next hops. It has to be noted here that this module does not take the final decision on which network interface/route will be selected, but only to extract the possible routes and give this result as input to the "interface selection" module (see below), acknowledging that maybe more than one network interfaces may be available in some devices. In the very constrained devices with a single network interface, of course the identification of the next hop can be done (in existing implementations) at this module. However, in our case this is not true, because there may be alternative routes to the destination (i.e. in a mesh network), so the interface selection module is delegated with this decision.
- **CR Agent:** this module substitutes the Interface Selection module that was described in D2.3, since it is assumed to be a more general component including the interface selection functionality. CR Agent is responsible for identifying the network interface to be used for the transmission of a packet (or a stream of packets), taking as input the available routes from the Routing module, as well as the QoS requirements of the specific service that generated the packet. The interface selection does not only select the network interface, but also **sets the transmission parameters** of the interface (using the Cognitive Radio technology when available – mainly on the Gateways), thus it has to be implemented in all RERUM Devices and RERUM Gateways, regardless if they have one or more interfaces. The CR Agent comprises of several sub-components [31][32][33][34][35][36][37] that enable the devices to have cognitive radio capabilities. More details for the internal functionalities of the CR-inspired agent will be given later on in this section due to the importance of this component in the RERUM architecture.
- **Scheduling:** this module is related with ensuring the QoS of the data streams, mainly dealing with queuing, scheduling and prioritisation of the outgoing packets/frames. This module is the final one before transmitting the frames and prioritizes them according to their service requirements. In intermediate nodes (e.g. routers, gateways), it is also possible to implement mechanisms to discard packets that have been delayed more than their service class requires, which saves resources from the network.
- **Communication security:** RERUM has a major focus on enhancing the security and privacy of the IoT, so the communication security is one of the basic working areas of the project. The idea is to design and implement security and privacy enhancing mechanisms that allow to protect the communication of the RERUM Device. Many recent studies (e.g. [38]) have shown that one of the most important issues of IoT security and privacy is the lack of use of encryption protocols in the transport layer, as well as the lack of efficient device authentication procedures. RERUM aims to really advance beyond the current state of the art in IoT communication security and provide lightweight mechanisms for protecting the confidentiality and the integrity of the data in transit, i.e. using among other things encryption procedures running on the sensing devices. Although data encryption is not always lightweight and usually requires high processing and storage capabilities for storing the encryption keys, RERUM is developing adaptive procedures that can work in both constrained and unconstrained devices. The mechanisms will take into account both the device capabilities and the services in order to identify the level of protection and security to be applied. For Communication Security several security components have to interact in order to secure the communications between the different elements of the RERUM system. Figure 20 shows an example identification of the components that contribute to the secure communications. Furthermore, Figure 21 shows the

 

interaction of those components for an End-to-End communication (transmission and reception). The descriptions of these security components will be given in Section 3.10. Briefly, for the transmission, the sender needs to apply Privacy Enhancing Technologies and the needed encryption and integrity generation, and then it identifies the reputation of the receiver. At the receiver, the authentication of the transmitter is being done (which may also include privacy enhancing technologies), and the decryption and integrity verification of the message follows. After that, the reputation of the sender is verified/changed according to the data he sends and if needed the required security/privacy/access policies are enforced. The order of the components is not fixed and not all components take place in every communication.

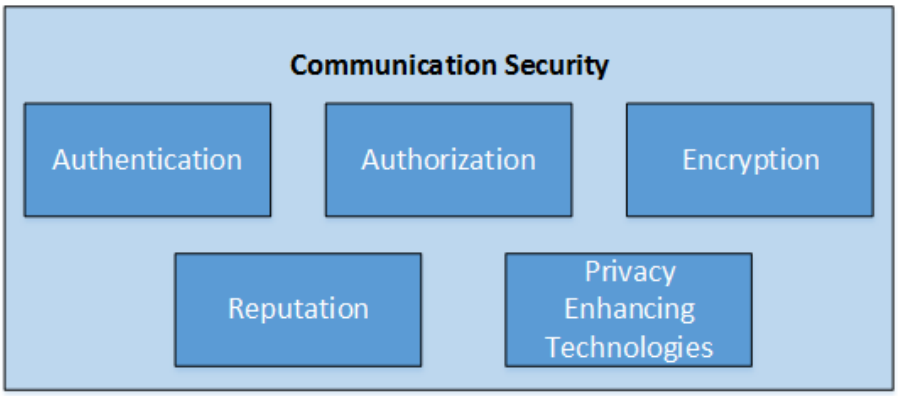


**Figure 20 – Security components in the communication layer**

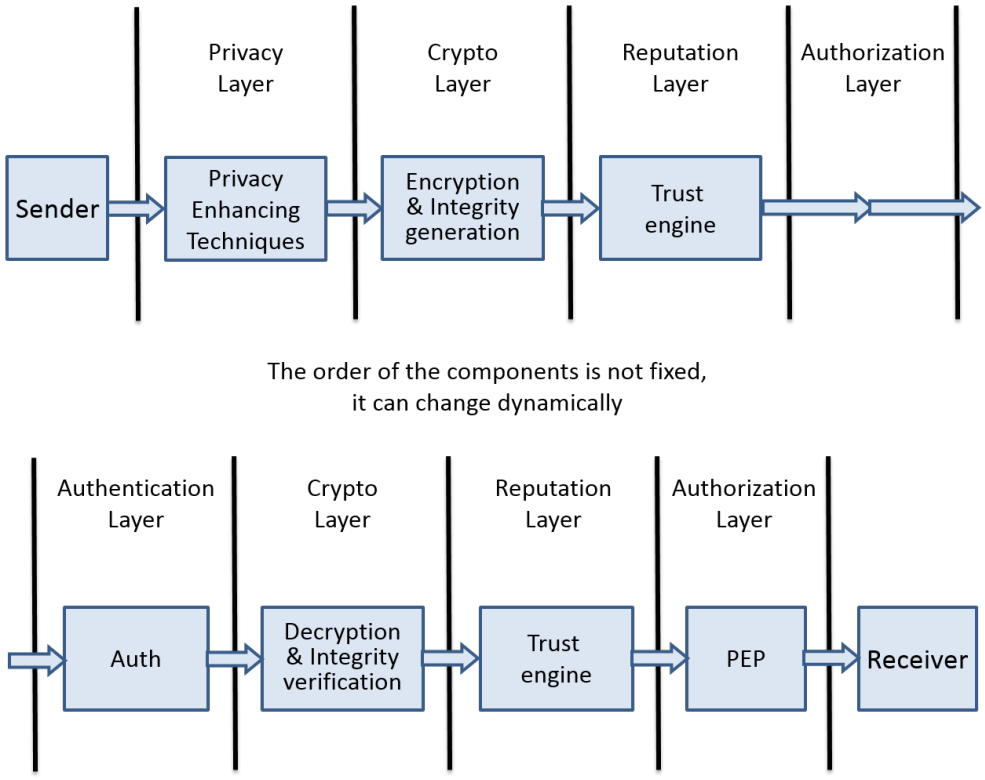


**Figure 21 – Interaction among security components (from sender to receiver)**

#### *3.4.1.2 Interfaces between internal modules of the communication manager*

Figure 19 shows also the interconnectivity of the internal components of the communication module and the internal interfaces. The external interfaces of this component are described in Section 3.1. The Communication Manager receives a transmission request by the IF_RCN interface and after it employs the Communication Security functionality contacting the SPT Manager that runs locally on the RD (or the RERUM Gateway), it has to perform the necessary operations to handle the transmission. This process will be described in the Message Sequence charts in the next subsection. The internal interfaces of the components of the Communication Manager are described below:

- IF_PT_RT: this is the interface accessed after the protocol translation in order to request the routing of the packet that has to be transmitted.
- IF_RT_PT: this is the interface accessed when the Routing component identifies that there is a need for network protocol translation (i.e. from ipv6 to ipv4) for transmitting the packet to the next network hop.
- IF_RT_CR: this is the interface accessed after identifying the appropriate route, in order to request from the CR Agent to identify the appropriate network interface and to configure its parameters for transmission.
- IF_CR_SC: this interface is accessed after identifying and configuring the interface for adding the packet to the scheduling queue of that interface.

There interfaces with the Network Manager (as mentioned in the beginning of Section 3.4) are:

- IF_NM_CoM that connects (i) the Routing Manager of the Network Manager with the Routing component in order to send commands for reconfiguring the routing table, (ii) the Spectrum Manager of the Network Manager with the CR Agent component in order to send commands for reconfiguring the components of the CR Agent after the execution of the centralized spectrum management mechanisms and (iii) the Network Monitor of the Network Manager with the CR Agent in order to provide statistics regarding the status of the battery and the IF buffers for using them in the spectrum decisions.

#### *3.4.1.3 Message sequence charts for communication manager*

The following diagrams show examples of message sequence charts for the communication of RERUM Devices in three scenarios.

In Figure 22 the direct communication (single hop) of two devices that are in the same network is depicted. The RD Adaptor has to transmit a packet, so it requests from the SPT Manager to secure this packet and prepare it for transmission. Then, the RD Adaptor sends it to the Communication Manager that handles the transmission. First of all, the Protocol Translation handles any type of end-to-end (or Network) protocol translation (if needed for the specific transmission, otherwise this function is null and just forwards the packet to the Routing component, which in turn tries to resolve the id of the target device and finds the most appropriate route for the packet. If required by the Service that creates the packet, then trusted routing will also be applied by contacting the SPT Manager. The next step is to contact the CR Agent for identifying the target network interface and configure its parameters for transmission and then forward the packet for scheduling, queuing the packet if required. Then the packet is transmitted and received by the target RD (RD#2) directly by the scheduling component of the on-device Communication Manager which forwards the packet tot eh CR Agent for the MAC checking and then to the RD Adaptor. The latter has to send the packet to the SPT Manager in order to perform the opposite part of the Communication Security, namely the integrity verification, decryption, etc. (as described in the communication security part above). Finally the raw data are sent back to the RD Adaptor to be handled by the respective RERUM Services Wrapper.

 

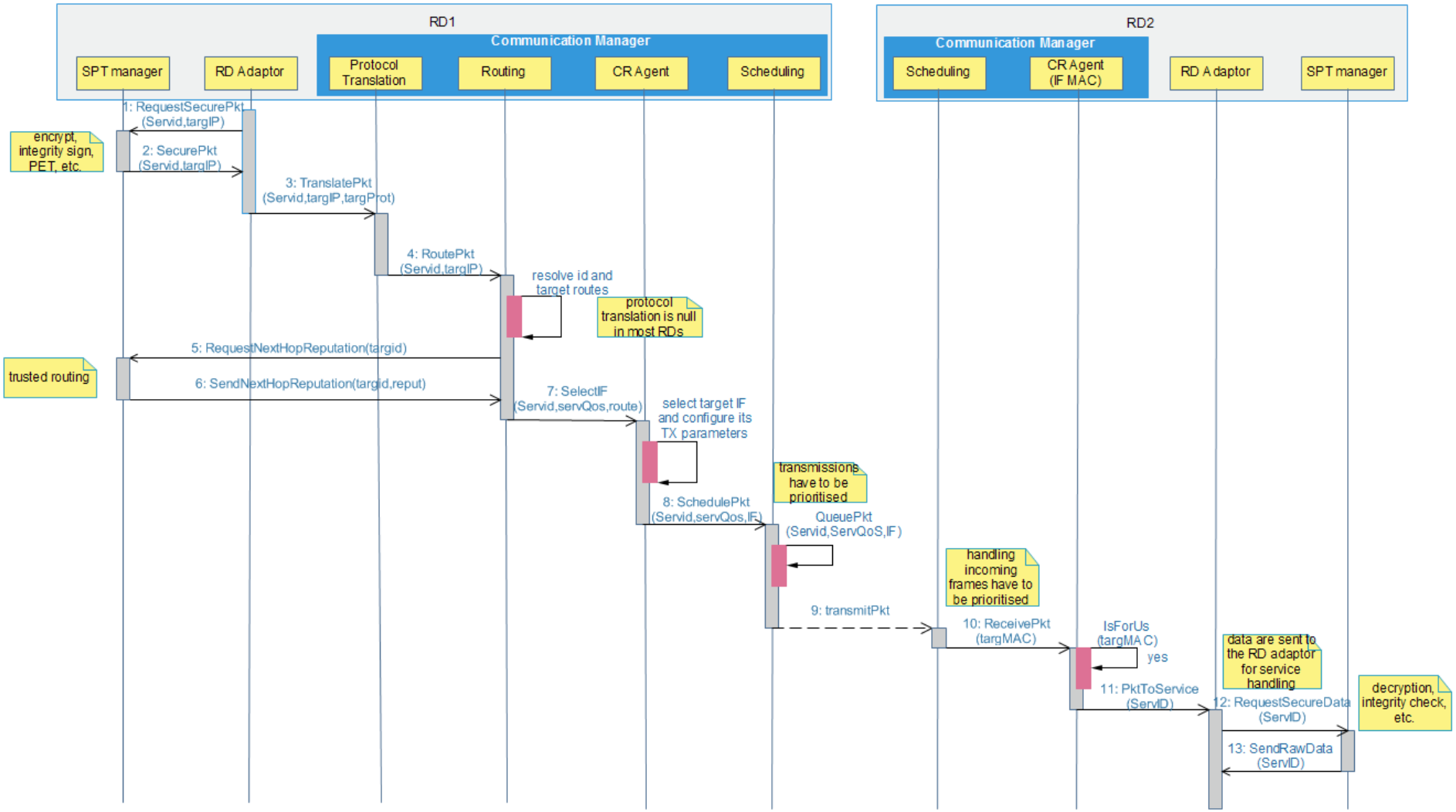


**Figure 22 - RD to RD communication within the same network**

In this example scenario the two devices are assumed to be using the same networking technology (i.e. IEEE 802.15.4 [40]) and have IPv6 addresses of the same domain, so no protocol translation is needed for their communication. Thus, the messages that are sent from the service are passing through the communication security mechanisms, are routed to the next hop, enqueued and sent via the wireless channel to the next device RD#2. There, the frame is received, checked in the MAC layer if it was indeed sent to this device and is forwarded to the respective receiving service.

In Figure 23 the multi-hop communication of two devices that are in the same network is depicted. In this scenario the two devices are using the same networking technology (i.e. IEEE 802.15.4) and have IPv6 addresses of the same domain, so no protocol translation is needed for their communication. However, in this scenario, the devices are not within transmission range and they need an intermediate device to route the packets. In this figure, only the functionalities of the intermediate RD that plays the role of the router/forwarder is shown, while the transmission of the packet from the originating device and its reception by the target device are not shown for the sake of simplicity as they are depicted in Figure 22. The receiver intermediate node checks if the packet is sent to it and if not it checks to see if it can route the packet to its final destination, so it calculates the next hop (using trusted routing if required), it selects the target interface, setting its parameters and enqueues the packet to be sent via the wireless channel to the next device.

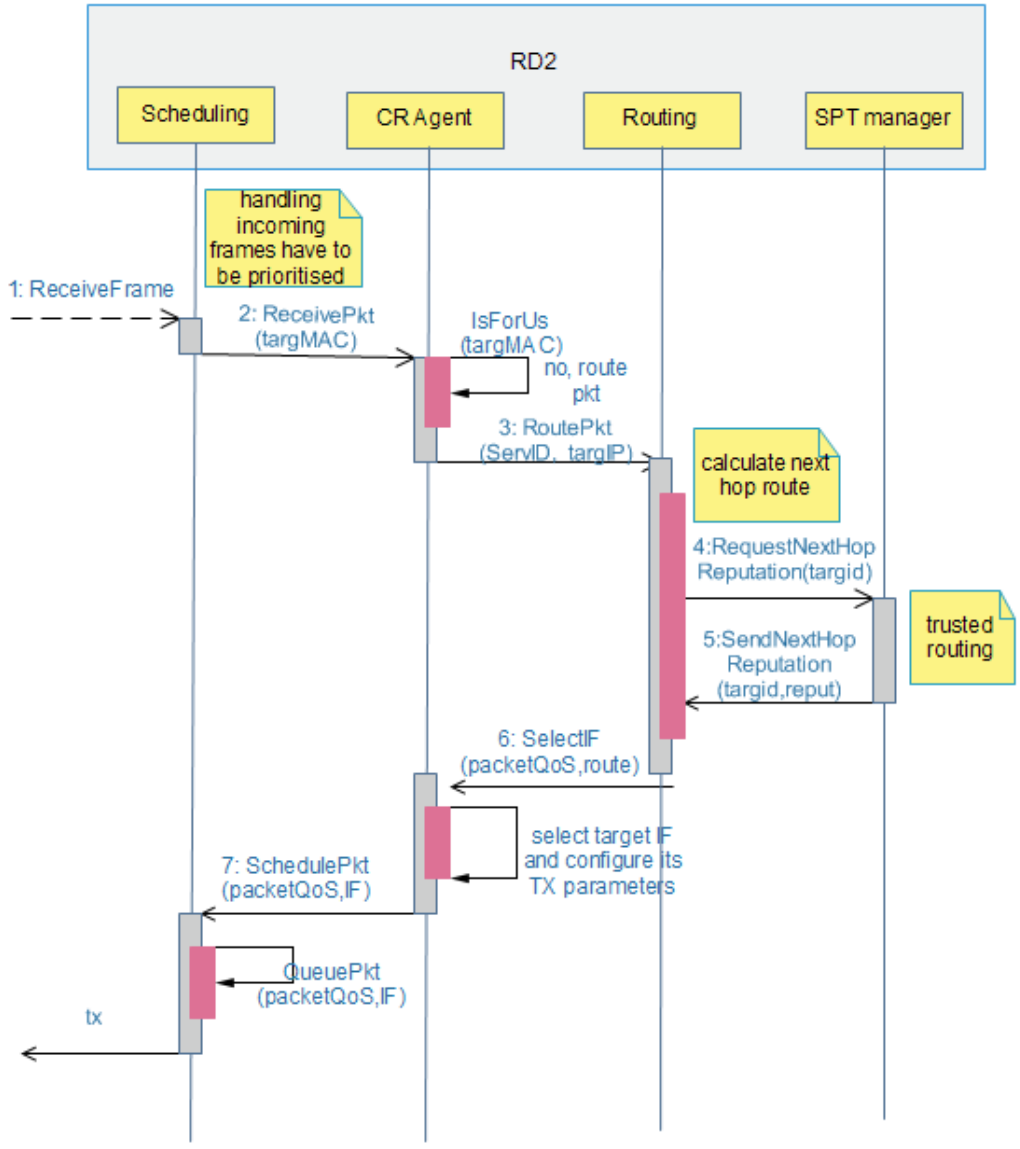


**Figure 23 - RD#2 plays the role of relay (multihop)**

 

In the Figure 24 another example communication scenario is depicted, in which two devices (origin and destination) belong to different network domains so their communication have to pass through a Gateway. Here, for sake of simplicity we depict only the internal path of the packet within the GW. Upon reception the frame is forwarded to the Communication Manager of the GW to decide for the next route of the packet after the end-to-end translation. The id of the destination is being resolved and the target routes are extracted. The integrity of the packet is verified, the trusted routing mechanism is executed (if required) and the packet is decrypted and encrypted again with a different key (if required). Then, the packet is being translated to the appropriate network protocol and then the packet is sent to the interface selection module to select the target network interface and then it is sent out for transmission to the destination.

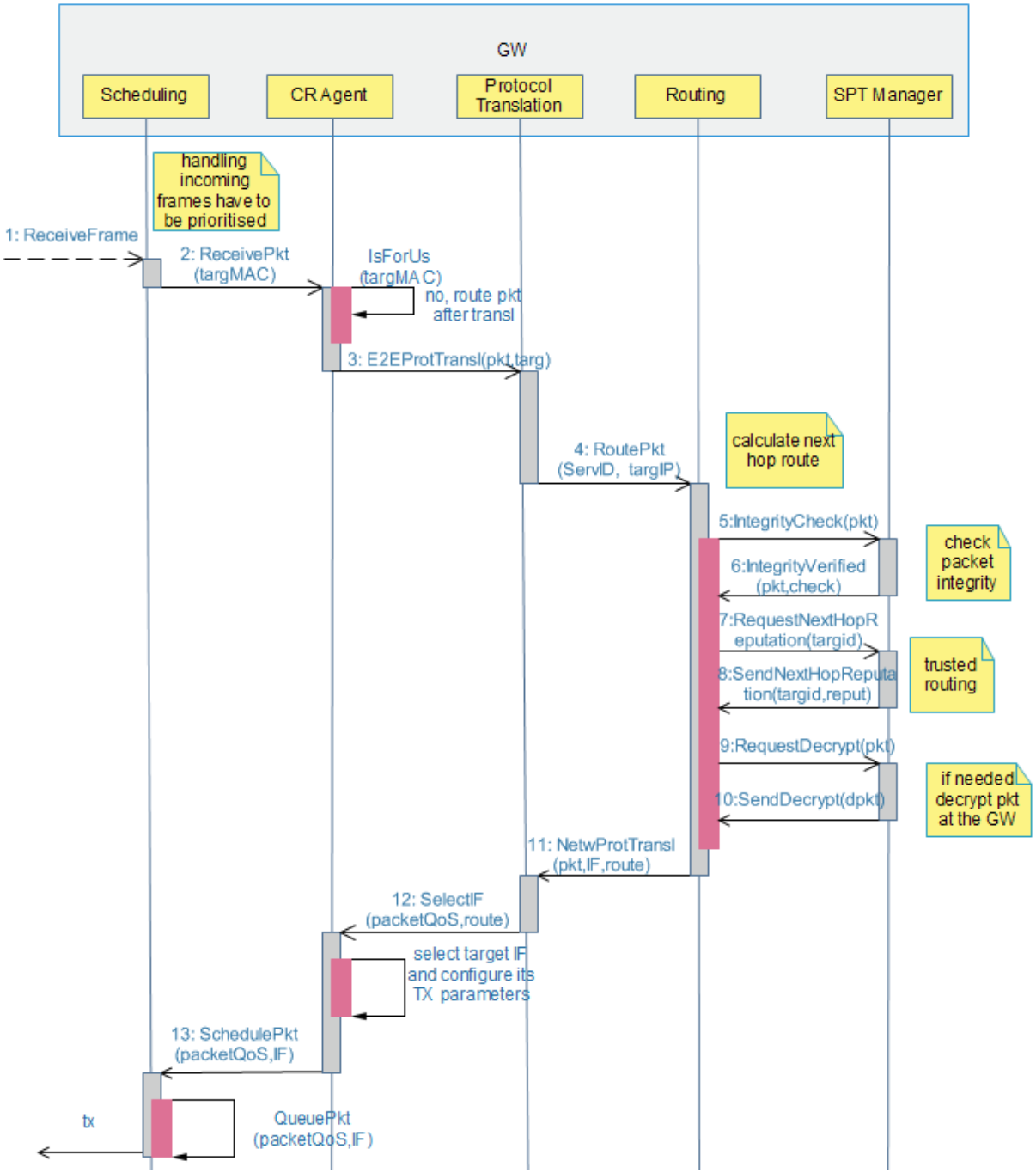


**Figure 24 - RD through GW communication**

### *3.4.1.4 CR Agent*

As mentioned above, the CR Agent module is the one that implements the cognitive radio mechanisms that allows for flexible spectrum usage and sets the transmission parameters of the interfaces of the devices. The CR Agent comprises of several functional components as depicted in the Figure 25 below.

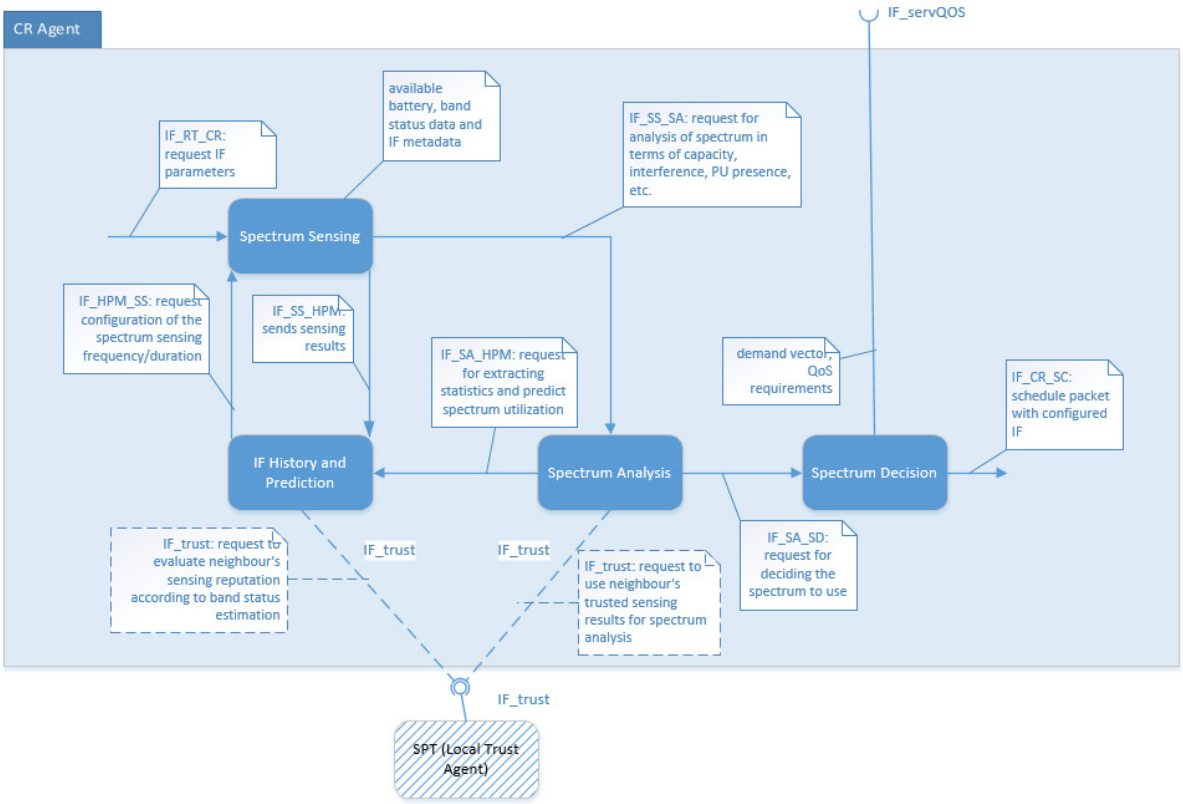


**Figure 25 – CR-inspired agent internal components**

The internal modules of the CR-inspired Agent as described in deliverable D2.3 and in [36] are:

- **Spectrum Sensing Module (SSM):** The SSM senses the spectrum in order to identify if a specific spectrum band is being utilized or not, mainly by Primary Users (PUs). Spectrum sensing can utilize several techniques, e.g. energy detection, cyclostationary feature detection, matched filter detection, etc. [34]. The most commonly used technique in the literature is by far the energy detection, which measures the energy at a specific spectrum band and comparing the result with pre-defined thresholds it can identify if a transmission exists or not. However, this technique cannot identify what type of transmission it is, namely if it originates from a PU or other cognitive users. On the positive side, this technique is the simplest one that consumes the lowest amount of resources at the receiver. The SSM takes as direct input from the radio interface the Band Status and some Sensed Data together with some Interface Metadata used for specifying the characteristics of the interface and the spectrum band under consideration (i.e. threshold). Thus, this module is used to establish the wireless channel availability at the time of transmission, but it can't really determine if the transmission comes from a licensed user or not. Spectrum sensing can also be done periodically in order to keep statistics about the utilization of each spectrum band and these statistics are sent to the IF History and Prediction Module. Finally, the SSM is the module that controls the sensing parameters of the

 

interface (spectrum frequencies to examine, sensing duration, sensing period, etc.) according to the input it receives from the IF History and Prediction Module.

- **Spectrum Analysis Module (SAM)**: This module is responsible for analyzing the sensing results that it receives from the SSM in order to identify the characteristics of the examined spectrum frequencies in terms of capacity, condition, interference, occupancy, PU presence and other link-layer related parameters. Since in CR there is no standard definition for a "channel", the RD can select its own central operating frequency and bandwidth, according to its transmission requirements, without being limited by the channel width of i.e. 2MHz for IEEE 802.15.4. In this respect, the RDs can use as much bandwidth as it is available for high resource-demanding applications. The only limitation comes from the hardware capabilities for the band width they can access. The output of the SAM goes both to the Spectrum Decision Module for selecting the spectrum band to use and to the IF History and Prediction Module.
- **IF History and Prediction Module (HPM):** this is the module that identifies statistics about the occupancy model of the spectrum bands/channels and keeps either a respective database or a model of the spectrum occupancy that is updated periodically. The goal is to keep a prioritized table of the spectrum bands in order to see which are more frequently utilized (so that the RD won't try to transmit in these very frequently) and which are mostly underutilized. This module helps also the SSM to adjust the sensing periods per spectrum band, in order to avoid sensing occupied bands very frequently and save energy. The HPM sends also statistics of spectrum occupancy to the neighbour CR-agents (after the required validation of the statistics by the Trust Engine) when cooperative spectrum sensing is utilized.
- **Spectrum Decision Module (SDM):** the SDM is the module that does the actual selection of the spectrum band that will be accessed. The prerequisites from the original communication demand according to the service class(es) of the RD are compared with the given sensed information which has been handled in the other modules of the CR-Agent. The available spectrum bands are weighted to the expected energy drainage for the upcoming transmission and the QoS requirements of the service and then, the most suitable spectrum frequency and bandwidth are selected. The selection is also based on the expected validity duration on each channel, i.e. switching spectrum bands too frequently might drain the battery more than finding a more stable band. Utilizing the statistical data from the IF History and Prediction module a preferable and stable spectrum band could be selected. When the most suitable spectrum band and interface have been selected, the most suitable Modulation and Coding (M&C) to be used by the chosen interface for the selected spectrum band are also selected, in order to consider any additional energy savings which still meet the specified requirements. The choice of modulation and coding might face constraints and regulations depending on predefined protocols, e.g. LTE and GSM face big differences in their modulation possibilities. In such a case the protocol is pre-set and there is no room for M&C variations. In other systems, where there is more room for adaptive M&C, the choice can be further refined to achieve lower energy consumption.
- **Local Trust Agent (from SPT Manager)**: this is a module for evaluation of trust for cooperative sensing and access decisions. It is contacted by both the IF History and Prediction Module and the Spectrum Analysis module to get the reputations of the measurements of the neighbour devices both for optimizing the sensing and the analysis results.

When an RD needs to make a transmission related with some service, a "demand vector" characterizing the traffic volume and QoS requirements is passed to the CR Agent. **The Spectrum Decision Module** will determine if the spectrum selection process needs to be initiated for handling the transmission or the current selected spectrum can satisfy the demand. This means that if the

current spectrum selection cannot meet the service demands then the CR-agent has to find another spectrum band (with i.e. larger bandwidth). If the transmission request can be served with the current spectrum selection, then the sensing procedure may be skipped and the CR agent will directly use the last known transmission option. Furthermore, the sensing procedure can be partly skipped if the transmission has fixed setup requirements or only has to sense a small part of fixed channels.

If a new sensing/selection procedure has to be performed then the demand vector will be passed to the **Spectrum Sensing Module** (SSM), which will determine which spectrum bands will be sensed. As it is seen in Figure 26, the SSM can be split into two sub-modules, namely the Instant Spectrum Sensing and the Periodic Spectrum Sensing. The first performs the spectrum sensing whenever it is required to identify the status of the spectrum for a specific transmission request, while the second performs the spectrum sensing periodically in order to capture the history of the spectrum occupancy for optimizing the decisions.

The sensed spectrum will be based on a prioritization table generated by the **IF History and Prediction Module** and the available energy, e.g. if the device has a fully charged battery the SSM might sense a wider spectrum over longer periods, while if the battery is low the SSM might only sense statistically safer spectrum channels for a shorter duration. The SSM will send interface parameters such as selected frequencies and sensing duration to interfaces that execute the sensing. The sensing results are analysed by the **Spectrum Analysis Module** to identify the characteristics of the spectrum band and then are stored in the **IF History and Prediction Module**, updating the IF prioritization table before passing on to the **Trust Agent**. The sensed data carries information such as IF Metadata and RF band status. This information will be treated by the Trust Agent to evaluate if the spectrum is trustworthy or not (usually comparing the results with similar results of neighbouring RDs). The results of the spectrum analysis are passed to the **Spectrum Decision Module,** that will select the most applicable option based on the given traffic demand vector, available power and other QoS related information. This process is also shown in Figure 26.

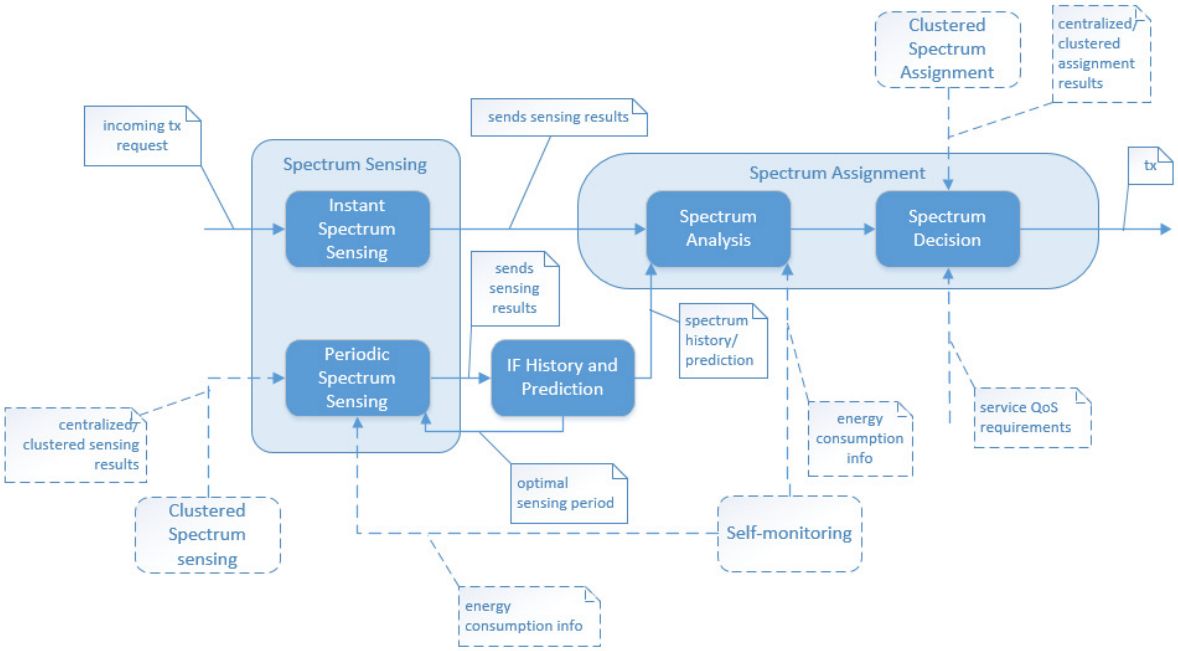


**Figure 26 - Spectrum management process within the CR Agent**

 

### 3.4.1.5 Interfaces between internal modules of the CR Agent

The internal interfaces of the CR Agent, as seen also in Figure 25, are the following:

- IF_SSM_HPM: this is the interface that sends the results of the spectrum sensing procedure to the HPM module in order to store them and process them for extracting the models of the spectrum occupancy.
- IF_HPM_SSM: this is the interface that sends the commands to the SSM for reconfiguring the spectrum sensing period according to the historical data and the prediction about future occupancy.
- IF_SS_SA: this is the interface that is used for sending the sensing results from the SSM to the Spectrum Analysis module, in order to analyse them and extract information regarding the capability of that spectrum band to serve the requested Service meeting its QoS requirements.
- IF_SA_HPM: this is the interface used by the SAM to request the statistics for the historical occupancy of the current spectrum band, as well as a prediction for future occupancy for using them in the spectrum analysis.
- IF_SA_SD: this is the interface used to forward the results of the spectrum analysis in order to decide which of the available spectrum bands will be used for the current transmission.

### 3.4.1.6 Message sequence chart for spectrum management

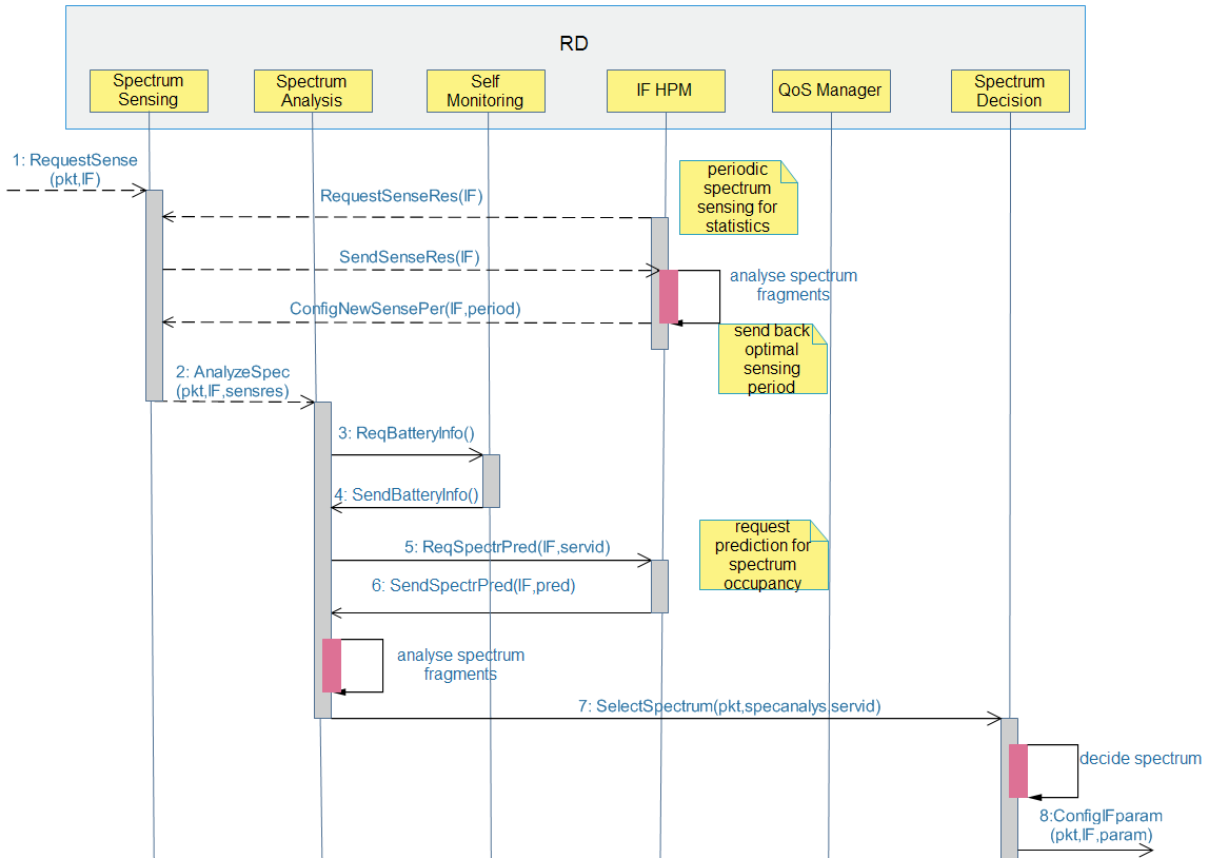


**Figure 27 - Message sequence chart for spectrum decision when there is new tx request**

In Figure 26 the message sequence chart for deciding the spectrum to use for a new transmission request is depicted. The new request is being initially received by the Spectrum Sensing module that has to sense the spectrum to see if the currently selected spectrum band is available for transmission or a new spectrum band has to be found/selected. As also seen in the figure, the HPM module sends periodically requests to the Spectrum Sensing module for sensing results for extracting the optimal period and the history of the occupancy at each band. The Spectrum sensing module forwards then the request to the Spectrum Analysis module, along with the sensing results. The SAM accesses the stats of the self-monitoring component for identifying the battery status, which is used for the analysis of the spectrum results in terms of power-efficiency. The SAM accesses also the HPM for receiving the prediction for the spectrum occupancy at each band to see if they will be available in the near future (assuming that perhaps a transmission will utilize more than one timeslots). The results of the analysis are sent to the Spectrum Decision module that takes the final decision regarding which spectrum fragment to use.

## 3.4.2 Network Manager

### *3.4.2.1 Internal description*

The Network Manager is responsible for managing and configuring the components that deal with the network connectivity of the device. In IoT-A the goal of the overall Management functionality groups is to implement the FCAPS [45] framework for network management. FCAPS stands for the functionalities of Fault, Configuration, Accounting (Administration), Performance and Security. However, within RERUM, we have allocated the Fault, Accounting and Security functionalities under the Security, Privacy and Trust functionality group, because of the high importance we give into security and privacy. Parts of the Configuration and the Performance are also included in the Configuration and Monitoring Manager (see Section 3.8). The device oriented approach of RERUM, enabled us to consider that Network Management should be a key part of the architecture for allowing the seamless networking of the devices, ensuring their high availability. The Network Manager includes a set of functional components that enable taking cooperative decisions between the RERUM Devices, allowing various modes of decision making, i.e. centralized, distributed or hierarchical clustered solutions.

The functional components of the Network Manager are depicted in the Figure 28, as well as their interactions with the SPT Manager, the Communication Manager and the Configuration & Monitoring Manager through three external interfaces. The dotted components are the ones that are considered useful for the RERUM framework but are not the focus of the project since optimal solutions already exist in the literature.

As defined within the requirements in D2.2, the RERUM architecture should be able to ensure system reliability with respect to system and device failures. The detection of a failure should be done as fast as possible, and the mean time between failures should be maximized. To address this requirement with respect to network failures, the network management group of RERUM includes a specific functional component the "**Network Monitor**". This component can be realized as a **self-monitoring** [46] mechanism through which the devices will be able to monitor themselves and their neighbours (in terms of network-related parameters) in order to identify when there are some failures. Parameters for network monitoring may include: (i) device status (on/off), (ii) battery lifetime, (iii) link quality, throughput and delay, (iv) transmission queue size, (v) number of collisions, (vi) packet error rate and (vii) other critical networking statistics. Many network monitoring frameworks use a centralized approach with a central monitor (i.e. in a gateway or even in a central server) that gets statistics from all nodes in the network, analyses them and identifies failures and misbehaviours. However, the continuous transmission of information induces large energy consumption on the devices, as well as heavy signalling in the network for transferring the statistics from all devices to the central node. Other approaches follow a distributed monitoring approach, so that each node monitors itself and the nodes

 

are exchanging statistics in order to find problems in the networks, but this can't be done in heterogeneous networks where devices have different physical layers and networking technologies. Those approaches although they do not create much signalling in the network, they induce much larger energy consumption on the nodes due to the need for running more complex monitoring algorithms on top of the many transmissions of statistical data.

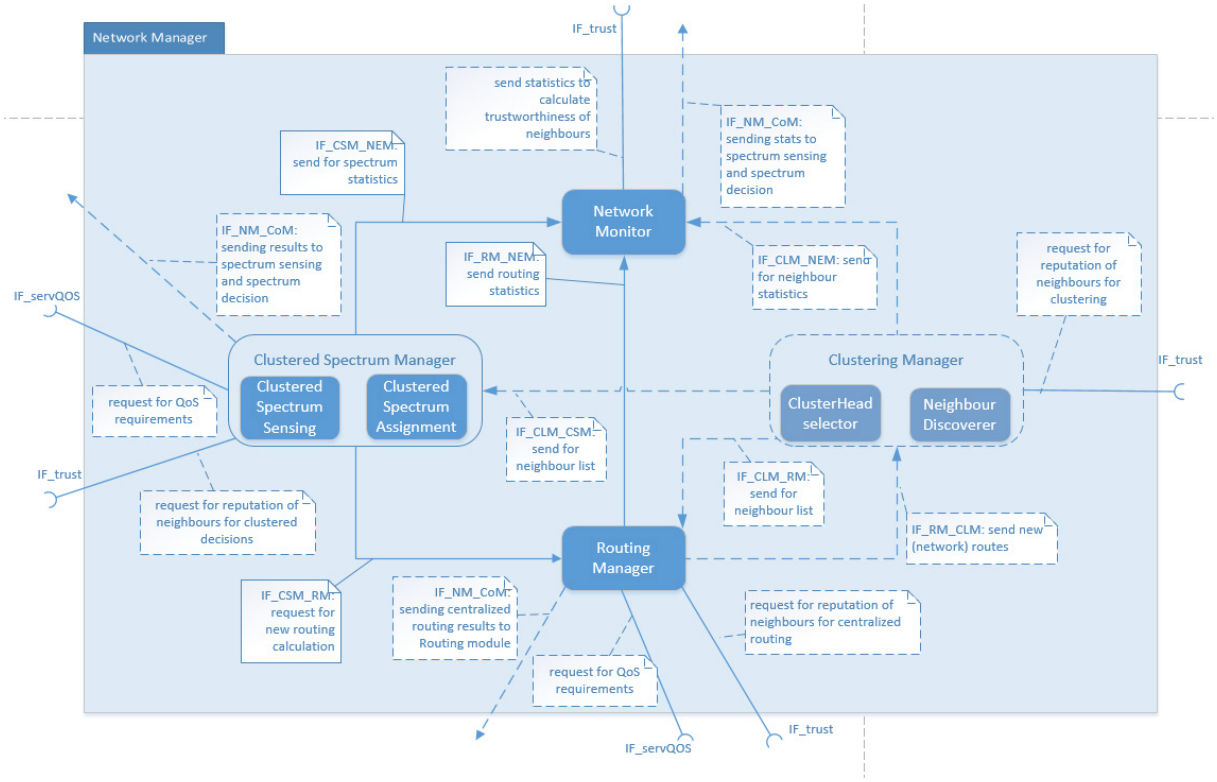


**Figure 28 – Network manager internal components and interfaces**

Another architectural requirement was related to the system resilience and robustness, meaning that the system must be able to prevent failures and to be able to recover quickly from failures without affecting significantly the system performance and operation. Furthermore, the system should ensure high availability of the device data, meaning that the devices should be able to send them (according the QoS requirements) whenever they are requested, regardless of the available network technology. We can link these requirements with the "**Clustered Spectrum Management (CSM)"** module, which is directly related with the CR Agent of the Communication Manager [35][36], enabling the cooperative implementation of spectrum sensing and spectrum assignment. The CSM module can work in either a centralized or a clustered approach and is responsible for managing the network interconnectivity of the nodes that are within the same network. In this respect, the nodes will be able to exchange spectrum availability statistics in order to identify the behavioural patterns of the available wireless spectrum and ensure optimal decisions regarding spectrum assignment. Utilizing collective knowledge on the status of the spectrum will ensure the avoidance of effects like shadowing, which lower the performance of wireless networks. In this respect, clustered spectrum management approaches would be quite suitable for managing the spectrum in small areas, so that the spectrum can also be re-used among distant clusters. Thus, this module is responsible for exchanging spectrum related statistics with other neighbour devices, in order to perform clustered or distributed (according to the topology requirements) spectrum sensing and assignment. The specific protocols used within RERUM were described in D4.1. When accurate estimations about the unutilized frequencies are available at the devices, the latter can take better decisions on which frequencies they will use either in a clustered or a centralized approach and this can prevent link congestion and link failures. Furthermore, the

spectrum management module can also respond very quickly to failures and security events (e.g. jamming) performing fast spectrum handovers and utilizing new frequencies to continue the smooth operation. The CSM module is directly connected with the Communication Manager through the IF_NM_CoM interface, which is used for sending the results of cooperative spectrum sensing/management for optimization of local decisions.

The requirement for high scalability of the RERUM system and supporting a large number of devices, as well as the requirement for allowing centralized management of constrained networks and the requirement for partitioning the network into clusters are all identified as key requirements for ensuring a high performance of the RERUM network. In this respect the "**Clustering Manager"** component performs the clustering of the network as a suitable solution for improving the scalability of the networking mechanisms [17]. Hierarchical clustering and layered clustering approaches can be used in large wireless sensor networks in order to allow more flexibility in managing the network and save energy in transmissions. The **"ClusterHead Selector"** is the component that allows the nodes to select (through a specified procedure) the cluster heads (that in most cases can be devices with more resources) and forward the data to them (when we consider the data plane) or send network monitoring statistics in order for local management of the network. The cluster heads are communicating with each other in order to e.g. forward the data until the gateway or decide about spectrum management or routing. For efficient clustering algorithms there is a need for accurate and efficient neighbour discovery, so that each node knows who its neighbours are in order to communicate with them to elect the cluster head or to propose them to join the cluster [47]. The specified module for **"Neighbour Discoverer"** is responsible for exactly this functionality. The neighbour discovery can assist in most of the network management mechanisms, i.e. for routing or spectrum management. Within RERUM, the notion of "cluster" and "neighbour" can also be extended to consider not only geographic location but also semantics, i.e. within a large area we could cluster together nodes based on the service they provide and this type of optimization may be critical for the network performance.

**"Routing Manager"** is the functional component that is responsible for creating and updating the routing table of each device. It is the component that executes the routing algorithm (at the network layer) that runs within the network (as i.e. the RPL [48] protocol in wireless sensor networks or the OLSR [49] in wireless mesh networks) and decides the next hop addresses and the respective interfaces. The Routing Manager module is thus the one that communicates with the respective modules of the other devices in the network to identify the optimized network paths for the current services. Due to the dynamic nature of the wireless spectrum these network paths may not be always the serving the service QoS requirements, so routing updates are also executed frequently. This can also be triggered when new services are requested from some nodes and the new data flows may congest some network routes causing low performance. Routing management in cognitive radio networks is very often related with spectrum management, since the routes should include links with nodes operating in the same frequency. In clustered approaches, the cluster head decides or assists the decisions about the routes within the cluster and then the cluster heads decide about the routes between them. As shown in Figure 28 the Routing Manager module sends the results of the routing mechanisms to the Routing module of the Communication Manager (described in Section 3.4.1) in order to update the routing table.

Interfaces between internal modules

As seen in Figure 28, the internal interfaces between the modules of the Network Manager are the following:

- IF_CSM_NEM: this interface is accessed by the Network Monitor component for requesting statistics regarding the spectrum occupancy, the link quality, IF throughput, etc. from the Clustered Spectrum Manager component.

 

- IF_CSM_RM: this interface is accessed by the Clustered Spectrum Manager component to send spectrum related statistics to be used by the Routing Manager for a new route calculation utilizing a spectrum-aware routing mechanism like in [73].
- IF_RM_NEM: this interface is used by the Network Monitor component to receive routing statistics by the Routing Manager.
- IF_CLM_RM: this interface is used by the Routing Manager in order to request the neighbouring list that will be used for the clustered/centralized routing mechanisms.
- IF_RM_CLM: this interface is used by the Clustering Manager in order to receive the new routes from the Routing Manager and initiate the clustering procedures to change/optimize the clusters if necessary.
- IF_CLM_NEM: this interface is used by the Network Monitor in order to receive statistics regarding the clustering mechanisms.
- IF_CLM_CSM: this interface is used by the Clustered Spectrum Management module in order to receive the neighbor list for performing the clustered spectrum management techniques.

## 3.5 Service Manager

As mentioned in Section 3.1, the Service Manager is responsible for handling both VE and RERUM Services. It consists of the following functional components (see Figure 29) that RERUM has borrowed from IoT-A, since it is not within RERUM's scope to deal with implementing these high-level functionalities:

- VE Resolution
- VE Service
- Service monitoring (which includes the monitoring of both VE Services and RERUM Services)
- RERUM Service Resolution (substituting the term IoT Service Resolution that is used in IoT-A).
- RERUM Service (substituting the term IoT Service that is used in IoT-A).

Figure 29 shows also the external interfaces of this functional entity as they have been described in Section 3.1.

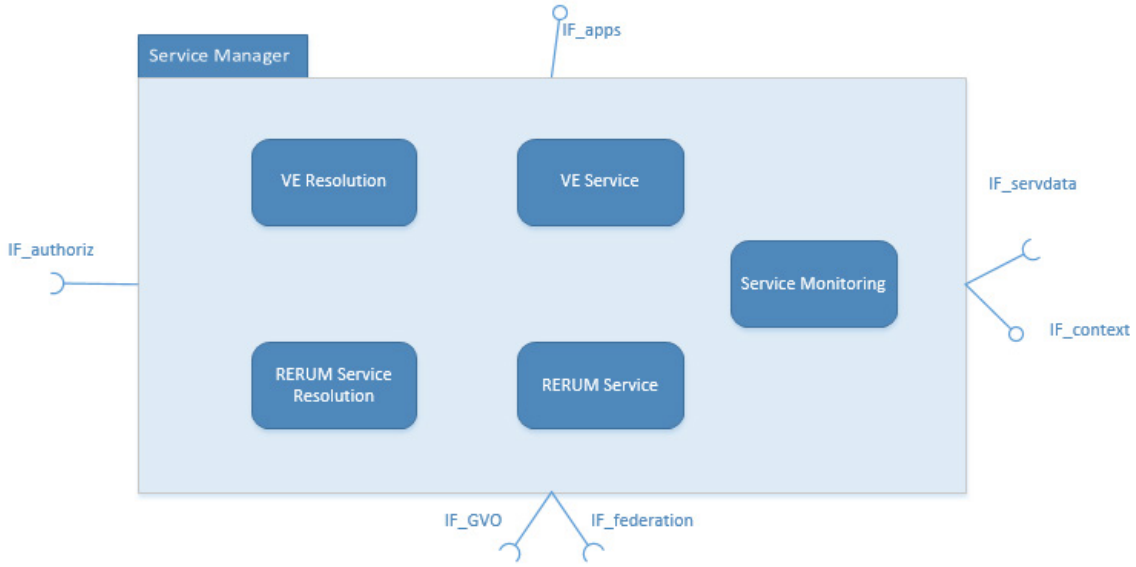


**Figure 29 – Service manager functional components**

### 3.5.1 Service/Resource model – ontologies

RERUM intend to support the facile and secure development of IoT applications regardless of the underlying exposed physical resources. RVDs are supporting these goals by bridging the low level sensing and actuation capabilities of various devices to the their virtualized and anonymized interfaces.

IoT applications involve the usage of sensing and actuation services applied to various physical entities, which are strongly dependent on the scenario and the actual context of the execution. As a result, predefined, hardcoded resource mapping is not an option due to high dynamics of the IoT environment.

In order to support the addressing of the required physical resources as well as the addressing of the associated access services, a proper description mechanism had to be adopted. RERUM makes use of the linked data paradigm when describing not only the exposed virtual entities but also the services, which expose these entities. This kind of metadata, which benefits from the linked data capabilities, is stored in the RERUM semantic stores. The stores provide the description mechanisms as well as a flexible retrieval interface.

The semantic description is also extended at SLA level in order to provide a generic mechanism for contracting as well as for monitoring service KPIs.

As a result, the RERUM ontology has been developed. An overview of the ontology relations can be seen in Figure 30.

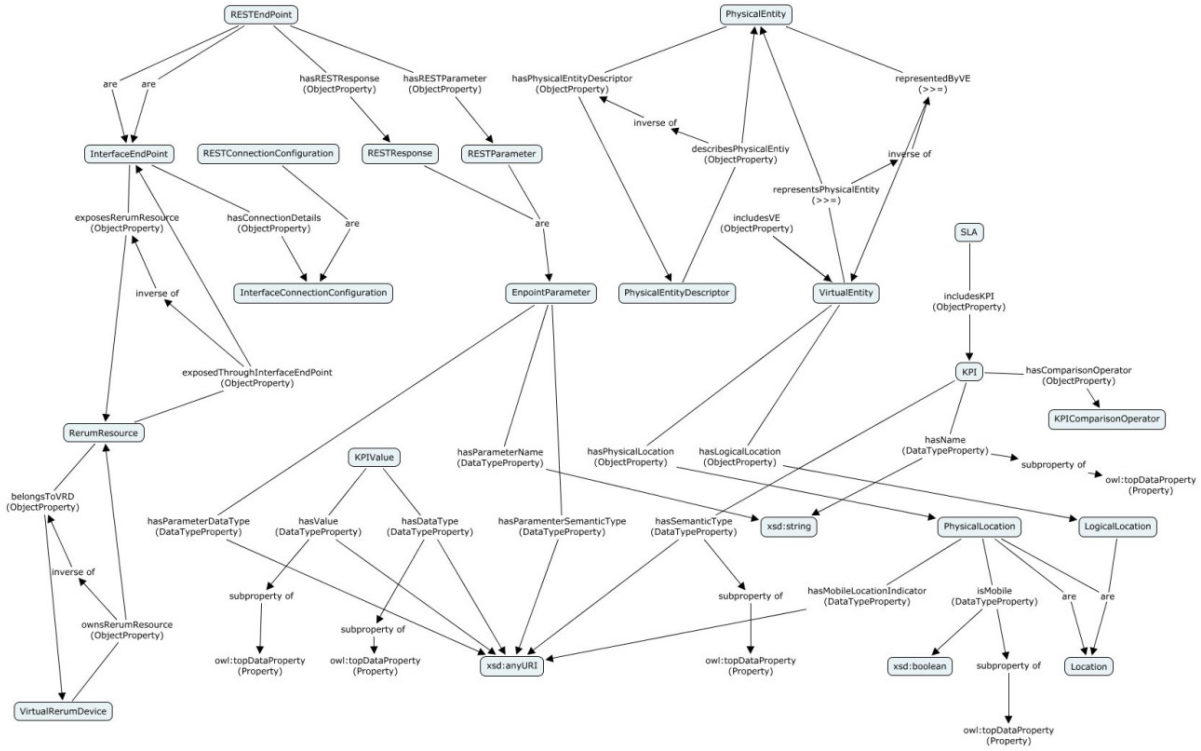


**Figure 30 – RERUM Information Model.**

The key concepts of the ontology and the most relevant relations will be described in the following paragraphs using a cloned mode representation. This means that, for the sake of readability, some of the classes might be duplicated in the graph representation.

 

Figure 31 depicts the relations which link the low level physical entities to their virtual counterparts, the virtual entities. Along with the actual information regarding the description of the physical entities, the virtual representation also includes an extended description of the location. The reason for placing the location description at VE level is to provide a uniform description and retrieval mechanism based on the actual physical entity location. Besides the physical location based on the entity's absolute coordinates (latitude and longitude) the information model includes the support for the description of a logical location as well as that for mobile entities. The logical location describes the VE's position from a semantic perspective (e.g a VE is located on room 502, on the 2nd floor of a specific business building from a specific town). In the case of mobile entities, their location cannot be stored into the semantic store due to the constraints related to the expected frequent updates. Nevertheless, along with the attribute indicating a mobile entity, the model also includes a placeholder for the entity's location indicating service. This acts as a referral mechanism which transfers the location indication to an external service.

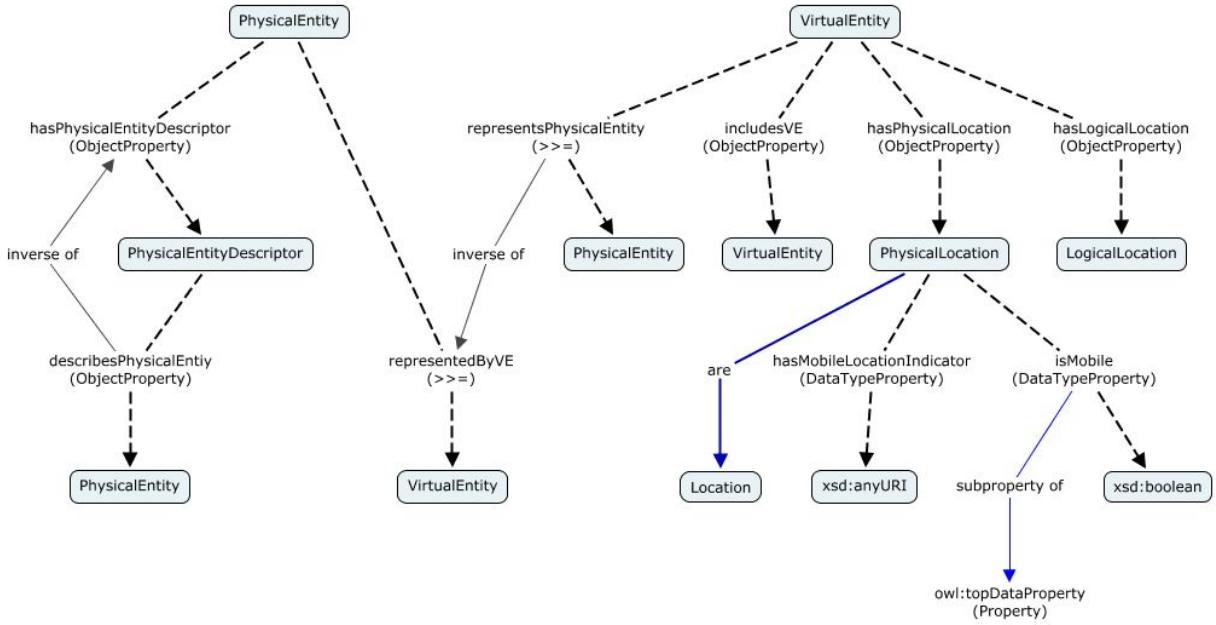


**Figure 31 – RERUM Ontology - partial view 1.**

In Figure 32 the relations between the virtualized device (the Virtual RERUM Device) and the externally accessible endpoints are depicted. RERUM resources, which provide the first level of virtualization, on top of the actual sensors and actuators encapsulated into a VRD, are exposed through REST endpoints.

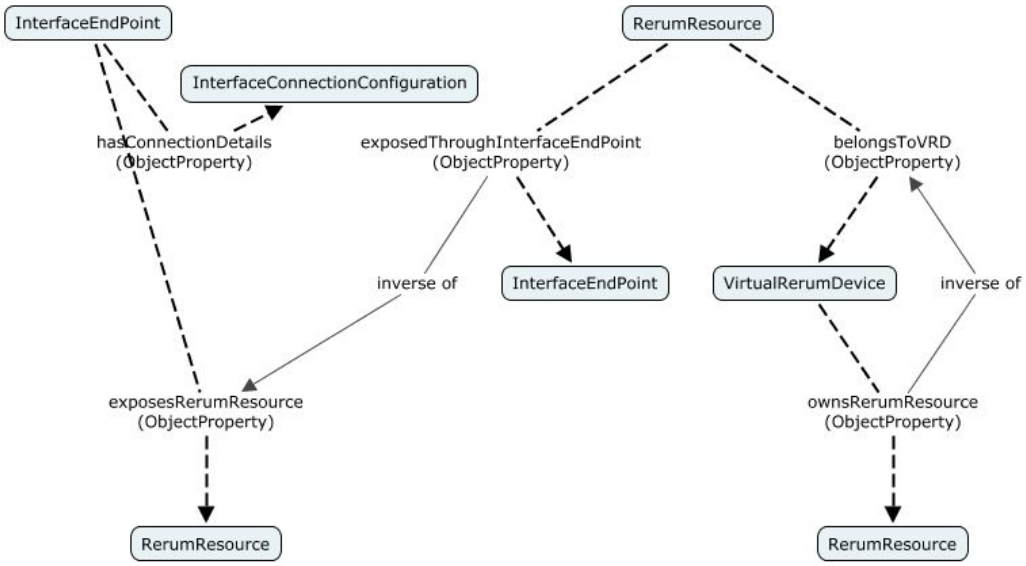


**Figure 32 – RERUM Ontology - partial view 2.**

Figure 33 depicts a brief description of the RERUM interface endpoints. As seen, the description of a RERUM REST interface does not include only the data type description but also the semantic one. This way we can express the fact that, for instance, a service not only returns a double value but a double value representing a temperature expressed in Celsius, thus extending the description capabilities.

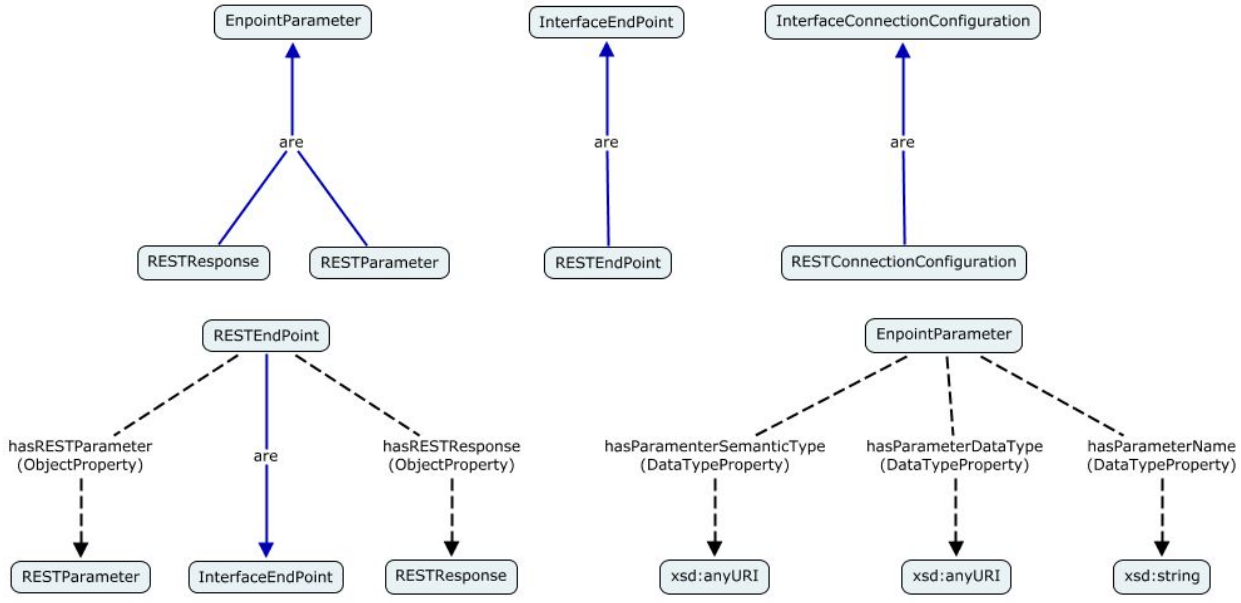


**Figure 33 – RERUM Ontology - partial view 3.**

The support for SLA (Service Level Agreement) descriptors has been designed so that it is generic and application or domain independent. An SLA is described as a set of triples describing KPIs (Key Performance Indicators) as seen in Figure 34. These triples include the KPI name, a comparison operator and a value acting as a threshold during the KPI evaluation. Besides the name, a semantic descriptor of the KPI is also supported.

 

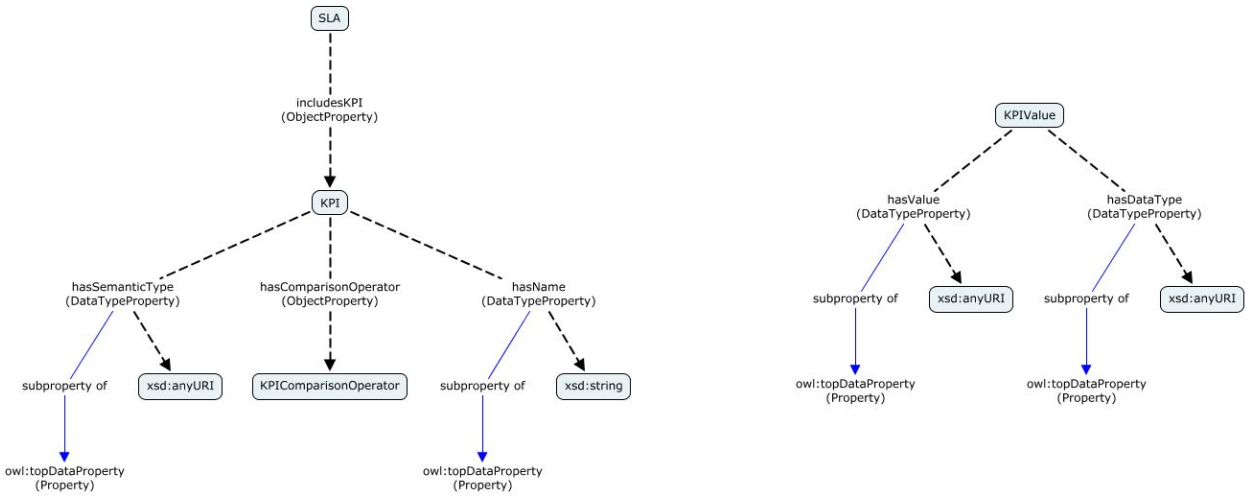


**Figure 34 – RERUM Ontology - partial view 4.**

As a result, each RERUM interface endpoint can be accompanied by an SLA descriptor set which can be used by monitoring applications.

All the above-mentioned semantic descriptors facilitate a rich description of the RERUM entities and their retrieval based on multiple criteria.

## 3.6 GVO Manager

The GVO Manager is responsible for identifying, registering and managing GVOs, as well as for discovering VRDs. It consists of three functional components as shown in Figure 35 and explained below. This figure also shows the external interfaces of the GVO Manager as described in Section 3.1.

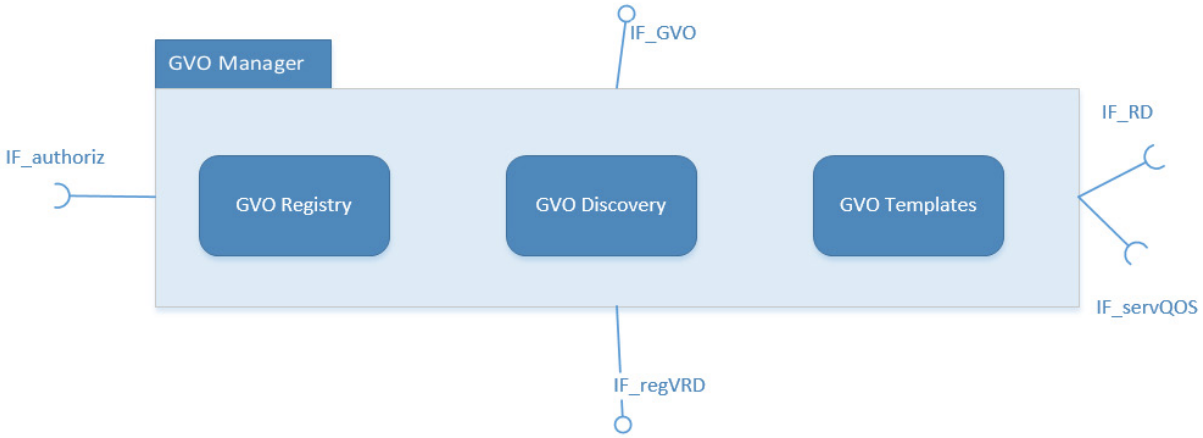


**Figure 35 – GVO Manager functional components**

### 3.6.1 GVO Templates

The GVO Templates functional component is a repository that stores a query-able collection of unique GVO templates in a specific format. Each template contains a semantically functional and syntactic description of its capabilities and, respectively, of its data formats. A GVO template can describe either a simple RERUM Device, a Physical Entity or a more complex VRD Federation - a composition of other VRDs or other federated VRDs, cooperating in an orchestrated effort in order to satisfy a specific condition or configuration for a needed Service (e.g. that of combining sensing and actuation

operations). The creation of VRD Federations is handled by the **VRD Federation Generator** described in Section 3.7.

### 3.6.2 GVO Registry

The GVO Registry contains a collection of existing GVOs (VRDs and VEs) linked to various contexts, resources and – in case of VRDs, to VEs. The registry, while conceptually being presented as a single centralized entity, can be in practice distributed among several domains (i.e. several instances of the Registry can exist on the RERUM Gateways, for identifying the RDs that are connected to the Gateway). Each entry represents a valid GVO acquired by a trusted agent and contains a semantic description using Resource Description Framework (RDF) triples, such as its name or unique identifier, its location, and its functionalities. In case of VRD Federations, the description may also include the context (VE Service) for which the federation was created, as well as its compositing VRDs.

### 3.6.3 GVO Discovery

Similar to its IoT.est Service Discovery component counterpart, the GVO Discovery component's task is to retrieve VEs or VRDs based on some criteria specified by the user. The search uses the VE/VRD descriptions stored in the previously described GVO Registry component, and uses a logic based language for modelling the query. In addition of retrieving plain VE/VRD entries from the Registry, the Discovery component also identifies the VE/VRDs for a previously defined Federation based on a logic which factors the user's and application's requirements, the available VEs/VRDs and their resources (as described by the Resource Monitor), special and temporal assumptions. The Discovery component might have to consider and mitigate the dynamic factor introduced by the mobility of the nodes and, if necessary, to reactively handle subsequent changes such as reassigning new VEs/VRDs to the Federation in order to compensate for variations in VEs/VRDs availability, readings or precision – scenarios which are detected by the Resource Monitor component.

### 3.6.4 Message sequence charts

Figure 36 shows an example of a message sequence chart for discovering a VRD that can serve a specific user application request. The VE Resolution component receives the service request and then contacts the GVO Discovery to identify the VEs that are associated with the request. Then the VE Resolution component contacts the RERUM Service Resolution component to identify RERUM Services that can serve the user application request through the service associations. The RERUM Service Resolution component identifies the RERUM Service that can serve the request and contacts the GVO Discovery component to find the suitable VRD that exposes this service. The GVO Discovery component searches the GVO Registry for the VRD templates and sends the response back to the RERUM Service Resolution component for accessing the specific RERUM Service of that VRD.

 

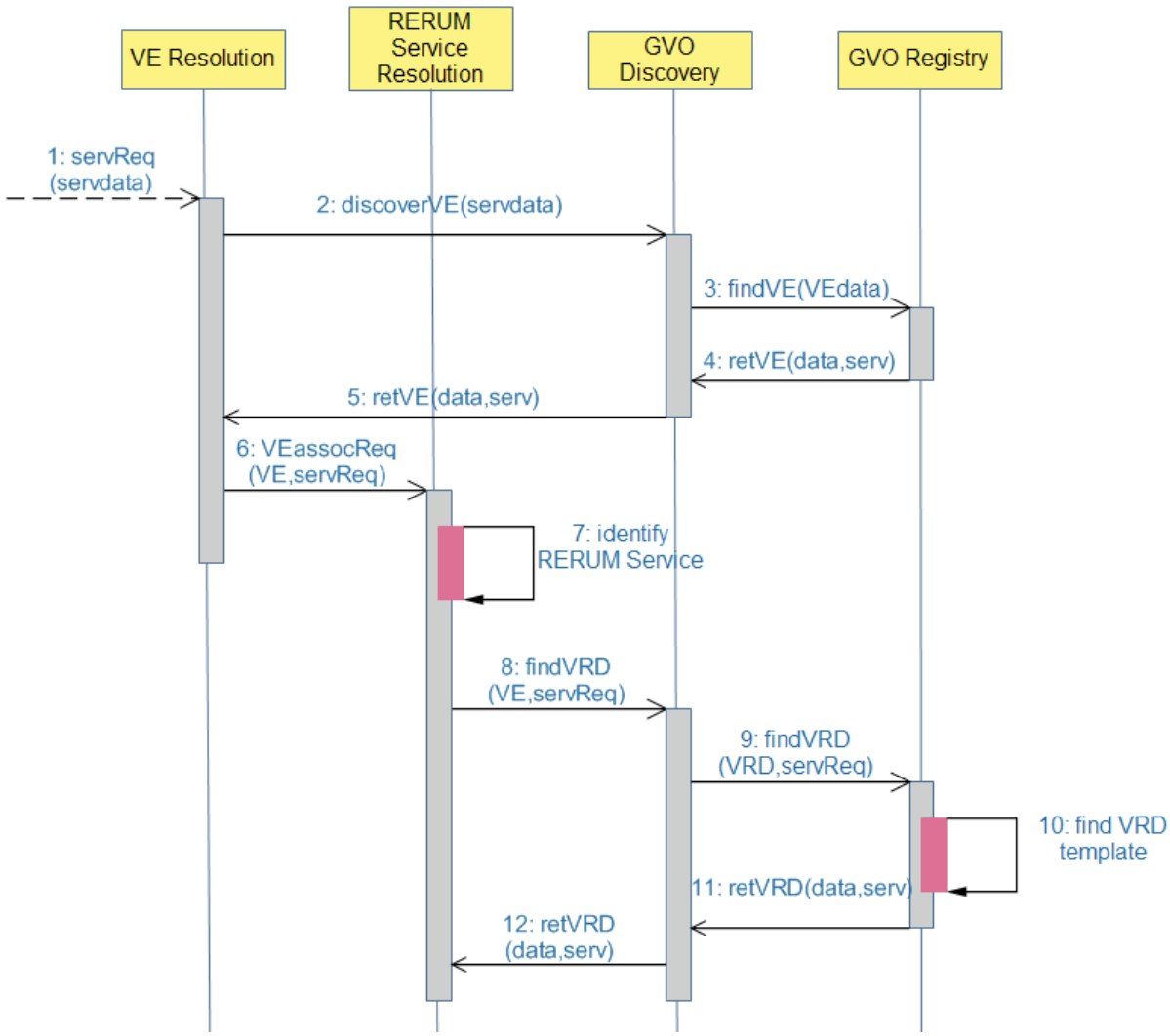


**Figure 36 – Message sequence chart for VRD discovery.**

## 3.7 Federation Manager

The Federation manager is responsible for generating and executing Federations of VRDs that are required to provide a composition of services associated with VEs. The Federation Manager receives a federation request, generates and executes a new Federation, when the user request cannot be met by a single RERUM Service. The request comes from the Service Manager and in order to generate a Federation, the Federation Manager communicates with the GVO Manager in order to identify the VRDs that have to be federated. Basically, the Federations can be seen as compositions of RERUM Services with or without some business logic. In the first case we can imagine an example of the air-condition controller, the temperature sensor inside the house, the temperature sensor outside the house and the window controller create a Federation in order to provide proper room temperature with minimum energy consumption, i.e. when inside_temperature > 25 degrees and outside_temperature < 20 degrees open window and shut down the air-condition. The second case can be seen as a simple filtering or aggregation of data gathered through invoking services of many sensors. The Federation Manager consists of two functional components as shown in Figure 37 and has three external interfaces as described in Section 3.1.

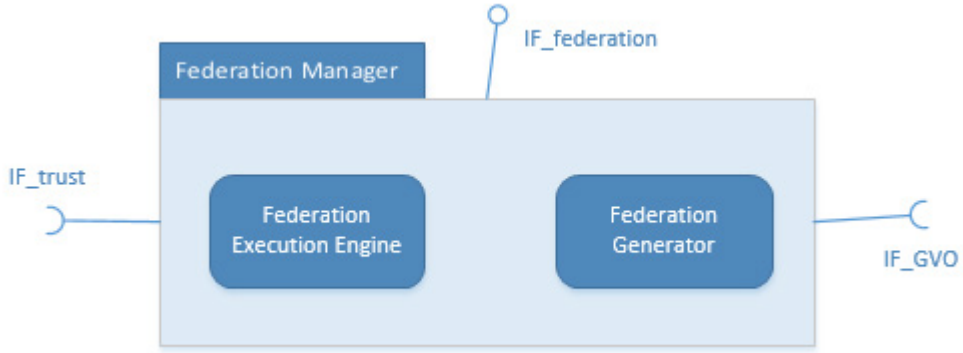


**Figure 37 – Federation manager functional components**

### 3.7.1 The Federation Generator

The Federation Generator is responsible for generating new VRD Federation instances. The creation of new federations can occur when the service request by the user/application cannot be fulfilled by a single RERUM Service, but only by grouping a set of VRDs and composing their Services. The VRDs in the federation's composition are chosen by the Federation Generator based on the relevant required properties, i.e. the type of sensor the corresponding RDs have, the quality of their measurements, if they have actuators, the physical location or the available resources and computation capabilities. Besides the previously described functional composition, the Federation Generator might also define hierarchical compositions – e.g., VRDs of rooms as part of VRDs of a building, part of VRDs of a block.

### 3.7.2 The Federation Execution Engine

The Federation Execution Engine is the main functional component of the Federation Head and is responsible for executing the program associated to a VRD Federation. The program which needs to be executed by the component is specified by the application in its requests for the IoT service. Within this document we frequently utilize the term Federation Head to refer to the functional entity that is responsible to manage and execute Federations.

### 3.7.3 Message sequence charts

Figure 38 shows the message sequence chart for the creation of a Federation, which is explicitly requested by the RERUM Service Resolution component when it cannot identify a single RERUM Service that can meet the user application requirements. The Federation consists of

- The description of the required resources and properties,
- The description of the logic which is to be executed.

The VE Resolution component identifies first the VEs that are associated with the user application request and then contacts the RERUM Service Resolution component to identify RERUM Services that can serve the user application request. When no single RERUM Service can do that, the RERUM Service Resolution component sends a federation request to the Federation Generator. The latter contacts the GVO Discovery component to identify the VRDs that can perform the federation and if successful, a Federation template is generated and registered to the GVO Templates. A Federation template contains, (i) the description of the required resources extracted and semantically annotated with both service request and returned VRDs-extracted information, (ii) the logic itself which is to be executed

 

by the Federation Execution Engine, and (iii) a description of the type of logic, e.g. if-else rules used for automatically adjusting of the ambient temperature or used for the lowering/rising of a roller blind, which can be later used to during the search for already existing Federations performing a certain role. Finally, an instance of the federation is also registered to the GVO Registry.

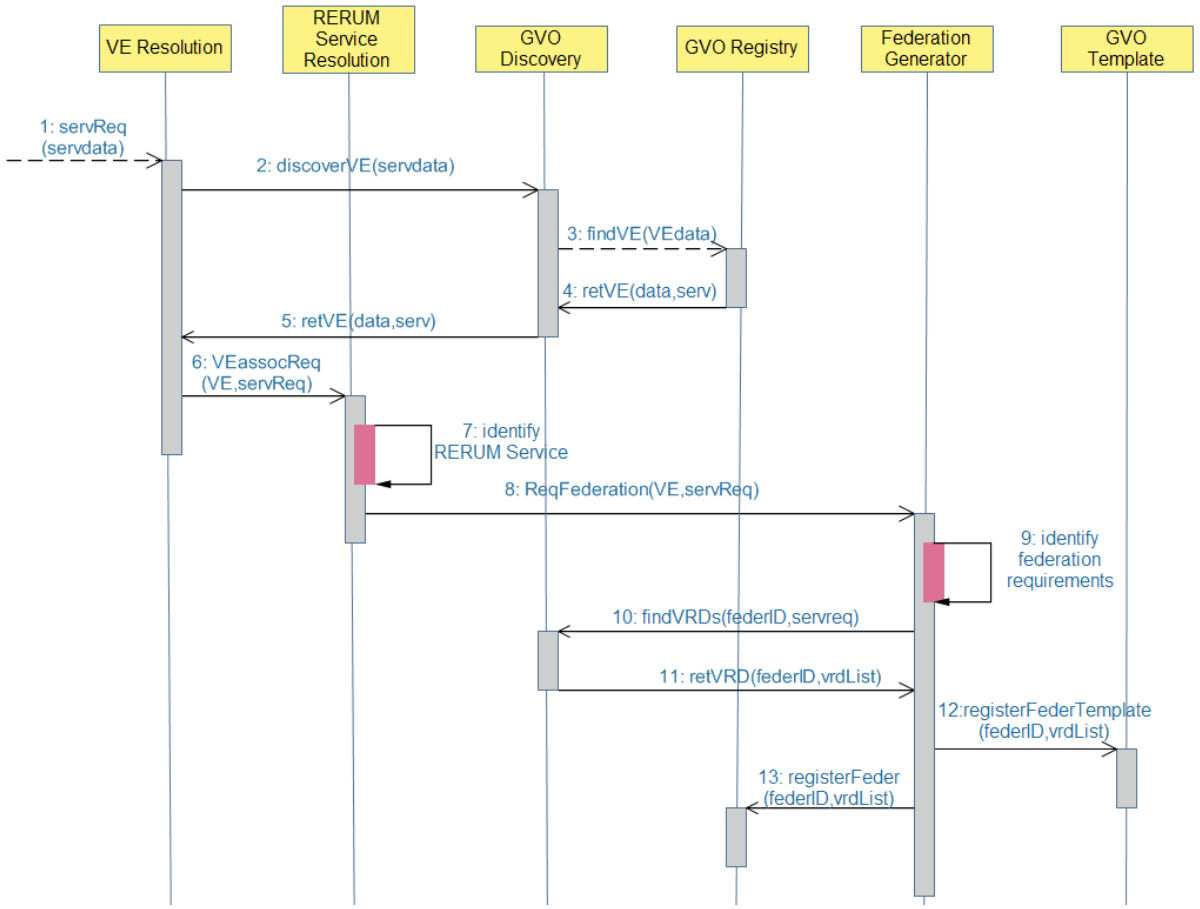


**Figure 38 – Sequence diagram depicting the creation of a Federation by the Federation Manager.**

## 3.8 Configuration and Monitoring Manager

This functional entity is responsible for (i) the configuration of both the middleware components and the RDs, (ii) their bootstrapping, (iii) the monitoring of their state and of their resources and (iv) the creation of alerts in abnormal situations. The Configuration and Monitoring Manager (depicted in Figure 39) consists of (i) the Configuration Manager, and (ii) the Monitoring manager, which in turn consist of several internal components, as described in the following paragraphs. The figure shows also the external interfaces of the functional entity as described in Section 3.1.

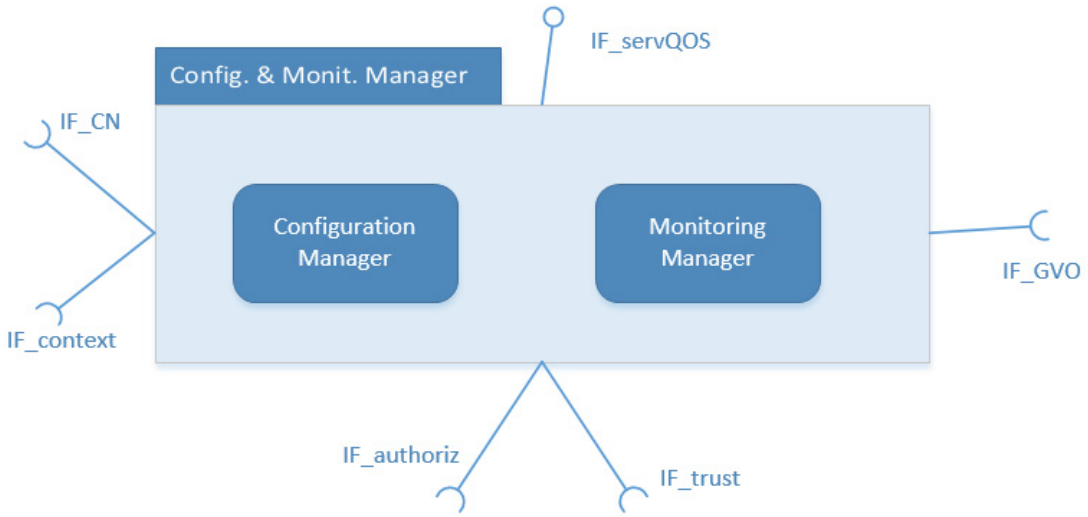


**Figure 39 – Configuration & Monitoring Manager functional components**

### 3.8.1 Configuration Manager

The Configuration Manager is responsible for configuring and re-configuring the RDs, as well as for initializing the various MW components with appropriate settings in a bootstrap stage. It consists of six functional components as seen in Figure 40 and explained in the subsections below.

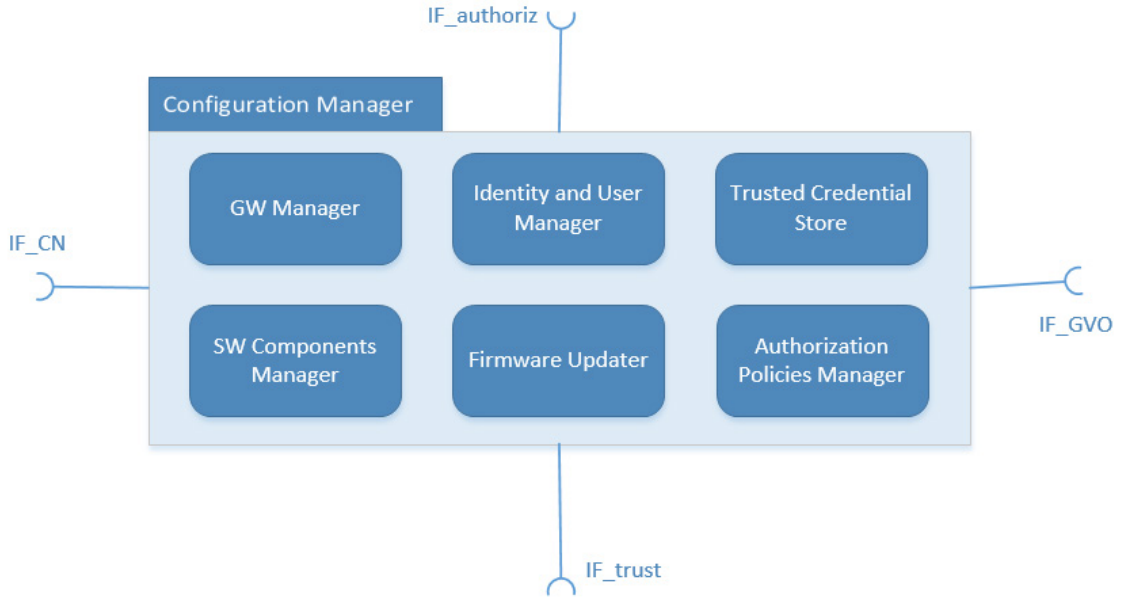


**Figure 40 – Configuration Manager functional components**

#### *3.8.1.1 Gateway Manager*

The role of the Gateway Manager is to generate and deploy gateway instances, which then detect, bridge and register RDs – and therefore their VRD provided services - with the RERUM middleware. Gateways can be generated according to various criteria like the physical location, various organizational or hierarchical domains, the requested service and features, etc.

 

#### *3.8.1.2 Identity and User Manager*

The Identity and user Manager is responsible for:

1) Guaranteeing that the user that is trying to access the system is who he claims to be, by checking that he is providing valid trusted credentials that have been previously assigned to him.
2) Providing a set of attributes associated to the user and guaranteeing that their values are the ones provided by the administrator;
3) Letting an administrator of the system to manage the above features

When managing the Users, this module receives the information of the user and its associated credentials and outputs an "OK" and an update of the user information. When checking the user's identity, this module receives the credential and identity of the user and outputs a decision with the attributes of the id of the user.

#### *3.8.1.3 Trusted Credential Store*

This component is responsible for storing the credentials in a secure and trusted way. In this respect, the store restricts unauthorized read or write access to credentials stored in it. Regarding read and write access this means that RERUM assumes to store secret credentials, like cryptographic tokens and keys in this store and assumes that the access to those is restricted to only the component that is authorized to access the credential. With respect to only write access, the Trusted Credential Store allows to provide a public, but read-only, list of public keys that are associated with another component or service and that this relation is marked as trusted. This would allow storing a public-key certificate of the application server, a certification authority, or another trusted entity here. The Trusted Credential Store can be built with the help of the Secure Storage component (described in Section 3.10.1.10).

#### *3.8.1.4 SW Components Manager*

The SW components manager is responsible for managing the software installed on the RDs. It has the following functional components:

- **RD Adaptor**, although is not part of the SW Components Manager, is the last and necessary step for finally update the firmware in the RERUM Devices. This Adaptor is the part of the Middleware located inside the Devices and it supports the update firmware process with its Firmware Installer module as is described in section 3.3.
- **OAP downloader** is the component responsible for downloading the SW components (also called artefacts) from the External Trusted Repository (see below) stated by an URL and checking their integrity and precedence. It is also responsible for invoking the OAP Builder once it has downloaded the artefact. The input of the OAP downloader will be an URL from where get the artefact. The Output of the OAP downloader will be the artefact downloaded or an error if there is an error in the downloading process or the downloaded module fails its verification.
- **OAP Builder** is the component responsible for building the artefacts downloaded in order to obtain a firmware. After building the firmware, it will sign it so its integrity can be certified later. It is also responsible for invoking the OAP deployer for the suitable groups of RDs.
- **OAP deployer** is the server counterpart of the RD Adaptor. It is responsible for selecting the suitable RD(s), which will have to receive the built SW component. This component makes use

of the VRD Discovery component of the Middleware to find out the suitable RDs that require a SW upgrade. The OAP deployer receives as input the criteria for selecting the available RD plus the firmware to be deployed along with its signature. The OAP deployer is not responsible for checking the validity of the library to be deployed, as this has already been done by the OAP builder. The OAP deployer will send the firmware and its signature to the Middleware. The output of the OAP deployer will be "ok" if everything worked, otherwise it will show the errors produced.

- **Trusted Repository** is a component **not part of the RERUM platform** as it is meant to be provided by third parties, but it is explained here to understand how it interacts with RERUM. The Trusted Repository is responsible for providing the artefacts for each suitable platform and for listing the set of artefacts that it provides. The input of the Repository will be the name/URL of the artefact to be provided and the metadata tags that allow the search filtering those artefacts. The output of the Repository will be the artefact requested for the OAP deployer or a list of artefacts available for that service to be sent for the PRRS finder, depending on which component invoked the Trusted Repository.
- **PRRS Event Manager** receives events, parse them and extract what kind of Alarm has been produced and the context information related for sending it to the PRRS Configuration.
- **PRRS Configurator** is a component in which you can add the Rules that define when a reaction must be taken and the Context Variables that can be used in those rules. The component is responsible for checking the compliance of the active rules every time that an event occurred and taking into account the context variables values in that moment for that concrete event. PRRS Configurator is also in charge of invoking the PRRS Finder when is required because of the compliance with a rule. The PRRS configuration is not a daemon, but a component that can be invoked either for a change on the configuration criteria or as the result of an incoming event. The input of the PRRS configurator will be a configuration based in a set of rules and variables and also the reaction to be triggered when the rule is met; it can be defined through the web GUI or the REST API. The output of the PRRS Configurator will be the criteria for searching suitable artefact or any other predefined reaction.
- **PRRS Finder** is a functional component responsible for finding in a repository the SW component that suits the criteria that are provided by the PRRS Configurator. The input of the PRRS finder will be the criteria provided by the PRRS Configurator. The output of the PRRS finder will be the location of the selected artefact. The following Figure 41 shows how the PRRS Configurator and Finder interact with the OAP component to get the components installed on the RD.

As Figure 41 shows, the PRRS Configurator provides the PRRS Finder with constraints that is the criteria for choosing the suitable artefacts. The PRRS Finder looks for a suitable set of artefacts (binary executable files ready to be deployed, named artefacts in the figure) and locates them on the Marketplace (that is not part of the RERUM system itself but will be provided as a proof of concept), and the location of those artefacts is supplied to the OAP [58] component so it can deploy it on the target RD, on their operating system (i.e. the ContikiOS [57]).

 

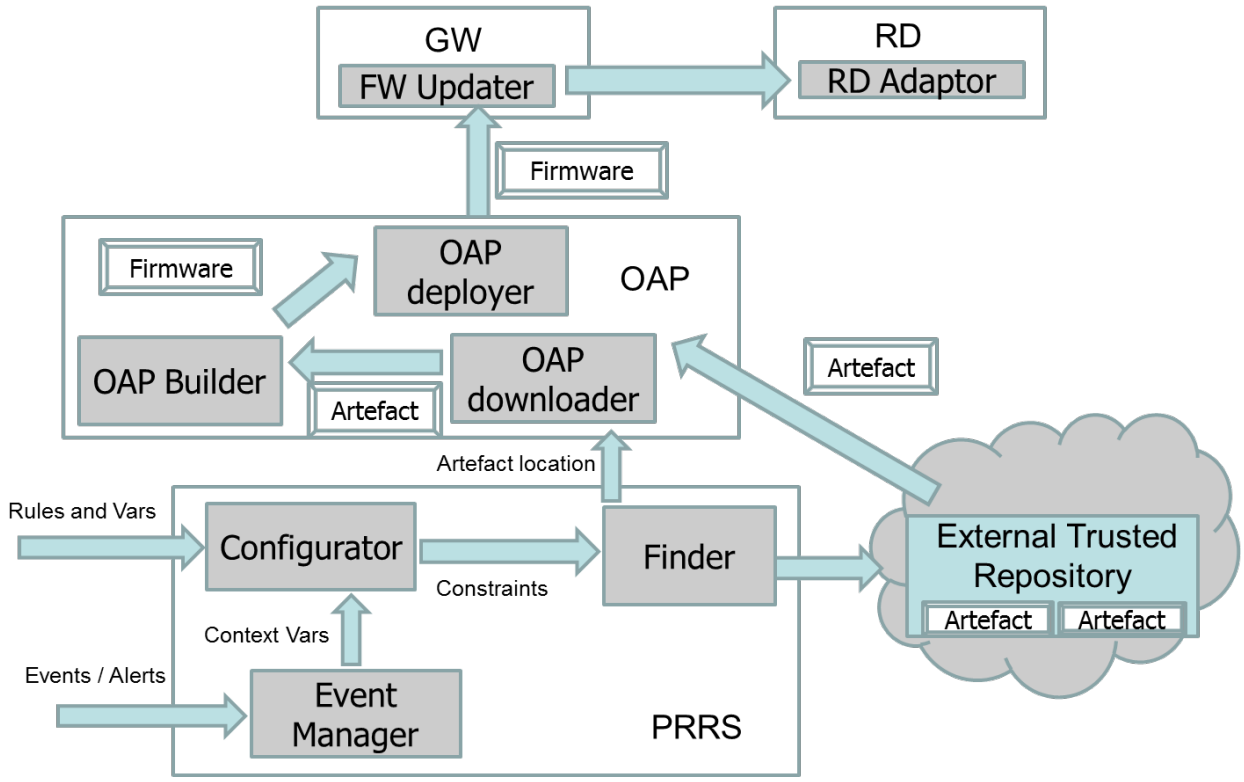


**Figure 41 – Installing new security components on the RD using the PRRS system**

#### 3.8.1.4.1 Interfaces between internal components

As can be seen in Figure 41 there are the following internal interfaces for the SW Components Manager:

- IF_EM_CONF: this interface is used by the Event Manager in order to send contextual information to Configurator regarding alerts and events.
- IF_CONF_FIND: this interface is used by the Configurator to send the constraints to the Finder for filtering the available solutions related to the firmware that should be deployed on the devices.
- IF_FIND_OAPDL: this interface is used by the Finder to send the selected Solution to the OAP for deploying it on the RDs.
- IF_OAPDL_OAPB: this interface is used by the OAP downloader to send to the OAP Builder the downloaded SW component and its signature in order to be built by the Builder in a format that can be understood by the RD.
- IF_OAPB_OAPDP: this interface is used by the Builder to send the built SW component and its signature to the Deployer for sending it to the RDs to be installed.

#### 3.8.1.4.2 Message sequence charts

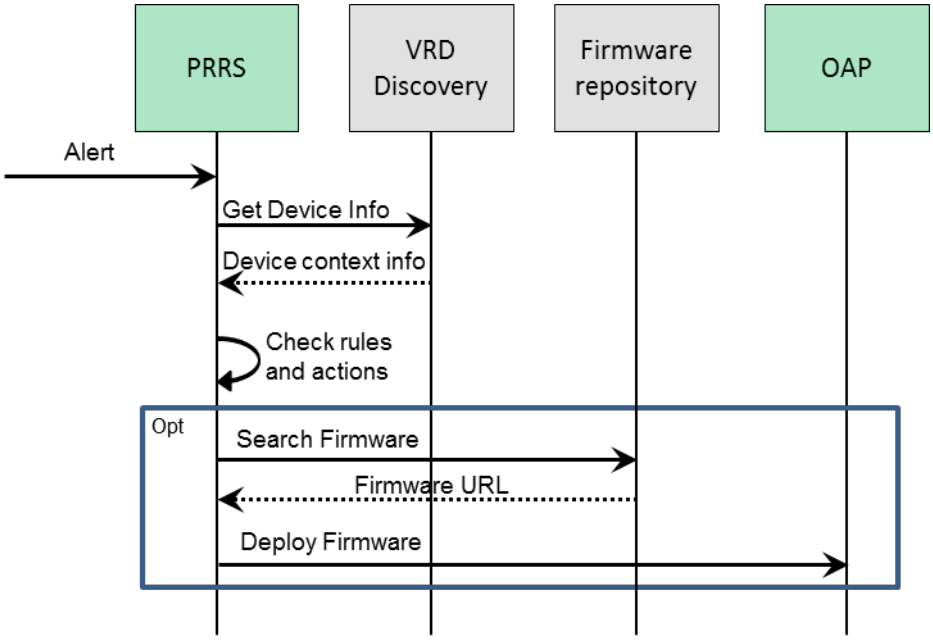


**Figure 42 - PRRS sequence chart.**

The firmware update process is started when the OAP is contacted by the PRRS as can be seen in figure 42. Then the OAP will initiate the firmware update process as can be seen on Figure 43.

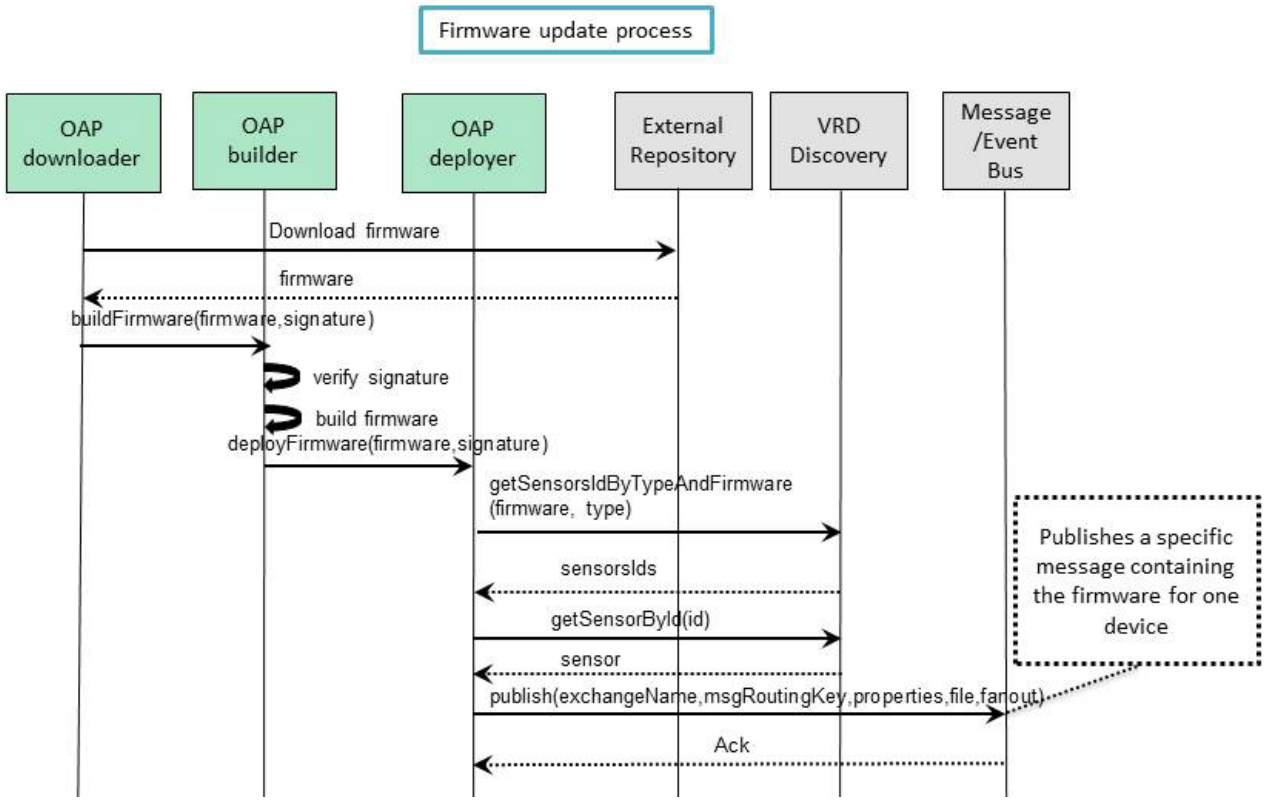


**Figure 43 - Firmware update sequence chart.**

First the OAP downloader gets the firmware from the external repository, then it sends it to the OAP builder, which verifies the signature and then builds the firmware. Once the firmware is built it contacts

 

the OAP deployer, sending it the signal to deploy the firmware. The OAP deployer will then query the VRD discovery to obtain the list of sensors that the firmware will be deployed to. Once it has the list the OAP deployer will contact the Message/Event bus to publish the firmware and the signature for each sensor, after which the OAP finishes its function in the firmware deployment process.

#### *3.8.1.5 Firmware Updater*

Firmware Updater is the component that is responsible for monitoring the firmware updates and sending these updates to targeted RERUM devices. The firmware updater runs inside the RERUM MW as a back process. It is designed to satisfy RERUM security requirements, so it uses DTLS over CoAP protocol to communicate with the RERUM devices.

##### 3.8.1.5.1 Internal Architecture

The Firmware Updater is composed of three different software modules, as shown in. It supports JSON topology of RERUM Framework.

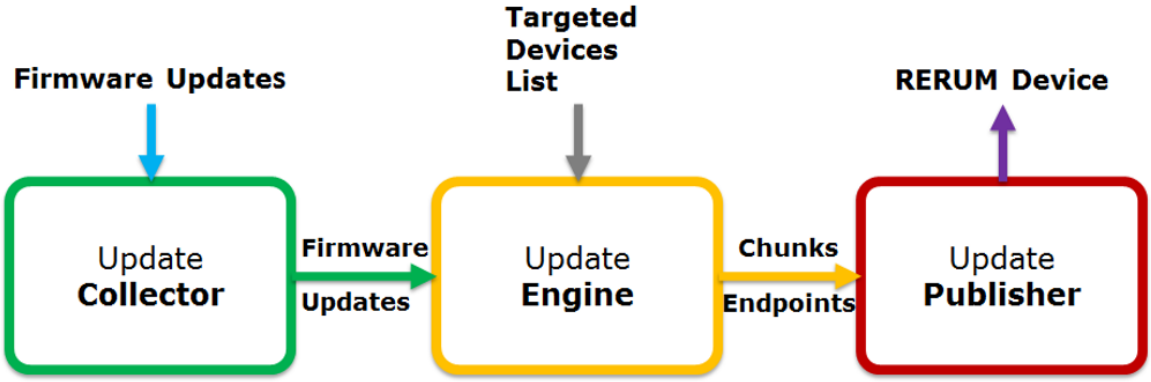


**Figure 44 - Firmware Updater - System Architecture.**

The main data flow of Firmware Updater is as follows: *Update Collector* subscribes to firmware update queues generated inside RERUM MW, collects updates from these queues and streams the updates to *Update Engine*. Input updates entering the *Update Engine*, are subjected to signature check for validation. If validated, the update file is split to 32-bit chunks. This chunk array and the list of targeted devices are forwarded to *Update Publisher*. *Update Publisher* module post the chunks one by one to targeted devices while checking the success code for each post event. If a problem occurs, this module repost the necessary chunks repeatedly. The three functional modules represented in this architecture are introduced in the following subchapters.

##### 3.8.1.5.2 Update Collector

The Update Collector is an internal module for Firmware Updater, which is responsible for subscribing to firmware update message queue, consume the data and stream it to Update Engine component. In RERUM, AMQP publish/subscribe mechanism is used to collect firmware update data. The module periodically checks for if any new update queues generated in the RERUM MW. If any changes occurred in the queues, update collector will subscribe to new ones, and unsubscribe from the omitted ones.

##### 3.8.1.5.3 Update Engine

The Update Engine is the main internal module for Firmware Updater. It is responsible for validating the firmware update file, creating 32-bit chunks and gathering the targeted RERUM Devices list. Once the module has been fed with a new firmware update file, as a first step it checks if its signature is valid or not. If the signature is not valid, this module ignores the received update file. After the signature validation, the file is split into an array of 32-bit chunks. Then the module queries the MW Gateway

 

module to receive the list of targeted RERUM Devices. This list includes the uri of the CoAP endpoint of the RERUM Devices. As the final step, Update Engine merges the array of chunks and the list of endpoints to an internal object and passes this object to Update Publisher module as an input.

#### 3.8.1.5.4 Update Publisher

The *Update Publisher* module is responsible for sending the firmware updates to RERUM Devices. It takes the internal object that consists of the array of chunks and the list of endpoints as an input and starts posting the values in the array to every endpoint defined in the targeted device list. The module uses DTLS over CoAP protocol to post the chunks. After each post, the module checks if the post event is successful or not. If it is successful, it posts the next chunk. If not it tries to post the last chunk. It repeats this process for 3 times. After 3 unsuccessful post events, it skips the endpoint, and starts updating the next devices in the list. After finalizing the update procedure, it informs the RERUM GW about the result of the updates as successful or not for each device as a list.

#### 3.8.1.5.5 Message Sequence Chart

The basic message sequence involving the Firmware Updater is shown in Figure 45 and described as;

1. Update Collector subscribes to Update Queue
2. Update Collector receives a new firmware update from the queue
3. Update Collector sends the update file to Update Engine
4. Update Engine queries the RERUM Gateway for targeted devices list
5. RERUM Gateway returns the targeted devices list as response
6. Update Engine creates the update object with the array of chunks and devices list and passes it to Update Publisher
7. Update Publisher posts the chunks to RERUM Device endpoint using DTLS over CoAP
8. RERUM Devices returns success/error as response
9. Update Publisher informs RERUM Gateway with the status of device updates

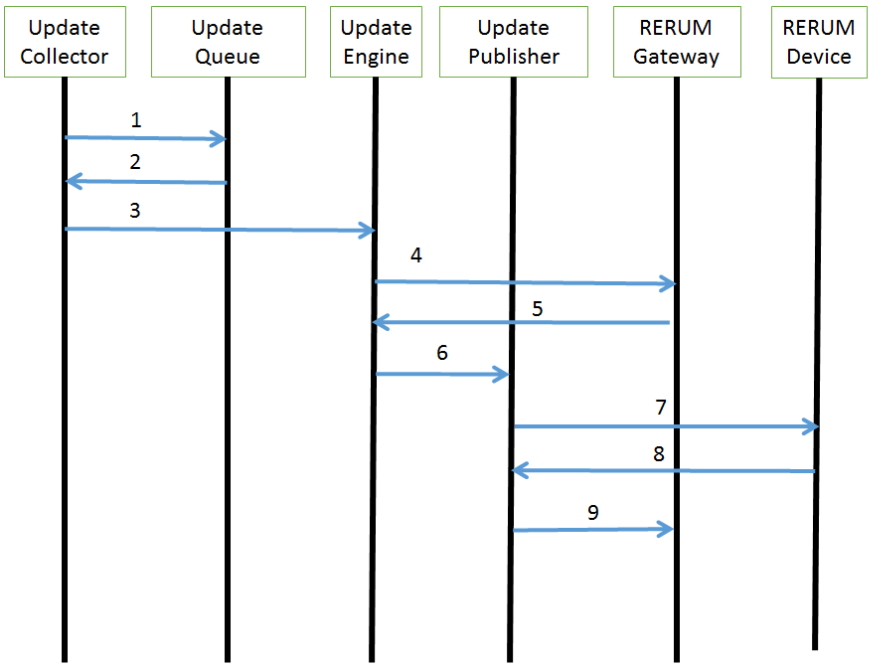


**Figure 45 - Message sequence for the Firmware Updater.**

 

#### *3.8.1.6 Authorization Policies Manager*

The authorization policies manager allows defining the security criteria used for deciding whether to grant or reject access to the RERUM system. These criteria will be saved in a standardized format so they can be obtained later by the PRP (see Section 3.10.1.9) to be provided to the PDP (see Section 3.10.1.8) so it can evaluate them against the request to take a decision. The Authorization policies manager supports the standard operations for adding, deleting and modifying policies. The Authorization Policies Manager takes as input the policy files provided by the administrator of the system and gives as output the Policy files to be saved to the **Security Policy Store**, which can be considered as a specific instance of the Trusted Credentials Store described above.

Figure 46 shows the relationship between the Authorization Policy Manager and the rest of the components involved in the evaluation of the security policies of the system:

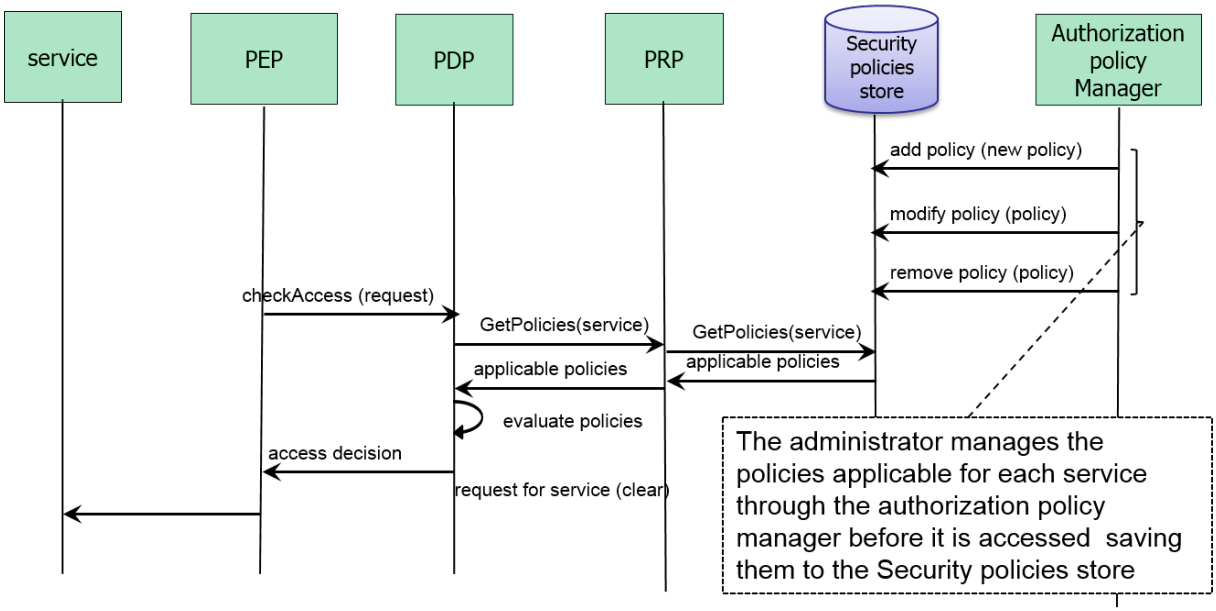


**Figure 46 - Managing security policies.**

Note: The interactions for the “modify” and “delete” operations are basically the same as for the “add” operation. That is, they modify the set of security policies applicable for a given request and these are finally supplied by the PRP.

### 3.8.2 Monitoring Manager

The Monitoring Manager is responsible for monitoring the state of the various functional components of the MW, logging their actions, raising alerts when there is an abnormal situation and acting in order to resolve it. Furthermore, it is responsible for monitoring the state and the hardware and network resources of the RDs and the provided QoS. It consists of four internal components as described in the following paragraphs and depicted in Figure 47.

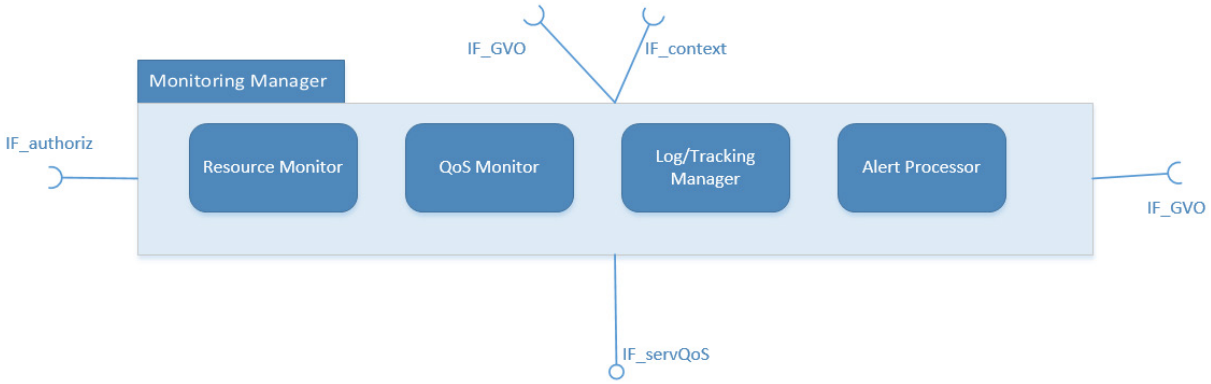


**Figure 47 – Monitoring Manager functional components.**

### *3.8.2.1 Resource Monitor*

The function of the Resource Monitor component is to provide a near real time (depending on the capabilities and the battery availability of the RDs) view of the hardware resources (i.e. memory, storage, CPU, battery, network connectivity, etc.) of each one of the available RDs. The state of the hardware resources is relevant especially during the selection of suitable VRDs by the GVO Discovery component. A second task of the Resource Monitor is to detect issues (readings of a given hardware or software resource which are below a context - or application relevant - threshold) and forward appropriate messages to the Alert Processor. The Resource Monitor is closely related with the Network Monitor functional component that was described in Section 3.4.2 and is included in the Communication and Network Manager. The idea is that the Network Monitor is the component that exists on the various types of RDs and also on GWs in order to allow the self-monitoring of the network components. These statistics are forwarded periodically to the Resource Monitor (through the RD Adaptor) in order to have a centralized view of the state of the RD's hardware resources within the RERUM system.

### *3.8.2.2 QoS Monitor*

The task of the QoS Monitor is to track the real time availability, utilization and performance of the registered VRDs receiving input from the Network Monitor that runs on the RDs. By analysing the data provided by the Log/Tracking Manager, deriving near real time statistics and comparing them to each service level objectives, the QoS Manager can ensure that SLAs are fulfilled by optimizing resource allocations in order to achieve a specific goal, e.g. to maximize the lifetime of the RDs, reduce measuring errors, etc.; finally, the QoS can provide feedback for further planning or SLAs negotiations.

### *3.8.2.3 Log/Tracking Manager*

The Log and Tracking Manager component has a dual role: At first, it logs all relevant events to, from and within the middleware layer. This includes (a) requests from users and applications, as well as (b) internal message exchanges, (c) configuration changes, (d) VRDs states, (e) resource usage, etc., in order to facilitate the auditing of the middleware components and RDs. The second role is to enable tracking of such events for each individual service that is provided. This enables, when required, the metering per user, application or service, which allows functionalities such as accounting and billing.

**Event Producer**

The Log/Tracking manager includes an "**Event Producer**" that is responsible for producing log entries that can be retrieved later from other MW components. The Event Producer receives the standard log output of each SW component in the system and outputs a persistent storage with the events produced from the logs retrieved in a common structure.

 

#### *3.8.2.4 Alert Processor*

The Alert Processor is responsible for processing specific alert messages within the MW, i.e. those concerning the normal operation of VRDs and MW components. In case of alerts related with VRDs, the Alert Processor needs to notify the appropriate components of the changes: e.g. to notify the VRD Federation Manager that VRDs which were initially included in a Federation cannot fulfil their role anymore. In case of MW-related alerts (e.g. the abnormal termination of one of the processes which provides Federation, QoS Management etc.), the Alert Processor needs to perform typical failback duties. The Alert Processor consists of the following components: (i) Alert Producer, (ii) Alert Reactor. In this project has been developed one of the possible implementations of an Alert Processor, namely the Inaccuracy Alert Producer, and the Inaccuracy Alert Reactor, but since these have specific focus on reputation and trust they are presented in Section 3.10.3.

##### 3.8.2.4.1 Alert Producer

The Alert Producer is not a single component in the system, but a set of them will be instantiated, one per each group of alerts to be generated on the system. The analysis of events and the alerts to be triggered are not the same for all types of events. For this reason, there will need to be a different alert producer component for each of them. In order to receive the proper events from the event dispatcher, the alert producer must have previously registered in it for that kind of event that is of interest. Note that alerts are events as well and as such, an alert might be captured by another Alert Producer to produce an even more refined analysis.

The Alert Producer obtains from the Event Producer the specific event that corresponds to the kind of events analysed by that concrete producer. The output is an alert corresponding to the security events being analysed. This alert must contain all relevant information regarding the involved events because the Alert Producer does not necessarily know which producers will be consuming its alerts or what information they will need.

##### 3.8.2.4.2 Alert Reactor

This component is responsible for reacting on a concrete alert. As there will not be a single type of alert, there will not be a single alert reactor in the system, but a set of them, one per each group of alerts to be generated on the system. Depending on the nature of the alerts their corresponding reactions are distinct. For this reason, there will need to be a different alert reactor component for each of them.

The Alert Reactors receive the alerts dispatched by the Message/Event Bus (see Section 3.9.4), which must contain all the information relevant to that alert. The output of the alert reactors will vary depending on the reactor itself and might possibly feedback the system, generating new logs to be analysed.

The relationship among the Alert Producers and Reactors and the rest of the components of the Alert Processor is shown in Figure 48. As the figure shows, each service produces its logs and sends them in its own format to the MW log / Tracking manager, which then sends a message to the Message / Event Bus, which acts as the Event Dispatcher of the system. These log events may be captured by any arbitrary set of alert producers, which may generate an alert event when suitable, and these alerts may be captured by any set of alert reactors. The output of the alert reactors is specific for each of them.

 

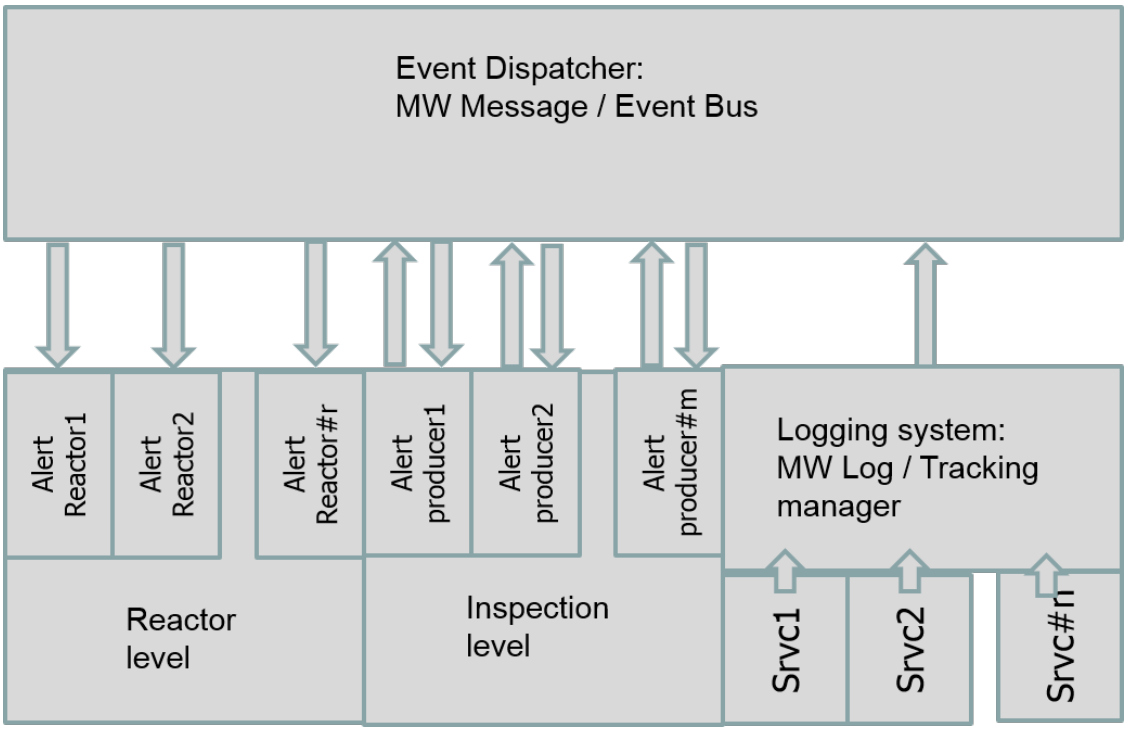


**Figure 48 – General Event Reacting mechanism.**

### *3.8.2.5 Interfaces between internal components*

The internal interfaces of the Alert Processor are:

- IF_BUS_AP: this interface is used by the alert producer to subscribe to the Message/Event Bus for receiving events.
- IF_AP_BUS: this interface is used by the Alert Producer to send alarms to the Message/Event Bus.
- IF_BUS_AR: this interface is used by the Alert Reactor to receive alarms by the Message/Event Bus.

### *3.8.2.6 Message sequence charts*

As is explained before, this is the message sequence for a concrete Alert Producer and Alert Reactor:

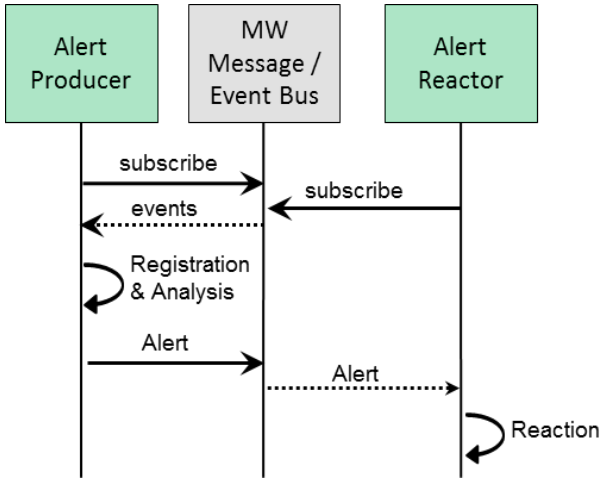


**Figure 49 - Alert Processor message sequence.**

 

## 3.9 Data and Context Manager

This functional entity handles all types of data processing within the MW and extracts context from them. RERUM will not develop these components from scratch (since they are not the focus of the project), but will utilize existing implementations (i.e. from IoT.est, OpenIoT or iCore) and will adapt the envisaged security, privacy and trust functionalities on these. Conceptually, the Data and Context Manager of RERUM consists of five components as shown in Figure 50.

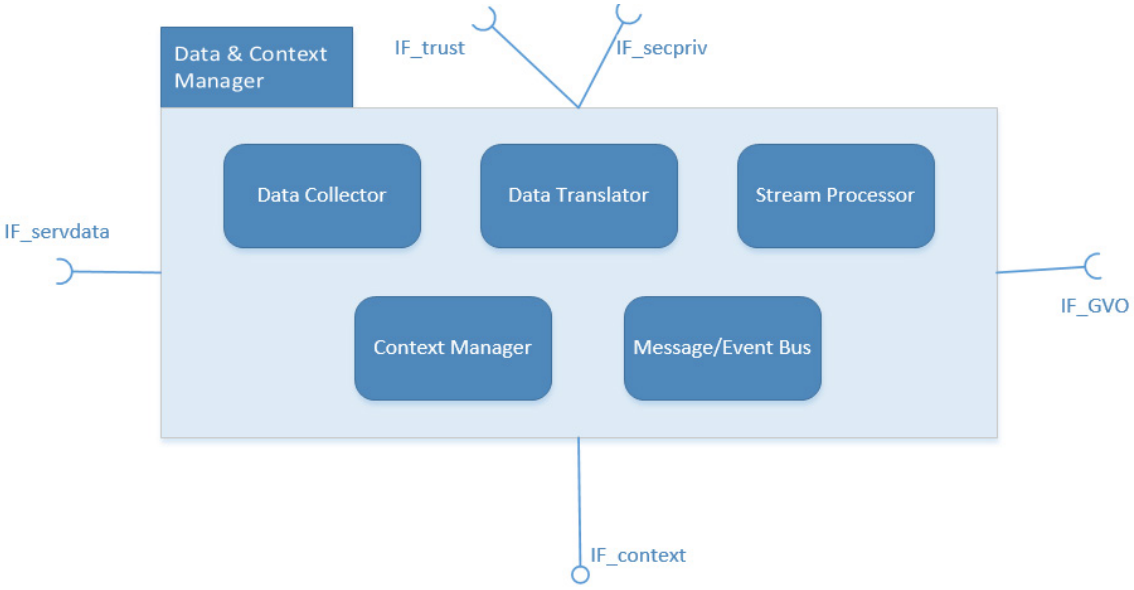


**Figure 50 – Data & context manager functional components.**

### 3.9.1 Data Collector

The role of the Data Collector is to collect the relevant data for the subscribed events from the RDs. If needed, the Data Collector may merge the data that relate with a VE and are received from various sources into a single event. The Collector will not validate data readings in order to filter out semantic/context erroneous readings; it will only perform syntactic conversions. The initial data format is defined in the GVO Templates.

### 3.9.2 Data Translator

The Data Translator performs the task of translating data into and from various formats, in order to enable data exchanges between various services. In addition, the Translator is also able to pack data in accordance to data constraints defined for the output format.

### 3.9.3 Stream Processor

#### *3.9.3.1 Description*

Enables the execution of functions such as data aggregation, the computing of mean values, minimum or maximum values and other window functions including filtering of noise, compression, etc. over data streaming from a single VRD or a Federation of VRDs. The data provided by federations represent a composition generated based on the data supplied by the individual included VRDs; the type of composition is defined by each federation, and is performed by the Federation Head/Federation Execution Engine.

**System Architecture**

The Stream Processor has distributed the functionality in four different software modules, as shown in Figure 51. It supports JSON topology of RERUM Framework. Specific to Stream Processor, the Data Streamer module is developed to be able to consume data from RERUM data source sources and to stream these data directly into Stream processing engine.

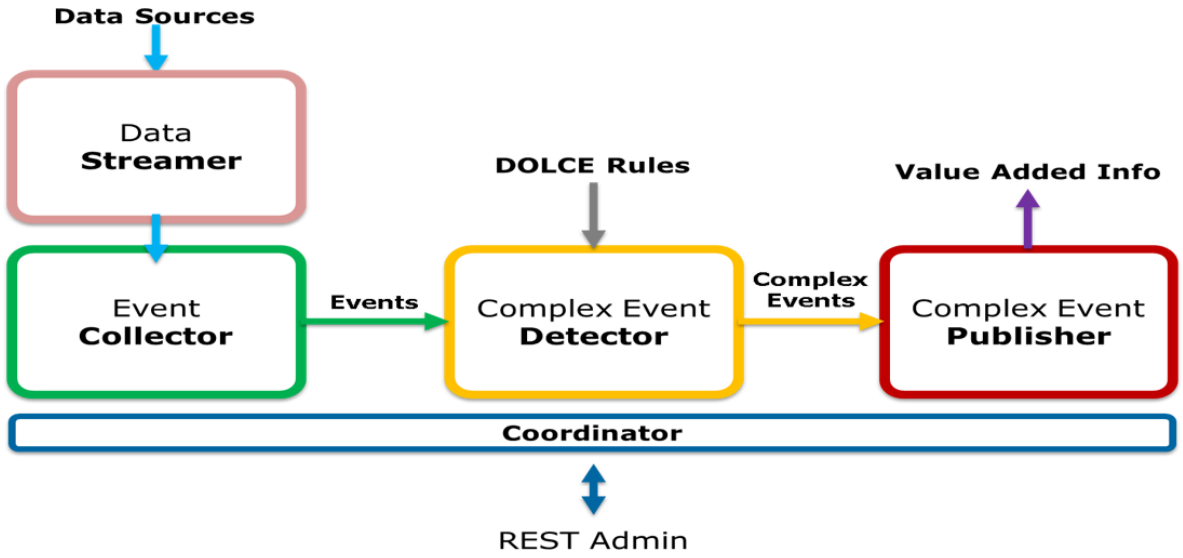


**Figure 51 - Stream Processor - System Architecture.**

The main data flow of Stream Processor is as follows: *Data Streamer* collects data from specified data source and streams them to *Event Collector*. Input events entering the *Event Collector* and coming out of the Complex *Event Publisher* in form of Complex Events, pass through the *Complex Event Detector*, where the decision about complex events is made by applying the *DOLCE Rules*. The five functional modules represented in this architecture are introduced in the following subchapters.

[*]*The Data Streamer is an internal component for Stream Processor. It is responsible for subscribing to specified message queue, consume the data and stream it to Event Collector component. In RERUM, AMQP publish/subscribe mechanism is used to stream data to Event Collector.*

**Event Collector**

The entry point to the Stream Processor is called *Event Collector*, whose primary goal is gathering all the information coming from different data sources, through different communication protocols and using varying data formats –this is known as data feeds or data sources–. Once the module has been fed with data, its secondary objective consists in transforming the information into a specific data format, an *Event*, which will be then outputted to the *Complex Event Detector* module. The following Figure 52 depicts its functionality.

Extracting the *Event Collector* module out of the kernel's engine provides better flexibility at the development stage. It is up to the *system developer* to choose which messaging protocol subscribe to –Apache Kafka, MQTT, AMQP, XMPP, etc. – or even provide the internal *Listener* sub-module the ability to request data periodically to a REST API data feed, for example. In this sense, when running the engine in a constrained hardware board, the *Event Collector* module can be written in a way that implements the less required functionality (software libraries *per se*), thus maximizing the amount of memory needed to operate. Concerning the *Decoder* sub-module, its purpose is to translate the acquired data into the internal representation understood by the Stream Processor, what in turn must match the *Event* clause as defined in the *DOLCE Rules* file –what will be explained in the following subchapter.

 

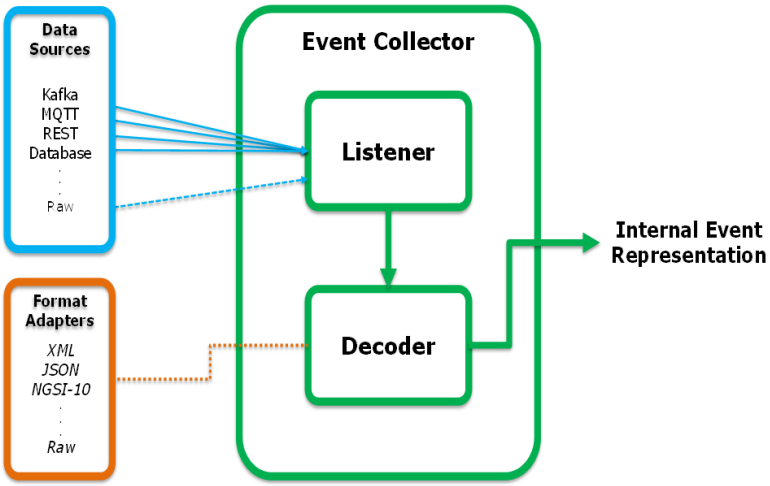


**Figure 52 - Stream Processor - Event Collector module.**

### Complex Event Detector

The *Complex Event Detector* module is the kernel of the Stream Processor engine. It detects events and produces expected results by using temporal persistence of volatile events until constraints of a rule(s) are entirely satisfied. Complex Event Detector module is based on the original SOL/CEP engine. However it has been re-written to use only standard C++ library to get rid of unnecessary external libraries. This more efficient implementation –less than 300 KB once compiled– allow creating a powerful module for real-time analysis of streaming data which might be deployed in an easy way through the variety of IoT-enabled hardware boards. In this sense, the engine is capable to run in major hardware boards running C++ compatible operating system, for instance the Raspberry Pi or the UDOO board.

Figure 53 depicts a high level perspective of internal submodules that take part in the complex event detection process.

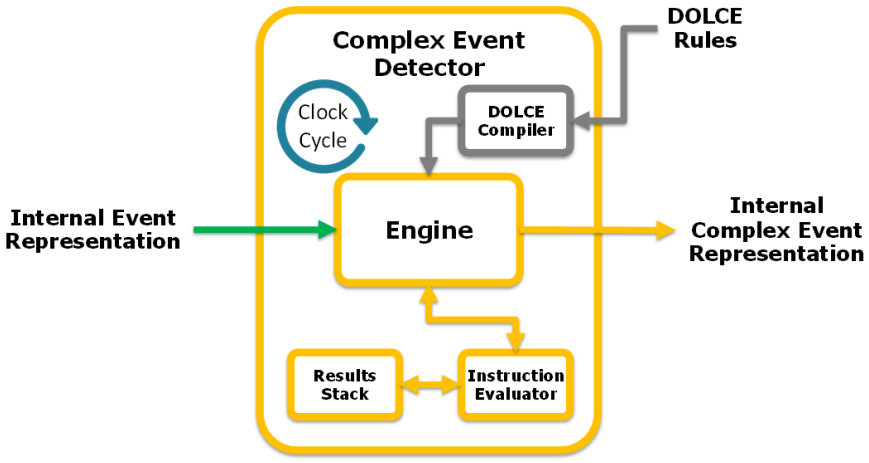


**Figure 53 - Stream Processor – Complex Event Detector module.**

Each *DOLCE Rule* file needs to be compiled, so that the converted rule is processed in a more efficient manner during runtime. In parallel, each time when internal clock dispatches a *tick* the *Engine* manages the sliding windows (in combination with the *Instructor Evaluator)*, where every *Complex clause* is checked according to given rule(s). When a *Detect clause* is valid the engine composes a *Complex Event* integrating the parameters included in the *Payload clause*, and send this Complex Event to the *Complex Event Publisher* module. The entire lifecycle process can be better understood by referring to the DOLCE Language Specification, which has been summarized and adapted in the following subchapter.

**DOLCE Rules**

DOLCE stands for "Description Language for Complex Events". It was designed for the CEP engine created by the Smart Objects Lab, part of ARI Research & Innovation, a division of ATOS Spain S.L. DOLCE language follows two main lines of development. On the one hand, it is conceived as a declarative language so as to minimize programming expertise to be able of using it. On the other hand, it considers real life scenarios from the design phase to make it simpler to use. This includes built-in types that deal with location and temporal awareness, as well as basic functions that facilitate the integration of business logic in the component. The DOLCE language functionalities were detailed in Deliverable 2.4.

**Complex Event Publisher**

The last step of the data lifecycle consists in traversing the *Complex Event Publisher* module. This module receives *Complex Events* and delivers them to the selected *Data Sinks*, reformatting the *Complex Event* portion into the data format selected by the *system developer* according to the expected application-defined external components. When used in the context of RERUM framework, this component publishes the detected events payload to specified AMQP topic exchange. It can be seen as the opposite functionality provided by the *Event Collector* module, as it also supports various communication protocols and data formats.

#### *3.9.3.2 Interfaces*

Stream processor provides one interface to manage its functionality in the context of RERUM. This interface enables the other modules to create a stream processor instance for a specified data stream, and receive the detected events. Stream processor needs the following parameters as input in order to successfully create an instance.

**Table 2: Required input parameters for the Stream Processor Instance.**

| Field | Mandatory | Description |
|---|---|---|
| **source** | Yes | Specification of the AMQP queue where Stream processor will subscribe to consume observations. |
| **dolceSpecification** | Yes | DOLCE specifications. |
| **complex** | Yes | Descriptions of "Complex" events in DOLCE. |
| **event** | Yes | Descriptions of "Event" section in DOLCE. |
| **external** | No | Description of external variables to be used in DOLCE rules. |
| **target** | Yes | Specification of the AMQP queue where Stream processor will publish the detected complex events. |

#### *3.9.3.3 Message sequence charts*

The basic message sequence involving the Stream Processor is shown in Figure 54 and described as;

1. Discovery Module triggers Stream processor Coordinator to create an instance
2. Coordinator creates a new stream processor instance
3. Coordinator triggers Data Streamer module to subscribe to the specified exchange
4. Coordinator informs Discovery Module as an instance is created

 

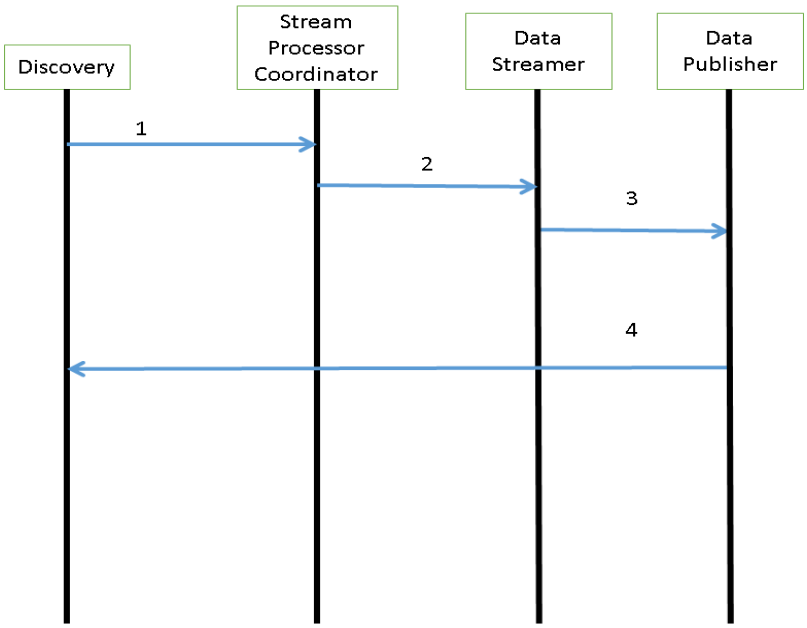


**Figure 54 - Internal message sequence of the Stream Processor.**

### 3.9.4 Message/Event Bus

The task of the Message/Event Bus is to enable and facilitate the communication between various running IoT services. The Message/Event Bus therefore allows the intelligent routing of messages between services, as well as performing the necessary data format adaptations through the Data Translator component between various communicating, heterogeneous services. The Message/Event Bus is also responsible with identifying high priority messages and dispatching them to the appropriate services first.

### 3.9.5 Context Manager

The functional role of the Context Manager is to create context-aware, intelligent personalized services for users and applications, therefore being able to offer customized observations based on the initial data received from the VRDs by notifying the Stream Processor component about the required relevant operations. In order to enable this, the Context Manager needs to monitor the service requirements and to derive a model which then monitors and captures the relevant contextual information. On the created model a reasoning process can be applied, deriving useful and refined information about the situation. Within RERUM we also consider that Context is able to influence access and privacy policies, when there are certain pre-defined situations (as detailed in [59]). For example, in a standard home deployment with various sensors, the Context of the “home situation” is “normal” and it allows the enforcement of the standard access/privacy policies. However, when the sensors identify that there is an emergency at home (i.e. the home owner is lying on the floor and not moving at all), the Context will change to “emergency situation” and then the access/privacy policies will change so that the Emergency Rescuers could open the door of the home and provide assistance to the owner.

## 3.10 Security, Privacy and Trust (SPT) Manager

This functional entity is responsible for handling all functionalities related to Security, Privacy and Trust within the RERUM platform and consists of three groups of components as seen in Figure 55. The external interfaces have been described in Section 3.1,

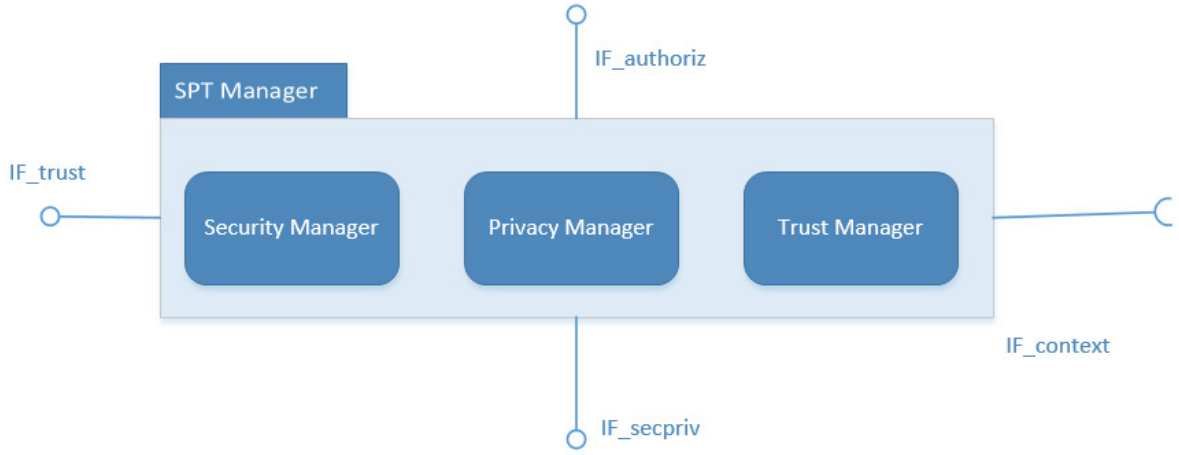


**Figure 55 - SPT Manager components.**

### 3.10.1 Functional components for Security

The functional component for security are depicted in Figure 56 and analysed in the following paragraphs.

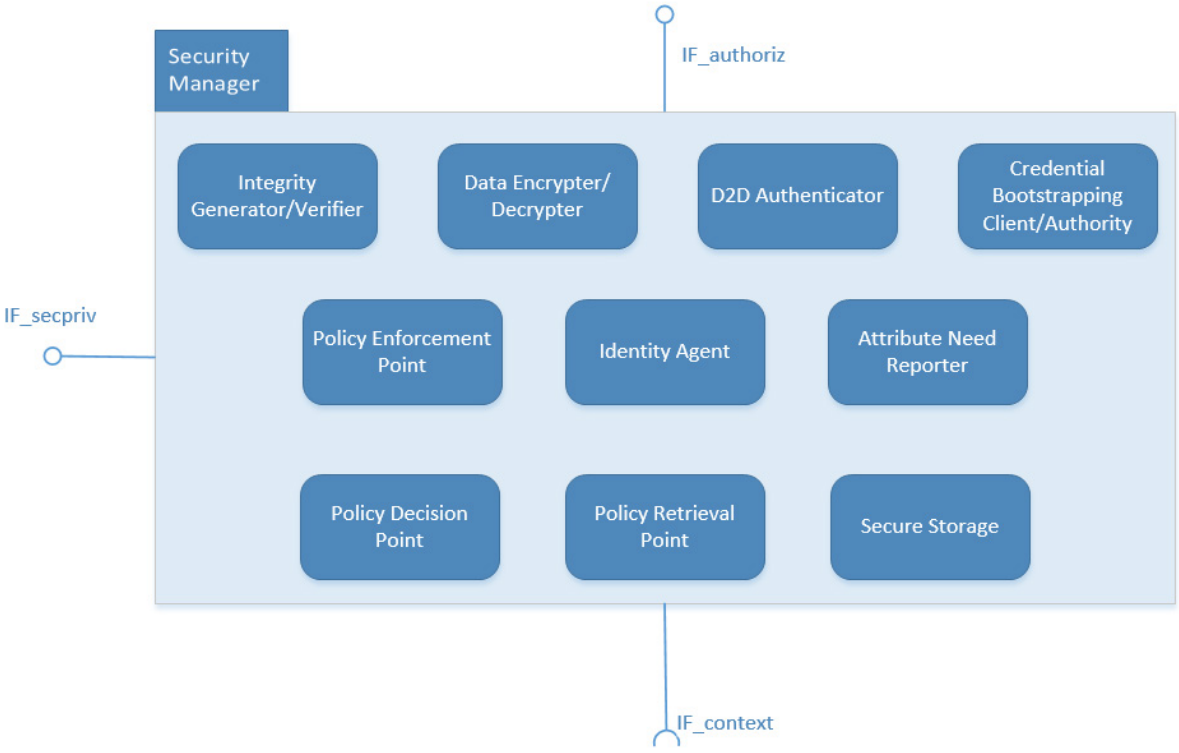


**Figure 56 – Security functional components.**

 

#### *3.10.1.1 Integrity Generator / Verifier*

Integrity means that data is in a verifiable state, i.e. that data is in the state of consistency or accuracy, internal quality, and reliability and all these are specified by the security policies for integrity**.** This property is verifiable by at least the party who applied the integrity protection and quite often is also required to be verifiable by a third party. Thus, it allows to corroborate that the integrity protected data have not been modified (or destroyed) in an unauthorised manner from the time when the integrity protection mechanisms have been applied to the data.

The Integrity Generator will generate an Integrity Check Value (ICV) [60]. This might require the use of secret keys to prevent third parties from generating valid integrity checksums for unauthorised modified data. This ICV is additional meta-data, which will enable a corresponding Integrity Verifier process to select the correct key material (secret or public). If the Integrity Verifier has access to the corresponding cryptographic key material, then it is able to verify if the integrity of the received data holds or not. In the absence of any unauthorised change, the Integrity Verifier shall output true, otherwise false. To facilitate further judgement of suitability of data it will provide more information alongside; e.g. to what strength the integrity holds up to or if we have details of the integrity checks failure. With this, the Integrity Verifier shall aid further the reputation system.

Note:

- The Integrity Generator's role is to allow the Integrity Verifier to detect unauthorised modifications. The Integrity Verifier is concerned only with the detection and not the prevention of changes.
- RERUM does not consider Integrity checks that are not built on cryptographically secure secrets (asymmetric or symmetric) mechanisms. E.g. a CRC [60] is not an ICV for the scope of this component, as CRCs in general do not provide, on their own, protection against malicious modification by an authorized party that has access to the data.
- This definition does not withhold the use of symmetric cryptography based message authentication codes, but allows choosing methods (for instance based on asymmetric cryptography) that provide a strong assurance of integrity to external third parties.
- The definition does explicitly not exclude authorised modifications. What constitutes an unauthorised or authorised change must be precisely specified in the relevant integrity security policies that must be enforced to the full extent by the integrity protection mechanism. Hence, by choice of the protection mechanism the applying entity defines what constitutes an unauthorised change.
- The definition sees data integrity in accordance with ISO 10181-6 as "a specific invariant on data" [61]. Following this, the integrity protection mechanism "detects the violation of internal consistency". "A datum is internally consistent if and only if all modifications of this item satisfy the relevant integrity security policies."
- If required, the definition is also concerned with allowing the detection of integrity violations by third parties.

Relation of Integrity Verifier and the capability to authenticate the origin of integrity protected data:
If the Integrity Verifier is able to bind the cryptographic secret used to an entity or a set of entities, the Integrity Verifier is able to judge which set of entities applied the ICV. For example, if a shared group key was used, the verification of the ICV under the group's key corroborates that a group member applied the Integrity Protection. If the secret used for generating the ICV can be bound to a single device, the Verifier can conclude that this device has applied the ICV. This component shall be capable of running on a RERUM Device and on a RERUM Gateway.
The regular ICV would be generated by invoking the Sign algorithm, as depicted in Figure 57. This requires the use of key material usable with the specific algorithm chosen for generating the ICV. As discussed in RERUM Deliverable D.31 and D3.2 in great detail, many different algorithms might be selected here. The component is cable of generically supporting all of them by the interface of Sign.

Sign gets as input the data to be integrity protected, as a JSON string. JSON was chosen due to its wide applicability and usage in the higher layers of applications in the Internet of Things, e.g. node-red. RERUM currently implements prototypes and subjects them to laboratory experiments. The interface sign is also supplied with an AlgorithmID selecting the desired ICV algorithm, which requires specific parameters (AlgInfo) as well as the signer's secret signing key. In this components interactions we assume that the needed key material (secret as well as trusted public keys) can be retrieved from the RD's local Trusted Certificate Store or Secure Storage.

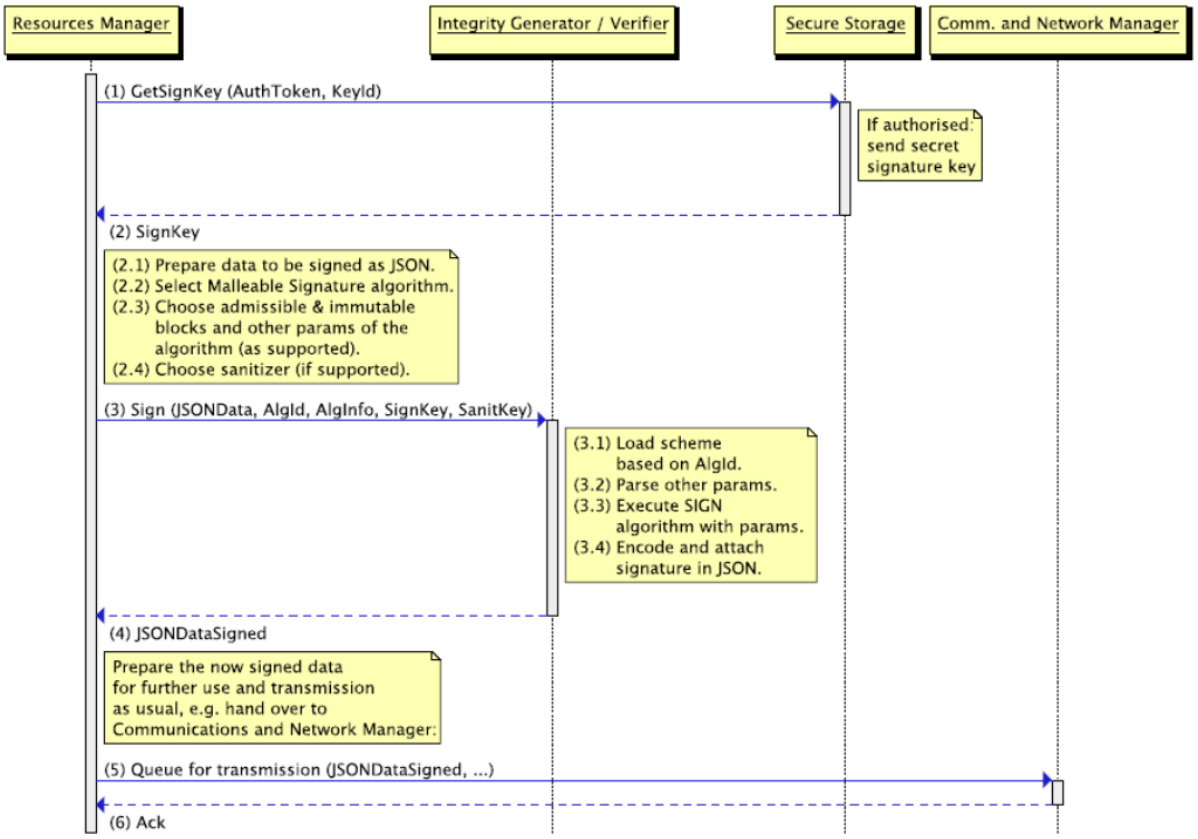


**Figure 57 - Message sequence chart for generating the Integrity Check Value (ICV) with the Integrity Generator / Verifier.**

If data in the form of a JSON object has been integrity protected, we call it signed, and denote it with JSONDataSigned in the message sequence charts. This data structure can then be validated. Regular validation would output TRUE if the data's integrity has not been breached, or FALSE if it has been tampered with in an unauthorised way. Now with the introduction of so called authorised subsequent changes due to privacy, the need for a more differentiated status for the verification arises. RERUM now allows to further differentiate –only for integrity protection algorithms that allow subsequent authorised changes— who is responsible for the current version of a message. The original signer is responsible if no subsequent change (authorised, as well as obviously for unauthorised subsequent changes). The original signer is not responsible if the message was changed subsequently. If there was an authorised change than the accountability feature of the malleable signature algorithms still allow to hold the party who did the authorised change accountable (this party is called the sanitizer in D3.1 and D32.). Henceforth, the two message sequence charts in Figure 58 and Figure 59 show the detailed output of the algorithm Verify.

 

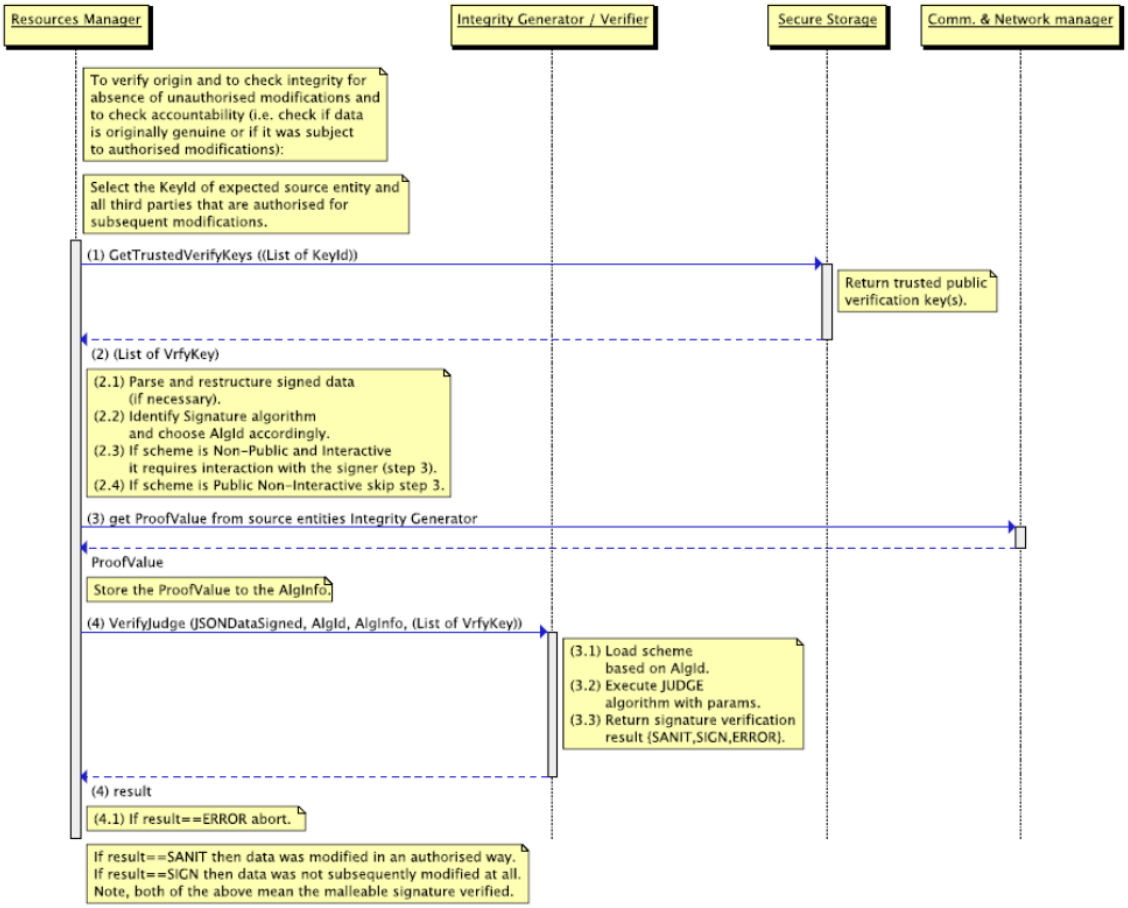


**Figure 58 - Interactively proofing who is accountable even for a subsequently authorised modified message when using the privacy enhanced Integrity Generator / Verifier.**

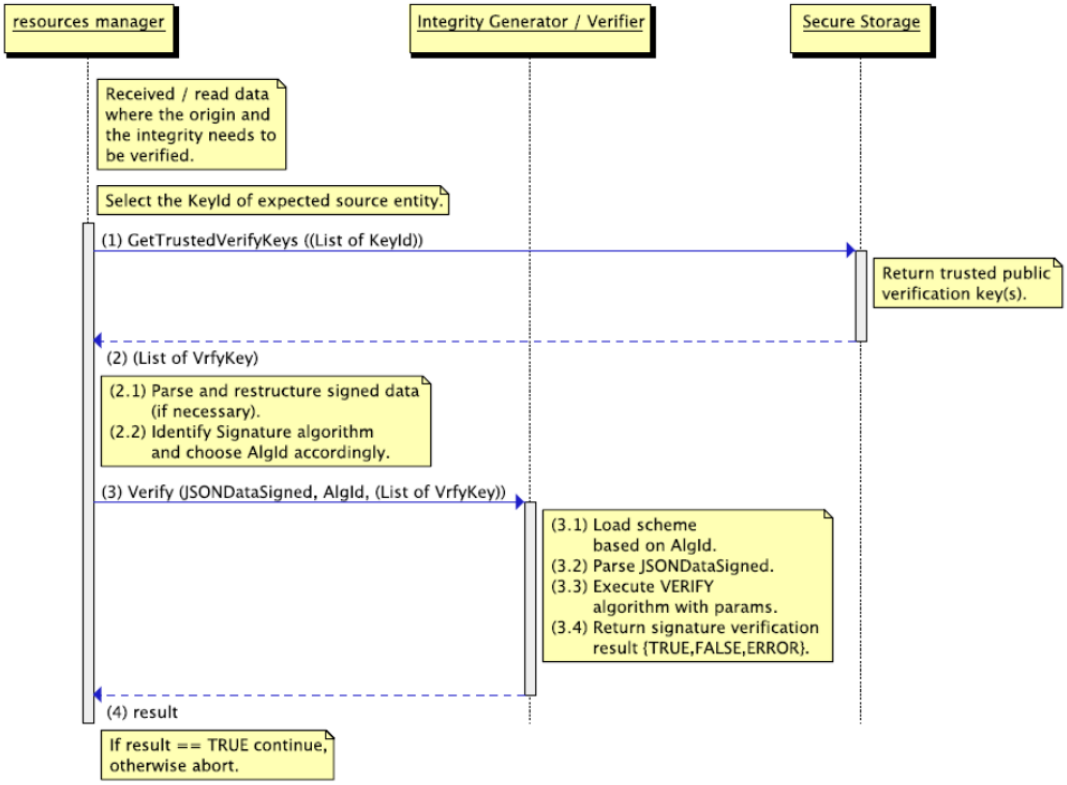


**Figure 59 - Direct verification with proof of accountability even for a subsequently authorised modified message when using the privacy enhanced Integrity Generator / Verifier.**

As Figure 60 shows, when the ICV permits authorised subsequent changes, e.g. a sanitizable signatures was applied, the Integrity Generator component's Sanit function can be called. The Sanit API is supplied with the original integrity protected message, denoted as JSONDataSigned, and additional information on how to change the message and most notably the key material that must be supplied to proof that one is cryptographically authorised to do the subsequent change.

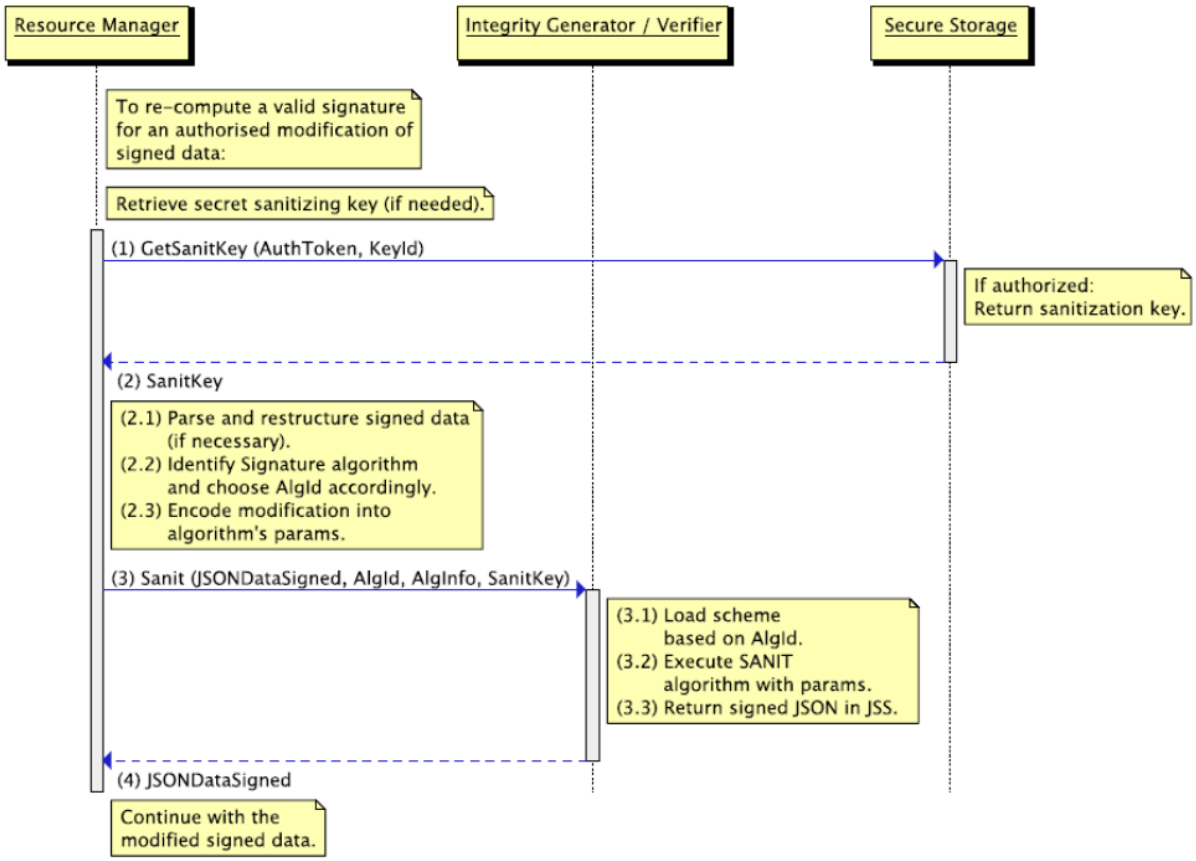


**Figure 60 - Message sequence chart for sanitizing integrity protected data using the privacy enhanced Integrity Generator / Verifier.**

### *3.10.1.2 Data Encrypter / Decrypter*

This component is used for (i) communication and (ii) storage purposes:

**Communication**

The Data Encrypter and Decrypter for communication is responsible to provide the confidentiality of data sent between two entities. This relates with RD to RD communication, RD to RERUM Gateway communication or RERUM Gateway to internet communication. The Data Encrypter receives the data to be sent and encrypts them based on an agreed encryption key. The Data Decrypter on the receiving side decrypts data, so that the content can be further operated on. As the communication links and the involved entities may provide different properties, different encryption schemas may be appropriate. This component is part of the Secure Communication functional component responsible for confidentiality protection as described in Section 3.4.1.

**RD storage**

The Data Encrypter and Decrypter for RD storage is responsible for providing the confidentiality of data at rest. The Data Encrypter receives the sensitive data to be stored and encrypts them with a storage encryption key. In case the clear text of the data is required later on, the Data Decrypter takes the

 

encrypted data and decrypts them with the storage encryption key. The component is part of the Secure Storage functional component as it is described in Section 3.10.1.10 below.

**Message sequence chart for CS encryption/decryption**

Figure 61 shows the exchanges of messages needed between an RD and the GW for encrypting/decrypting measurements using Compressive Sensing. The RD Adaptor contacts the part of the SPT Manager/Encrypter that runs on the device in order to encrypt a block of measurements. The SPT Manager/Encrypter contacts the Secure Storage component to get securely the encryption key and then encrypts the measurements (after calculating the compression rate) and sends them to the RD Adaptor, which forwards them to the Communication Manager for transmission. At the GW, the Communication Manager receives the packets and forwards them to the Decrypter that waits for a whole block in order to start the decryption (after contacting the Secure Storage component for the decryption key).

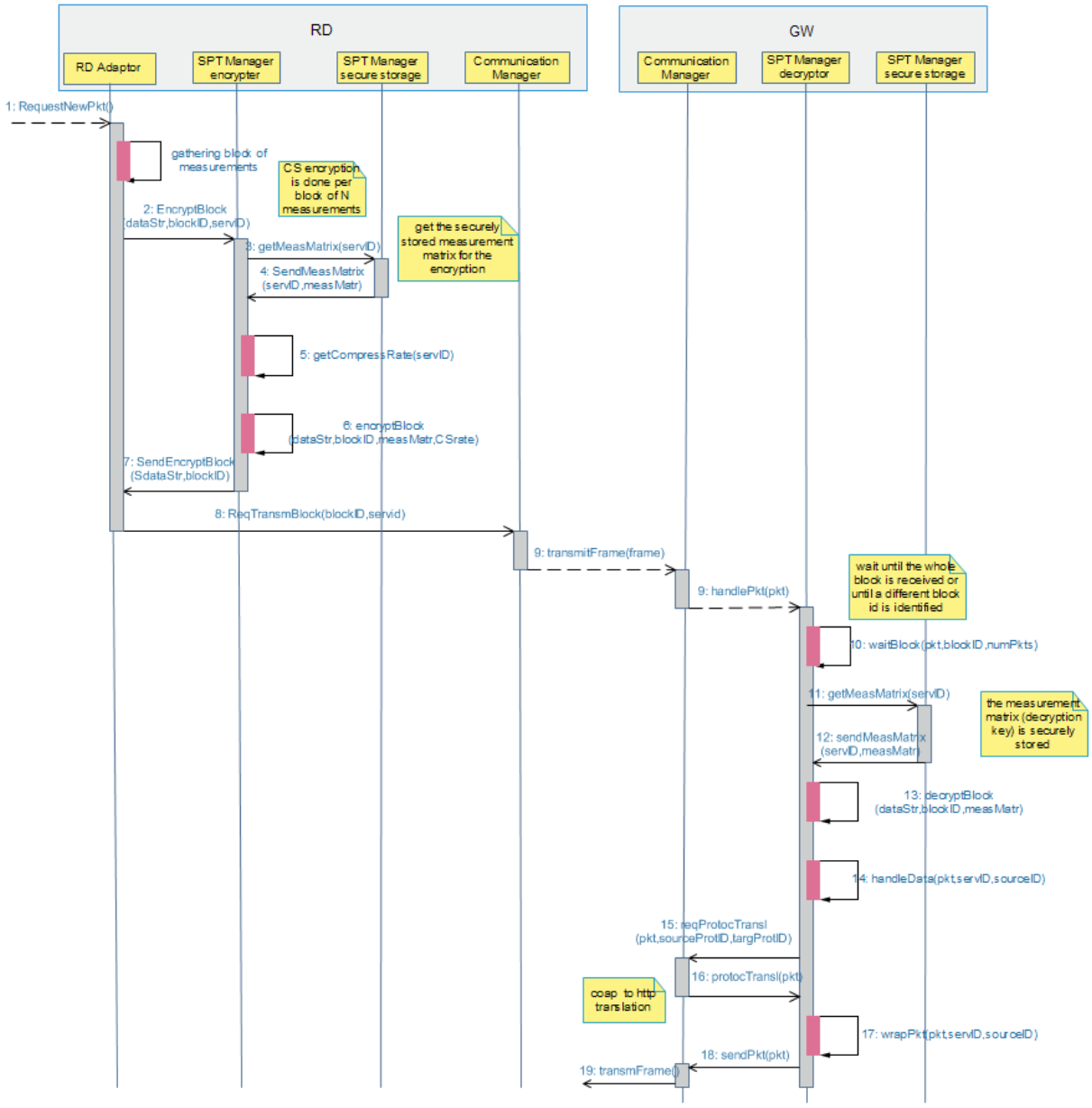


**Figure 61 - Message sequence chart for CS based encrypted D2D communication.**

#### *3.10.1.3 D2D Authenticator*

Entity authentication is the process whereby one party is assured of the identity of a second party involved in a protocol. For RERUM an entity is any computer system with the possibility to communicate over a network with other computer systems, so this also includes RERUM Devices. The D2D Authenticator component deals with the process of entity authentication carried out between two devices. This component runs on the RDs and could also run on the RERUM Gateway.

RERUM will not require this process to produce evidence used for non-repudiation (for an explanation of non-repudiation, see for instance [62]), but does not exclude such stronger mechanisms to be used. They can be used to generate evidence of the authentication that can be proved to hold also to a third party that did not participate in the process.

The two parties involved in an entity authentication dialog can be distinguished. RERUM calls them **Verifier** and **Prover**. By Verifier we denote the party that is, after a successful run of the protocol or functionality, assured of the identity of the Prover. The Prover is a second party involved in an entity authentication protocol, or as the author of a particular piece of data. With Prover we denote the party that is generating a proof of its identity in an entity authentication protocol.

Achieving the goal of D2D authentication is crucial in RERUM as only then the recipient of a message can identify from which device the message originated. Or in other words the device acting as Verifier is able to authenticate the other device. Hence, the identity of one device can be confirmed to another device, thus protecting against a masquerade attack [63].

Note that it is required to bind the authentication to some long term token and that this token must be safe-guarded against theft by both parties, as the authentication protocols usually only allow to corroborate the claimed identity of other parties at the time the protocol is run.

Device authentication can be one-way (unilateral) or mutual. In a unilateral entity authentication, after a successful run of the protocol, the Verifier is convinced to communicate with the Prover. If the vice versa is also true, hence both participating devices are assured of each other's identity at the same time, it is a mutual authentication.

RERUM defines **mutual device authentication** as the process whereby each participating device participating is assured of the identity of the other device involved in an entity authentication protocol.

In order to carry out the authentication, the D2D Authenticator needs some form of means to check the presented identity. There are basically three options for this:

1. Verifier and Prover both have access to a shared secret; the authentication protocol will make use of symmetric encryption and decryption as cryptographic primitives, involving this token as a key.
2. The Prover has access to the secret signature generation key that corresponds to the Verifier's signature verification key; the authentication protocol is based on signature generation and verification algorithms using the public and secrets parts of an asymmetric key as a token. The prover's public key must be trusted or it must be traceable to a trusted origin; this is commonly known as a Public Key Infrastructure (PKI).
3. The Prover has access to some type of token (HW or SW), which generates one or several secret keys and a public but trusted parameter that corresponds to his ID. The public parameters play a role similar to certificates in public key infrastructure.

For the third case (where the association is built upon a proof of knowledge of a token), he required secret or trusted tokens shall be stored in the Trusted Credentials Store (see Section 3.8.1.3). The needed tokens depend on the cryptographic protocol, but either they need to be trusted themselves or their trust needs to be assessed (e.g. in the case of public keys) to form an association.

 

Hence, either the token itself or at least some information to generate or verify the token must be pre-installed in the Trusted Credential Store. The required tokens highly depend on the selected protocol the participating devices use. In detail, this is part of the analysis will be performed in WP3.

In the case of shared keys, we need a shared secret between each Verifier-Prover-pair. In the other cases, the Prover must have a device-unique secret key, for which a public key exists. For signature based authentication we require the Verifier to have trust into the signature verification key of the Prover. And even in the case that uses Identity Based Encryption (IBE) we need the Verifier to trust the ID-based encryption/signature scheme's public parameters. RERUM is aware of the problems of key management and will be exploring several solutions in WP3.

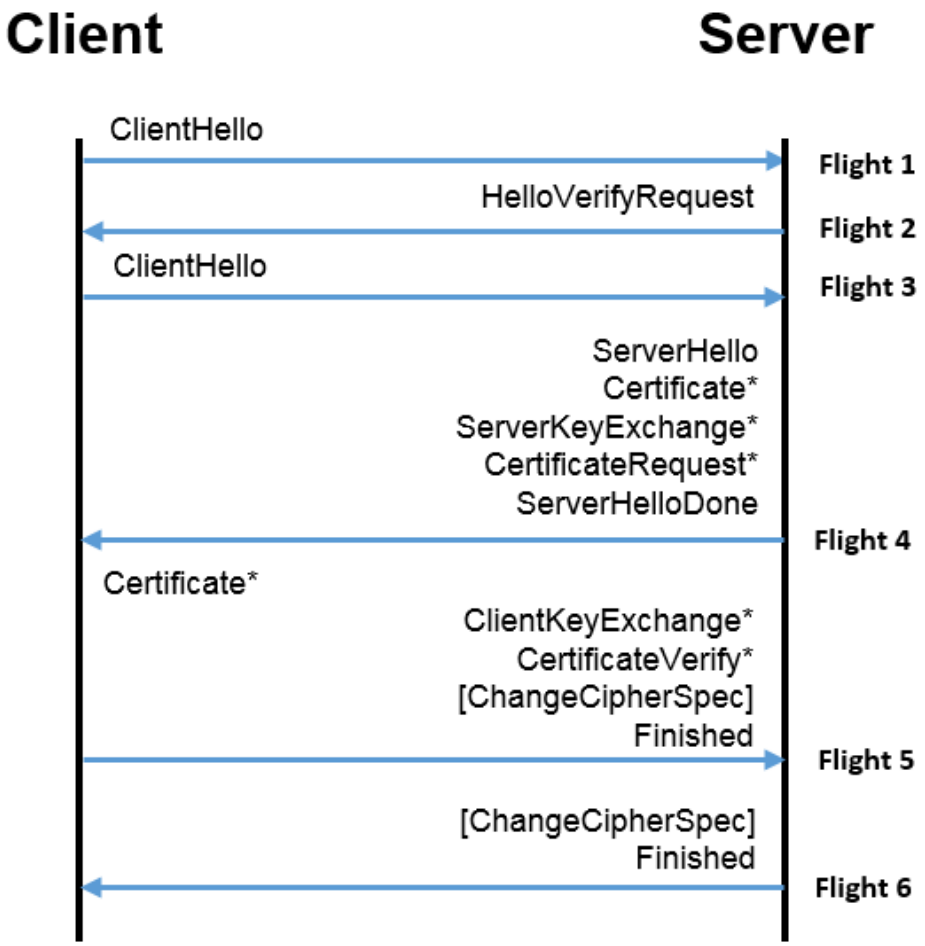


**Figure 62 – Message exchanges for device to device authentication.**

In Figure 62 an example of the message exchanges between two devices that are performing mutual authentication is depicted. One device is assumed to be the client and the other one the server.

#### *3.10.1.4 Credential Bootstrapping Client / Authority (CBC/CBA)*

The task of Credential Bootstrapping Client / Authority functionality is to provide the RERUM Devices with security credentials for secure configuration operation and management of the RERUM platform. The bootstrapping functionality is distributed between the CBC running on RERUM Devices and the CBA running in the Security Center, which can be a trusted security server either within the RERUM domain or in the internet.

The CBC is responsible for providing access to initial security credentials. With these initial credentials it is able to communicate securely to the CBA to get operational credentials. The operational credentials are used for further secure communication and for updating security credentials.

The CBA running in the backbone is responsible to provide the necessary credentials to authorized devices. For this task the CBA has access to the credentials or generates the credentials by its own. It provides the credentials to the devices using a secure communication channel, which needs to be further elaborated on in WP3.

CBC/CBA are able to support the provisioning of all types of credentials for the applied cryptographic mechanisms. These may be shared keys for symmetric crypto mechanisms or private/public key pairs certificates for asymmetric crypto mechanisms.

Figure 63 give an overview of the security components required for credentials bootstrapping.

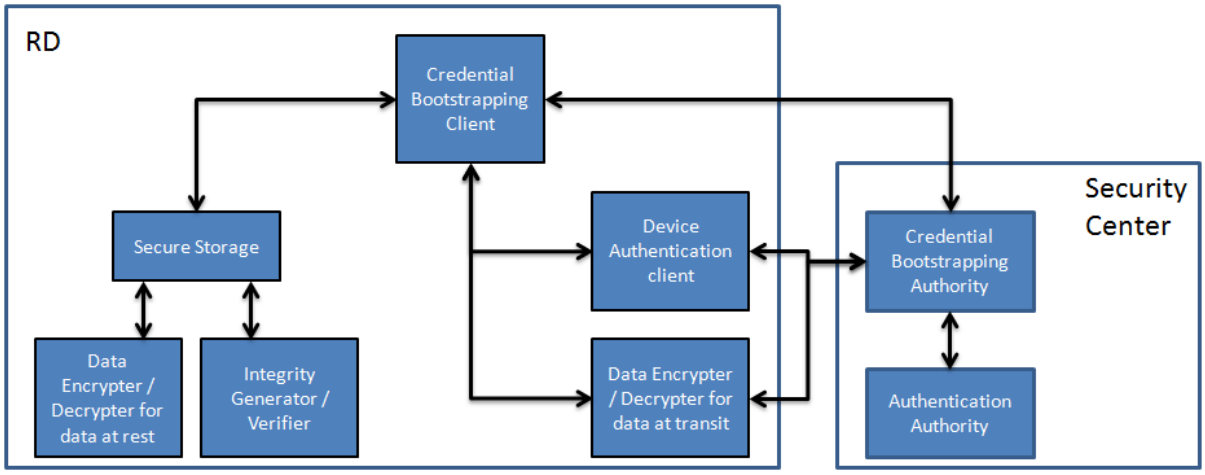


**Figure 63 – Relation between security components and their location.**

The CBC runs on the RERUM Device and invokes the different security components on the RD to get the required security services. The CBA runs on the Security Center. Depending on the scenario the Security Center is located in the RERUM network or in the internet.

Figure 64 shows the data flow for credential bootstrapping.

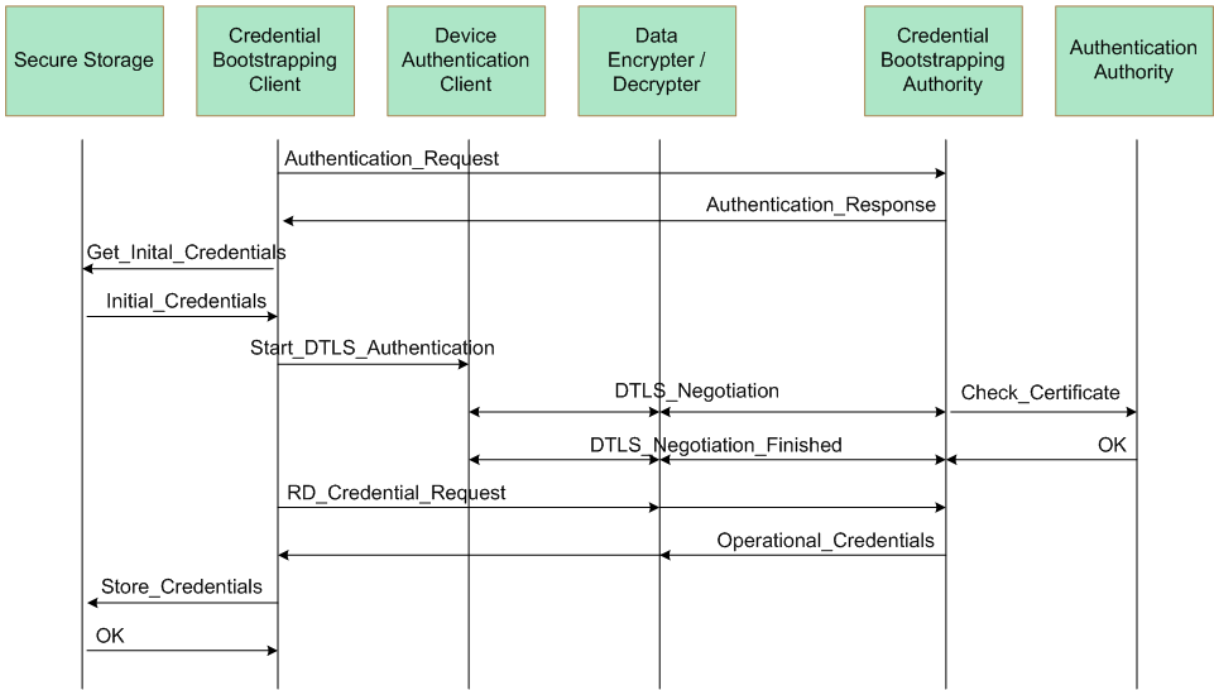


**Figure 64 – Data flow for credential bootstrapping.**

 

Precondition:

The device contains initial credentials that are provided (i) during manufacturing, (ii) during a pairing process prior to installation of the device or (iii) through the PRRS configurator (see Section 3.8.1.4).

Sequence:

- In a first step the keys for secure layer 2 communication and for secure end-to-end communication between RD and GW and between RD and Security Center are bootstrapped. The Credential Bootstrapping Clients (CBC) requests the join credentials from the local secure storage and submits an authentication request towards the Credential Bootstrapping Authority (CBA) that contains the identifier of the RERUM device and a join counter.
- The Authentication Response from the Credential Bootstrapping Authority contains the 802.15.4 network key and the key for communication with the Credential Bootstrapping Authority of the Security Center in the future.
- The CBC requests from the Device Authentication Client to establish a DTLS connection towards the CBA and to authenticate the RERUM device by the operational credentials received in the previous step.
- The CBA verifies the authentication credentials provided by the CBC via the Authentication Authority. In case the authentication was successful, the DTLS handshake is finished. In the other case, the DTLS handshake will be aborted.
- The CBC submits a request for operational credentials via the Data Encrypter Component (DTLS engine) towards the CBA and receives the requested credentials from the CBA.

The above sequence is necessary for each device that needs credentials. In the sense of credential bootstrapping the gateway is a normal device. The security association is always between the device that wants to bootstrap its credentials and the Security Center. Devices/gateways in between are transparent for the credential bootstrapping process.

#### *3.10.1.5 Policy Enforcement Point (PEP)*

The Policy Enforcement Point [64] is responsible for intercepting incoming service requests in the system and granting or denying access to them. To do so, it will retrieve all necessary input from the request in a protocol independent manner and invoke the Policy Decision Point (see below) to know whether to grant or reject the decision. Depending on that, the PEP will let the request pass or it will reject the request. The sensitivity and the requirements to access a certain set of data are described in the access policies. The input to the PEP will be the request itself, which must contain a security token with the information related to the attributes of the user issuing the request. The output of the PEP will be either letting the request to pass or rejecting it and returning a security error page.

#### *3.10.1.6 Identity Agent (IdA)*

Access policies base usually on attributes of the RERUM registered user utilized for accessing the system. For instance, a policy may grants access to a resource only if the attribute ‘user’ is set to ‘municipality staff’. But these values are not normally included in the request, and it is necessary that the authorization components have access to it in some way, asking it to the Identity Provider that holds them. Asking for attributes to an identity provider is both a complex and time-consuming task because there are many types of identity providers with their own mechanisms to get this information and the main standard to do it, which is SAML, is very complex and time-consuming to execute. The

IdA wraps the complexity of accessing the different possible Identity Providers and provides a cache of user attributes per each user with an active session to avoid having to ask for them for each incoming request.

#### *3.10.1.7 Attribute Need Reporter (ANR)*

The ANR works jointly with the IdA. Its function is to maintain a list of the names of all the user attributes needed by any policy in the system so the IdA can retrieve them. This user attribute list hast to be updated each time the set of policies in the system change. For this reason, the ANR is also invoked by the policy manager when updating the policies.

#### *3.10.1.8 Policy Decision Point (PDP)*

The Policy Decision Point [64] is responsible for deciding whether a request is granted or not, based on the information provided by the PEP and the security policies applicable for that request. The PDP utilizes its own specific data structure and it does not reuse the original request, aiming to be independent from the protocol that the request was originally carried on. The input of the PDP will be a protocol independent description of the request containing the information necessary to evaluate the security policies corresponding to the requested resource. The result of the PDP will be a protocol independent response stating whether to accept or not the request.

#### *3.10.1.9 Policy Retrieval Point (PRP)*

The Policy Retrieval Point [64] provides the security policies applicable for the requested services so that the PDP can evaluate them. The input of the PRP will be a protocol independent request including at least the data referring to the resource whose access wants to be decided. The output of the PRP will be the content of a set of policy files / rules applicable for the resource requested.

Figure 65 and Figure 66 show how PEP, PDP, PRP and the trust engine interact with each other to grant or deny access to the services requested:

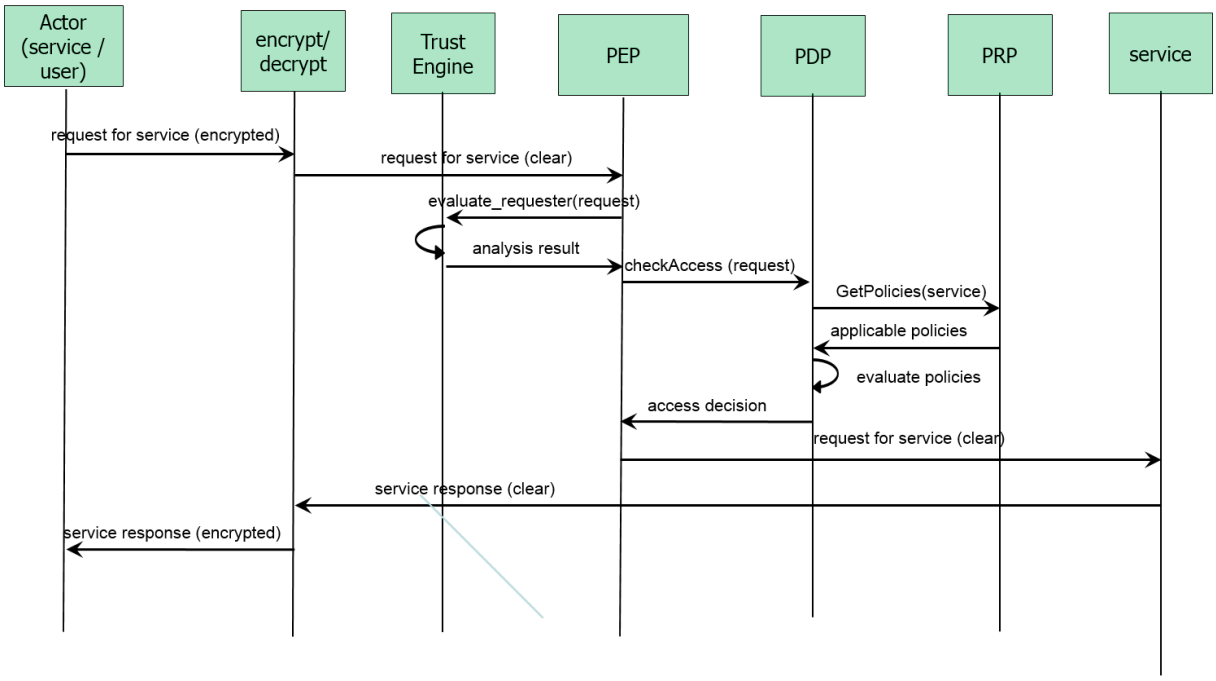


**Figure 65 – Authorization process when access to a service is granted.**

As the figures show, the PEP intercepts RERUM Service requests , enriches them by evaluating the reputation of the requester invoking the Trust Engine (see Section 3.10.3.3), and then it invokes the PDP to evaluate the request against the proper policies provided by the PRP. For authorization

 

purposes, it might be conceptually acceptable to only guarantee the integrity of the request to make sure that the data of the request have not been altered to fake the decision. For added security we assume the request or response is always encrypted including integrity protection.

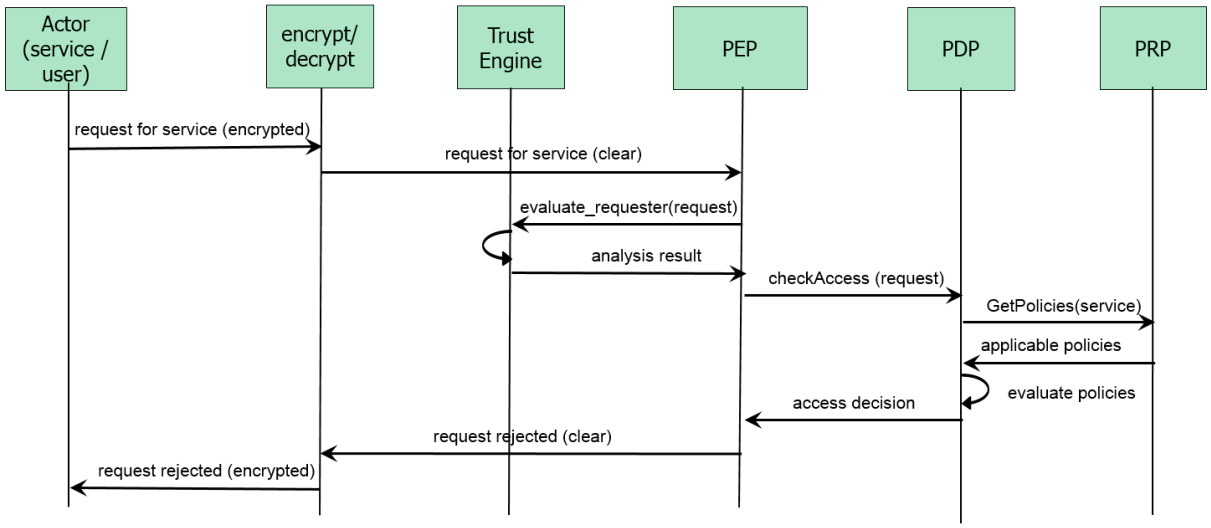


**Figure 66 – Authorization process when access to a service is rejected.**

#### *3.10.1.10 Secure Storage*

This is a functional component that aims to securely store data on a physical device. This component is used jointly with the Integrity Generator and Verifier (see Section 3.10.1.1) and the Data Encrypter / Decrypter (see Section 3.10.1.2) to securely store data on the RD. It will store the encrypted data on the RD and retrieve them decrypted, making sure that the storage is integrity protected by invoking the Integrity Generator / Verifier.

### 3.10.2 Functional Components for Privacy

A total of eight functional components support and enable privacy within RERUM as seen in Figure 67 and have three external interfaces as described in Section 3.1. Figure 68 presents the identified privacy components and their location in the RERUM architecture.

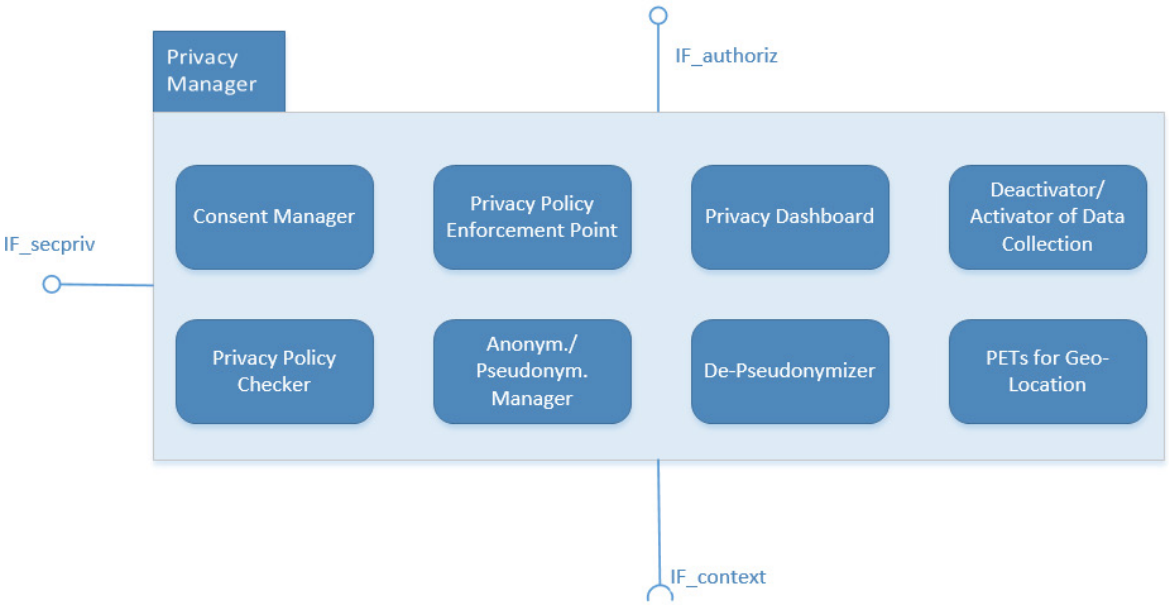


**Figure 67 – Privacy Manager internal components.**

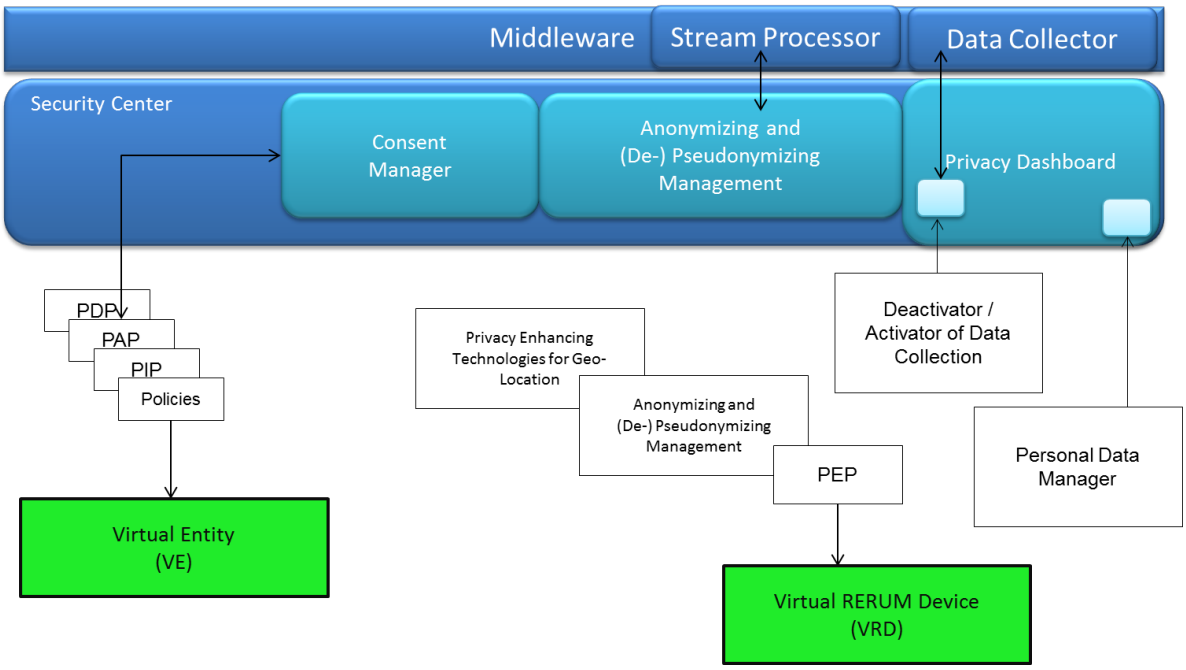


**Figure 68 – Location of Privacy Components in the RERUM architecture.**

### *3.10.2.1 Consent Manager*

When personal data are collected, generated, stored or processed, a preceding **Consent** from the data subject is needed, for instance because the law may require it. This is the responsibility of the Consent Manager. It is located conceptually at the Security Center, and interacts with the Virtual RERUM Devices and the data processing parties above the Middleware. The Consent Manager is a centralized point for users to give their informed consent in form of a mouse-click, a touch on their smartphone or otherwise.

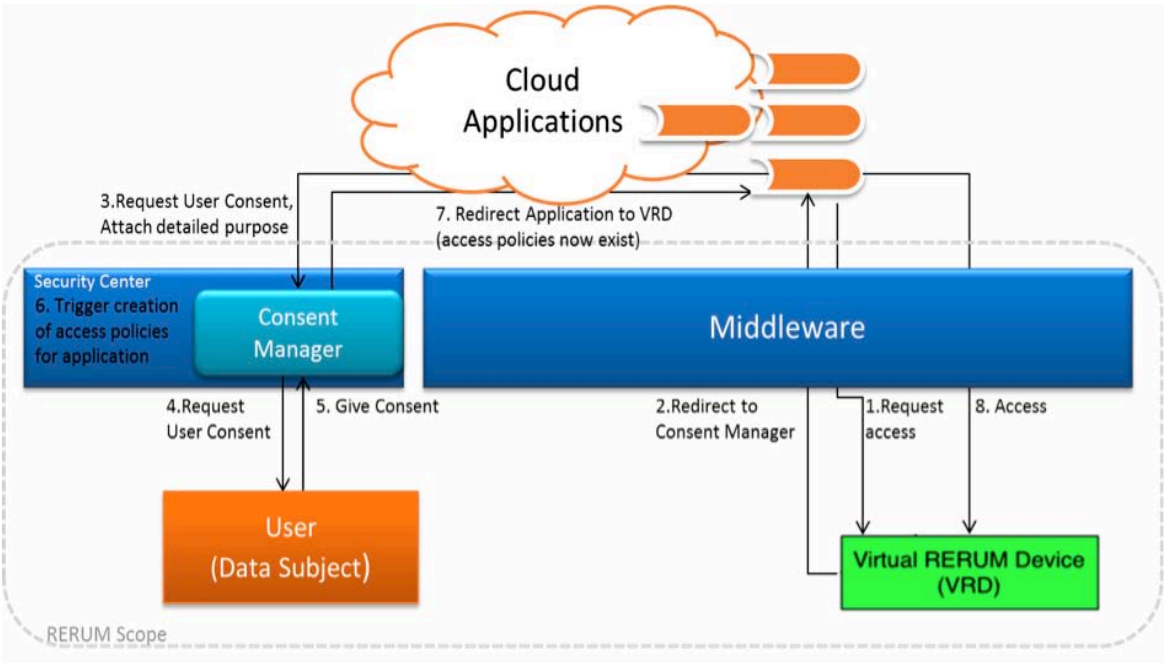


**Figure 69 - RERUM Consent Manager Interaction.**

Service providers and other data processing parties have to clearly and lawfully explain the purpose of data collection to the user, thus the Consent Manager must be accessed on a device with advanced

 

graphical user interface or with audio capabilities. After a successful consent confirmation by the user, the processing party will then receive the necessary credentials and service end point information to gather sensed data and to process it as described in the Consent Manager. This includes services that are coupled with a Physical Entity, e.g. manufacturer services of smart wear.

We assume a shared RERUM ”Consent Manager” being provided by every IoT infrastructure (RERUM ”Security Centre” for all of its data subjects (see Figure 69). The operator of the local IoT infrastructure most likely also operates the consent manager. It needs to be trusted by the data subjects and the data controllers. The RERUM ”Consent Manager” is located conceptually at the RERUM ”Security Centre”, and interacts with the data processing parties above the RERUM “Middleware” as depicted in Figure 69. In that figure a new data collector (application) (1) requests access to personal data residing in the VRD (or Virtual RERUM Object -VO), which (2) requires consent of the data subject. The consent manager supports both (3) the data collector and the (4) data subject in obtaining and (5) granting that consent, eventually offering (semi-)automated consent based on the data subject’s consent handling preferences. It (6) generates and deploys the corresponding privacy policies of the data subject and (7) informs the data controller of the consent grant, eventually providing OAUTH-style credentials and access point information. The data controller may then successfully (8) access the personal data at the VRD/VO. The detailed consent manager functionality is described in D3.2.

#### 3.10.2.1.1 User Plane and the User Agent

The data subject / user interacts with the consent manager via a graphical user interface (GUI) provided by a suitable user agent. The user plane in this case consists of the data subject and the user agent, which in turn may be a browser with or without a browser-based app or (frequently in the case of smart mobiles) a pure app. In Figure 69 the User (Data Subject) box comprises both the user and the user agent. The user agent displays the information to the user and accepts user decisions. It interacts with the consent manager as well as the privacy dashboard and also other RERUM components offering a graphical user interface. For reasons of simplifying the sequence charts in the following section, we omitted depicting the user and just drew the user's agent.

#### 3.10.2.1.2 Message sequence chart for Requesting Consent

When a data controller tries to access a data source or other resource involving personal data without sufficient consent, it must be told to obtain consent first. A possible request consent work flow is depicted in the sequence chart in Figure 70. We assume, the data controller has authenticated at a suitable identity provider first and is bringing along a suitable set of credentials. This e.g. may be a SAML assertion together with assertions for any required attributes. We further assume that multiple data subjects need to consent for that particular resource and purpose. We want to hide from the data controller which of the data subjects declines or lets expire a request for consent. As long as the data subject does not choose to grant a request for consent or otherwise to interact with the data controller explicitly (e.g. via non-anonymous feedback), the identity of the user is shielded from the data controller by the consent manager.

The data controller (1) initiates a session (session id: “SId”) with the RERUM IoT component containing the privacy policy enforcement point (pPEP). It requests data from the restricted data source. The pPEP verifies the authentication token and adds it to its security context. Then it asks (no extra message depicted, sketched in the comment field only) the privacy policy decision point (pPDP) for permission to serve that data controller with the data source. The pPDP (also no extra message depicted) consults its privacy policy repositories and various privacy policy information points. It needs to figure out, if a consent is needed for that data source, and whether all required consents are present (and valid!). The pPDP denies the access request and instructs the pPEP of the obligation that must be (2) passed on to the data controller. The data controller prepares a suitable request for consent (at best including the minimum set of data sources needed for the specific purpose at hand).

 

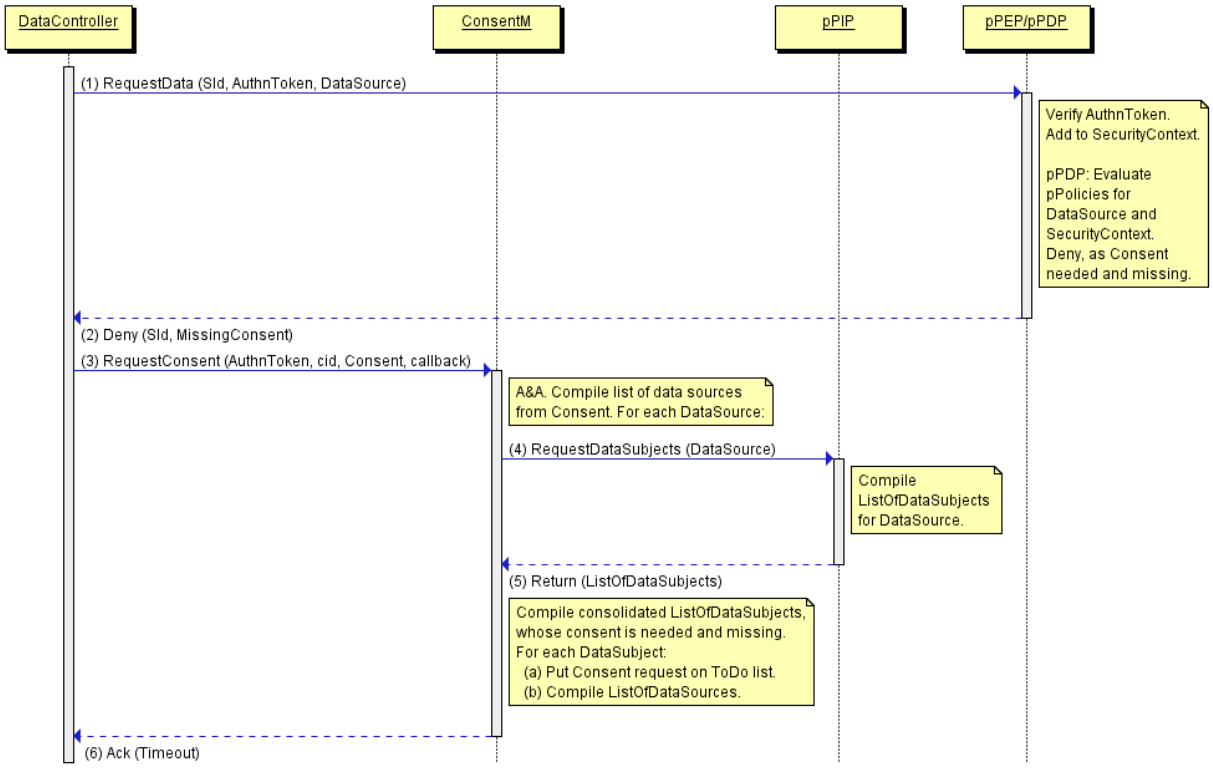


**Figure 70 - Consent Manager: Sample Need for Consent Sequence Diagram.**

The data controller (3) submits the authenticated request for Consent to the consent manager, stating a unique consent id “cid” for reference and specifies a call-back function. There the consent manager can leave a notification on the progress of the request-for-consent processing. The consent manager analyses the machine processable part of the request for consent that details the required data sources and their specifics. Then for each data source on that list the consent manager (4) requests from the privacy policy information point (pPIP) a list of data subjects associated with that data source, whose consent “cid” is still missing. The pPIP compiles such a list and (5) returns it to the consent manager.

The consent manager evaluates all the data sources for associated data subjects. Then it prepares a consolidated list of data subjects and (a) puts the request for consent “cid” on their to-do-list. Additionally it (b) compiles a list of data sources associated with each data subject. This list will be needed to indicate to each data subject the relevant data sources. Later on, after consent has been granted or denied, it serves to derive a suitable (minimised) privacy policy for that data subject. The data controller (6) is informed that the request for consent was submitted to the appropriate parties. It is also told, when that request for consent will expire, should it not have been resolved completely by then.

#### 3.10.2.1.3 Message sequence chart for Granting Consent

The data subject / user interacts with the consent manager via a graphical user interface (GUI) provided by a suitable user agent. When a data subject (who is a registered RERUM user) in their ToDo-List finds a request for consent awaiting their approval, they can select to review it. We imply that the data subject so far has not indicated consent handling preferences that in this case would allow for a fully automated consent handling.

 

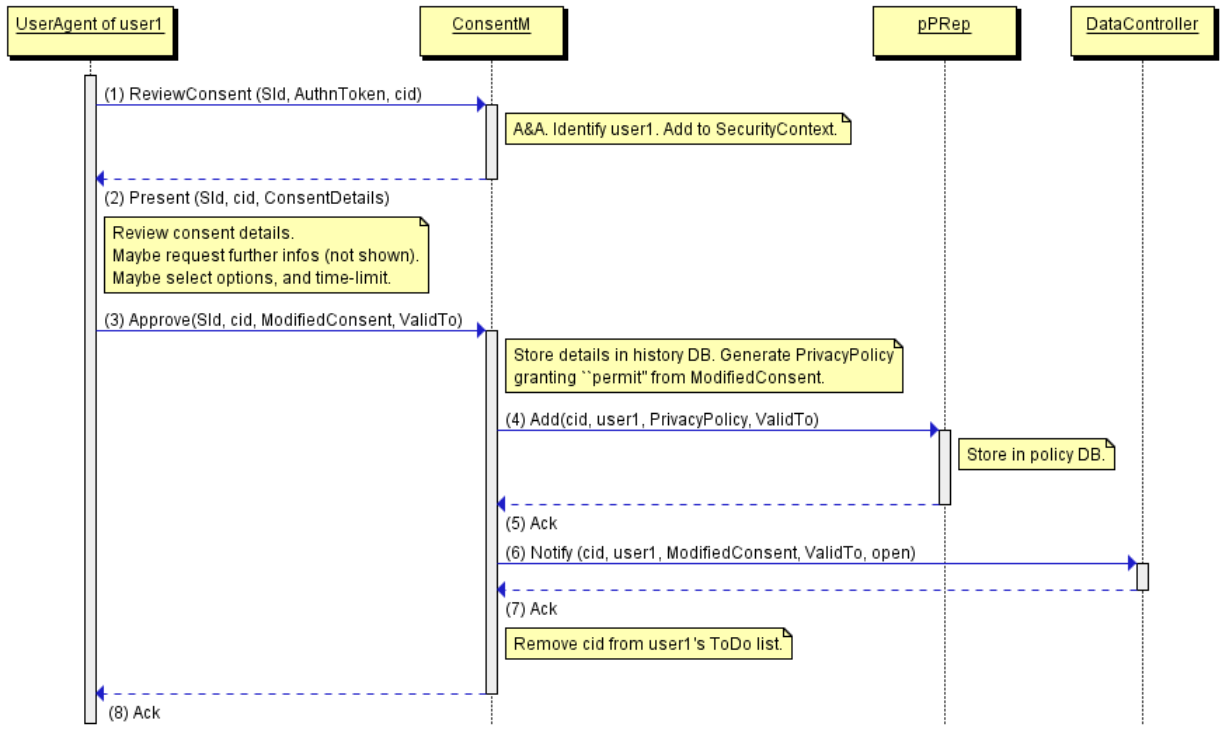


**Figure 71 - Consent Manager: Sample Manual Consent Granting Sequence Diagram.**

A possible manual consent work flow is depicted in the sequence chart in Figure 71, where the actual data subject in question is “user1”, who is interacting with the consent manager and the privacy dashboard via a user agent. We assume “user1” brings a suitable set of authentication and authorisation credentials.

The data subject (1) via the user agent initiates a session (session id: “SId”) with the consent manager and requests a certain consent (“cid”) for review. The consent manager authenticates and authorises the data subject, whose identity is “user1”. Then it retrieves the request for consent “cid”, and prepares a suitably enhanced presentation of it. The consent manager (2) submits the presentation to the user (=data subject), or rather the user agent, for review and approval.

The data subject with the help of the user agent reviews the consent details and may request further information on various aspects of the request for consent (not depicted separately). If the request for consent contains options to select from, the data subject can elect to do so. Eventually the consent might be time-limited. Once the data subject has finished the review, (3) approval (or denial) to the (maybe concretised and time-limited) request for consent can be sent to the consent manager via the user agent. The consent manager stores the consent decision in its history data base. This includes the original request for consent, the presentation provided to the data subject, the modifications made to the request for consent and all other relevant circumstances and context data.

**Privacy Policy Derivation and Deployment**

After the data subject has decided to grant or deny the request for consent, the consent manager (as depicted in Figure 71) transforms the (modified, time-limited) request for consent into an XACML-compliant privacy policy. That policy is applicable for that data subject, that data controller, and that “cid”, and addresses only those data sources with which the data subject is actually associated with (see step 5.b of Figure 70). The policy may eventually result in a “permit” statement. If consent was denied, it would be a “deny” statement. This privacy policy is (4) added to the privacy policy repository’s (pPRep) data base. The pPRep (5) acknowledges the receipt and the new policy is in effect by then.

User initiated changes to the consent content the consent manager may report to the data controller when informing it via call-back. The consent manager (6) calls back the data controller regarding the "cid" on the newly obtained consent grant including the specific details. If more consent approvals are pending from the same "cid", this is indicated by the status "open". Before timeout the status is "open", afterwards it is "expired".

### *3.10.2.2 Privacy Policy Enforcement Point*

User consent is an agreement between a data subject and a processing party **on a purpose**, which describes why personal data are collected and processed. This purpose needs to hold at all times, whenever personal data are processed. The requirement of "purpose" is realized by the definition of Privacy Policies found in [64].

These policies can be enforced before disclosing data by checking if the requesting party can fulfil them. The enforcing component is called Policy Enforcement Point (PEP). There are two main cases, where an enforcement point can be placed.

- Directly at the Virtual RERUM Devices, the Privacy PEP will check fulfilment of an adequate policy for a certain requested data set.
- Decoupled, a trusted Privacy PEP will check policy protected data in transit. The trusted Privacy PEP has to know the adequate policy for this data set or the data set carries it alongside or has a link to the adequate policy.

RERUM considers both Privacy PEP versions. The first one is integrated in the Virtual RERUM Device in the RERUM architecture (Figure 72). The second one can be deployed in RERUM's IoT Services Layer, universally available for checking the data in transit, as RERUM will ensure that the generated data has Policy enforceability build into it.

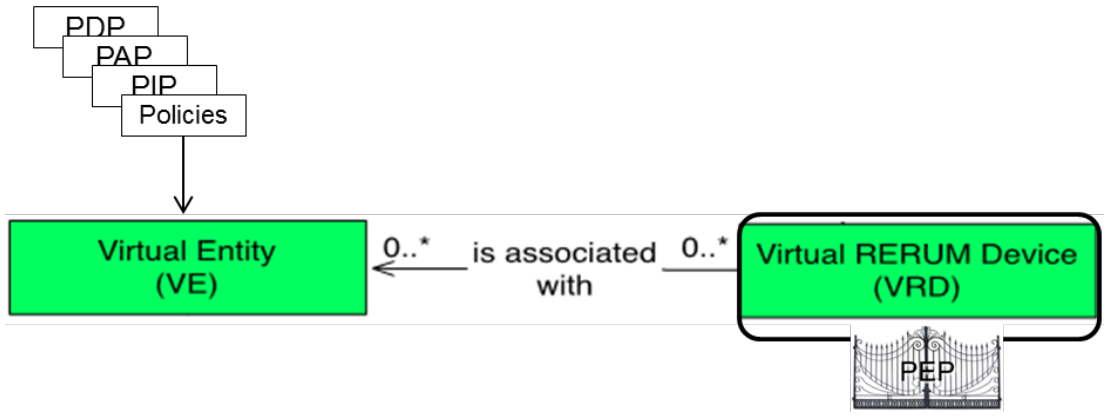


**Figure 72 – Privacy PEP is located at the VRD.**

#### 3.10.2.2.1 Message sequence chart for Accessing PEP protected data

The sequence of messages for accessing a VRD protected by a PEP is as shown in Figure 73.

1. An authenticated application requests access to data hosted in a VRD.
2. The VRD does not decide itself upon granting or denying access, it redirects the request to the Policy Decision Point, located at the associated Virtual Entity.
3. The PDP checks which policies apply for the data requested, and if the requester can fulfil them.
4. If needed, the PDP will request more contextual information from the VRD for the decision.

 

5. The PDP sends the decision to the PEP, which will act accordingly. The decision types are of the form: allow access, deny access, request undecidable.
6. The PEP will act according to the decision and either send the requested data or deny the access. (We will assume “deny” as default, if no policy applies positively or in case of conflict.)

It should be noted, that while the policy enforcement point and the policy definitions for privacy and security are functionally different, the underlying infrastructure is the same.

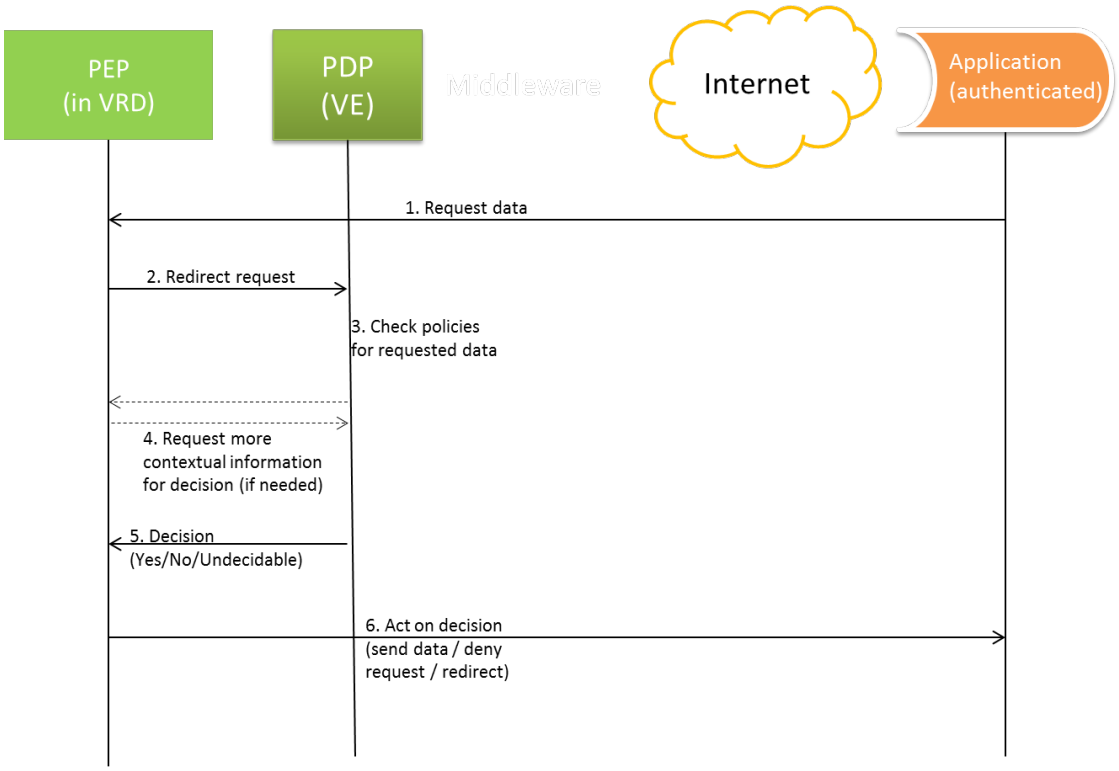


**Figure 73 – Sequence for the access of PEP protected data.**

#### 3.10.2.2.2 Message sequence chart for Granting Exceptions to Privacy Preferences

Figure 74 shows a message sequence where a potential conflict between a data subject’s privacy preferences and a new request for consent is resolved by granting exemptions. The work flow starts as described in Section 3.10.2.1.3 with data subject “user 1” desiring to (1) review the request for consent “cid”. The consent manger authenticates the user. The consent manager starts compiling the presentation of the request for consent “cid”. It needs information whether current privacy policies based on privacy settings would shadow policies based on that request for consent.

To this end the consent manager creates a hypothetical security context (HSecurityContext). This pretends the data collector is placing a request for action. The consent manager analyses the request for consent and lists all basic actions referenced in it. For instance the data controller wants to read the data from sensor 1 with a certain precision. The consent manager then (2) interacts with the pPDP. It wants to find out, which of the actions the pPDP decides about on basis of policies created from privacy preferences, and what these are (``reason=yes’’). The pPDP decides this hypothetical question and indicates blocking preference(s) in its (3) “deny (reason)” response. Step by step the consent manager compiles the conflict information. These the consent manager includes in the (4) presentation of the request for consent.

As before, the data subject via the user agent starts reviewing the information provided. In the course of the review the data subject is informed about the policy conflicts to be expected and about the privacy preferences involved. The data subject via the user agent decides to (5) grant exceptions to

some preferences for this consent “cid”. For this the data subject may log into the privacy dashboard with the help of the user agent. After (6) successful granting of exceptions the data subject can (7), also via the user agent, approve of the request for consent and the work flow proceeds as described before in Section 3.10.2.1.3.

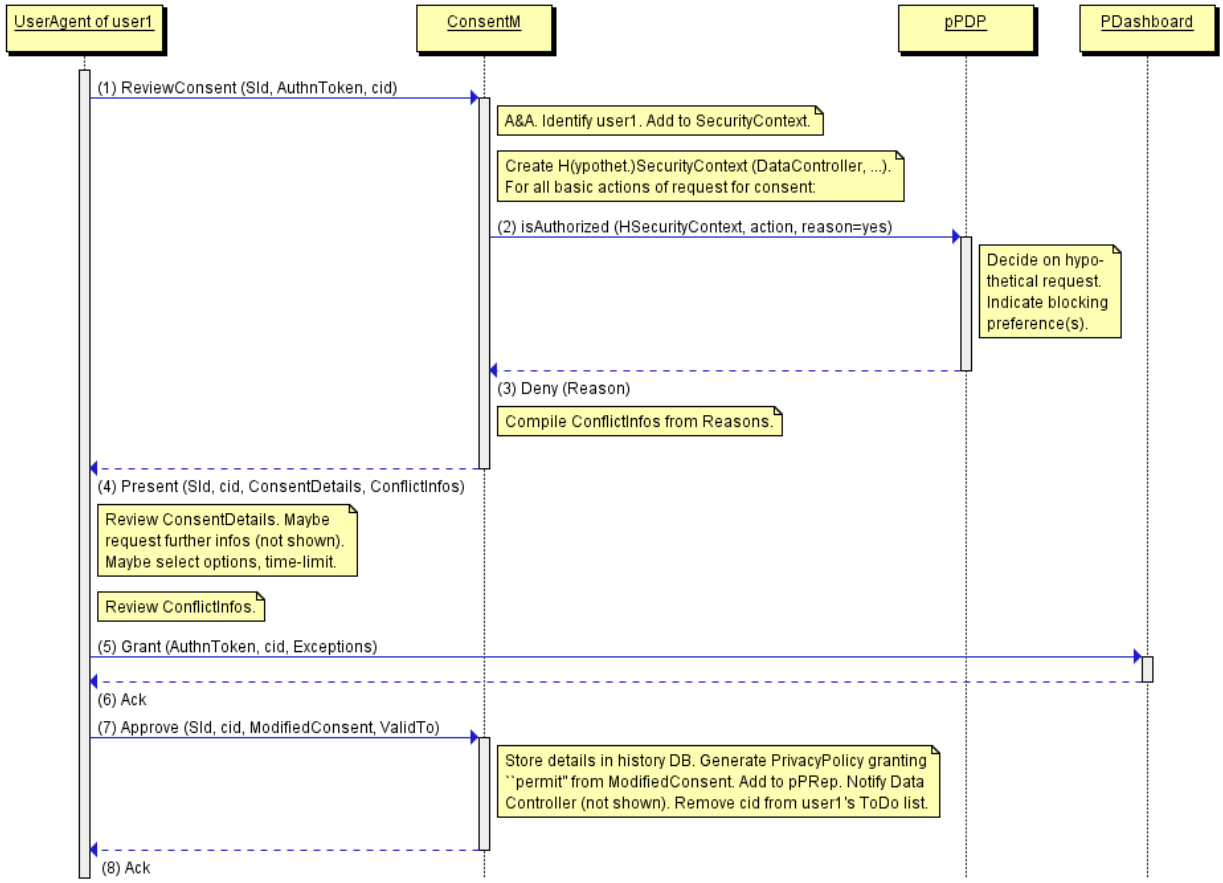


**Figure 74 - Conflict Resolution during Consent Granting Sequence Diagram.**

### *3.10.2.3 Privacy Dashboard*

Due to the fact that not all people that utilize IoT applications have technical background, it is not viable to elicit a detailed policy language editor for users, such as a XACML editor, to define privacy policies. This will be done in the Privacy Dashboard instead. The Privacy Dashboard [66][67] is a graphical user interface, which visualizes a RERUM Device’s behaviour and allows setting a specific behaviour according to users’ preferences (Figure 75). The user preferences will then be translated to detailed XACML policies without the user’s assistance. Additionally, it allows tracking how many Physical Entities are connected to the RERUM Middleware and which kind of data they are disclosing.

The Privacy Dashboard is a graphical user interface, it interacts with most privacy components in order to visualize the users preferences. The implementation of the dashboard is use case specific, it can either be deployed as a per application component, adding an access point like “Impressum / Contact” in web based systems, or as a deployment user-side, such as an app in the user's smartphone.

 

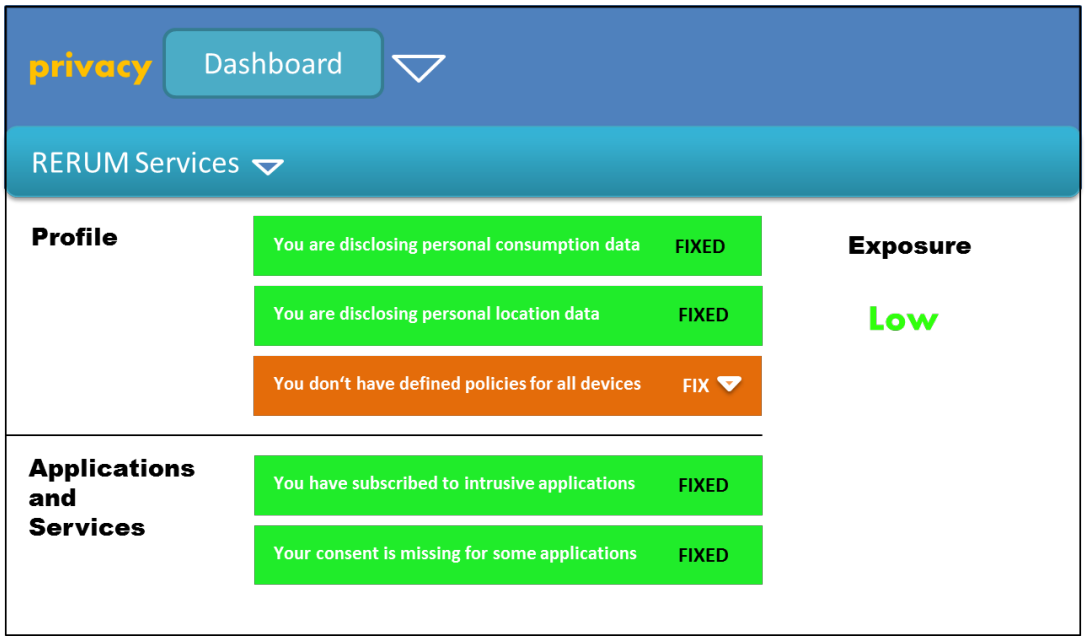


**Figure 75 – RERUM Privacy Dashboard.**

Figure 76 shows the interaction of the Privacy Dashboard with the Consent Manager. The interaction is very straightforward in the case of a required consent from a user.

A service requests access to user data. The consent manager searches for existing policies to verify if consent exists. If no consent exists, the consent manager requests consent from the dashboard. The dashboard visualizes the request, notifies the user and presents a readable form to the user. If the user accepts, the privacy dashboard will confirm the given consent to the consent manager, which will trigger a policy generation and storage. The dashboard will also handle the interaction between a user and his devices. As devices will adopt different pseudonyms, the dashboard will interact with the anonymizing / pseudonymizing manager to retrieve the real identities, in order to visualize activities to the user.

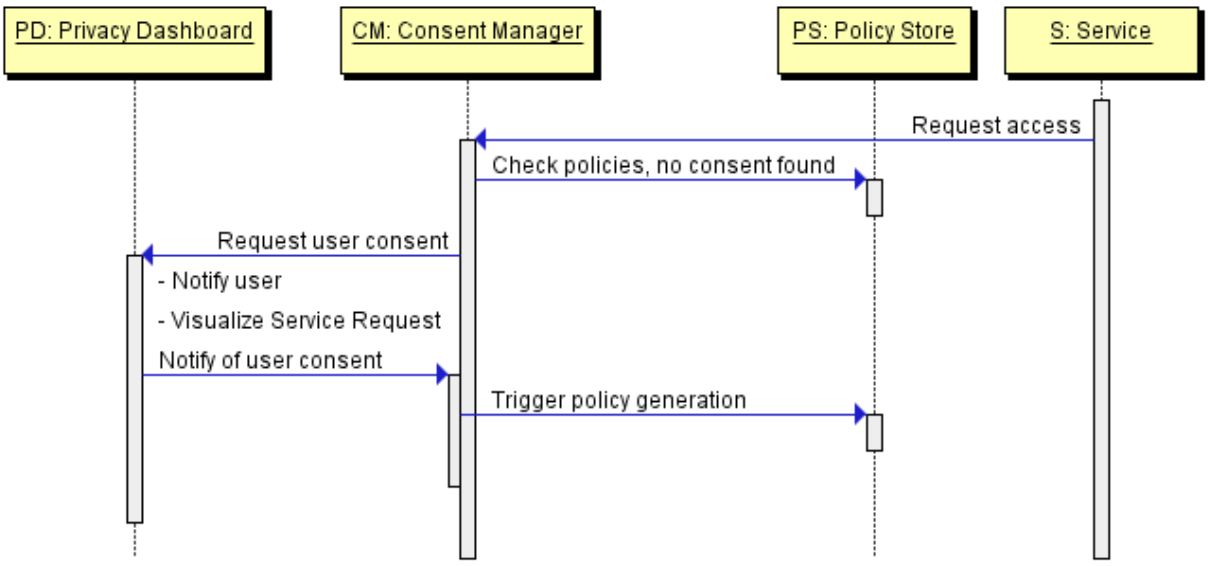


**Figure 76 - Interaction of Privacy Dashboard and Consent Manager.**

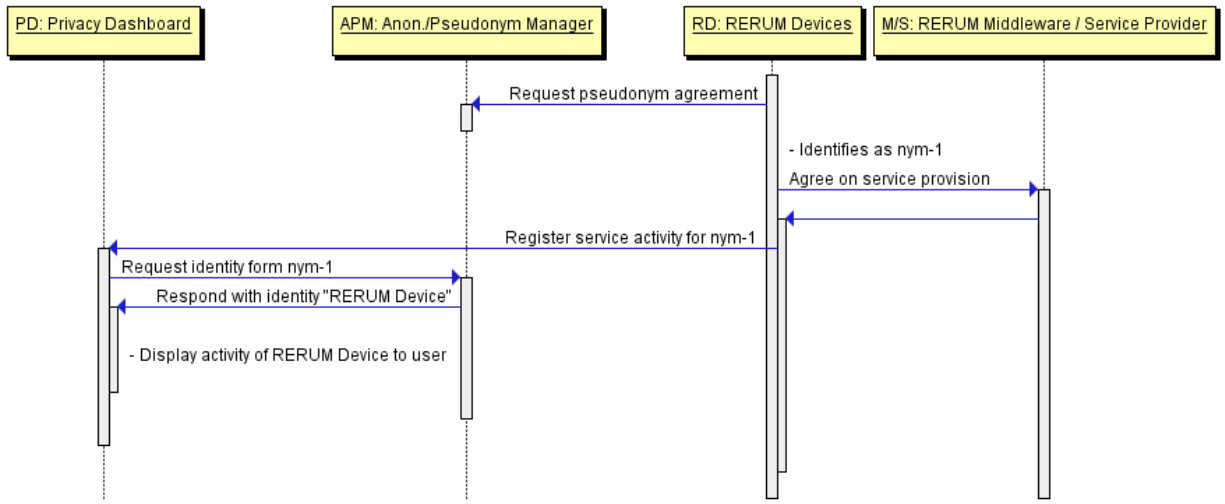


**Figure 77 - Interaction of Privacy Dashboard and Anonymizing and Pseudonymizing Manager.**

Figure 77 depicts the message flow. We simplify the sequence unify the RERUM Middleware and the service provider without detailing the actions between both.

As a first step, a device agrees on a pseudonym. It might be the case that the device does not differentiate between different parties, therefor always identifying itself under a pseudonym. We assume that a service agreement has been reached and that a service will consume the data from the device. This agreement will be displayed at the dashboard. To avoid displaying different pseudonyms in agreement for the user, the dashboard requests the real identity of devices at the anonymizing / pseudonymizing manager. With the real identity, the service agreement can be shown to the user in an understandable and readable way, while the technical agreement and service provision has been reached under the device's pseudonym.

The dashboard is the main interface for the user in terms of privacy control. The user is allowed to interfere anytime, e.g. in case he wants to opt-out from a service. The responsible component is the activator / deactivator of data collection with the privacy dashboard as its interface, as seen in Figure 78. Service agreements and device's activities will be shown in the dashboard. The user will have a possibility to opt-out of the service with the click of a button, sending a command to the activator / deactivator and triggering a collection stop at the RERUM middleware. A detailed description of the activator / deactivator is given below.

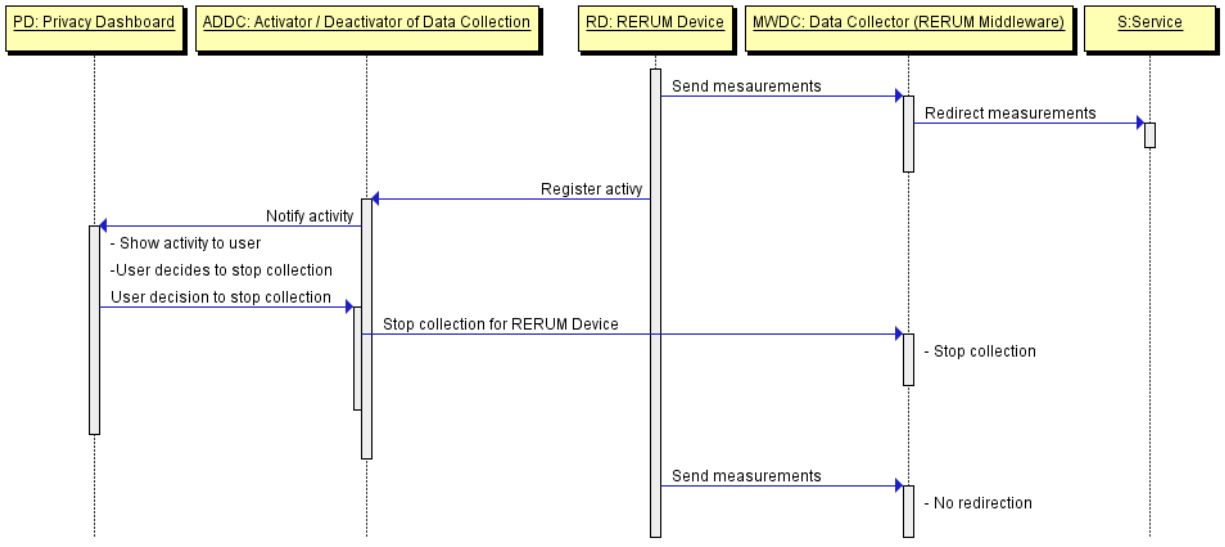


**Figure 78 - Interaction of Privacy Dashboard and Activator / Deactivator of Data Collection.**

 

### *3.10.2.4 Deactivator / Activator of Data Collection*

Data minimization is one of the core principles of privacy-by-design. RERUM will ensure on a scenario-basis that the collection of personal data is, as far as possible, dispensed with. If personal data collection is unavoidable in some scenarios, RERUM will follow an opt-in approach. That means that data will be collected, only if the user actively allows a RERUM Device to do so. An activator / deactivator of data collection will be specific for RERUM scenarios, such as in the smart transportation use case. The user will start transmitting data, when he/she actively installs a smartphone application and switches on data collection.

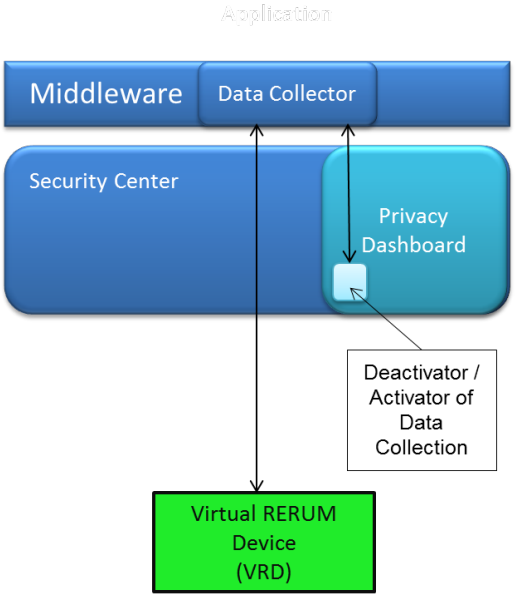


**Figure 79 – Location of Activator / Deactivator of Data Collection.**

Figure 79 displays the location of the Activator / Deactivator of Data Collection and its relation with other Middleware components. For third parties and services, the RERUM Middleware will allow to exclude an application or device by demand of the user, and so to deny their access to user data, to stop unwanted data distribution and to enforce the opt-out wish of the user. This can be done inside the user Privacy Dashboard (see below) for scenarios that do not actively support an opt-in approach. The Privacy Dashboard will display the user's associated devices and applications, and will allow him to deactivate data collection, even if the application or the device is unable to act accordingly. In this case, the deactivator component will notify the Data Collector in the Middleware and advise him to deny any request or communication by the disabled application or device.

The Data Collector is located at RERUM's Security Center and interacts with the Data Collector and the Privacy Dashboard (see previous chapter and Figure 79). In case of a opt-out, the user would click on a button for service drop-out in the privacy dashboard to start the following sequence:

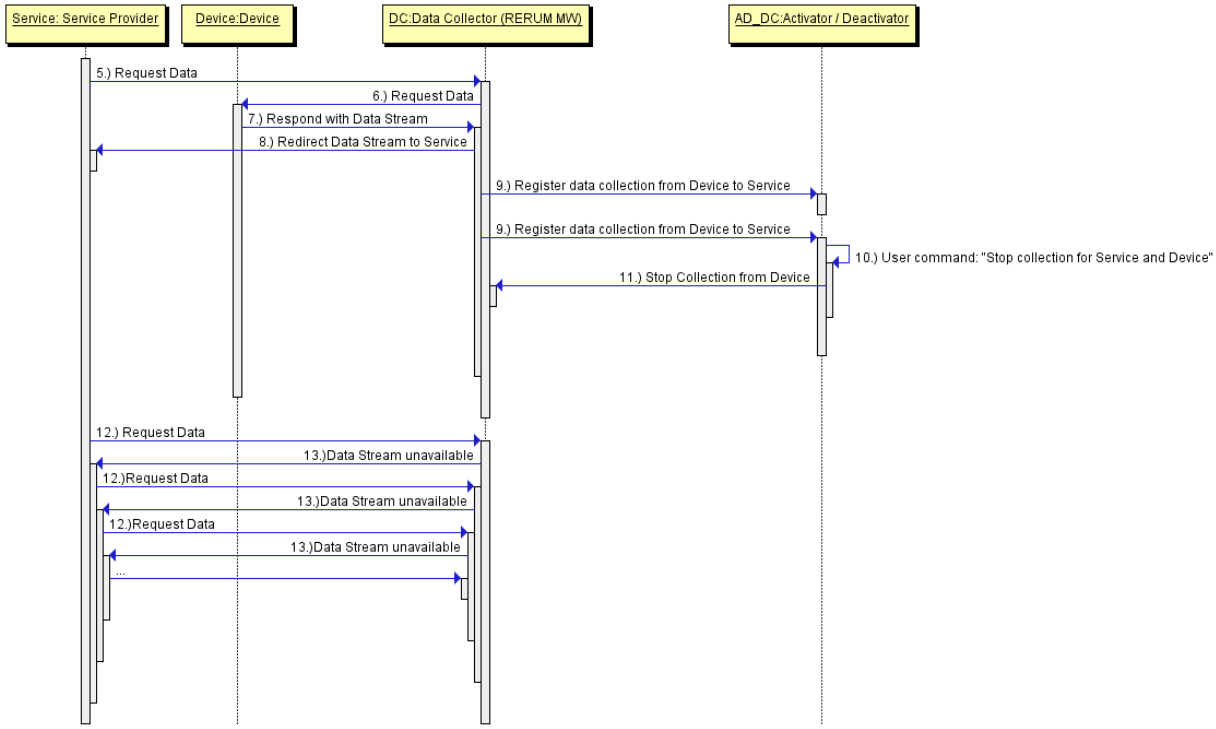


**Figure 80 - Activator / Deactivator in case of service opt-out.**

We assume in Figure 80 a service agreement (steps 1 to 4 of figure 26 in D3.2, section 3.4.2) were a service requests data from a user. The request is directed to the RERUM Middleware and redirected to the device. The device creates measurements collected by the Middleware and resends them to the Service Provider. The activator / deactivator is aware of the device's behavior (awaiting requests). By user interaction, the activator / deactivator sends a message to the RERUM middleware (more specific to the data collector) to stop collecting and sending data from the device. The user has now successfully opted-out of the service, new requests will be blocked with a service unavailable message, which does not reveal if a user has opted-out or the device has simply become unavailable. The reason for this is described in D3.2, section 3.4.

### *3.10.2.5 Privacy Policy Checker (PPC)*

The pPEP checks the policy files for accessing the data served from the services and / or devices, but the authorization process is likely to need to access to sensible data, because it will access user attributes. Though the privacy and access control to the information of user attributes is theoretically a responsibility of the Identity Provider that provides them, many Identity Providers, especially legacy ones, may not possibly include privacy checks. For this reason, RERUM provides a Privacy Policy Checker that checks privacy policies for the user attributes to be retrieved in the authorization process.

The pPEP is meant to be an independent component while the PPC is conceptually meant to work jointly with the idA to ensure it does not access user attributes that It should not do. Besides, the privacy policies that the PPC works with are necessarily more limited than the ones that the pPEP work with. Besides, the PPC must necessarily not work with policies based on any other user attribute than the user-id, because the goal of these policies is precisely to control the access to those attributes and would cause a recursive cycle of attribute retrieval.

### *3.10.2.6 Anonymizing and Pseudonymizing Management*

As mentioned above, data minimization is a key concept for true privacy-by-design. Anonymisation mechanisms will be implemented conceptually in RERUM scenarios, whenever possible. They will reside in RERUM Devices and anonymize data directly after they are sensed, delete identity traces

 

when data are sent, and anonymize the sender's identity. Adequate and optimized cryptographic mechanisms are considered per scenario.

Pseudonymisation requires extensive management, depending on the type of pseudonymisation that is considered. Generally, there are four types of pseudonymisation techniques, based on (i) asymmetric encryption, (ii) identity-based attributes, (iii) group signatures and (iv) symmetric encryption. RERUM considers identity-based schemes as best suited for pseudonym agreement and management.

Pseudonyms mostly rely on a trusted issuing party, which also manages the pseudonym identity mapping. This party has high connectivity and is required to authenticate a pseudonymized entity. This party also takes care of pseudonym updates, changes and revocation. It resides on the Security Center and is closely coupled to the authentication authority and the Stream Processing component of the MW for data aggregation management. Figure 81 visualizes the location of the Anonymizing and Pseudonymizing Management.

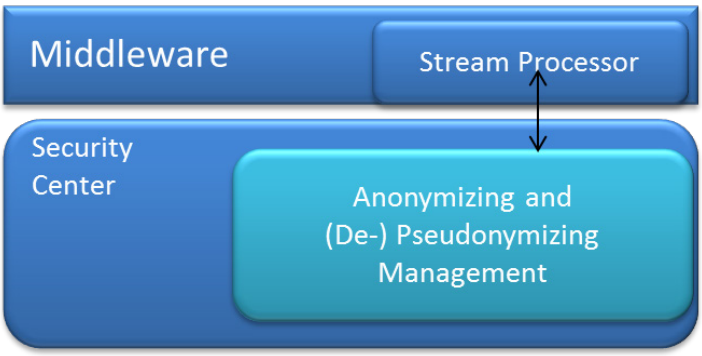


**Figure 81 – Location of the Anonymizing and Pseudonymizing Management.**

For the anonymizing and pesudonymizing manager (abbreviated "A/P Manager") the architectural integration remains the same as in D2.1, it resides on the Security Center. The anonymizing and pseudonymizing manager mainly interacts with devices and user agents which want to recover the devices identities. An exemplary interaction is given in Figure 82.

The interaction of the anonymization and pseudonymization manager is very use case specific. The manager might re-link pseudonyms for user agents (such as the RERUM UC-I1 smartphone app) or the privacy dashboard (see Section 3.10.2.3). In Figure 82 the user, a simplification of a user agent which could be a device as well, agrees on a service which is allowed to consume data from a user's device. Again, we simplify the transmission between service provider and device, adding the RERUM Middleware as part of the Service Provider role.

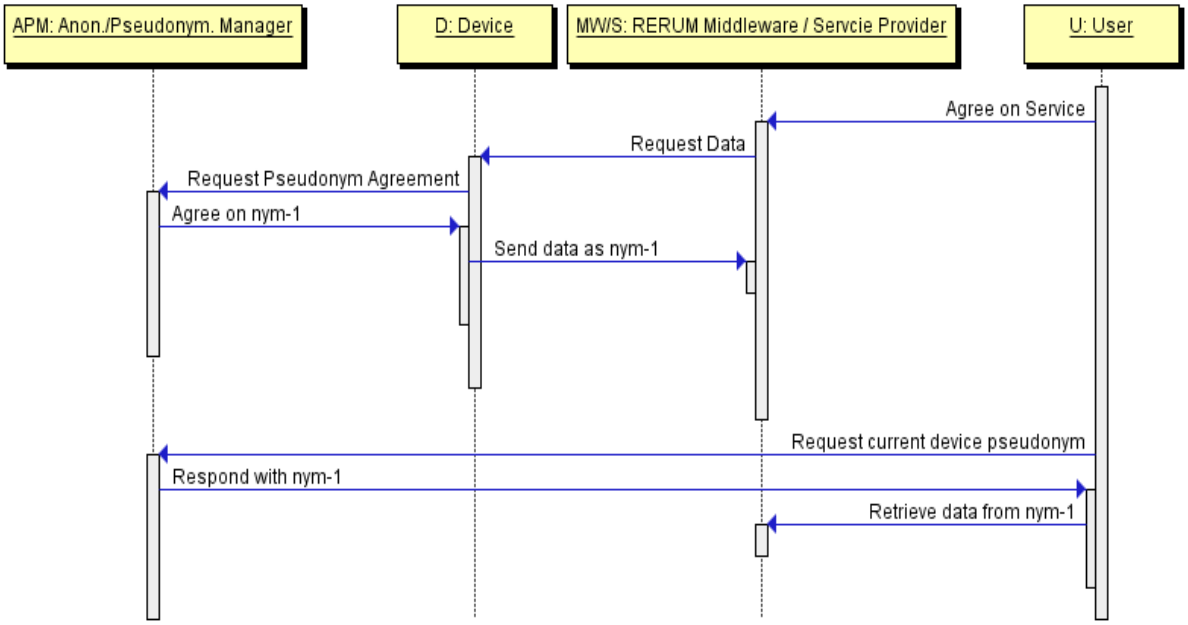


**Figure 82 - Interaction between Anon/Pseudonym Manager, Device and User.**

The Middleware / service provider request data from the device. If the device does not have pseudonym yet, it requests a pseudonym agreement with the A/P manager. Upon agreement, the device and the manager have synchronized the pseudonym generation of the device. This will allow the manager to re-link pseudonyms (for details see D3.2 section 4.5.7). The device will transmit messages under one or different pseudonyms.

The user might want to retrieve his data from the service provider. The user has to know under which pseudonym the data was sent, processed and stored. He requests the pseudonym from the A/P manager and retrieves the data from the service provider with it.

In RERUM, the A/P manager's actions are very use case dependant. It will interact with many different parties, thus being dependant from additional authentication and authorization mechanisms. Therefore, the A/P manager will have a close relation to the authorization and authorization mechanisms in the security centre and the stream processor,

#### *3.10.2.7 De-Pseudonymizer*

For billing or legal purposes, it might be necessary to de-pseudonymize certain entities and to track the identities that were behind certain actions. This is done by the De-Pseudonymizer. The de-pseudonimizing mechanisms differ depending on the used scheme.

RERUM's proposed mechanism of relinking or de-pseudonymizing is dynamical, the party that re-links the identity and the pseudonym does not have a list of identities and pseudonyms, it generates the pseudonym that an identity must have used. This is possible, if the re-linking party is trusted, if it has a root- or sub-secret (as described in D3.2, section 4.5.7) and the re-linking party knows the real identity of the system participant.

#### *3.10.2.8 Privacy Enhancing Technologies for Geo-Location*

Smart transportation is one of the main scenarios of RERUM and requires special technologies for geo-location privacy protection. Figure 83 visualizes the data that are gathered in GPS-based traffic measurement. In the simplest scenario, apart from a user's IP-address, information about his exact position in certain points of time is transmitted. Figure 83 shows measurements M1 and M2 as an example. In every measurement, the user's position is given as a latitude/longitude set, including the time of the measurement.

 

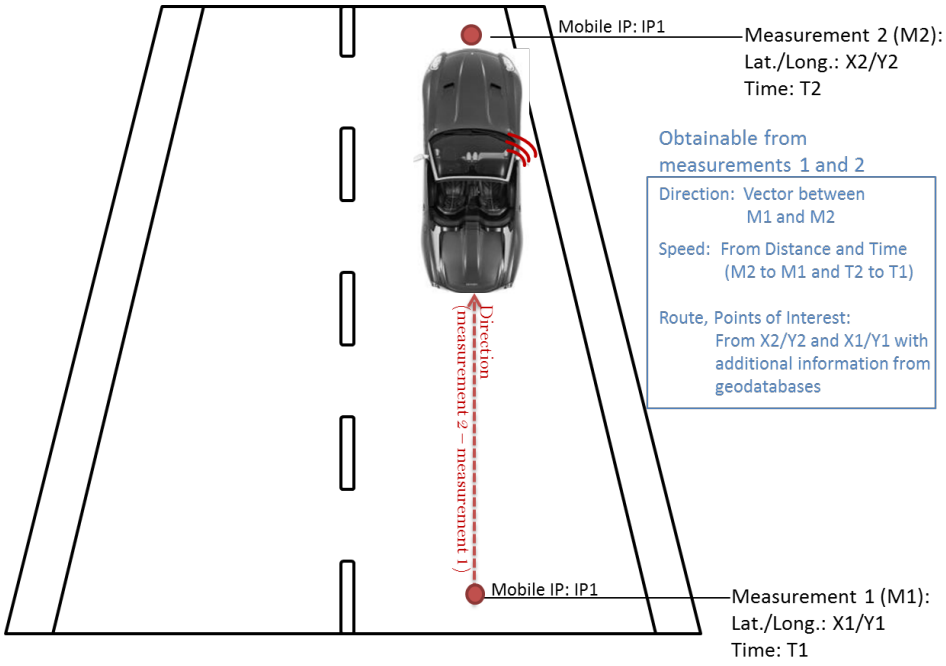


**Figure 83 – Geo-location data in traffic measurement.**

This is enough to calculate a vehicle's speed (difference of distance and points of time between measurements) and direction (difference between locations), as well as to disclose intrusive information about the traffic participant, for example the user's daily route and his daily points of interest (by mapping of measurements to geodatabases).

RERUM will use cryptographic solutions to minimize the personal data collected. These solutions will help to generate the data that are needed for traffic estimation, without the transmission of detailed coordinates. At the same time, it is needed to anonymize the participant's true identity. This will be done by design, at RERUM Device level, which is the point of data generation.

The resulting data set will strongly differ from measurements M1 and M2 in Figure 83. The measurements will be sent in such a way, that the sending participant would not be possible to be assigned as the source of the data. Depending on the scenario, measurements will contain information about a certain street/route/location and the average speed in a (close to real-time) point in time. The addition of pre-processed, aggregated values and the omission of detailed coordinates and the user's identity will impede the disclosure of personal information, while allowing the service provider to obtain the needed information for traffic analysis and forecast.

Figure 84 shows the interplay of the geo-location privacy component with other RERUM components. The geo-location PET is a part of the on device S&P&T mechanisms and resides in the RERUM Device. It receives sensing data from the GPS sensing element in the device to create random vectors as described in D3.2, section 4.7.1. Finished vector data is sent to the RD Adaptor to be transmitted to the service provider by the communication manager and the RERUM Gateway.

The geo-location privacy component maybe switched off by policies (see D3.2, sections 4.7.2. and 4.7.3), this might be achieved in several ways:

The GPS sensing element is switched off when a geo-location policy applies. The geo-location PET will stop receiving GPS data and generate aggregation vectors.

The geo-location PET receives an opt-out message from RD Adaptor (not shown in Figure 84); the geo-location PET generates an aggregation vector and stops until an opt-in message arrives. This alternative is less desired, as the GPS sensing element would continue sending data to the geo-location PET.

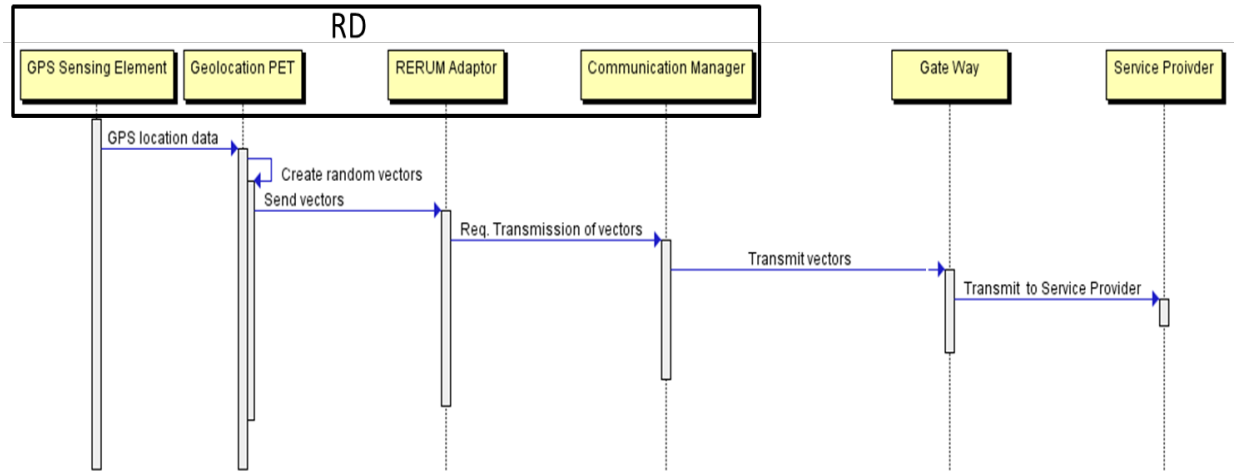


**Figure 84 - Interaction of the Geo-Location PET with RD components.**

### *3.10.2.9 Interfaces between privacy components and MW/on device components*

Privacy Components communicate with authorization components via the creation of Privacy policies. These privacy policies are XACML policies. As such, they can be evaluated by the authorization components in a similar way that they do for the access policies. In fact, the pPEP is actually a PEP that runs against the set of Privacy Policies against the access ones.

Consequently, the component that interface between the privacy components and the authorization ones is the Policy Manager, which deploys all policies, including the privacy ones on their respective policy stores.

In the general form, the interfaces of the SPT Manager with the other functional entities were described in Section 3.1. In Table 3, we provide more details about the internal interfaces between the privacy components and between the privacy components and the MW.

**Table 3: Description of interfaces of the Privacy components.**

| Interface name | Connected Components | Connection Title | Exchanged data description |
|---|---|---|---|
| IF_consApp | Consent Manager, Application | Request Consent, notify of granted consent | When there are no policies for handling an application request for user data, the consent of the data subject should be requested by contacting the Consent Manager. This interface is also used by the Consent Manager to notify about the granted consent. |
| IF_consPIP | Consent Manager, PIP | Request Data Subjects | The Consent Manager requests from the Policy Information Point the list of Data Subjects and their respective policies for identifying which subjects it will contact when there is a need for their consent. |

 

| | | | |
|---|---|---|---|
| IF_consUA | Consent Manager, User Agent | Review Consent, Grant/deny consent | When the data subject needs to review or change the policies regarding granting consent he has to contact the Consent Manager who replies with all the respective details or a failure in case no information exists for the specific data subject. This interface is also used by the user for granting or denying a specific request for consent. |
| IF_consPPDP | Consent Manager, PPDP | Evaluate Authorization Conflicts | When there is a need for finding out if there are conflicts between the privacy preferences and the requests for consents, the Consent Manager has to contact the PPDP to request for the evaluation of the conflicts. |
| IF_consPPREP | Consent Manager, pPRep | Add/delete privacy policy | After receiving the response of the user related with a specific request for consent, the Consent Manager contacts the pPRep to request an addition of a new privacy policy or the deletion of an existing one. |
| IF_PrDUA | Privacy Dashboard, User Agent | Change user privacy preferences, Grant Exception | The user sends requests to the Privacy Dashboard for changing his privacy preferences. The user sends also requests for granting exceptions from the privacy preferences for specific requests for consent. |
| IF_Act_DC | Activator/De-activator of Data Collection, Data Collector | Opt-out/in of data collection | When there is a need for changing the data collection status, the Activator/De-activator of Data Collection sends the respective request to the Data Collector in order to start or stop the data collection for a specific service. |
| IF_PrDAPs | Privacy Dashboard, Anonymiser and Pseudonymiser | Change or renew user or object identity | The Privacy Dashboard sends commands to the Anonymiser/Pseudonymiser in order to hide or renew the identity of a specific user/RD as the result of the applied PETs. |
| IF_geoRMan | Geo-Location Privacy Component, Resource Manager | Hide Position | The Geo-Location Privacy Component that exists on the RD sends processed data sets to the Resource Manager that either do not contain the user position (due to geo-privacy policies) or have a hidden position. |
| IF_AnPdPs | Anonymising and Pseudonymising Manager, De-Anonymiser/Pseudonymiser | Relink Pseudonym and Identities | When it is requested by the Administrator, the Anonymising/Pseudonymising Manager sends requests to the De-Anonymiser/Pseudonymiser in order to relink a previous or current pseudonym and reply with the identity of that pseudonym or a failure if linking is not possible. |

### 3.10.3 Functional components for Trust (Trust Manager)

The functional components of the Trust Manager are depicted in Figure 85 along with the external interfaces as described in Section 3.1. The Trust Manager is responsible for assessing the level of trustworthiness of an element of the RERUM system. The Trust Manager includes the following functional components: (i) Trust Configuration Manager, (ii) Reputation Rules Configurator, (iii) Trust Engine, (iv) Inaccuracy Alert Producer and (v) Inaccuracy Alert Reactor.

Trust and reputation require different (non static) evaluation and treatments for incoming and outgoing messages. For instance, incoming messages need to be authorized and the reputation of the requester must be evaluated for this purpose, but this is not needed for outgoing messages.

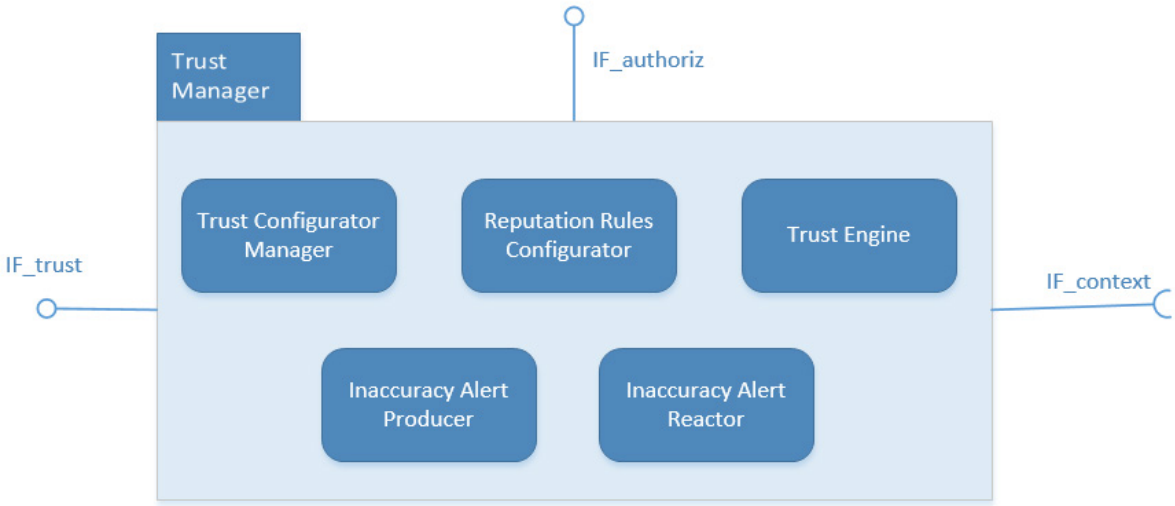


**Figure 85 – Trust Manager internal components.**

RERUM deals with these different needs by grouping the Trust components in 3 different levels (depicted in Figure 86): (i) a common trust infrastructure for those components that are needed to evaluate the trust and reputation, and one specific level for each (ii) incoming (Requester) and (iii) outgoing (Producer) message. More specifically:

- The **Requester layer** comprises the components that require the evaluation of trust or reputation and are specific to incoming request messages. For the moment, this level only comprises the distinct PEPs of the system (described in Section 3.10.1.5) but it is feasible that the architecture is expanded with additional components for the incoming messages.
- The **Producer layer** comprises the components that require the evaluation of trust or reputation and are specific to outgoing messages, which will normally correspond to data produced by the services of RERUM. For the moment, this layer only comprises the distinct Inaccuracy Alert Producer components (see Section 3.10.3 below), but it is feasible that the architecture is expanded with additional components for the outgoing messages.
- The **Trust infrastructure layer** comprises the components for defining and evaluating the trust and reputation criteria. The components that reside in the requester and producer layers invoke the ones in the Trust infrastructure layer for evaluating trust or reputation.

Note that the previously described layers are only conceptual ways of grouping the Trust components for ease of understanding; but they are not components themselves. Figure 86 shows how these components are grouped and related with each other:

 

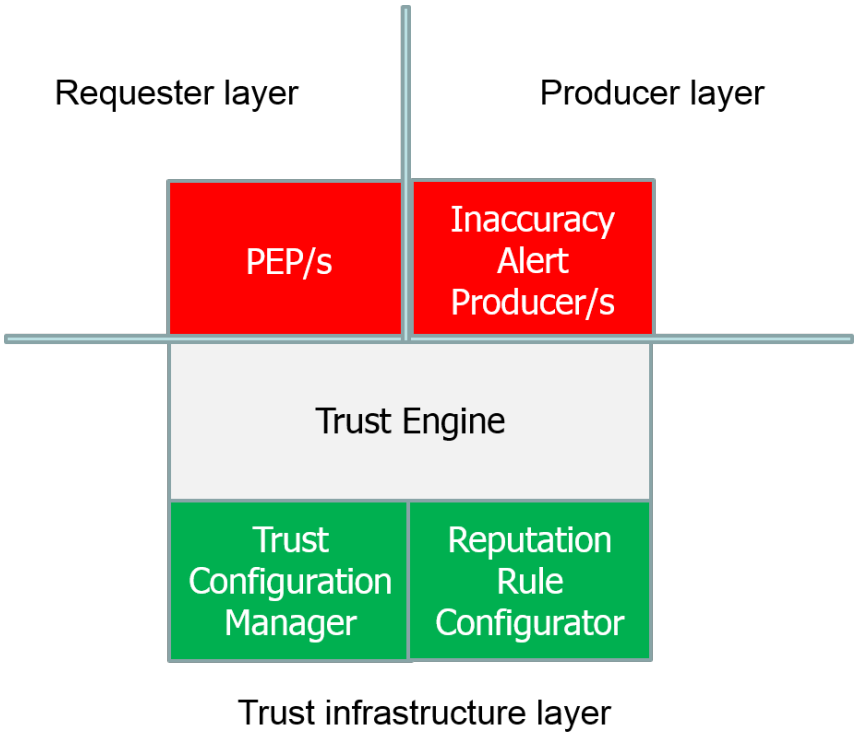


**Figure 86 – Trust components.**

### *3.10.3.1 Trust Configurator Manager*

This component will offer a GUI that will allow an administrator to graphically manage both the trust and reputation criteria. The input of the Trust Configuration Manager will be the interaction with the administrator for choosing the operations to be performed. The output of the Trust Configuration Manager will be the trust configuration already present in the system

### *3.10.3.2 Reputation Rules Configurator*

This component allows defining the criteria for evaluating the reputation of a given requester, which will be able to be used during the authorization process. The input of the reputation rules configuration is a set of valid reputation criteria provided or approved by at least one human being. The input of the Reputation Rules Configuration is a set of valid reputation criteria expressed in a standardized formal language. The output of the Reputation Rules Configuration is a set of valid reputation policies that will feed the Trust Engine (see below) and be stored on a store specific for this purpose.

### *3.10.3.3 Trust Engine*

The Trust Engine is responsible for evaluating the reputation criteria obtained from the Reputation Rule Configurator. The trust engine is built on top of the reputation rule configurator, because it retrieves its reputation rules from there. The Trust Engine is invoked at 2 different levels: **Requester** and **Producer**, depending on whether it is used for evaluating the reputation of a requester or the accuracy of the produced data respectively. Hence, as the purpose for using the Trust Engine is different for these two levels, the API for accessing it may differ as well. In any case, the Trust Engine retrieves the reputation rules from the store of the reputation policies and executes them to produce its output. The input to the Trust Engine is the applicable reputation policies and the request for incoming requests or an event with the response for the produced data. The output of the Trust Engine is a reputation measurement.

Figure 86 (above) shows the relation of the Trust Engine with the rest of the Trust components. At the requester level, it is invoked by the PEP to obtain a measurement of the reputation of the requester

before authorizing their requests. At a producer level, it is invoked by the Inaccuracy Alert Producer to update the reputation of the VRDs that produce the data.

Figure 87 shows the generic procedure for setting up and evaluating the reputation at a producer level. For a concrete example of a component evaluating the accuracy of the VRD sensor, see the Inaccuracy Alert Producer and Reactor components in Section 3.10.3.4 and Section 3.10.3.5 below.

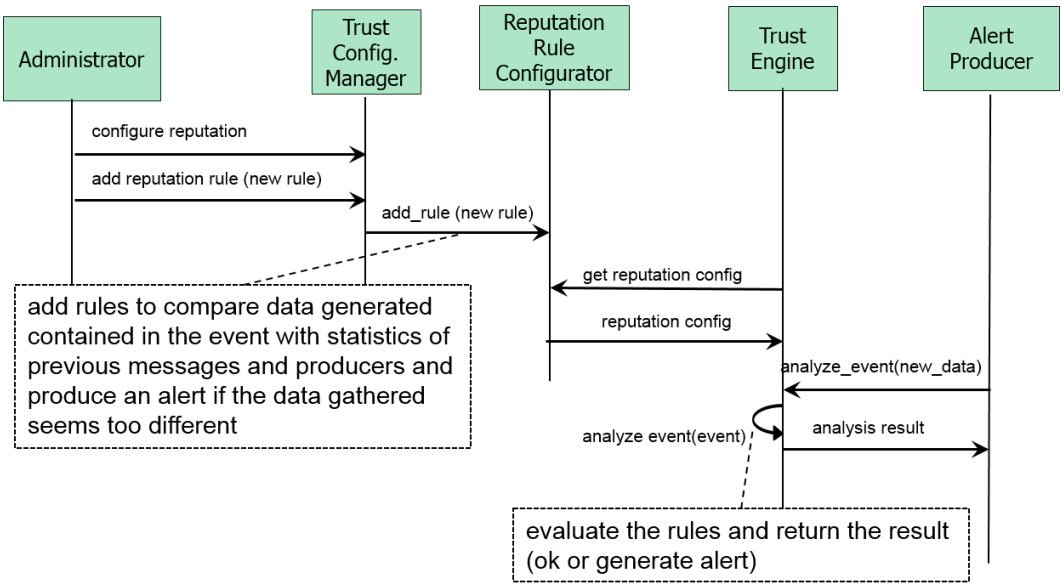


**Figure 87 – VRD reputation evaluation.**

Figure 88 shows the generic procedure for setting up and evaluating the reputation at a requester level. The general authorization procedure follows this procedure.

#### *3.10.3.4 Inaccuracy Alert Producer*

The Inaccuracy Alert Producer is a specific example of the Alert Producer described in Section 3.8.2.4.1. This component analyses outgoing measurements from the installed sensors comparing them against the existing ones and producing alerts in case their measurements are significantly different from the already obtained ones. The input for the Inaccuracy Alert Producer is the current and passed measured values and the reputation rules. The output for the Inaccuracy Alert Producer is none, in case no anomalous behaviour is obtained or an inaccuracy alert in any other case.

 

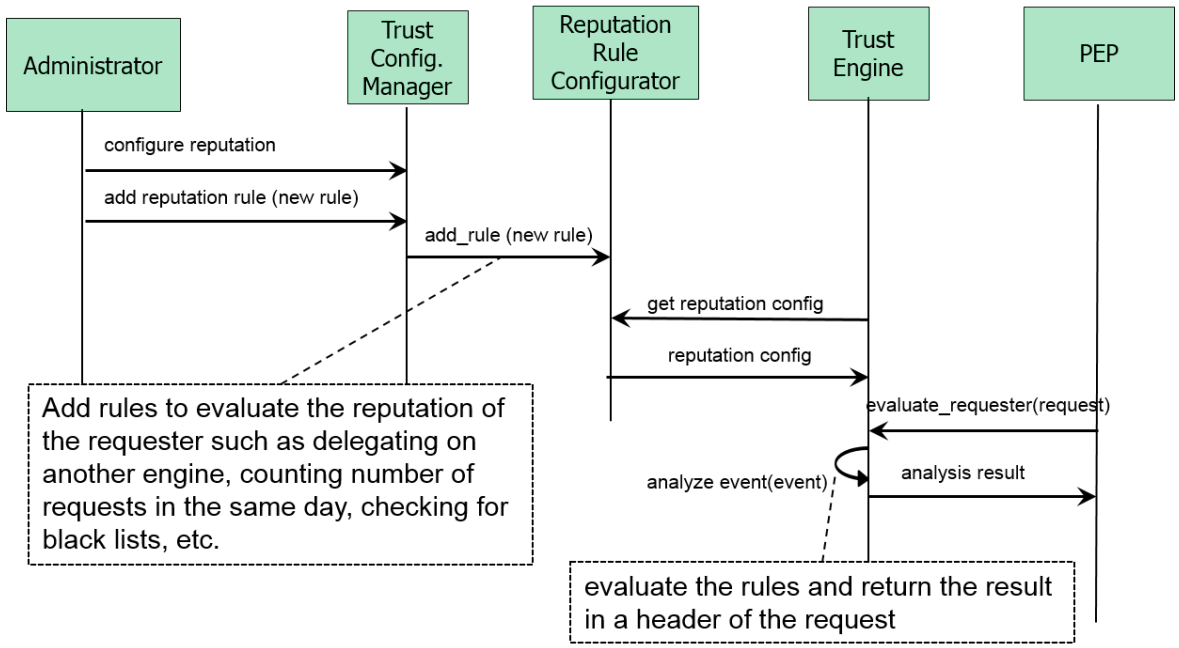


**Figure 88 – Requester reputation evaluation.**

### *3.10.3.5 Inaccuracy Alert Reactor*

The Inaccuracy Alert Reactor is a specific example of the Alert Reactor described in Section 3.8.2.4.2. This component produces reactions for inaccuracy alerts. More specifically, it will generate a message for the administrator when a very inaccurate measurement has been received. In order to do so, it communicates with the Trust Engine (described above) to analyse the measurement and produce the alert. The input for the Inaccuracy Alert Reactor is the alerts generated by the Inaccuracy Alert Producer. The Inaccuracy Alert Producer and Reactor are good examples of how the alert / producer mechanisms work in the system, and in this concrete case, for evaluating the reputation of a VRD from the measurements that it is producing. Figure 89 shows how the general reputation mechanism fits with the Inaccuracy Alert reputation mechanism.

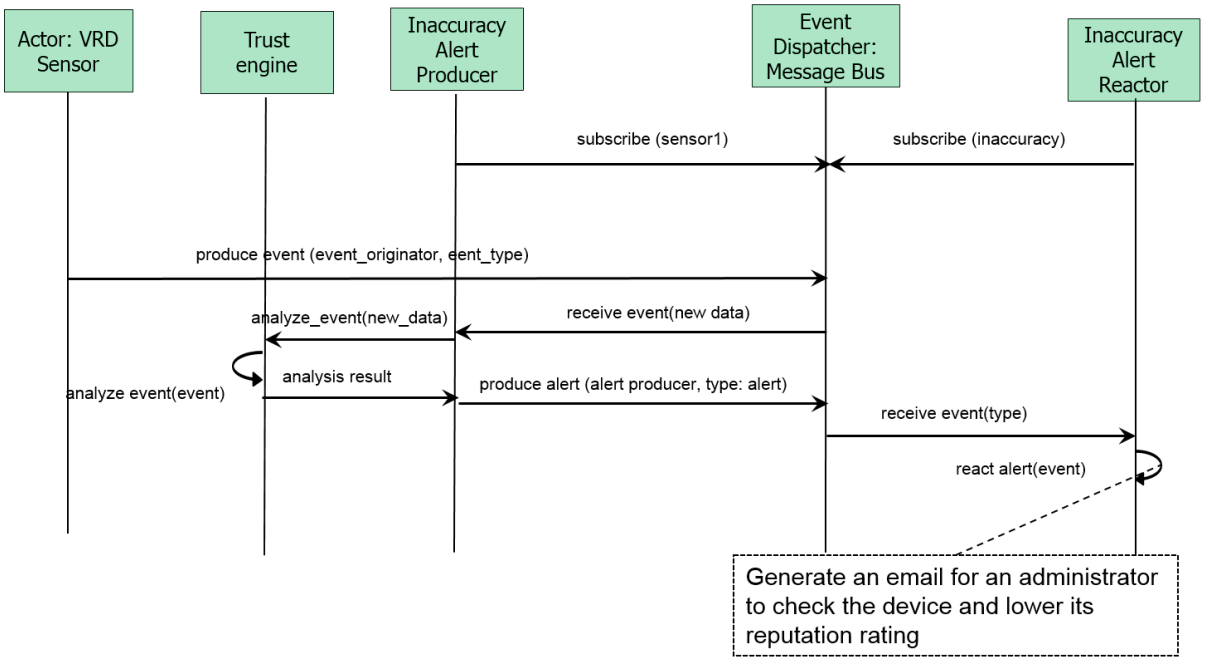


**Figure 89 – Inaccuracy reputation mechanism.**

#### *3.10.3.6 Interfaces for Trust components*

The interfaces of the SPT Manager with the other functional entities were described in Section 3.1. Of significant importance are the interfaces with the Message/Event Bus of the Data & Context Manager as described below:

- IF_alPro_Mbus: this is the interface between the Alert Producer and the Message/Event Bus and is used by the Alert Producer to subscribe to a data queue of the Message/Event Bus in order to retrieve information that can be evaluated to generate alarms. The alarms are also published in the Message/Event Bus in another specific queue in order to be read by the Alert Reactor.
- IF_alRe_Mbus: this is the interface between the Alert Reactor and the Message/Event Bus and is used by the Alert Reactor to subscribe to the Message/Event Bus in order to retrieve the alarms that are generated for managing them and updating the trust status accordingly.

## 3.11 Application layer

As mentioned before, the application layer is basically out of RERUM's scope. This means that RERUM will not conceptually define functional components on how to develop applications or how to present the results to the end user. Nevertheless, RERUM will develop its own draft applications for the trials that will be executed within the activities of WP5, but only for demonstration purposes. However, in order to have a fully functional system, the RERUM framework will need to receive the application/service requests with a specific format through an API. This API will be defined in the next deliverables of WP2, when we will have more knowledge on the exact input that is needed from the applications.

From a security perspective the service access/request from the application to the RERUM framework needs to be secured. Either the application is authenticated directly from the RERUM framework or the user that sends the service request (through an application) is authenticated. In the first case it will be a fixed security association between the application and the RERUM framework, whereas in the second case it will be a dynamic one based on the concrete user access request. Part of the authentication process is also typically some kind of dynamic key negotiation between the application and the RERUM framework to be applied for further protecting the communication payload like confidentiality or integrity protection. As user/application authentication is not in the research scope of RERUM, RERUM will apply existing standard security mechanisms like username/password authentication, pre-installed PKI credentials or similar applicable mechanisms.

The User should be able request a service no matter how complex it is through a well-defined interface, using a meta language that describes the resources and the services that the VRDs expose. As mentioned in [14] one option is to use a business process language, i.e. XPDL [69], BPMN [70], etc. However, these languages are of general purpose and designed to model many workflow environments, which makes them very complicated for IoT. As a simpler solution, OpenIoT considers the use of rule-based description languages, such as JBoss Drools XML Rule Language, JenaRules, RuleML, SWRL, SWSL or SPARQL [71]. Similar with OpenIoT, iCore also uses SPARQL as the basic language for the user service request. RERUM will evaluate the requirements that the functional entities have from the applications in order to make a similar decision on which language will be selected for requesting a service from the application point of view and this will be included in the next deliverable D2.4 due end of February 2015.

 

## 3.12 Interplay of Middleware components

Within RERUM we have used the term "Middleware" to refer to the collection of functional components that are interworking for enabling the RERUM Devices to provide services to the users as described in the introduction of this chapter. The functional entities that comprise the Middleware can be seen in **Figure 90**, which depicts also the interactions between the functional entities. However, in the next paragraphs we show in more detail the interconnectivity of the functional components that comprise the Middleware, as well as two examples of data flows when a user requests a simple service and when he requests a federation of VRDs.

### 3.12.1 MW functional components interconnectivity

Middleware internal components interconnectivity. The following Figure 90 shows an example how the Middleware functional components can be typically interconnected to each other.

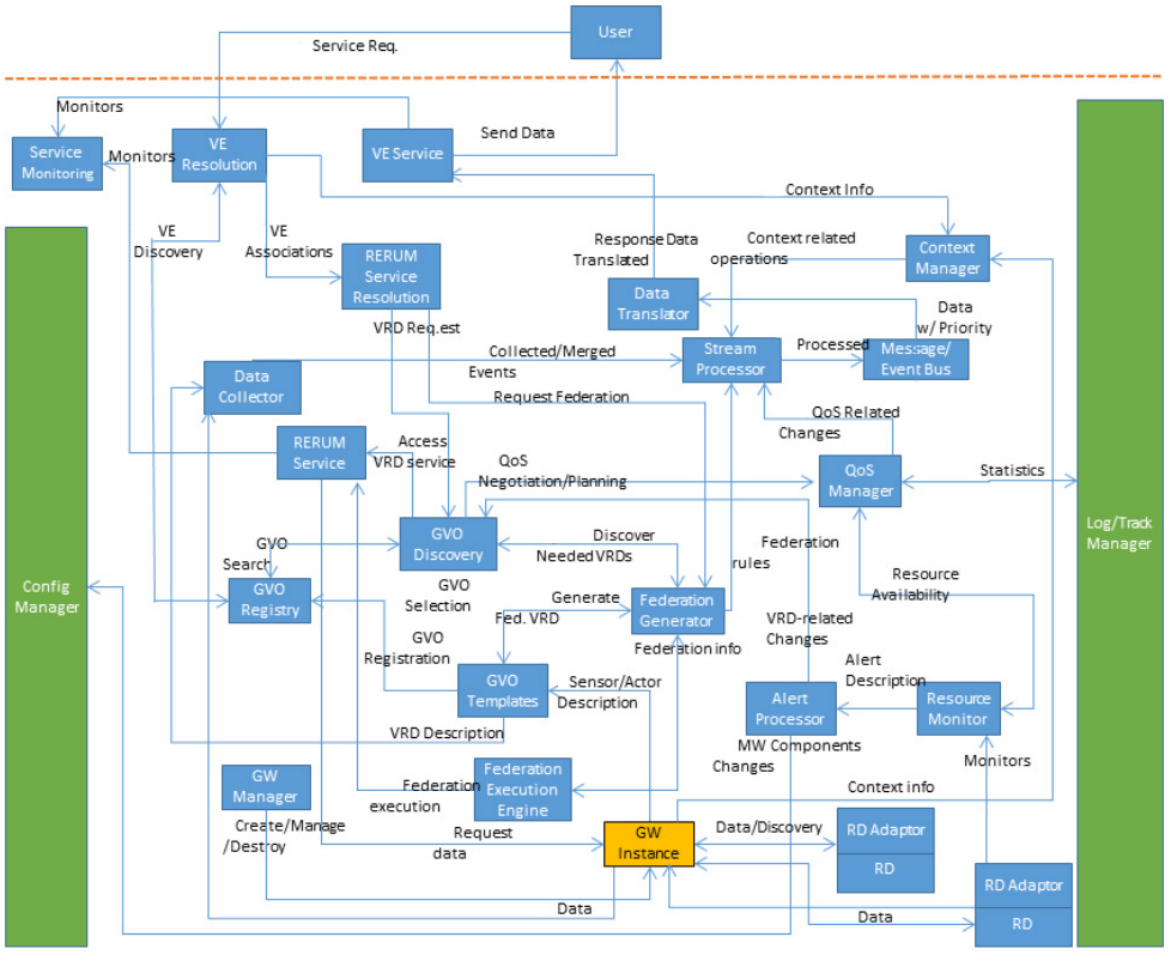


**Figure 90 – RERUM Middleware functional components interactions.**

### 3.12.2 Basic temperature sensing example

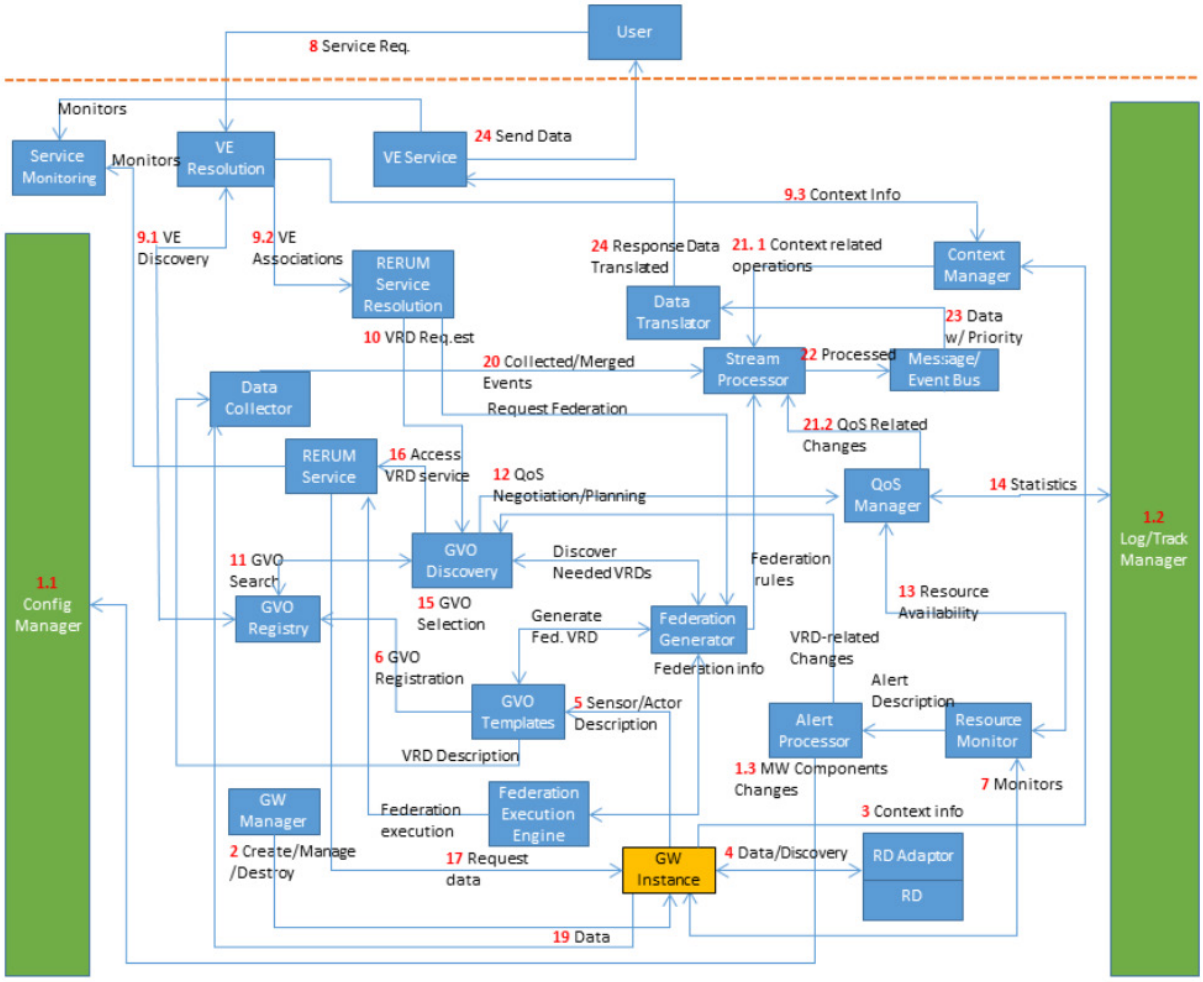


**Figure 91 – Middleware components interactions for addressing a service request.**

The steps of the data flow for the example are described below (and shown in Figure 91):

1. Bootstrapping:
   1.1. The Configuration Manager helps the bootstrapping of the MW components
   1.2. The Log/Tracker Manager starts tracking the states of the components and the exchanges between them.
   1.3. The Alert Processor updates the Configuration Manager with alerts regarding the MW components (if needed).
2. The GW Manager initializes and sets up the GW Instance.
3. The GW Instance sends context information to the Context Manager. This might be location, domain, etc.
4. An RD that has a temperature sensor registers itself with the GW Instance - or is discovered by it, through the RD Adaptor.
5. The GW takes the RD's description and maps it to an existing or adds a new matching VRD Template.
6. The VRD Template receives the RD information, links them with the GW and other information and stores them in the searchable GVO Registry.

 

7. The Resource Monitor component starts monitoring the temperature sensor RD.
8. A temperature reporting application requests a service which enables it to monitor the temperature of i.e. the bedroom of a home. It describes the needed services and, optionally, also provides some context information. This also includes QoS/SLA descriptions.
9. Service Request:
   9.1. The VE Resolution component analyses the service request, and discovers the VE that is associated with the specific service request (i.e. finds the bedroom for which the user asked the temperature) through a query in the GVO Registry.
   9.2. Then the VE Resolution sends these associations to the RERUM Service Resolution component and the respective context of the service to the Context Manager
10. The RERUM Service Resolution receives the associations and tries to identify the services that can provide information about temperature and sends request to the VRD discovery to identify which VRDs can provide these services (i.e. which are the temperature sensors in the bedroom?).
11. The VRD Discovery makes a search in the registry to identify these VRDs
12. If suitable VRDs are found, the Discovery Component also sends the QoS requirements with the QoS Manager.
13. The QoS Manager processes the requirements descriptions, checks the near real-time availability of the requested resources and sends statistics to the Log/Track Manager
14. The statistics are stored into the Log/Track Manager.
15. After the VRD Discovery, a suitable VRD is selected and
16. the respective service of the VRD is invoked.
17. The service requests data from the device (through the GW and the RD Adaptor)
18. Temperature readings start to flow from the RD, and
19. through the GW Instance, to the Data collector.
20. The Data collector receives data, processes them, merges data concerning an event, creates events, etc. and sends them to the Stream Processor.
21. (21.1 and 21.2) If needed, context and QoS Related operations like accuracy, are performed
22. Data are pushed on the Message/Events bus, where data is prioritized
23. Data are sent to the Data Translator, which formats the data into the format understood by the App and
24. sends them to the VE service that forwards them to the application.

 

### 3.12.3 Basic Federation example

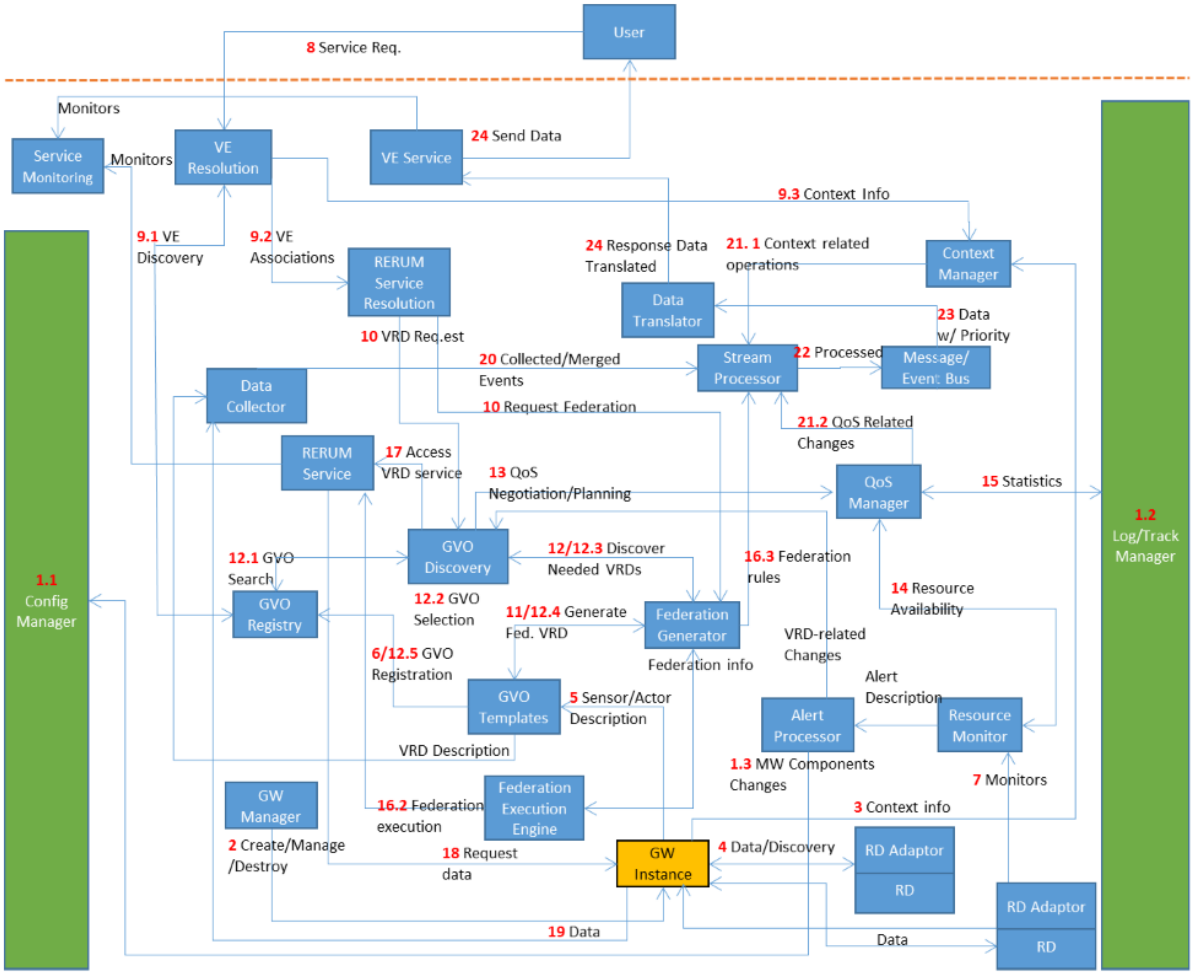


**Figure 92 – Interactions for addressing a service request that requires a federation of RDs.**

The steps of the data flow for the Federation example are described below (and shown in Figure 92).

1. Bootstrapping:
   1.1. The Configuration Manager helps the bootstrapping of the mw components
   1.2. The Log/Tracker Manager starts tracking the states of the components and the exchanges between them.
   1.3. The Alert Processor updates the Configuration Manager with alerts regarding the MW components (if needed).
2. The GW Manager initializes and sets up the GW Instance.
3. The GW Instance sends context information to the Context Manager. This might be location, domain, etc.
4. An RD#1 that has a temperature sensor and another RD#2 that has a Window Controller register themselves with the GW Instance - or are discovered by it, through the RD Adaptor.
5. The GW takes the descriptions of the RDs and maps them to the appropriate VRD Templates (or creates new templates).
6. The VRD Template stores and links the RDs description and associations in the searchable GVO Registry.

 

7. The Resource Monitor component starts monitoring the RDs.
8. An application controlling windows sends a service request using the rule “Open-Window-if-Temperature-too-High”. It describes the needed services, the model of the Federation and the program that is to be executed by it (simple, rule-based) and, optionally, also provides some context information. This also includes the QoS/SLA descriptions.
9. Service Request:
   9.1. The VE Resolution component analyses the service requests, and discovers the VE that is associated with the specific service requests (i.e. finds the bedroom for which the user asked to control the temperature) through a query in the GVO Registry.
   9.2. The VE Resolution sends these associations to the RERUM Service Resolution component and the respective context of the service to the Context Manager.
10. The RERUM Service Resolution receives the associations and tries to identify the services that can provide the requested service from the user application. The component identifies that this request cannot be met by a single service and a Federation is required. Thus, it analyses the services that have to be composed for this federation and sends this information to the Federation Generator.
11. The Federation Manager receives the services associations and queries the GVO Templates to find a Federation Template which is returned to the Federation Generator.
12. The Federation Generator sends information to the VRD Discovery to discover the needed VRDs for the Federation:
   12.1. The VRD Discovery Component, upon receiving the description of the service, searches for suitable VRDs in an already existing Federation.
   12.2. The selection of VRDs is sent back to the VRD Discovery, which
   12.3. forwards the information to the Federation Generator and stores the federation as a new GVO Template which
   12.4. is then added in the Registry.
13. If suitable VRDs are found, the Discovery Component also checks the QoS requirements with the QoS Manager.
14. The QoS Manager processes the requirements descriptions, checks availability of the requested resources and sends statistics to the Log/Track Manager
15. The statistics are stored into the Log/Track Manager.
16. Federation execution:
   16.1. The Federation Execution Engine receives from the Federation Generator the program that is to be executed.
   16.2. The Federation Execution Engine starts executing the Federation program, as provided by the App accessing the Services of the respective VRDs.
   16.3. The Stream Processor gets the Federation rules about i.e. filtering/aggregating the data streams from the various RDs that are federated.
17. The respective services of the Federated VRDs are invoked.
18. The services request data from the RDs (through the GW Instance and the RD Adaptor)
19. Status updates and other information of relevance to the application start to flow from the Federation Head to the Data collector.
20. Data collector receives data, processes them, merges data concerning one event, creates events/etc. and sends them to the Stream Processor.
21. (21.1. and 21.2) If needed, context and QoS Related operations like accuracy, are performed

22. Data are pushed on the Message/Events bus, where data is prioritized
23. Data are sent to the Data Translator, which formats the data into the format understood by the application and
24. sends them to the VE service that forwards them to the application.

### 3.12.4 Interfaces between MW components

Table 4 presents the details regarding the internal interfaces between the Middleware components that are depicted in Figure 90.

**Table 4: Middleware internal interfaces description.**

| Interface name | Connected Components | Connection Title | Exchanged data description |
|---|---|---|---|
| IF_Us_VER | User,<br>VE Resolution | Service Request | REST Service call by the User to the VE Resolution component, describing the feature of interest and the need services. E.g., the description of an intersection and the current, local average temperature. |
| IF_VES_Us | VE Service,<br>User | Send Data | REST Service (or, by request, other formats) exposing to the User the data initially request, provided, filtered and processed by the RERUM middleware. |
| IF_RSer_Smon | RERUM Service, Service Monitoring | Monitors | Monitoring and statistical information related with the status of the respective service. |
| IF_VEser_Smon | VE Service,<br>Service Monitoring | | |
| IF_VER_CoMan | VE Resolution, Context Manager | Context Information | The contextual information related to both the User and to the selected GVOs is forwarded to the Context Manager component, which then can process User-relevant events using the additional information. |
| IF_VESer_Dtran | VE Service,<br>Data Translator | Response Data Translated | If required by the User, the RERUM Middleware can provide specific formatting and end-point interfaces to the requested information. |
| IF_VER_RSRes | VE Resolution,<br>RERUM Service Resolution | VE Associations | |

 

| | | | |
|---|---|---|---|
| IF_RSRes_Disc | RERUM Service Resolution,<br>GVO Discovery | RD Request | The discovery of VRDs associated to relevant VEs. |
| IF_RD_GWI | RD Adaptor,<br>GW Instance | Data Discovery | RDs registration and resource-readings being send to the GW. |
| IF_GWM_GWI | GW Manager,<br>GW Instance | Create/ Manage/ Destroy | The information required by a RD in order to configure itself, re-configure itself or to disable itself from acting as a GW Instance |
| IF_GWI_CoMan | GW Instance,<br>Context Manager | Context Info | Contextual information provided by the GW Instance to the Context Manager.<br><br>The Gateway Instance has a local awareness, and forwards relevant information. |
| IF_DC_GWI | Data Collector,<br>GW Instance | Data | Any reporting forwarded by the GW Instance to the RERUM middleware, stemming from the underlying RDs. |
| IF_Temp_GWI | GVO Templates,<br>GW Instance | Sensor/ Actor Description | An initial (on registration) description of the model of the RDs or of the PEs. |
| IF_Temp_Reg | GVO Templates,<br>GVO Registry | GVO Registration | The RDs and PEs description, appended with information provided by the GW Instance. |
| IF_Disc_Reg | GVO Discovery,<br>GVO Registry | GVO Search | Query prepared based on the required resources, as described by the User and augmented by the intermediary VE resolution components. |
| IF_Disc_QoS | GVO Discovery,<br>QoS Manager | QoS Negotiation Planning | QoS requirements (provided by the User) are to be considered during the VRD discovery process. The QoS Manager provides the required parameters for possible matches in the GVO Registry. |
| IF_QoS_SProc | QoS Manager,<br>Stream Processor | QoS Related Changes | QoS-related information (e.g. processing time restrictions, accuracy of the reporting, interval) are considered during the processing of the reported properties by the Stream Processor. |
| IF_Mbus_DTran | Message/Event Bus, | Data with Priority | Messages and events are, based on their priority, forwarded to the corresponding end-points. |

| | | | |
|---|---|---|---|
| | Data Translator | | |
| IF_SProc_Mbus | Stream Processor,<br>Mesage/Event Bus | Processed | Generated events and reports are, once the processed, forwarded into a prioritizing message queue |
| IF_Disc_FedM | GVO Discovery,<br>Federation Manager | Discover Needed VRDs | When the composition of a Federation is required, the GVO Discovery forwards its description, constraints, etc. to the Federation Manager component. |
| IF_FedM_Temp | Federation Manager,<br>GVO Templates | Generate Fed. VRD | Generated VRD Federations are stored as a template, similar to RDs and PEs. |
| IF_ResM_QoS | Resource Monitor,<br>QoS Manager | Resource Availability | The Resource Monitor periodically forwards information related to the resource availability of the underlying sensor networks to the QoS Manager, which then can provide input based on its current representation of their states during processing and selection processes. |
| IF_ResM_AlProc | Resource Monitor,<br>Alert Processor | Alert Description | Once a resource's state reaches a certain threshold, an alert is generated and sent to the Alert Processor component. |
| IF_AlProc_Disc | Alert Processor,<br>GVO Discovery | VRD-related Changes | Describes the alert-type and the affected RDs, which can trigger a re-discovery of suitable VRDs or Federations of VRDs, if required. |

 

# 4 RERUM deployment scenarios

This section presents the final version of the deployment view of RERUM considering the use case scenarios that were described in deliverable D2.1 [2], the trial scenario topologies described in D5.1 and the initial version of the deployment view described in D2.3. The functional view of the architecture described in the previous sections does not assume the location of each of the functional entities and their components on physical devices (apart from the RD Adaptor, which should always be located on the RDs of course). This is mainly left as a design choice for network or service operators that are willing to deploy a RERUM system. However, here we provide a generic view on the possible topologies of RERUM system deployments, as well as the physical devices that are assumed to be part of the RERUM system, their interconnectivities and some example data flows between them. The generic view is an updated version of the one presented in D2.3 and was extracted from the description of the use cases in D2.1 with the assistance of the functional view of the architecture and the domain model.

## 4.1 RERUM network topology

In this part, the high level network topology of RERUM is being presented, alongside with examples of the deployment of a RERUM system. The section starts with a high level design of the network deployment, identifying the possible scenarios for the various types of physical components that exist within a network, as well as their interconnectivities and with a brief description of the network interfaces between the components. Then, example deployment scenarios for the indoor and outdoor use cases are given (considering these two general categories of the RERUM use cases as described in D2.1). Furthermore, an example of a possible hybrid/hierarchical deployment of RERUM systems is also given within this section. The goal is to describe a generic RERUM topology that can include all use cases as special cases with as few modifications as possible. We also note that these topologies and scenarios are not the final ones and may be revised until the final RERUM architecture deliverable, according also to the research results of the technical WPs.

### 4.1.1 High level design

Figure 93 presents the high level view of the RERUM network topology. The basic physical components are (i) the RERUM Devices, (ii) the RERUM Gateways, (iii) the Application Server, (iv) the Security Center and (v) the Middleware Server. As described in the previous sections, the Middleware may be distributedly implemented in various physical components, but here for the sake of simplicity we assume to have it in a single physical component. The figure also denotes the components that are inherent to the RERUM System, which are all components apart from the Application Server, which, as has been mentioned before, is outside the scope of the project. The figure presents also two special cases of the RERUM Devices, namely (i) the RD as an Aggregator/Cluster Head that interconnects other RDs and (ii) the RD as a mobile phone, which does not always connect to a GW and can be connected to the Application Server or the Middleware through Wide Area Network (WAN) connectivity via its e.g. 3G interface. The figure also shows the various network interfaces that are used for the interconnectivity of the physical components. These network interfaces are explained in the next subsection.

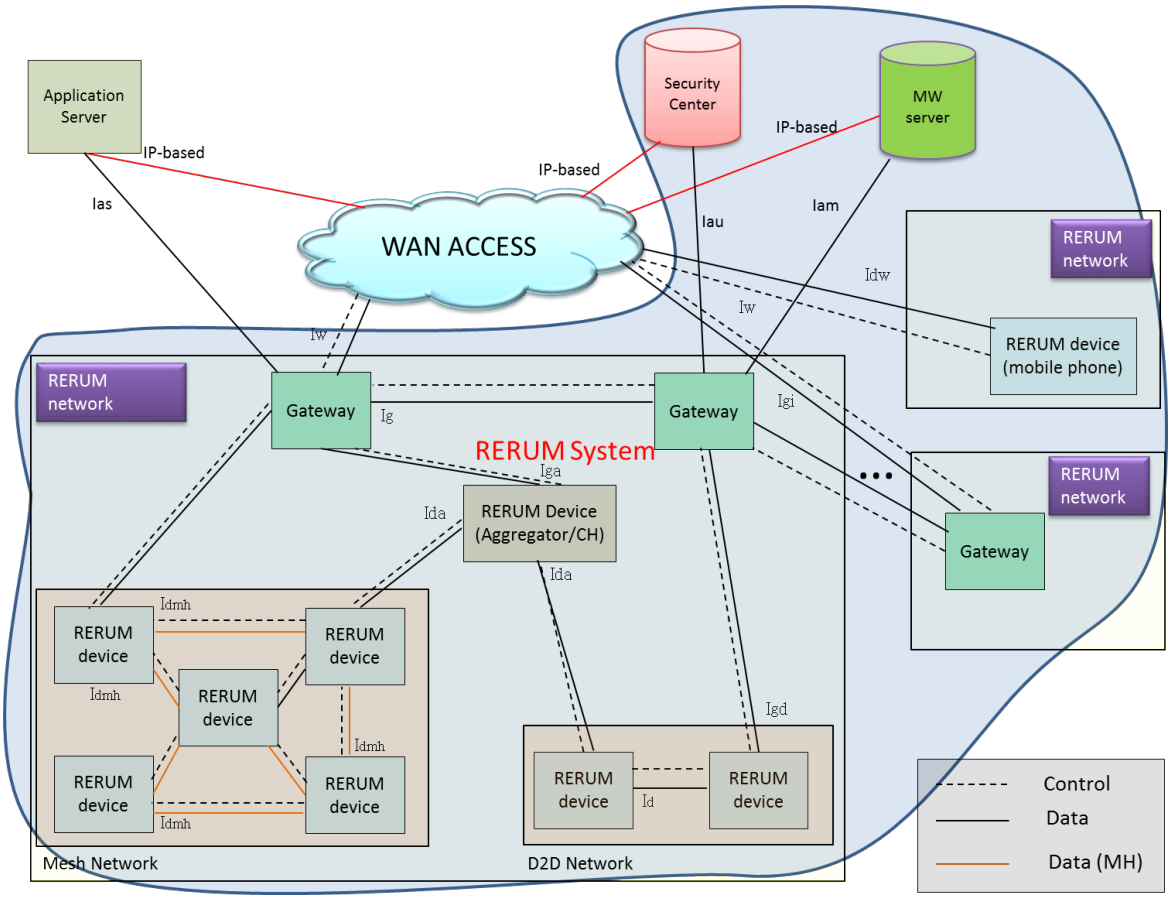


**Figure 93 – High level view of the RERUM network topology.**

## 4.1.2 Interfaces

**Table 5: Network interface description.**

| Interface name | Adjacent Elements | Interface description |
|---|---|---|
| Id | Network Interface between RDs | *Physical layer protocol*<br>2.4GHz IEEE 802.15.4,<br>Effective rate: 250Kbps<br>Transport layer<br>TCP, UDP<br>Routing layer<br>IPv6 (6LowPAN)<br>*Application protocol*<br>HTTP, CoAP |
| Idmh | Network Interface between devices via hops (can use IEEE 802.15.4) | |
| Igd | Network Interface between devices and gateway | |
| Ig | Network Interface between gateways in the same network | |
| Igi | Network Interface between gateways in different networks | N/A in RERUM |

| | | |
|---|---|---|
| Ida | Network Interface between devices and aggregators (Border GWs) | Same as for Id interface above. |
| Iga | Network Interface between gateways and aggregators | Same as for Id interface above. |
| Iw | Network Interface between gateways and WAN | Option #1 (indoor UCs):<br>*LAN*<br>Ethernet<br>*WAN*<br>ADSL: ITU-G.991.1<br>ADSL2+: ITU-G.992.5<br>VDSL2: ITU-G.993.2<br>*Application protocol: HTTP*<br>Option #2 (outdoor UCs):<br>GPRS: 171,2 kbit/s,<br>EDGE: 384 kbit/s<br>HSPA: 14 Mbit/s,<br>HSPA+: 168 Mbit/s<br>*Application protocol: HTTP* |
| Ias | Network Interface between gateways and Application Server | |
| Iau | Network Interface between gateways and Security Center | |
| Iam | Network Interface between gateways and Middleware | |
| Idw | Network Interface between devices and WAN | |

### 4.1.3 Example deployments

In this subsection, we will provide some examples of deployments of the RERUM system in indoor and outdoor scenarios, mapping the deployments to the RERUM Use Cases as they were described in D2.1 and after evaluating the initial scenarios presented in D2.3 together with the deployment scenarios described in D5.1.

#### *4.1.3.1 Indoor deployments*

For the indoor scenarios we assume that the MW can be implemented on an indoor server and partly on the GW. Depending on the design choice of the administrator, the MW could be fully implemented on the GW, however here we want to capture the general case where the MW is separated from the RERUM GW. In D2.3 we had presented also scenarios with full implementation of the MW on the GW. We also assume that the security/privacy server (Security Center) is located at the indoor RERUM Network, because there is no need of having an external server managing the security/privacy of the internal system. Even if the Security Center is located in the internet, there is no real change in the network topology.

Acknowledging that some indoor applications may also need external information, we include in the figures the box "**internet resources**". One example of usage of such resources is if we have a federation for controlling the Air Condition with regards to the city temperature and we want to get somehow the temperature of the city area without having to deploy additional outdoor sensors. In this case, the Federation Manager on the GW should access an external web service to get information on the average city temperature. However, this information should also pass through the trust mechanisms

to identify that it can be trusted before used. Furthermore, we can see an RD playing the role of an aggregator, combining data from other RDs.

Figure 94 shows the generic indoor network topology for a RERUM system. The figure depicts also the various options for the deployment of the components, namely the MW, the Security Center and the Application Server. Throughout the scenario figures, the **red arrows** represent control plane exchanges, while the **green arrows** represent data plane exchanges.

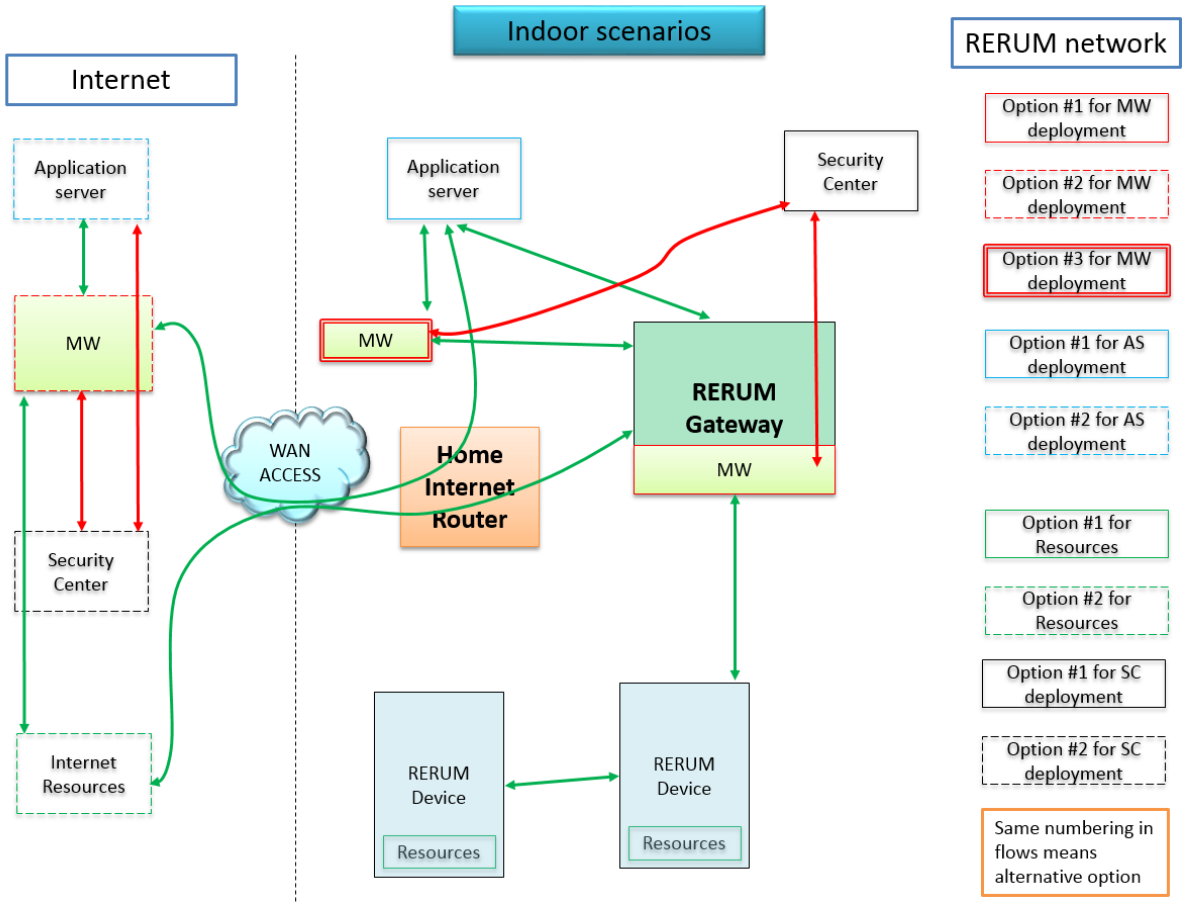


**Figure 94 – Generic RERUM Indoor Topology options.**

Figure 95 presents a simplified example scenario of the indoor deployment, assuming that the Application Server lies within the indoor deployment, meaning that the RERUM applications are installed and utilized only by the inhabitants of the house. For simplifying the figure we have removed the “Internet Home Router” assuming that its role is captured by the Gateway.

 

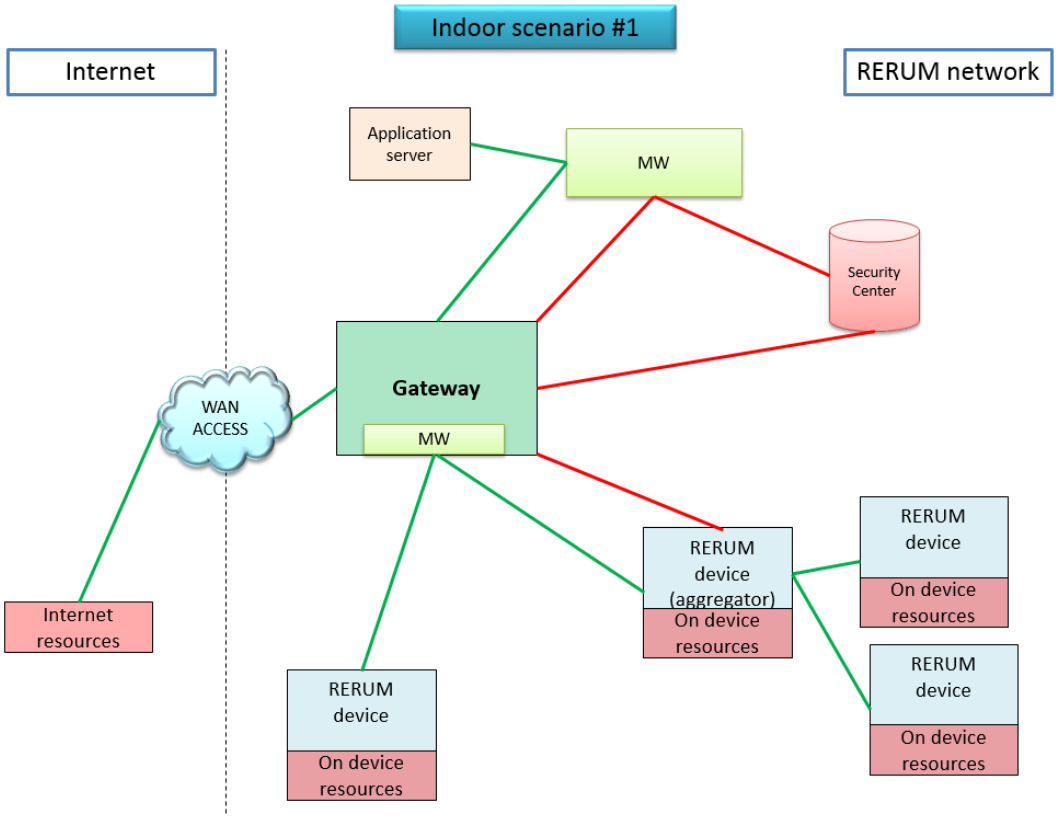


**Figure 95 – Scenario Indoor 1 – Both Application Server and MW indoor.**

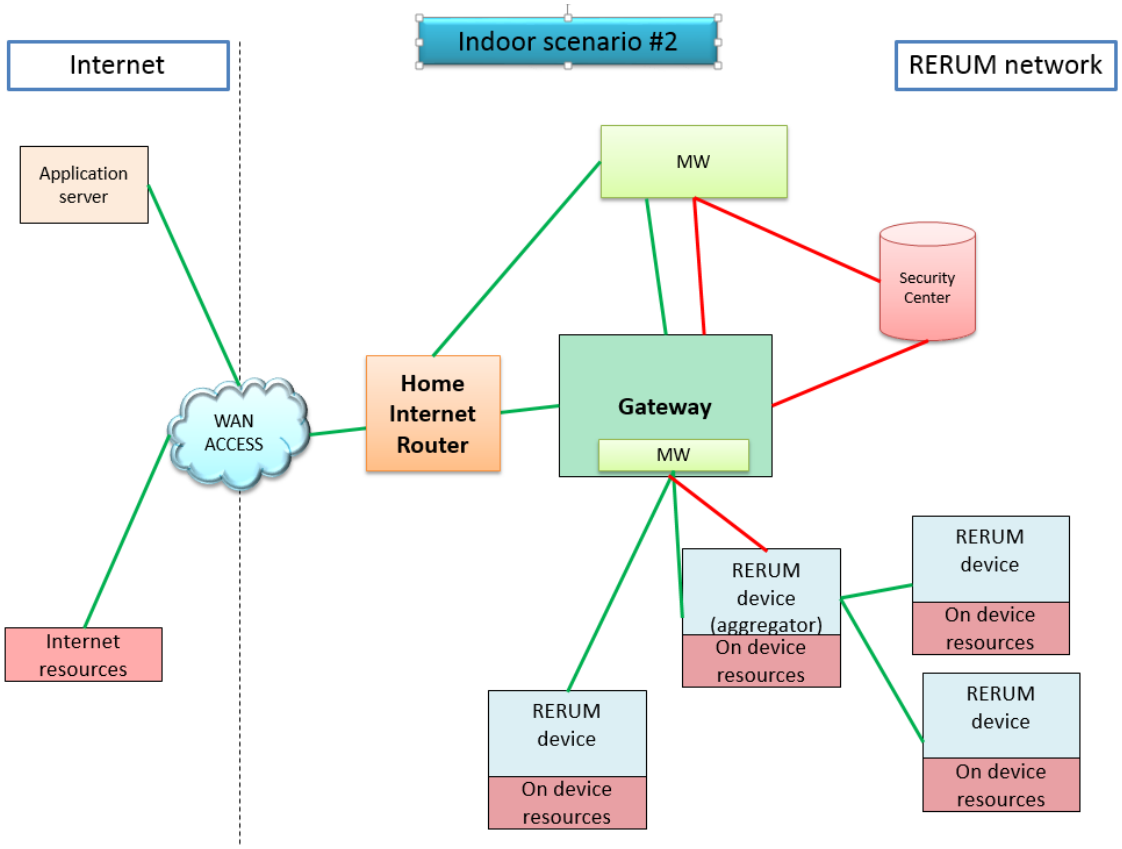


**Figure 96 – Scenario Indoor 2 – Application Server in the internet and MW indoor.**

Figure 96 shows the second example scenario of an indoor deployment, where the Application Server exists outside the RERUM Deployment (in an internet server). This is a very realistic scenario, since many companies providing smart home applications are installing their applications on their own servers and not inside the users home. In this scenario we include also the Home Internet Router, because it is the one that first receives the application requests and forwards them to the MW to be handled accordingly.

Figure 97 shows the third option of the indoor deployments, where both the Application Server and the MW are installed in internet locations (i.e. the Cloud) outside the user's premises. In this case, some parts of the Security Center have to be located in the internet too, i.e. the Authorization and Access Control functionalities, while other parts can be located indoor, i.e. the privacy and trust modules, as well as the encryption, integrity verification, etc.

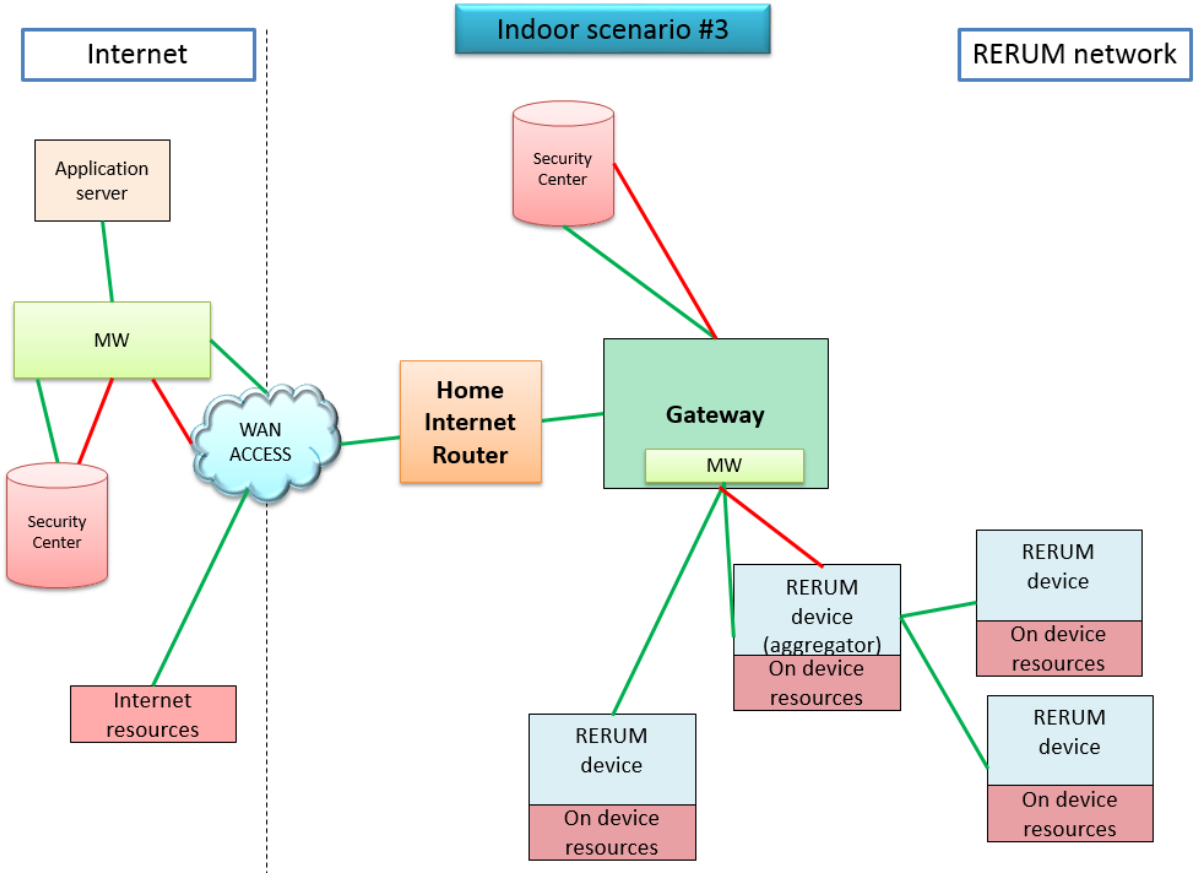


**Figure 97 – Scenario Indoor 3 – Both Application Server and MW in the internet.**

### *4.1.3.2 Outdoor deployments*

In the outdoor deployments, most of the MW functionality is always implemented in an internet server, which we assume that is secure and trusted. However, the GVO Registry and the VRD Discovery can be distributedly implemented in the Gateways to improve the scalability of the system. The Security Center and the application server are also located in the internet. The general topology of the outdoor RERUM systems is presented in Figure 98.

 

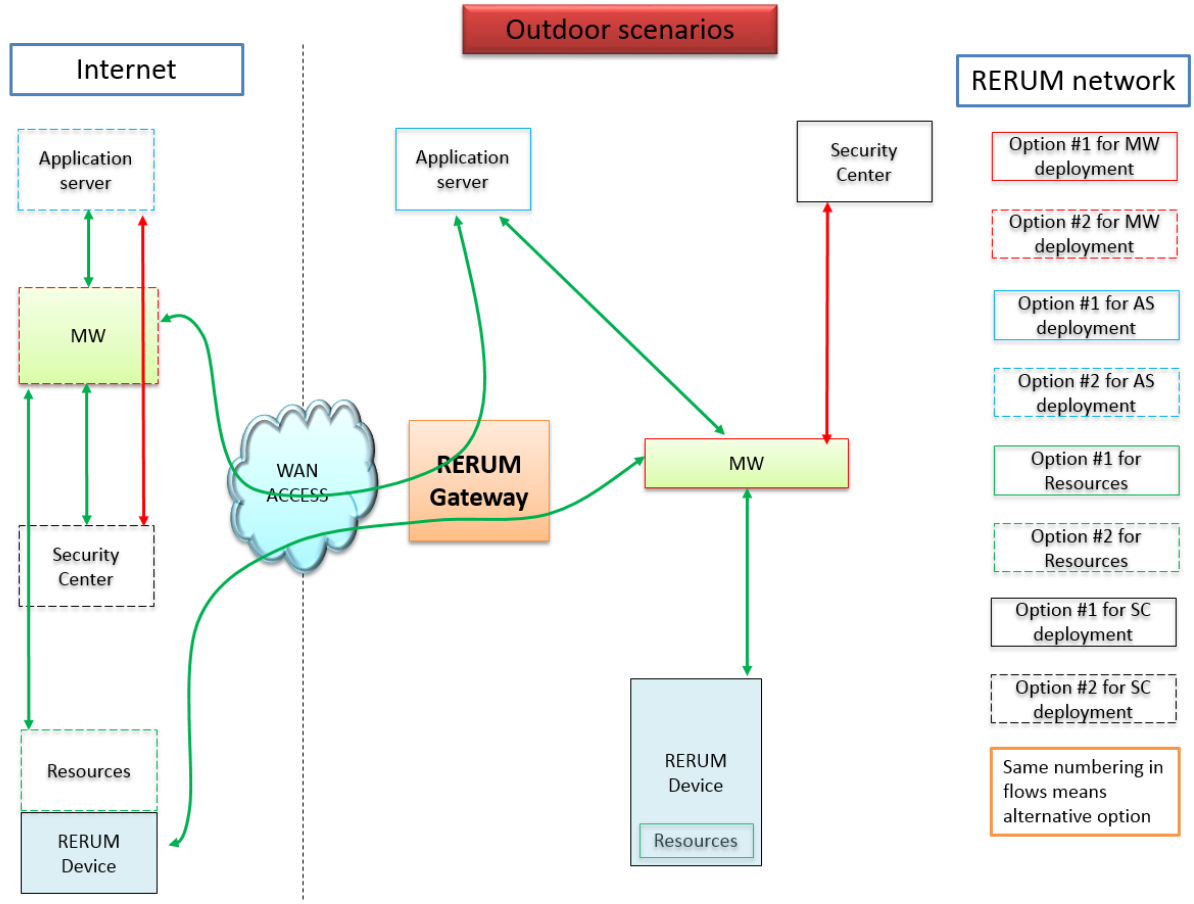


**Figure 98 – Outdoor scenarios: Deployment options.**

We assume that we have two scenarios. In Scenario 1 in Figure 99, all RDs are connected to the MW via the GW. Similarly to the indoor scenarios, the RDs can be connected directly to the GW or through other RDs that play the role of aggregators. In Scenario 2 in Figure 100, we depict the connectivity of mobile RDs that should mostly be non-constrained devices (i.e. smart phones). These can either be connected through a GW (that will change over time as they are moving in a city area) or they can be directly connected to the MW via i.e. a 3G interface. The capability of the RDs to connect directly to the MW allows a more flexible deployment of the RERUM network, since there will be no requirement to install e.g. dozens or hundreds of GWs around the city to connect mobile devices for the smart transportation use case.

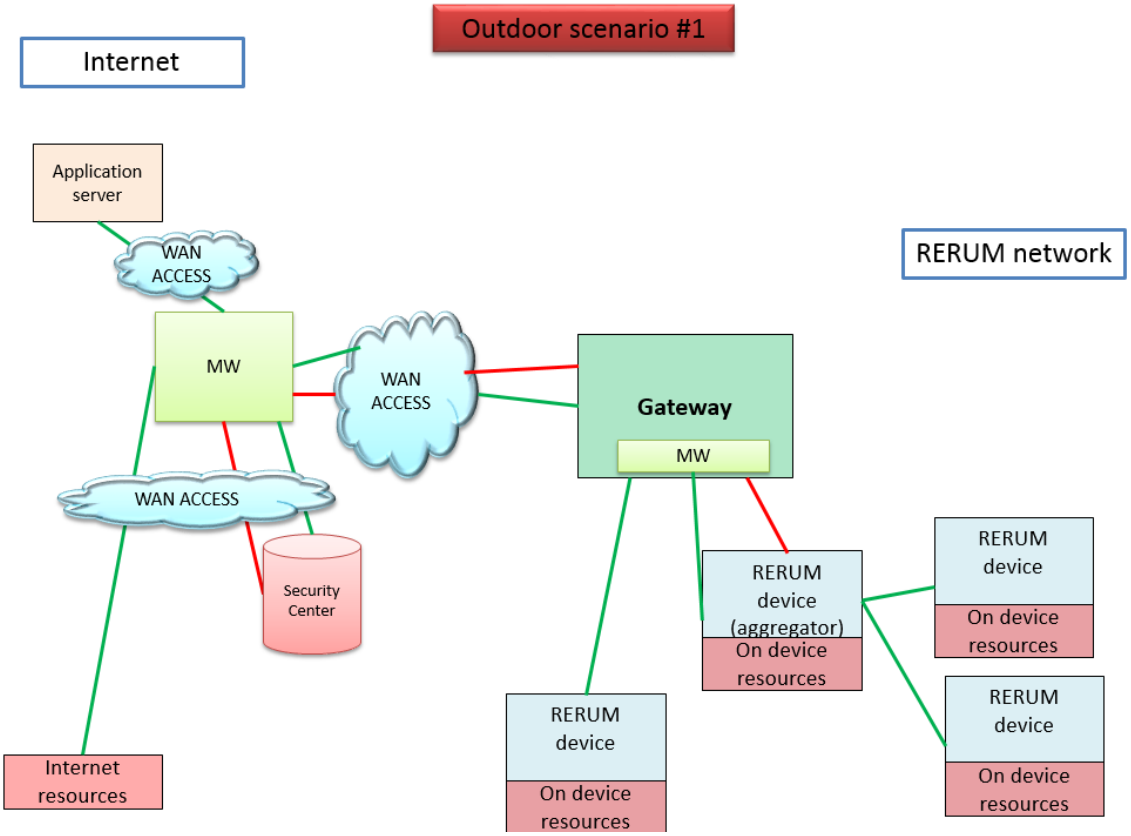


**Figure 99 – Scenario Outdoor 1 – All RDs connected via Gateway.**

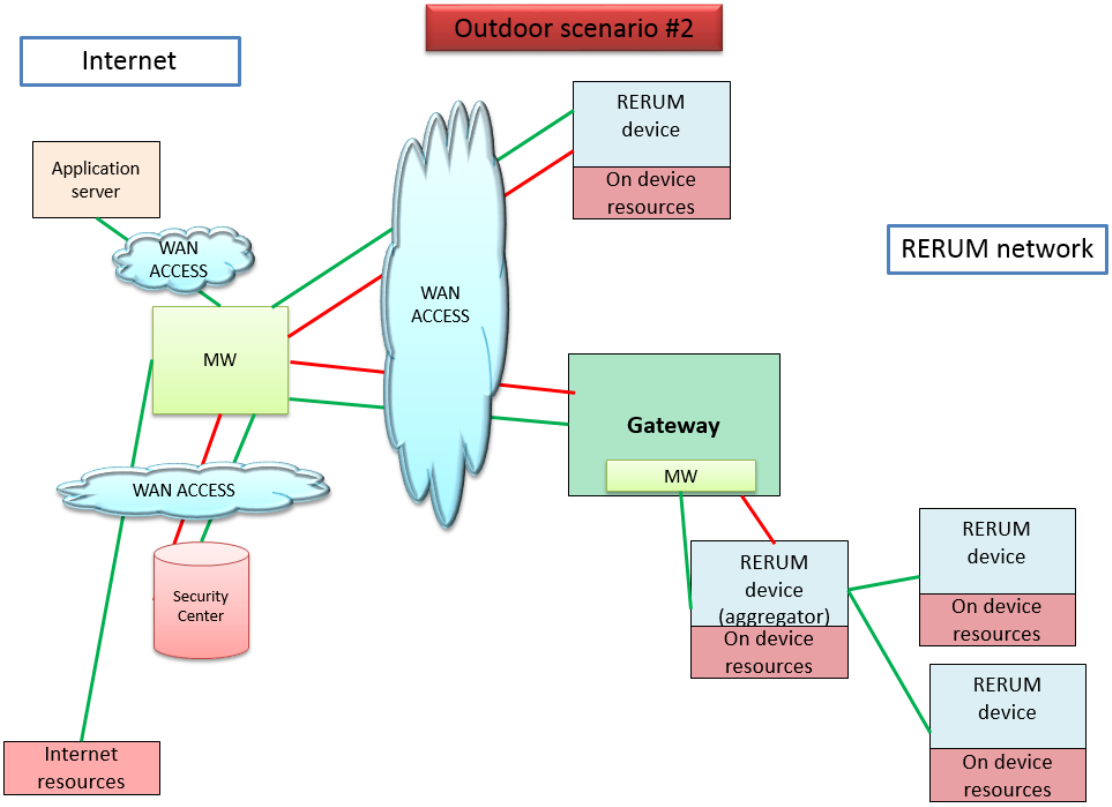


**Figure 100 – Scenario Outdoor 2 – RDs connected either via Gateway or directly to the MW.**

#### *4.1.3.3 Hybrid/hierarchical deployments*

RERUM acknowledges the importance of building a system that should not be closed or restricted, but it will be able to coordinate with other IoT frameworks to provide truly IoT applications. The idea here is that there will be situations where an indoor deployment of the system (e.g. in a home apartment) may be needing to cooperate with other deployments either indoor (e.g. with the deployment of the neighbour or of the overall building) or outdoor (e.g. with the neighbourhood or the city deployment). In such a case, to ensure the privacy of the data of the indoor deployment, the home administrator will be capable of selecting and defining the services that will be provided to the external world. These services will be installed by the administrator according to his security/privacy preferences and he will give proper access and privacy rules and policies for the external users. These services may probably be federations/compositions of indoor RERUM Services and will probably be installed on the GW (or another indoor server). Now, in order to include the indoor deployment in a hierarchical outdoor deployment, the outdoor deployment doesn't have to (or better shouldn't) know specific information for the indoor deployment, e.g. the type of RDs that the indoor deployment has or the type of services they expose. The only thing that the outdoor deployment has to know is the external services that the indoor deployment provides and that are allowed to be accessible from the outside world. In this respect, the indoor deployment can be registered in the MW of the outdoor deployment as a single VRD that exposes some services and has some characteristics, i.e. location, resources, etc. That way, the externally available services of the indoor deployment can be discovered and accessed from the outdoor deployment the same way like the services of a normal RD.

In a similar way, we can imagine the cooperation of two indoor RERUM deployments. For example, if two neighbours (A and B) want to use applications that require information from the neighbours, then they can install external services for their deployments. That way, neighbour B will be able to access the services of neighbour A as a normal external service accessing **internet resources** as described in Section 4.1.3.1.

Figure 101 shows an overview of the hybrid/hierarchical deployments. RERUM Deployment #1 has some internal services Service#1.1-1.3. They are composed/federated into an external Service#1.4 that is public and allowed to be accessed by anyone. In the outdoor city deployment #m, the indoor Deployment #1 is seen as the VRD#m.1 that exposes the Service#m.1 that corresponds to the Service#1.4. The figure shows also the case in which the neighbour Deployment#2 needs to access a Service of Deployment#1 and since the administrator of the deployment allows that to happen, the Service#2.5 is created out of a composition of Service#2.1 and Service#1.3.

 

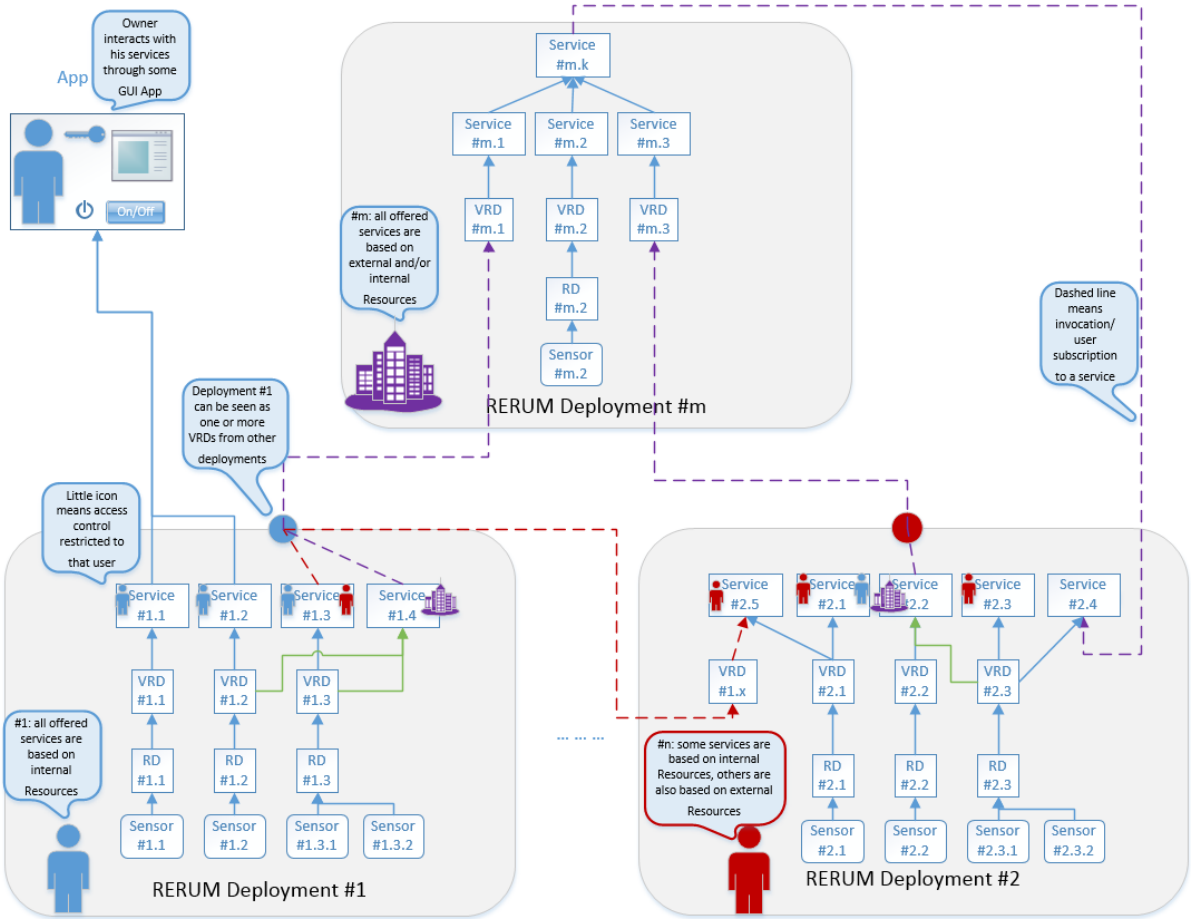


**Figure 101 – Hybrid/hierarchical deployment scenario**

## 4.1.4 Data flows examples

As described in D2.1, there are two basic type of data exchanged between components as described below, the Control Plane Data Exchanges and the Data Plane Data Exchanges, which are summarized here for the sake of the document's completeness.

### *4.1.4.1 Control Plane Data Exchanges*

These are data flows are used to monitor and control the use-case deployment's behaviour. Control plane data exchanges concern:

- network management messages,
- routing control messages,
- neighbour discovery messages,
- messages for link-layer associations,
- messages for TCP session establishment handshakes,
- network management requests and responses
- RD registration messages and responses
- Network monitoring messages

The data involved in these flows are the Command and Control Data (C&C-DATA) as described in D2.1, which are data and metadata used to control, monitor, and manage the system's overall state, in order to ensure correct operation. Based on the RERUM scope identified at the beginning of this threat analysis as well as the use-case definition, we identify the following C&C-DATA:

 

- EUI-64 identifiers for network interfaces (e.g. IEEE 802.15.4-compliant RF transceivers) on sensor nodes and the gateway.
- 6LoWPAN prefix information used for the communication within a 6LoWPAN mesh
- IPv6 routing tables
- Spectrum/channel usage information/history
- Neighbour Discovery caches
- Application configuration data
- Application version updates
- Registration requests/responses

When in transit, these data are part of control plane data exchanges.

#### *4.1.4.2 Data Plane Exchanges*

The data plane exchanges involve application layer network traffic, which is visible and of interest to end users, even in cases where they are not necessarily user-initiated. We use the terms "application traffic" or "user traffic" to refer to this class of traffic, with the two terms being equivalent. Furthermore, we make a distinction between *data in transit* and *data at rest*. The term "data in transit" is used to describe data being transmitted over the network, while the term "data at rest" describes data stored at a device or a network entity. Part of the data plane communication takes place over network infrastructures outside RERUM's influence, such as the Internet or an ISP's infrastructure, where the exact technologies used to deploy such infrastructures may be unknown.

These User Data (U-DATA) can be further broken down into the following categories:

- Sensed Data (S-DATA)
- Actuation Data (A-DATA)
- High-level Application Data (H-DATA), including service requests

All user data are part of data plane exchanges. The subcategories listed above include the transmission of actual data readings, as well as user queries that may have triggered them. For example, the transmission of an ambient temperature reading is considered U-DATA. The user request that triggered this reading's transmission is also considered U-DATA.

### 4.1.5 Message sequence charts

The following diagrams show some basic examples of message sequences in a basic deployment scenario. Note, that these examples are not UC-specific and could be applied with minor modifications to all RERUM UCs. The basic actions/operations described in the following figures are: (i) user logins to RERUM web server (or Application Server) to request an application, (ii) RD Discovery and RD registration, (iii) CoAP/http proxy, (iv) human user requests access to a Service, (v) user gets data from RD, and (vi) the RDs are federated in order to provide a Service.

#### *4.1.5.1 User login to RERUM web Server*

The process for the user login to a web server for accessing a RERUM application is depicted in Figure 102. The steps are as follows:

1. The user submits a HTTPS URL to the Local Browser.
2. The Browser gets the URL HTTPS address and hits the public IP of the Web Server
3. The Web Server responses with a HTTP "200 OK" message to the Browser for its availability to serve
4. The user sees the RERUM login page.

5. The user enters the sign-in credentials (username and password). The account has been created at the local database of the Web Server.
6. The Browser submits over HTTPS (POST) the credential data.
7. The Web Server responds with Set-Cookie
8. User sees the result.
9. The Browser submits a HTTPS GET new page to the Web Server.
10. The Credentials validity is checked with the Security Center/user authentication component based on Cookie.
11. The Security Center responds with the result (user found, or user not found).
12. If the user is found and password matches, then the RERUM main page is sent to the Browser.

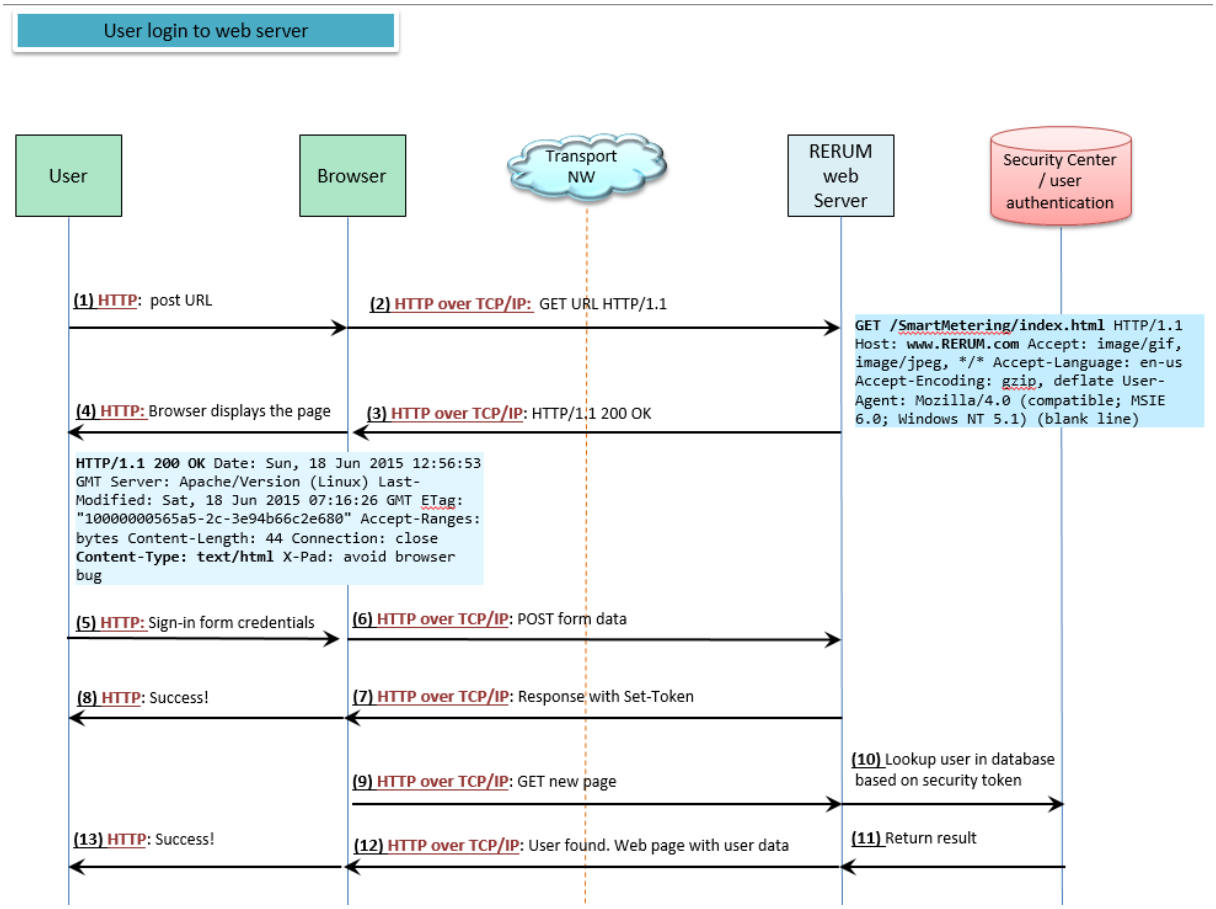


**Figure 102 – Login to RERUM web Server.**

### *4.1.5.2 User requests access to a RERUM Service*

The process for requesting a service by a HU is depicted in Figure 103. The steps are as follows:

1. The human user (HU) requests access to the Service that resides for example on an Application Server (AS). This may be done either through a WAN or not depending on the location of the AS.
2. The AS will check whether the HU is authenticated to use the requested service/application, through a request to the Security Center.

 

3. The Security Center requests from the AS the authentication credentials of the HU (e.g., encryption keys, certificates, etc.).
4. The AS will request the authentication credentials from the HU, unless there is a secure mechanism that ensures the secure storage of the HU credentials on the AS. In that case, the AS will send the credentials to the Security Center. Otherwise, the HU will send the credentials to the AS, which will forward them sequentially to the Security Center.
5. The Security Center will calculate according to a specific security mechanisms, whether the HU can be authenticated to use the requested service. In case the HU is authenticated, the authentication confirmation will be sent to the AS and then to the HU.

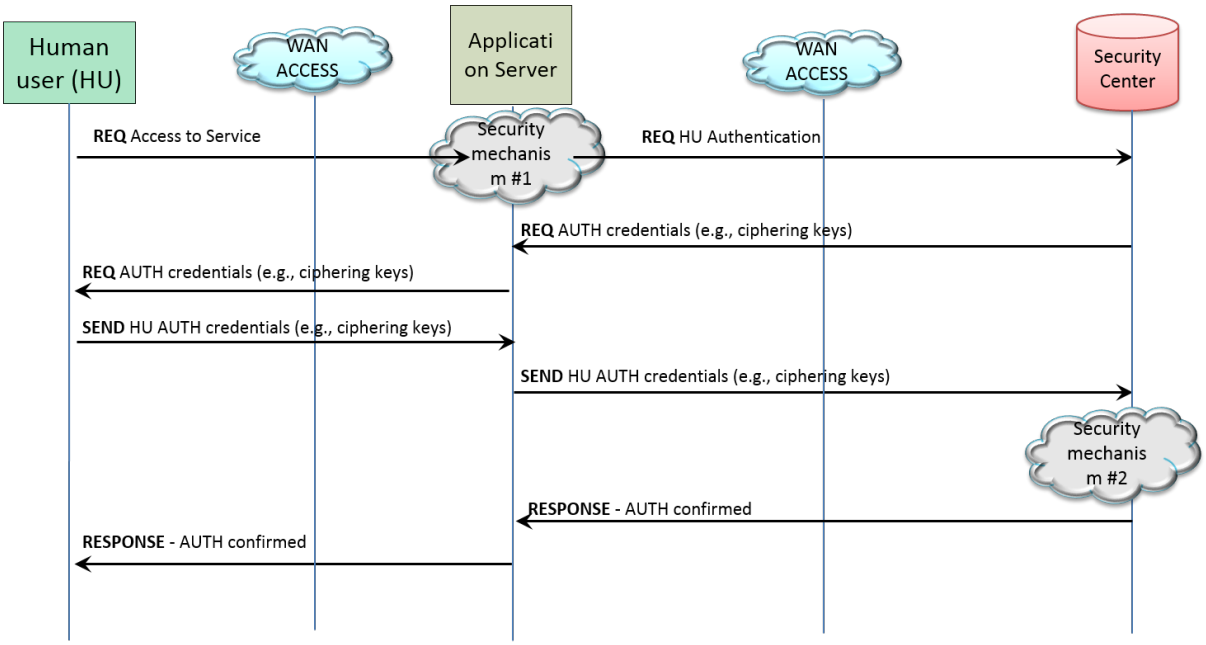


**Figure 103 – Human User requests access to a service.**

### *4.1.5.3 RD Discovery and RD registration*

The process for the RD Registration and Discovery is depicted in Figure 104:

1. The RERUM GW sends an HTTP message to the MW (GVO manager) for adding the GW to the GVO Registry.
2. The GVO Manager responds with a HTTP success message.
3. The RERUM GW sends a multicast message (**CoAP**: GET request for the resource /.wellknown/serverInfo) in order to get the information about the RDs that are attached to the GW.
4. The RDs respond with a CoAP message (JSON format of resources).
5. The RERUM GW sends a HTTP POST: addSensorToRegistry (sensorAdd) (doPost) to the GVO Manager

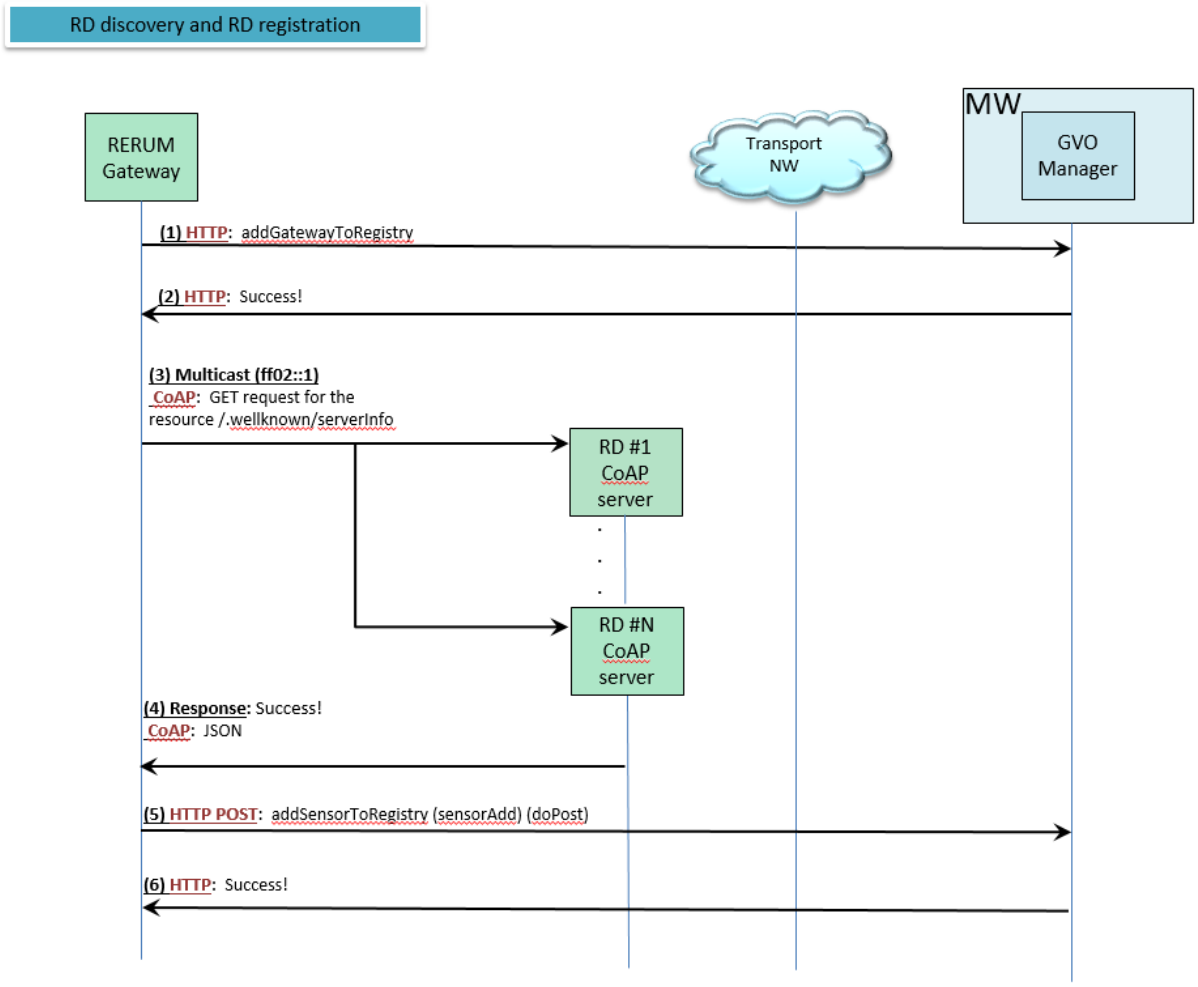


**Figure 104 – RD Discovery and RD registration.**

### *4.1.5.4 User gets data from an RD*

The process for receiving data from an RD is depicted in Figure 105. The steps are as follows:

1. The HU registers to the AS as described previously.
2. The RDs are registered to the corresponding GW as described previously.
3. The HU requests a specific service/application.
4. The AS resolves the GW that is involved in the requested service.
5. The AS forwards the HU service request to that GW.
6. The GW in turn resolves the RDs that are involved in the requested service.
7. The GW gets the accesses control information for the RDs through the Security Center.
8. The GW requests the data from the RDs.
9. The RDs response and send the requested data to the GW, which in turn sends them to the AS.
10. The AS sends the requested data to the HU.

 

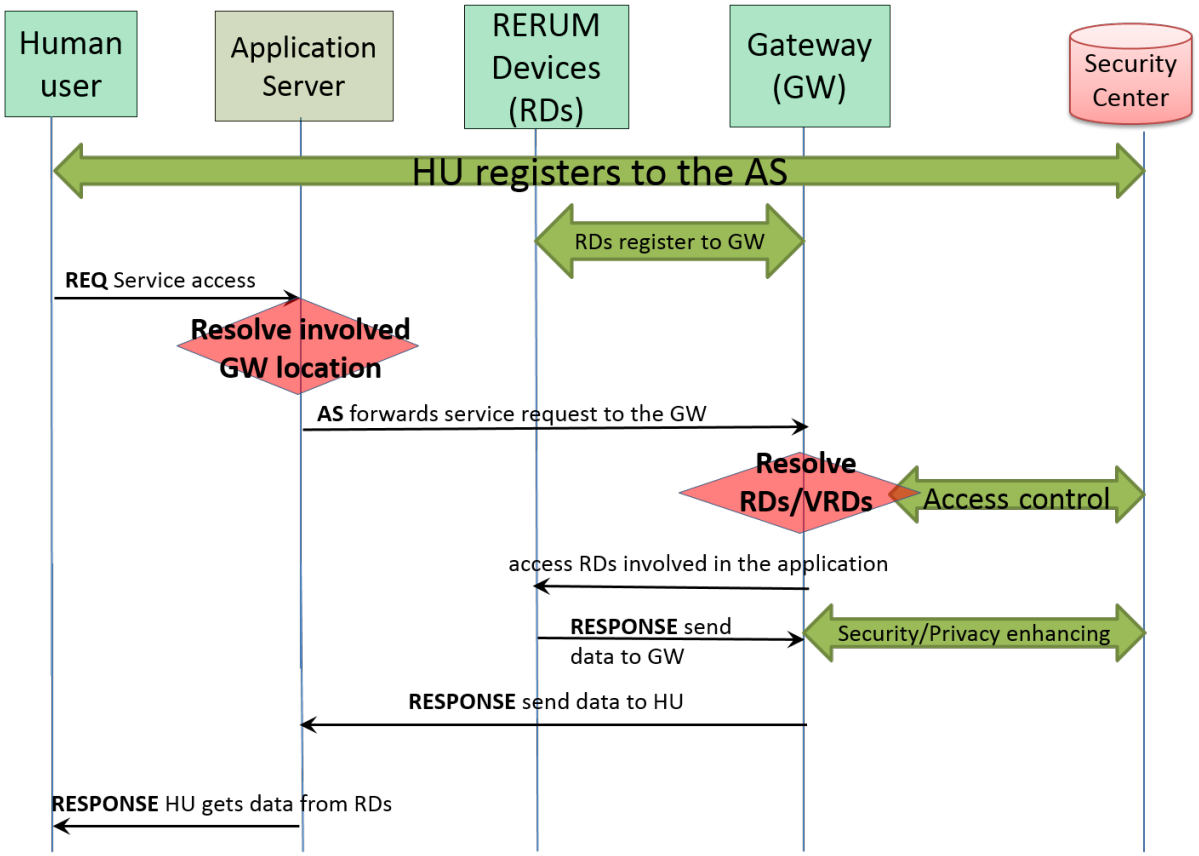


**Figure 105 – HU gets data from an RD.**

### *4.1.5.5 CoAP/HTPP proxy*

In most cases a translation between CoAP and HTTP protocol is needed, since the user client may not be CoAP-aware. This process is depicted in Figure 106 The steps are as follows:

1. The HTTP client sends a HTTP request to the RERUM GW/proxy.
2. The "Protocol Translator" component on the GW receives the HTTP request and translates it to CoAP.
3. The GW sends the CoAP request to the RD/CoAP server.
4. The CoAP server responses with a CoAP message (the resources are sent in JSON/xml, etc. format).
5. The GW receives the CoAP message and translates it back to HTTP
6. The GW sends the HTTP response to the HTTP client.

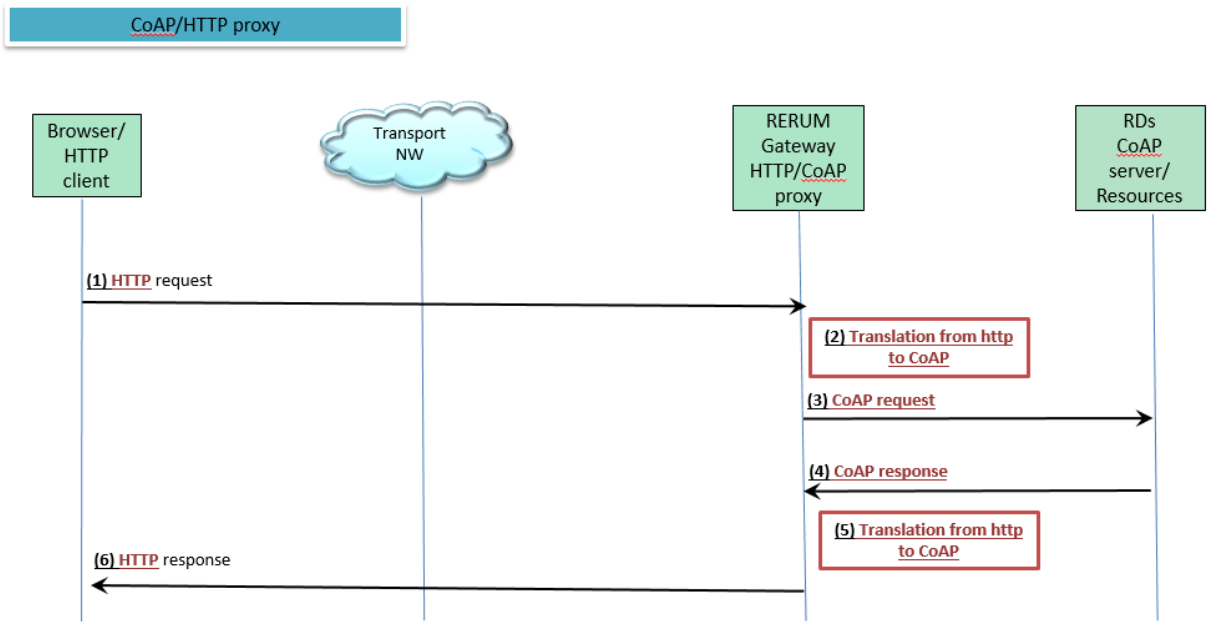


**Figure 106 – CoAP/HTPP proxy.**

### *4.1.5.6 RDs federation*

The process for RDs federation is depicted in Figure 107. The steps are as follows:

1. The RD#1 registers to the GW as described previously.
2. The RD#2 registers to the GW as described previously.
3. The HU registers to the AS as described previously.
4. The HU requests access to a specific service.
5. The AS resolves the GW that is involved in the requested service.
6. The GW resolves the RDs that are involved in the requested service (i.e., RD#1 and RD#2).
7. The GW creates the VRD federation. The Security Center is involved in this process to ensure the secure creation of the federation between trusted and authorised RDs.
8. The data are sent from the RDs to the GW
9. The GW sends the aggregated/filtered data to the AS to be forwarded to the HU.

 

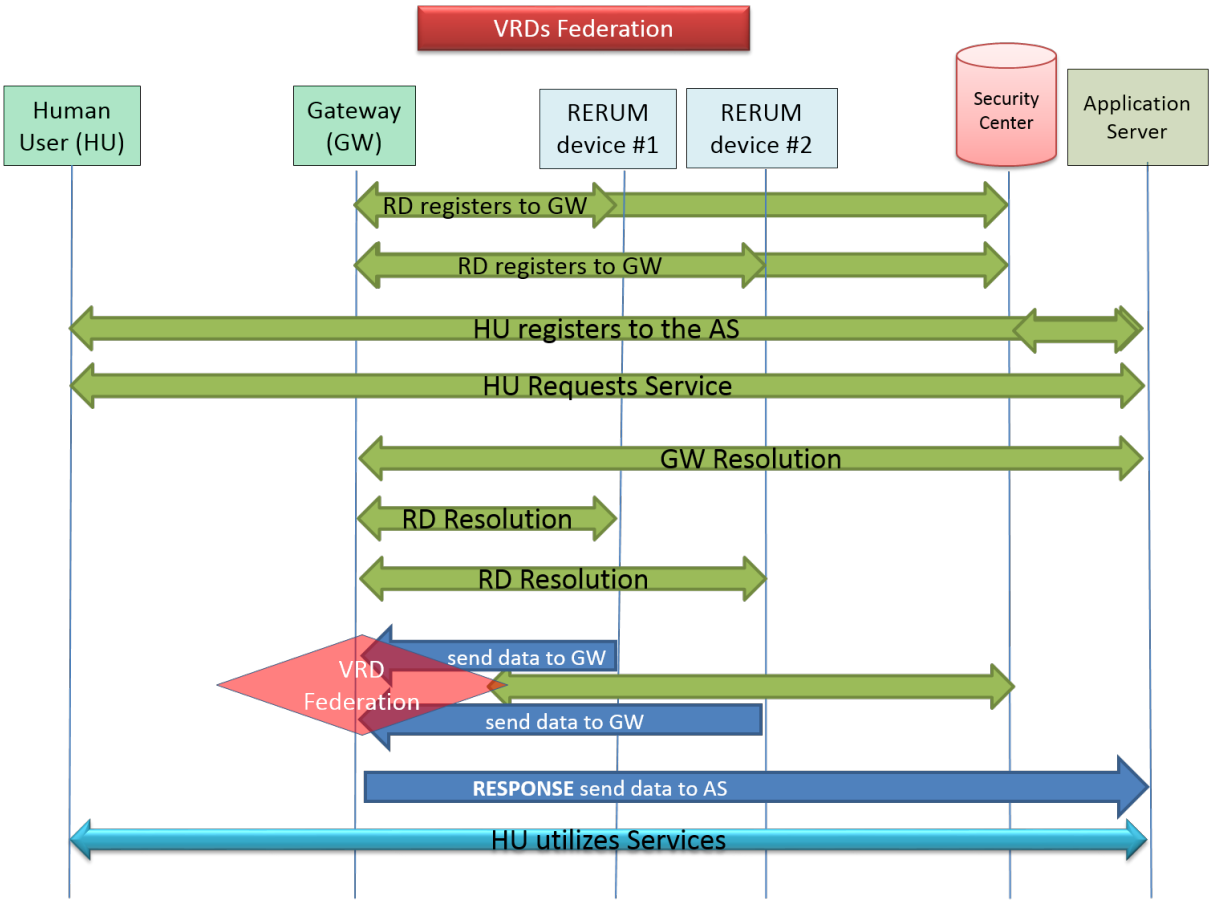


**Figure 107 – RDs federation.**

## 4.2 RERUM physical components

This section presents the basic physical components of RERUM and gives a first attempt for a detailed description of the functionalities that may exist on these components. As it can be seen in the previous figures, there are three basic physical components of RERUM: (i) the RERUM Gateway, (ii) the RERUM Device and (iii) the Security Server (or Security Center). As it has been mentioned throughout this document, the Application Server is conceptually out of the scope of RERUM and should only include communication security functionalities, i.e. encryption. decryption, etc. with some specific keys that should be decided beforehand or at the time of the service request. For this reason the Application Server is not described in this section. Furthermore, since the Security Server can be implemented in any powerful physical device and basically it includes the security, privacy and trust functionalities, it is also not described in detail here.

### 4.2.1 RERUM Gateway

The RERUM Gateway was described as a component of the use cases in D2.1 and its respective requirements were identified in D2.2. The initial description of the functionalities on the gateway was detailed in D2.3 and here we will do an update.

#### *4.2.1.1 General requirements*

In the following, based on the use case description and the requirements analysis in D2.2, the requirements of the RERUM GW are extracted in terms of installation, network access and communication, management and security. The scope of this analysis is to define the GW in terms of functionalities and architectural design.

| Requirement | Description |
|---|---|
| Installation | UC-O1 & UC-O2: Installation by experts, since special needs are required for physical security and protection. |
| | UC-I1 & UC-I2: Installation by end-users in a plug-n-play fashion. |
| Management and performance monitoring | UC-O1 & UC-O2: Remote management and performance monitoring of the GWs should be possible by the authorized entities. |
| | UC-I1 & UC-I2: Remote management of the GWs and their performance monitoring should be possible by the respective authorized entities. Service/network providers should have access to management operations without being able to have access to user data and personal information. End-users should be able to access the management of the gateway both locally and remotely. |
| Remote access | All UCs: The human user (depending on the user role and rights) to remotely access the gateway either through the internet or the intranet. This is required for users that have the role of an Administrator, e.g. for configuration and monitoring purposes. |
| Security | All UCs: Depending on the use case and its requirements (e.g., energy consumption, communication requirements, etc.) different security mechanisms may be applied for securing both the communication with the devices within the RERUM intranet or the devices/servers/cloud through the internet. |
| Connectivity | All UCs: The connectivity with the devices within the RERUM intranet or the devices/servers/cloud through the internet is possible via a set of communication protocols. |

#### *4.2.1.2 High level architecture*

The GW plays a central role in the RERUM system whenever it is included in the deployment. The following Figure 108 depicts in general the internal functionalities of a GW. As it is depicted in Figure 108 the RERUM Gateway includes standard networking functionalities for protocol translation, routing between interfaces, etc. Furthermore, it includes various security, privacy and trust mechanisms that are needed for secure communications and policy enforcement, as well as for trust management. Moreover, it includes all the Configuration and Monitoring components with regards to the configuration of the devices and the monitoring of their performance, raising alerts when any events are detected. The GW can include also MW functionalities for registering the RDs and discovering them, as well as for executing federations, managing data and context and ensuring the QoS of the services. As mentioned in the hierarchical/hybrid scenario, the GW can also play the role of a VRD exposing services of indoor deployments, thus it includes parts of the RD adaptor.

 

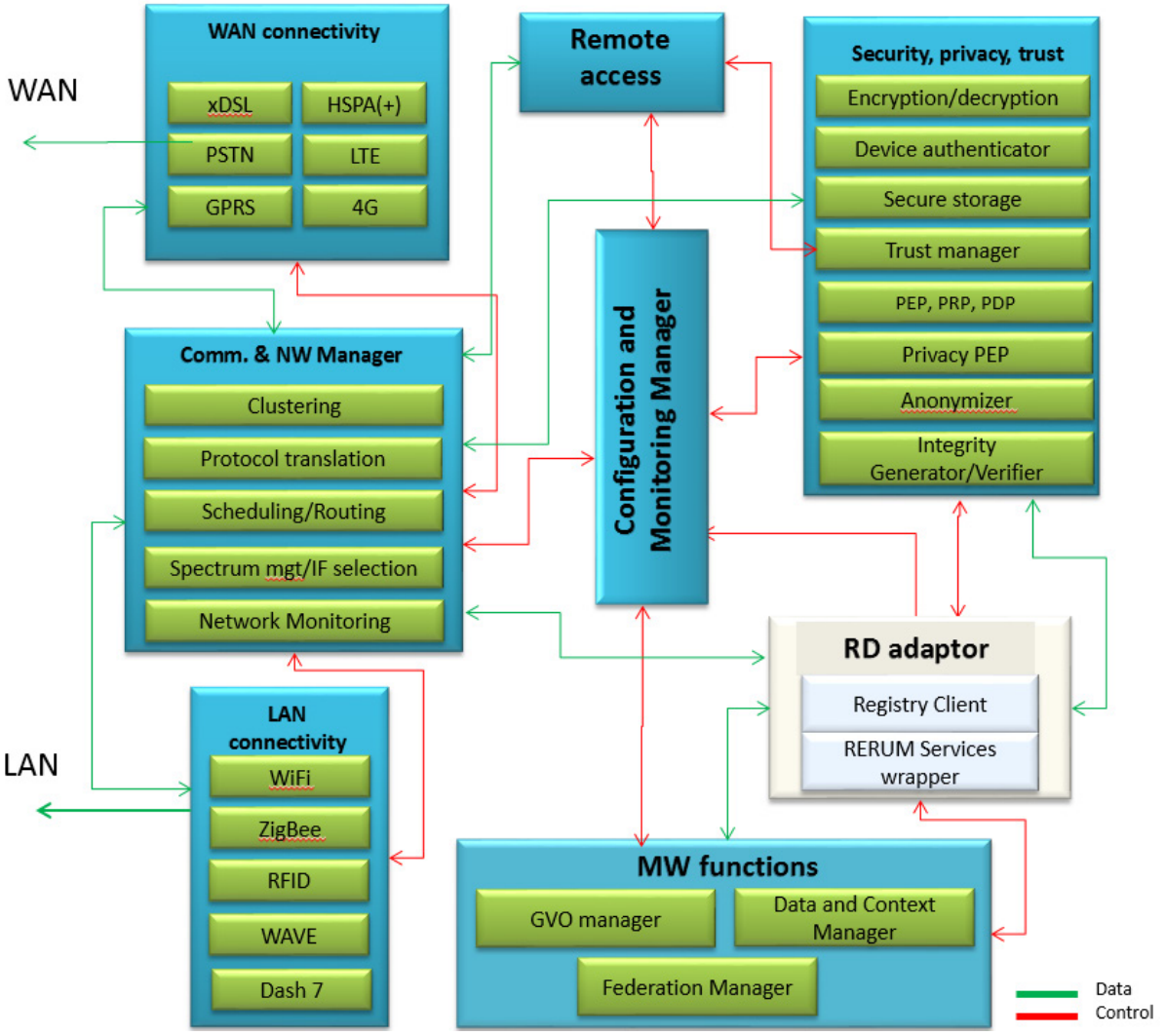


**Figure 108 – The RERUM gateway. High level overview of supported functionalities.**

### 4.2.2 RERUM Device

As mentioned in D2.2 and in Section 3.3, a RERUM Device may include the following hardware components, but not necessarily all of them (For example, a simple RD might have acting elements, a yet even more simple, might not have sensing elements either and it would just be there acting as a router in a mesh deployment):

- Power Supply Unit
- Processing unit
- Communication interfaces
- Sensing elements
- Memory/storage unit
- Acting elements

The RD performs specific operations hosting Resources. Depending on the scenario, a RD may access the internet through a GW (connecting through the LAN connectivity interfaces) or directly through the WAN connectivity interfaces. Furthermore, the RD may employ sensors or/and actuators, or may be connected to them wirelessly or via wires.

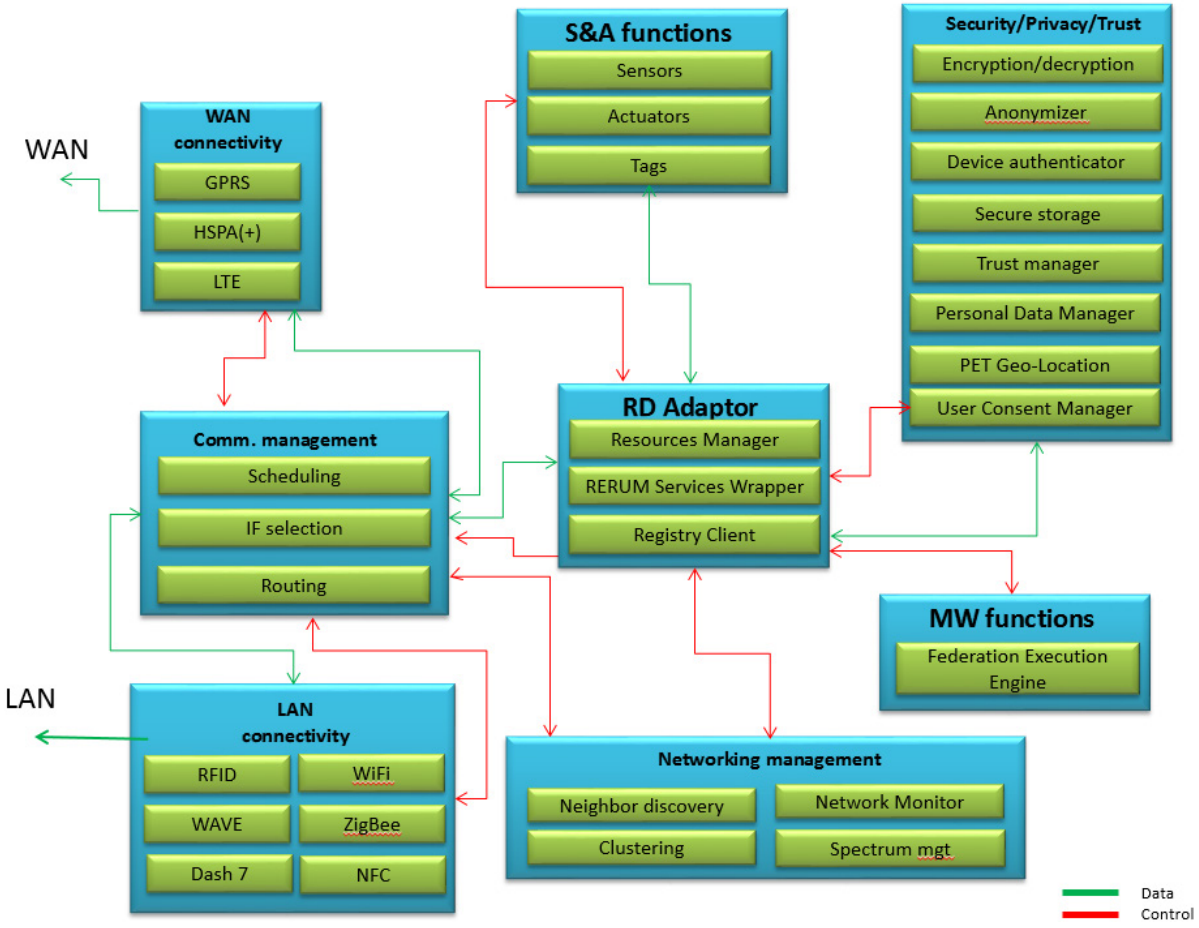


**Figure 109 – The RERUM Device. General high level overview of supported functionalities.**

As it can be seen in Figure 109, the RD includes most communication and networking functionalities, some SPT mechanisms for communicating securely, as well as the RD adaptor so that it can expose services and be registered in the MW. Finally, it can include the Federation Execution Engine, when the RD plays the role of a Federation Head.

In Figure 110 and Figure 111 the two versions of the RERUM Device used within the RERUM project are depicted. Figure 110 shows the unconstrained RD, namely a mobile phone and the functionalities it should support. The key points here are on the Security/Privacy part, where there are functionalities for Geo-location privacy, anonymization, personal data management and consent management. Furthermore, in Network Management, only the Network Monitor is required, since no clustering or centralized decisions are related with mobile phones (in a general case). Figure 111 shows the constrained RD, namely a sensor platform like Zolertia Z1 or the Re-Mote. Here, the privacy mechanisms are less than before and there is no WAN interface (in general).

 

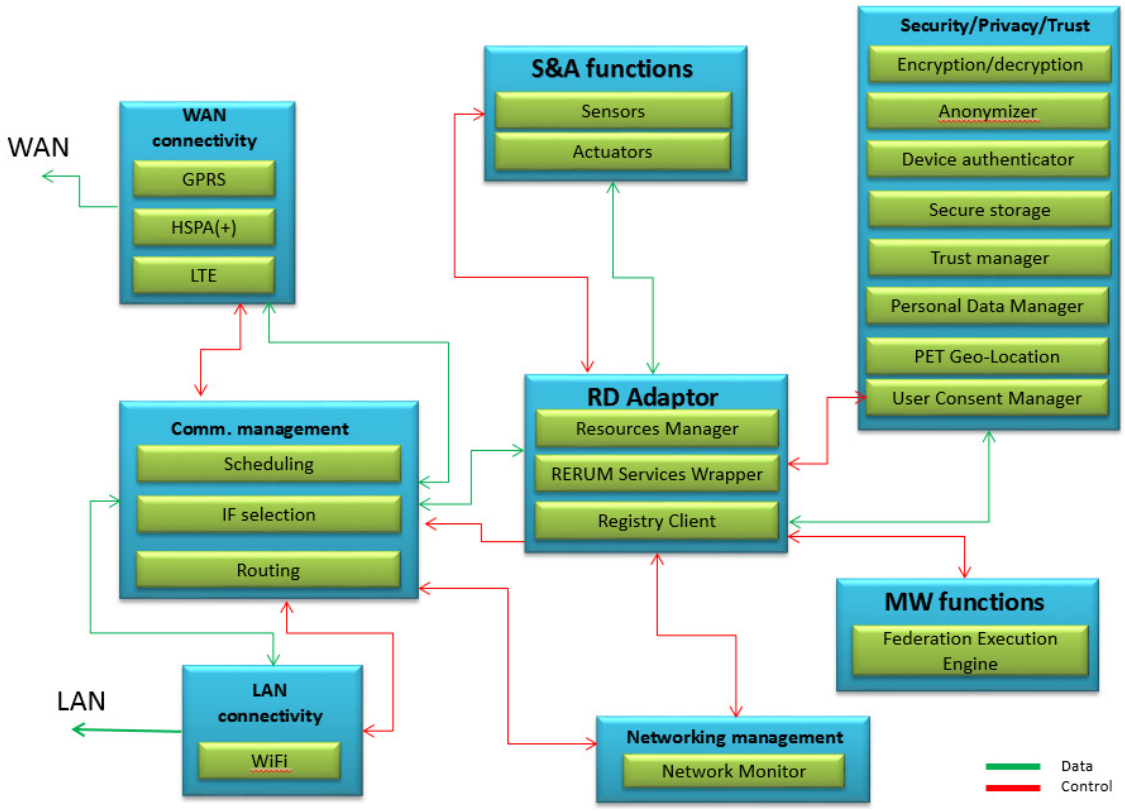


**Figure 110 – The Unconstrained RERUM Device (Mobile Phone) - supported functionalities.**

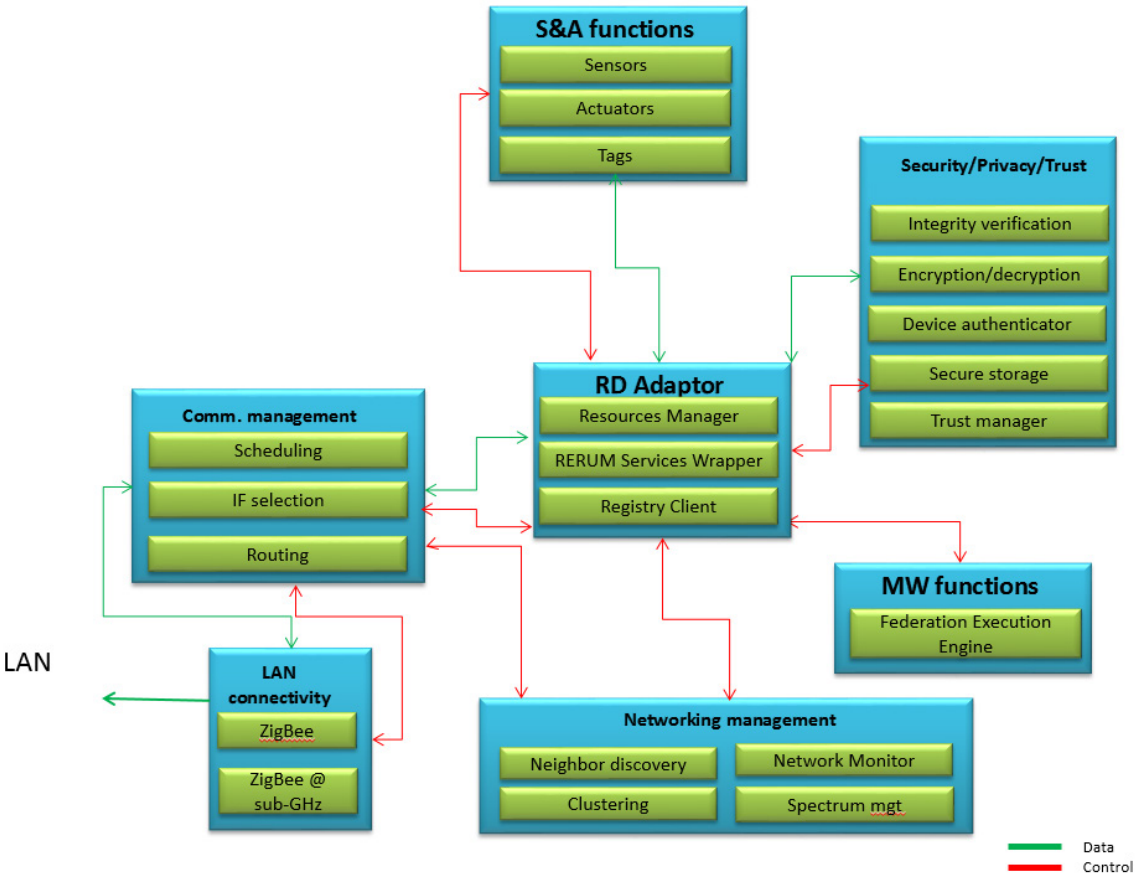


**Figure 111 – The Constrained RERUM Device (sensor platform) - supported functionalities.**

# 5 Conclusions

This deliverable presented the final draft of the overall RERUM architecture, as it was defined based on (i) the initial draft that was presented in D2.3 and (ii) the feedback from the technical workpackages during the activities of the second year of the project. In its final form, the RERUM architecture presents a complete description of (i) RERUM domain model, (ii) the architectural layers, (iii) the functional entities, (iv) their interactions and interfaces, (v) their internal functional components and their interconnectivities, (vi) the deployment model for a RERUM infrastructure and (vii) examples of models for the RERUM Devices and the RERUM Gateways.

The RERUM architecture is built in a way to reflect the basic design principles of the system, namely the concepts of security, privacy and reliability by design tailored to the requirements of the Internet of Things. In this respect, the initial version of the architecture was built after a detailed analysis of the functional and the non-functional requirements that were defined in deliverable D2.2. The very long list of security, privacy and trust-related requirements in that deliverable show exactly that RERUM tried to cover a wide range of solutions that can enhance the security of a generic IoT system, not focusing only on the RERUM applications.

The architecture is based on the foundations of IoT-A ARM, extending it with additional required functionalities with a specific focus on Security, Privacy, Trust and Network Reliability. The RERUM domain model was based on the IoT-A domain model, with the addition of the Context that was borrowed from BUTLER. RERUM also put its own touches to the IoT domain model, extending it with elements for security, privacy and trust, e.g. the respective policies that are attached either to the VEs or to the VRDs, the Reputation, the Data Subject and the Consent.

The RERUM architecture shows also the major focus of the project on the devices, acknowledging the fact that there is no point of securing a system if it utilizes unsecure devices. In this respect, the secure configuration of the devices has played an important role in the design of the RERUM system. With the design of a secure bootstrapping process and a framework for the reconfiguration of the firmware of the devices, RERUM ensures that the devices will have secure credentials and will be communicating with each other in a secure way. The RERUM architecture uses the RD Adaptor as the intermediate component that transforms a device into a RERUM Device capable of communicating with the rest of the RERUM components. It is well-known though that most IoT devices are constrained devices that are not able to run complex software solutions. To address this issue, RERUM targeted to design and develop techniques that are lightweight and can be installed and run even in constrained devices. As presented in in Section 4.2.2, several functionalities for security, privacy, trust and network reliability are envisaged to run on the RDs, improving significantly the security and the reliability of the overall system. Most of these functionalities are developed specifically for the RERUM project and can be found in the respective deliverables of the technical workpackages.

A key target of the project is ensuring people's privacy and protecting their data. This is obvious in the architecture presented in this deliverable by the powerful Privacy framework that is presented in Section 3.10.2. Privacy Enhancing Techniques are applied on a cross-layer manner in the RERUM system, in order to be able to provide multiple layers of protection to the user data for mitigating the threats that were presented in D2.1. PETs can be applied on the RDs, the RERUM Gateway or the Security Center that is tightly coupled with the RERUM MW functionalities. This design was made for ensuring that the data will be protected across their entire path starting from the RDs and going all the way up to the end user of an application.

Another key point of the RERUM architecture is the integration of an advanced trust and reputation framework, which has a threefold target for evaluating the reputation/trustworthiness of (i) the RDs, (ii) the Services and (iii) the users. Trust is very important for IoT systems that consist of a plethora of devices that are monitoring the physical world. However, it becomes even more important in cases of devices that act upon the physical world (actuators), which can potentially harm humans when they

 

are malfunctioning or hacked. RERUM evaluates the reputation of not only the users of the services (as normally in the Internet world) but also of the devices and the Services that are provided by the system to ensure that when there are sensitive data or critical situations, only trusted devices/services will participate in the system activities.

Furthermore, another important part of the architecture is related with ensuring the reliable interconnectivity of the RDs through advanced networking and communication management mechanisms and the introduction of the Cognitive Radio (CR) agent (see also deliverable D4.1) that is able to manage efficiently the access to the wireless spectrum on these devices that have the technical capabilities to do it. This is a key innovation of the project, since there were no previous attempts in past IoT projects to apply CR techniques on IoT devices. As analysed in the IERC Strategic Research Roadmap [72], *"New communications paradigms that use opportunistically the communication resources available at any given time will have to be adopted to provide seamless connectivity. Approaches based on the use of multiple radio bearers or* ***inspired by cognitive radio technologies*** *will have to be pursued to provide dependability, especially in harsh environments"*. RERUM acknowledges the importance of providing seamless connectivity to devices and the advantages of Cognitive Radio technology, and aims to become a pioneer into developing lightweight spectrum management mechanisms and low power CR-inspired RDs.

Finally, the deliverable also includes the final description of the RERUM deployment scenarios, taking into account the initial description that was described in D2.3 and the deployment scenarios of the trials as described in D5.1. The deployment scenarios cover not only the two different groups of use cases that have been identified in the project, namely indoor and outdoor, but also a hybrid scenario that combines both. The hybrid scenario came as a necessity after acknowledging the importance of keeping locally the information and the management of the indoor deployments. Thus, RERUM assumes currently that in hybrid smart city installations, an outdoor installation will not have to know the specific internal RDs and services of the indoor deployment, but the home owner (Administrator) will identify himself the exact services that the external world will be allowed to access. In a similar way we also describe the ability of the system to allow two neighbour indoor deployments to work jointly.

As a final remark, RERUM acknowledges that the Internet of Things is becoming an integral part of the everyday life of the citizens in smart cities. Smart city applications require data from sensors deployed everywhere in city environments to provide advanced services to the citizens and improve their quality of life. However, the personal information of the citizens is of outmost importance, and it should not be disclosed to unauthorized parties. The dangers of disclosure of personal information within smart city deployments are high, because until recently, the research interest for security and privacy in the IoT world was limited and existing deployments do not provide security or privacy enhancing mechanisms. RERUM identifies this issue as critical and focuses on enhancing existing IoT architectures with building blocks that will provide security and privacy by design, so that the information of the citizen will be transferred securely and will not be disclosed to third parties without the consent of the person of interest.

 

# References


[1] https://www.ict-rerum.eu/

[2] T. Mouroutis, et. al., RERUM Deliverable D2.1 - Use-cases definition and threat analysis, May 2014, https://bscw.ict-rerum.eu/pub/bscw.cgi/d14540/RERUM%20deliverable%20D2_1.pdf

[3] J. Cuellar et. al., RERUM Deliverable D2.2 - System requirements and smart objects model, May 2014,

https://bscw.ict-rerum.eu/pub/bscw.cgi/d14593/RERUM%20deliverable%20D2_2.pdf

[4] E. Tragos, et. al. RERUM Deliverable D2.3 – System Architecture, August 2014, https://bscw.ict-rerum.eu/pub/bscw.cgi/d18321/RERUM%20deliverable%20D2_3.pdf

[5] A. Bassi, et. al, Enabling Things to Talk, Springer, ISBN: 978-3-642-40402-3 (Print) 978-3-642-40403-0 (Online)

[6] uBiquitous, secUre inTernet-of-things with Location and contEx-awaReness, EU-FP& project BUTLER, www.iot-butler.eu

[7] Empowering the IoT with Cognitive Technologies, EU-FP7 project iCore, www.iot-icore.eu

[8] European Research Cluster on the Internet of Things, http://www.internet-of-things-research.eu

[9] Internet of Things - Architecture, IoT-A EU-FP7 project, www.iot-a.eu.

[10] D2.3 Architecture Reference Model, iCore Deliverable D2.3, June 2013

[11] D3.2 Integrated System Architecture and Initial Pervasive BUTLER proof of concept, BUTLER Deliverable D3.2, September 2013.

[12] Report on Reference Architecture For IoT Service Creation and Provision, IoT.est Deliverable D2.2, August 2012.

[13] IoT@Work D1.3 – Final framework architecture specification, H.–P. Huth et al, 04.07.2013, https://www.iot-at-work.eu/data/D1.3_IoT@Work_Architecture_final_v1.0-submitted.pdf

[14] D2.3 OpenIoT Detailed Architecture and Proof-of-Concept Specifications, OpenIoT Deliverable D2.3, March 2013.

[15] IoT-A project, D1.5 - Final Architectural Reference Model for the IoT, July, 2013.

[16] Abowd, Gregory D., et al. "Towards a better understanding of context and context-awareness." Handheld and ubiquitous computing. Springer Berlin Heidelberg, 1999.

[17] Abbasi, Ameer Ahmed, and Mohamed Younis. "A survey on clustering algorithms for wireless sensor networks." *Computer communications* 30.14 (2007): 2826-2841.

[18] Standard, Federal. "1037C." Department of Defence Dictionary of Military and Associated Terms in support of MIL-STD-188 (1996).

[19] Merriam-Webster Online: Dictionary and Thesaurus www.merriam-webster.com

[20] Ovidiu Vermesan and Peter Friess, 2013. The IERC Book. Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems, River Publishers Series In Communications, available at www.riverpublishers.com.

[21] G. Baldini, et. al., “IoT Governance, Privacy and Security Issues”, EUROPEAN RESEARCH CLUSTER ON THE INTERNET OF THINGS, IERC Activity Chain 05 Whitepaper, April 2014

[22] Generic Adaptive Middleware for Behaviour-driven Autonomous Services, EU-FP7 project GAMBAS, www.gambas-ict.eu


 

[23] Collaborative Open Market to Place Objects at your Service, EU_FP7 project COMPOSE, www.compose-project.eu

[24] ISO, "Open Systems Interconnection--Basic Reference Model", ISO7498, http://www.ecma-international.org/activities/Communications/TG11/s020269e.pdf.

[25] Vinton G. Cerf, Edward Gain, "The DoD Internet Architecture model", Elsevier Science Publishers B.V. (North Holland), 1983.

[26] E. Rescorla, N. Modadugu, "Datagram Transport Layer Security", April 2006, Network Working Group, RFC 4347, http://tools.ietf.org/html/rfc4347.

[27] T. Dierks, E. Rescola, "The Transport Layer Security (TLS) Protocol", August 2008, Network Working Group, RFC 5246, http://tools.ietf.org/html/rfc5246.

[28] G. Tsirtsis, P. Srisuresh, "Network Address Transaltion-Protocol Translation (NAT-PT)", Februrary 2000, Network Working Group, RFC-2766, http://tools.ietf.org/html/rfc2766.

[29] N. Kushalnagar, G. Montenegro, C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement and Goals", August 2007, RFC 4919, http://datatracker.ietf.org/doc/rfc4919.

[30] IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. (2012 revision).IEEE-SA. 5 April 2012. doi:10.1109/IEEESTD.2012.6178212.

[31] J. Mitola, "The software radio architecture," Communications Magazine, IEEE, May, pp. 26–38, 1995.

[32] J. Mitola III and G. Maguire Jr, "Cognitive radio: making software radios more personal," Personal Communications, IEEE, vol. 6, no. 4, pp. 13–18, 1999.

[33] J. I. Mitola, "Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio Dissertation," Dissertation Royal Institute of Technology Sweden, vol. 294, no. 3, pp. 66–73, 2000.

[34] 1. Haykin, S., Cognitive radio: Brain-empowered wireless communications. Selected Areas in Communications, IEEE Journal on, 2005. 23(2): p. 201-220.

[35] E. Tragos, S. Zeadally, A. Fragkiadakis, and V. Siris, "Spectrum assignment in cognitive radio networks: A comprehensive survey," IEEE Communications Surveys and Tutorials, Vol. 15, No. 3, 2013

[36] E. Tragos and V. Angelakis, "Cognitive Radio Inspired M2M Communications (Invited Paper)," in IEEE Global Wireless Summit 2013.

[37] George Stamatakis, Elias Tragos, Apostolos Traganitis, "Energy efficient collection of spectrum occupancy data in wireless cognitive sensor networks", Wireless Vitae 2014.

[38] HP Internet of Things Research Study, online at

http://fortifyprotect.com/HP_IoT_Research_Study.pdf

[39] Yucek, Tevfik, and Hüseyin Arslan. "A survey of spectrum sensing algorithms for cognitive radio applications." *Communications Surveys & Tutorials, IEEE* 11.1 (2009): 116-130.

[40] http://www.ieee802.org/15/pub/TG4.html

[41] http://www.3gpp.org/technologies/keywords-acronyms/98-lte

[42] http://www.etsi.org/index.php/technologies-clusters/technologies/mobile/gsm

[43] S. Fowler, et al. "Evaluation and Prospects from a Measurement Campaign on Real Multimedia Traffic in LTE vs. UMTS", IEEE (GWS'14), 2014.

[44] M. Lundgren, et al. “Energy-Aware Rate Selection in Cognitive Radio Inspired Wireless Smart Objects,” in proc. IEEE ISCC 2014.

[45] ISO/IEC 7498-4: Information processing systems -- Open Systems Interconnection -- Basic Reference Model -- Part 4: Management framework

[46] Hsin, Chihfan, and Mingyan Liu. "Self-monitoring of wireless sensor networks." *computer communications* 29.4 (2006): 462-476.

[47] Kohvakka, Mikko, et al. "Energy-efficient neighbor discovery protocol for mobile wireless sensor networks." *Ad Hoc Networks* 7.1 (2009): 24-41.

[48] Tim Winter, Pascal Thubert, Anders Brandt, Jonathan Hui, Richard Kelsey, Philip Levis, Kris Pister, Rene Struik, JP Vasseur, and Roger Alexander, “RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks”, RFC 6550.

[49] T. Clausen, P. Jacquet, “Optimized Link State Routing Protocol”, October 2003, RFC 3626.

[50] Tragos, Elias Z., et al. "Admission control for QoS support in heterogeneous 4G wireless networks." *Network, IEEE* 22.3 (2008): 30-37.

[51] Angelakis, Vangelis, et al. "Probabilistic Routing Schemes for Ad Hoc Opportunistic Networks." *Routing in Opportunistic Networks*. Springer New York, 2013. 209-222.

[52] Pelusi, Luciana, Andrea Passarella, and Marco Conti. "Opportunistic networking: data forwarding in disconnected mobile ad hoc networks." *Communications Magazine, IEEE* 44.11 (2006): 134-141.

[53] Tragos, Elias Z., et al. "Access selection and mobility management in a beyond 3G RAN: The WINNER approach." *Telecommunication Systems* 42.3-4 (2009): 165-177.

[54] Mihovska, A., et al. "Policy-based mobility management for heterogeneous networks." *Mobile and Wireless Communications Summit, 2007. 16th IST*. IEEE, 2007.

[55] D.8.1.2: FI-WARE GE Open Specification – Security, FI-WARE Deliverable D.8.1.2, April 2013.

[56] W3C Incubator Group Report 27, Unified Service Description Language XG Final Report, October 2011, http://www.w3.org/2005/Incubator/usdl/XGR-usdl-20111027.

[57] Contiki: The Open Source OS for the Internet of Things, http://www.contiki-os.org/.

[58] Ehmke, Edward L., and Mark T. Stair. "Intelligent over-the-air programming." U.S. Patent No. 5,381,138. 10 Jan. 1995.

[59] Neisse, R.; Steri, G.; Baldini, G.; Tragos, E.; Nai Fovino, I.; Botterman, M. Dynamic Context-Aware Scalable and Trust-based IoT Security, Privacy Framework. Chapter in Internet of Things Applications - From Research and Innovation to Market Deployment, IERC Cluster Book, 2014;

[60] U. Herberg, T. Clausen, “Integrity Check Value and Timestamp TLV Definitions for Mobile Ad Hoc Networks (MANETs)”, RFC 6622, ISSN: 2070-1721.

[61] Information technology, Open Systems Interconnection, Security frameworks for open systems: Integrity framework, ISO/IEC 10181-6:1996.

[62] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. Handbook of applied cryptography, CRC press, 2010.

[63] RFC4949, August. "Internet Security Glossary." (2007).

[64] OASIS standard, eXtensible Access Control Markup Language (XACML) Version 3.0, available at http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf

 

[65] SNEKKENES, Einar: Concepts for Personal Location Privacy Policies. In: Proceedings of the 3rd ACM Conference on Electronic Commerce. New York, NY, USA : ACM, 2001 (EC '01). ISBN 1–58113–387–1, pages 48–57.

[66] BUSCH, Marianne; KOCH, Nora ; SUPPAN, Santiago: Modeling Security Features of Web Applications. Version: 2014. http://dx.doi.org/10.1007/978-3-319-07452-8_5. In: HEISEL, Maritta (Hrsg.); JOOSEN,Wouter (Hrsg.) ; LOPEZ, Javier (Hrsg.) ; MARTINELLI, Fabio (Hrsg.): Engineering Secure Future Internet Services and Systems Bd. 8431. Springer International Publishing, 2014. – ISBN 978–3–319–07451–1, pages 119-139.

[67] EBINGER, Peter; HERNÁNDEZ RAMOS, JoséLuis ; KIKIRAS, Panayotis ; LISCHKA, Mario ; WIESMAIER, Alexander: Privacy in Smart Metering Ecosystems. Version: 2013. http://dx.doi.org/10.1007/978-3-642-38030-3_9. In: CUELLAR, Jorge (Hrsg.): Smart Grid Security Bd. 7823. Springer Berlin Heidelberg, 2013. – ISBN 978–3–642–38029–7, pages 120-131.

[68] MANULIS, Mark ; FLEISCHHACKER, Nils ; GUNTHER, Felix; KIEFER, Franziskus ; POETTERING, Bertram: Group Signatures: Authentication with Privacy. Cryptographic Protocols Group Department of Computer Science, Technische Universitaet Darmstadt. Version: 2012.

[69] XML Process Definition Language, www.xpdl.org

[70] Object Management Group Business Process Model and Notation, www.bpmn.org

[71] Paschke, Adrian, and Harold Boley. "Rule markup languages and semantic web rule languages." (2010): 1-24.

[72] Vermesan, Ovidiu, et al. "Internet of things strategic research roadmap." *Internet of Things-Global Technological and Societal Trends* (2011): 9-52.

[73] Ma, Huisheng, et al. "Spectrum aware routing for multi-hop cognitive radio networks with a single transceiver." *Cognitive Radio Oriented Wireless Networks and Communications, 2008. CrownCom 2008. 3rd International Conference on*. IEEE, 2008.

# Annex 1 - Terminology

| Term | Definition | Source |
|---|---|---|
| Acting element | An (embedded) device that has the capability to affect the condition of a Physical Entity, (like changing its state or moving it) by acting upon an electrical signal | RERUM/ IOT-A part of actuator [15] |
| Actuator | A smart device that includes one or several acting elements and receives (IT-based) commands translating them to electrical signals for the acting elements. An actuator can also include a sensor so that there is knowledge on the Physical Entity it acts upon, in order to translate correctly the command into the electrical signal. | RERUM/ IOT-A [15] |
| Application server | The point responsible for the end-user services (e.g., automation services, energy management, etc.). The Application server may reside either in the internet or in the RERUM domain and is responsible for accepting dynamic resource requests, executing the appropriate actions, and returning the results to the user. | RERUM/ IOT-A [15] |
| Context | Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. | [16] |
| Cluster | A group of wireless (mainly sensor) nodes that work together for a more efficient and scalable organisation and management of the network. | RERUM, based on [17] |
| Cluster Head (CH) | The RERUM Device that plays the role of the Head of a Cluster within the RERUM network. The CH is responsible for routing the data from the members of the cluster to the rest of the network, as well as to take centralized networking decisions. The CH is either pre-assigned or can be selected by the RERUM Devices. | RERUM, based on [17] |
| Clustering | The process of splitting the network in clusters and electing CHs. | RERUM, based on [17] |
| Consent | Within RERUM the user consent is used for privacy purposes, when the system will ask the user if he allows to send his data to an application that requests them. | RERUM |
| Device | It can be a single or a combination of the following elements:<br>• Sensors, which provide information about the Physical Entity | IoT-A [15] |

 

| | | |
|---|---|---|
| | • Tags, which are used to identify Physical Entities<br>• Actuators, which can modify the physical state of a Physical Entity | |
| Federation Head (FH) | A functional component that executes the process of the Federation of VRDs. It can be assigned to any powerful RD, the GW or a centralized server. | RERUM |
| Federation of Virtual RERUM Devices | Several Virtual RERUM Devices are forming a Federation if they cooperate to offer a joint service for a Virtual Entity (VE). The logic necessary to orchestrate the service is associated to the Virtual Entity that offers the service. | RERUM |
| Gateway (GW) | Network node equipped for interfacing with another network that uses different protocols. | Federal Standard 1037C [18] |
| Generic Virtual RERUM Object (GVO) | This is a software artefact that groups both virtualizations found in RERUM, namely the Virtual Entities and Virtual RERUM Devices that share properties like, that<br>• they allow to be discovered,<br>• they allow to be addressed, and they allow to be interacted with in a standardized manner. | RERUM |
| Internet Resources | These are sources of data/measurements that originate from outside of the RERUM domain and can be used as input for the applications. | RERUM |
| Middleware | Within RERUM, the Middleware is assumed to be a software layer or a group of functionalities that allows heterogeneous devices to be discovered, addressed and accessed by the applications in a seamless and unified way. The Middleware includes the virtualization of devices to hide their heterogeneity. | RERUM |
| Physical entity (PE) | A discrete, identifiable part of the physical environment which is of interest to the user for the completion of his goal.<br>Physical Entities can be almost any object or environment. | Merriam-Webster dictionary [19]/<br>IOT-A [15] |
| RERUM Aggregator | A RERUM Device can play the role of an Aggregator, when it collects, processes (aggregates, encrypts, filters, etc.) data/measurements from many other RERUM Devices and forwards them to the GW/Middleware/Application Server. A RERUM aggregator can be considered as an RD playing the role of a Federation Head and could be very helpful in terms of privacy, because this aggregation will avoid the leaking of personal information that may be contained in the data that are aggregated. | RERUM |

| | | |
|---|---|---|
| RERUM Device (RD) or RERUM Smart Object | A RERUM Device (RD) is a piece of hardware and software (incl. the Operating System) that is equipped with intelligence. It has one or more Resources that the RERUM Device is able to either fill with interpreted and pre-processed sensory data or able to read and interpret the commands that are given. The RERUM Device has some Sensing, Tag or Acting elements directly attached to it. | RERUM |
| RERUM Deployment | The specific topology of software components on the physical layer, as well as the physical connections between these components. | IoT-A [15]/ RERUM |
| RERUM Gateway | A RERUM Gateway is a physical device that plays the role of a network gateway interconnecting different RERUM networks. Furthermore, the RERUM Gateway is responsible for managing the RDs that are connected to it. In this respect it can also include various Middleware functionalities. | RERUM |
| Resources | Resources are software components that provide some functionality. When associated with a Physical Entity, they either provide some information about or allow changing some aspects in the digital or physical world pertaining to one or more Physical Entities. In general, they are typically sensor Resources that provide sensing data or actuator Resources, e.g. a machine controller that effects some actuation in the physical world. | IOT-A On-device Resources [15] |
| Sensing element | An (embedded) device that perceives certain characteristics of the real-world environment (Physical Entities), translating a change into an electrical signal. | RERUM |
| Sensor | A smart device that includes one or several sensing elements and is able to translate the electrical signal of the sensing elements to some type of information (digital representation) with specific value and semantic. | IoT-A [15] |
| (IoT/RERUM Service | Software component enabling interaction with resources through a well-defined interface, often via the Internet. | IoT-A [5], RERUM |
| Smart Object | See RERUM Device | RERUM |
| Virtual Entity (VE) | The digital synchronized representation of a Physical Entity. | IoT-A [15] |
| Virtual RERUM Device (VRD) | A Virtual RERUM Device (RD) is a digital representation of a RERUM Device. The same one physical RERUM Device at one time is represented by one Virtual RERUM Device. This is a software artefact, like a Virtual Entity (VE), but represents a RERUM Device (RD). | RERUM |

 

| | | |
|---|---|---|
| User | A Human or a software that interacts with a system for transferring information. | Based on IoT-A [15] |