ROBOTS IN ASSISTED
LIVING ENVIRONMENTS

UNOBTRUSIVE, EFFICIENT, RELIABLE AND MODULAR
SOLUTIONS FOR INDEPENDENT AGEING

Research Innovation Action

Project Number: 643892 Start Date of Project: 01/04/2015 Duration: 36 months

DELIVERABLE 5.1

Architecture of the RADIO ecosystem I

Dissemination Level	**Public**
Due Date of Deliverable	Project Month 15, June 2016
Actual Submission Date	1 April 2016
Work Package	WP5, Overall architecture of the RADIO ecosystem of services for the medical care institutions and informal care-givers
Task	T5.1, Designing the RADIO ecosystem interconnection and inter-facing
Lead Beneficiary	NCSR-D
Contributing Beneficiaries	TWG, S&C
Type	R
Status	Submitted
Version	Final

Project funded by the European Unions Horizon 2020 Research and Innovation Actions

Abstract

This report describes the design of the overall RADIO ecosystem of multiple RADIO Home deployments, applications for the informal care givers, and applications for the medical care institutions. It establishes the architecture and the interfaces of the various components that comprises the RADIO ecosystem.

History

Version	Date	Reason	Revised by
01	18 Jan 2016	First draft, establishing document structure and work allocation. Writing Introduction and outlining the content of the first chapter	NCSR-D
02	28 Feb 2016	Writing the Architecture section	NCSR-D
03	9 Mar 2016	Writing the Automation and Notifications sections	S&C
04	18 Mar 2016	Writing the Security section	TWG
04	28 Mar 2016	Pre final version	NCSR-D
05	31 Mar 2016	Internal review and corrections	RUB
Fin	1 Apr 2016	Final document preparation and submission	NCSR-D

Executive Symmary

RADIO Home deployments live and act inside the *RADIO ecosystem*. It is envisaged that RADIO Home deployments will seamlessly integrate in the RADIO ecosystem as nodes to an abstract network regardless of the different communication technologies and of the heterogeneous hardware and software components that comprise them.

RADIO ecocystem provides the necessary mix of components and interconnections in order to support useful operations such as clinical report inspection, privacy preserving data analysis, notifications and home automation. These operations must comply with requirements regarding the protection of the sensitive data produced in each RADIO Home.

This report establishes the architecture and interfaces for efficient and privacy-preserving data exchange between RADIO Home deployments, the formal and informal care-givers applications, and medical care institutions that operate across a wide area.

Abbreviations and Acronyms

REST API	Representational state transfer applications programming interface
IoT	Internet of Things
IPSec	Internet Protocol Security is a protocol suite for secure IP
Z-Wave	a wireless communications protocol for home automation
VPN	Virtual Private Network
RPC	Remote Procedure Call

CONTENTS

Contents v

List of Figures vi

1 Introduction 1
- 1.1 Purpose and Scope . . . 1
- 1.2 Approach . . . 1
- 1.3 Relation to other Work Packages and Deliverables . . . 1

2 Requirements 3
- 2.1 Connectivity Requirements . . . 3
- 2.2 Privacy Requirements . . . 3

3 Architecture 4
- 3.1 Entities and Roles . . . 4
- 3.2 Components . . . 4
  - 3.2.1 RADIO Home . . . 4
  - 3.2.2 IoT Platform . . . 5
  - 3.2.3 Care givers' application . . . 5
  - 3.2.4 Care institution's application . . . 5
  - 3.2.5 Clinical experimentation application . . . 6
- 3.3 Topology . . . 6
  - 3.3.1 Communication Channels . . . 7
  - 3.3.2 Discoverability . . . 7
  - 3.3.3 Changes in the topology . . . 7
- 3.4 Security . . . 8

4 Care institution application 10
- 4.1 Overview . . . 10
- 4.2 Use Cases . . . 10
- 4.3 Authentication and Authorization . . . 10
- 4.4 Interfaces . . . 10
  - 4.4.1 Formal Care Giver interface . . . 10

5 Privacy Preserving Data Mining 11
- 5.1 Overview . . . 11
- 5.2 Use Cases . . . 11
- 5.3 Privacy Protection . . . 11
- 5.4 Interfaces . . . 12
  - 5.4.1 Programming Language . . . 12

6 Automation and Notifications 13
- 6.1 Overview . . . 13
- 6.2 IoT Platform . . . 13
  - 6.2.1 Sensor Service . . . 13
  - 6.2.2 Event Service . . . 14
  - 6.2.3 Security . . . 15
- 6.3 Smart Home Controller . . . 15

6.3.1 24x7 Scheduler . . . 16
6.3.2 Security . . . 16
6.4 Notifications . . . 17
6.4.1 Notification Dispatcher . . . 17
6.4.2 Informal Care Giver's Interface . . . 17
6.5 Rule Engine . . . 17
6.5.1 Information Flow . . . 17
6.5.2 Rules . . . 18

7 Conclusion **20**

A Smart Home APIs **21**
A.1 Sensor Service API . . . 21
A.1.1 C# Example . . . 21
A.1.2 Java Example . . . 22
A.2 Event Service API . . . 24
A.2.1 C# Example . . . 24
A.2.2 Java Example . . . 24
A.3 Smart Controller API . . . 26
A.3.1 Java Example . . . 26

LIST OF FIGURES

1 Dependencies between this deliverable and other deliverables. 2
2 Information exchanged between the components of the RADIO Ecosystem 5
3 Interconnections and channels of communication between different RADIO deployments 6
4 Secure entity interconnection over public networks in RADIO ecosystem 9
5 A simplified example of an initiated distributed secure computation 12
6 Client-server method invocation 14
7 Smart Home Controller API 16
8 Rule Engine Information flow 18
9 A “blockly” visual representation of a rule definition 19

1 INTRODUCTION

1.1 Purpose and Scope

The purpose of this deliverable is to provide the design of the overall RADIO ecosystem of multiple RADIO Homes, applications for informal care givers, and applications for medical care institutions. This design aims to place the local network of each RADIO Home (developed in WP4) in the overall context of the RADIO ecosystem of communicating RADIO Homes, care givers, and care institutions; and to do so in a way that:

- Allows only relevant information to be shared and ensures the security of private data and extracted information;
- Can scale to a large number of RADIO Home deployments managed by a single care institution;
- Can handle heterogeneity in communication technologies and hardware and software components

This document establishes the architecture and interfaces for efficient and privacy-preserving data exchange between RADIO Home deployments, the formal and informal care-givers applications, and medical care institutions that operate across a wide area. The development and prototyping of methods that satisfy these specifications is outside the scope of this deliverable. The design of each RADIO Home is also outside the scope of this deliverable.

The rest of the document is structured as follows. Section 2 discusses connectivity and privacy requirements that the RADIO ecosystem must comply with and Section 3 presents the proposed architecture, discusses the various components of the ecosystem and their interconnections. The remaining sections discuss in details the various components and the main use cases. In particular, Section 4 discusses the access of the clinical data, Section 5 discusses the data analysis on a privacy-preserving environment and Section 6 presents the Smart Home services and discusses the notifications and automation functionalities of the RADIO Home.

1.2 Approach

Task 5.1 specifies and designs the interconnection structure and interfaces for efficient and privacy-preserving data exchange between RADIO deployments, the formal and informal care-givers applications, and medical care institutions that operate across a wide area. It is envisaged that RADIO deployments will seamlessly integrate in the RADIO ecosystem as nodes to an abstract network regardless of the different communication technologies and of the heterogeneous hardware and software components that comprise them.

NCSR-D leads this task and contributes with large-scale and privacy-preserving data management at the medical care institutions site. TWG, RUB and S&C contribute on specifying the network of heterogeneous communication technologies and hardware and software components.

1.3 Relation to other Work Packages and Deliverables

This deliverable is informed from *D4.1 Architecture for extending smart homes with robotic platforms I* about the architecture of individual RADIO homes.

This deliverable informs *D2.7 Guidelines for balancing between medical requirements and obtrusiveness II* about the RADIO ecosystem architecture to form the personal data retention and unobtrusiveness trade-off. In its turn, D2.7 provides requirements for the next iteration of the architecture *D5.2 Architecture of the RADIO ecosystem II*.

D5.2 supersedes this deliverable.

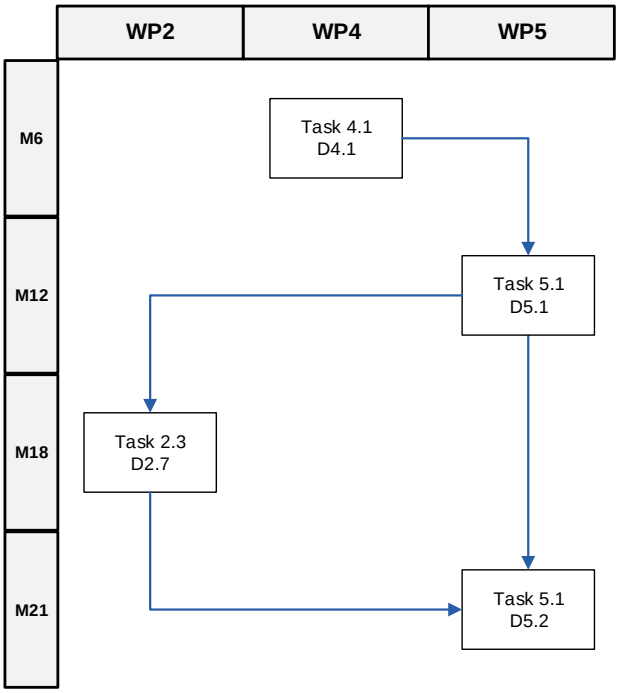

Figure 1: Dependencies between this deliverable and other deliverables.

2 REQUIREMENTS

The design of the RADIO ecosystem architecture should take into account several requirements on the connectivity and availability of RADIO home deployments and the management of the sensitive data transmitted from and to the RADIO home deployments and the clinical sites.

2.1 Connectivity Requirements

It is required that the clinical sites should connect and retrieve data from the RADIO home deployment. During connectivity loss, the RADIO home deployment should continue working autonomously and record data that can be later retrieved by the clinical site. However, RADIO home deployment should be able to transmit urgent notifications to the care-giver and clinical institution about the state of the deployment. However, in a such rare case the RADIO ecosystem must be able to identify the disconnected home deployments and notify accordingly.

2.2 Privacy Requirements

The majority of the data produced by the RADIO home deployment are considered private and certain privacy requirements must be taken into account in every layer of management (i.e. exchanging, retrieving and processing).

Exchange of private data Data exchange between RADIO home deployments and clinical institutes should be secured in the sense that no other party except the ones that are communicating can eavesdrop to the data exchanged. This fact includes parties from outside as well as from inside the RADIO ecosystem.

Management of private data Clinical institute applications can retrieve private data only from the RADIO home deployments that have permission to do so. Moreover, different users of the clinical institute application must have different levels of clearance. The application should be able to validate a user's identity when user credentials are provided.

Processing of private data Computations over data of all the RADIO Home deployments can be defined in order to retrieved potentially useful statistical results for clinical experimentation. Those results should be presented to authorized researchers through the clinical experimentation application. However, the computations over the private data of the RADIO Home deployments should be aggregated values and should not reveal individual data points of a known home deployment. In other words, one should be not be able to either distinguish an individual data point or the initial RADIO Home that has been retrieved from. Moreover, except from the result produced, other parties of the RADIO ecosystem should not deduce (or gain access to) other parties private data.

3 ARCHITECTURE

In this section we discuss the proposed architecture of a RADIO ecosystem that satisfies the requirements (see Section 2).

3.1 Entities and Roles

A *RADIO Home* deployment is the main entity of the RADIO ecosystem. A RADIO Home is essentially an appropriately setup space where the *robot* and the *primary subject* live and perform their daily activities. The RADIO Home is a Smart Home in the sense that smart sensors and actuators are deployed in that space. A RADIO Home can be, for example, a real house building or an appropriately configured room inside a health institution.

A *health institution* is an institution that provides care for the primary subjects and therefore have deployed several RADIO Homes. A health institution for instance can be a hospital or a rehabilitation centre. The medical personel that monitor and provide care for a primary subject are considered to be the *formal care givers* of that RADIO Home and should have access to the clinical reports produced by the RADIO Home deployment.

Apart from the formal care givers there exist the role of the *informal care giver* that are essentially people with no medical expertise but can be notified in case of an emergency that occured in the RADIO Home.

Last but not least, the *research centres* are also entities of the RADIO ecosystem. Those research centres are interested in conducting statistical analysis and data mining to the clinical data produced by the RADIO Home deployments. The certified personel that can analyse those data will be called *health researchers*.

3.2 Components

RADIO ecosystem comprises of several interconnected components. The relevant components and their interconnections are depicted in Figure 2. In this section we describe the responsibilities and operations of each one.

3.2.1 RADIO Home

RADIO Home is the main data provider of the RADIO ecosystem. It processes the raw data retrieved from the deployed sensors and the robot and securely stores the processed data. The main components that reside in a RADIO Home with respect to the other RADIO ecosystem components are:

Sensing and Recognition System that is the collection of sensors deployed in the RADIO Home and in the robot, and the system of algorithms that can recognize abstract events from sensor measurements. This system produces the events that are stored in the *EventLog* database.

Actuation System that is the collection of actuators in the RADIO Home and in the robot that can perform actions in the physical world.

Smart Home Controller that is responsible to integrate the sensor and actuation communication protocols (e.g. WiFi, Z-Wave), dispatch measurements from sensors to the *IoT Platform* and actuations from the IoT Platform to the actuation system.

Medical Report Generator that provides processed clinical data to the authorized personel of the care institute in order to evaluate the condition and wellness of the subject.

Notification Generator that provides alerts and notifications about urgent events occuring in the home to the registered care givers and care institute personel via the appropriate applications.

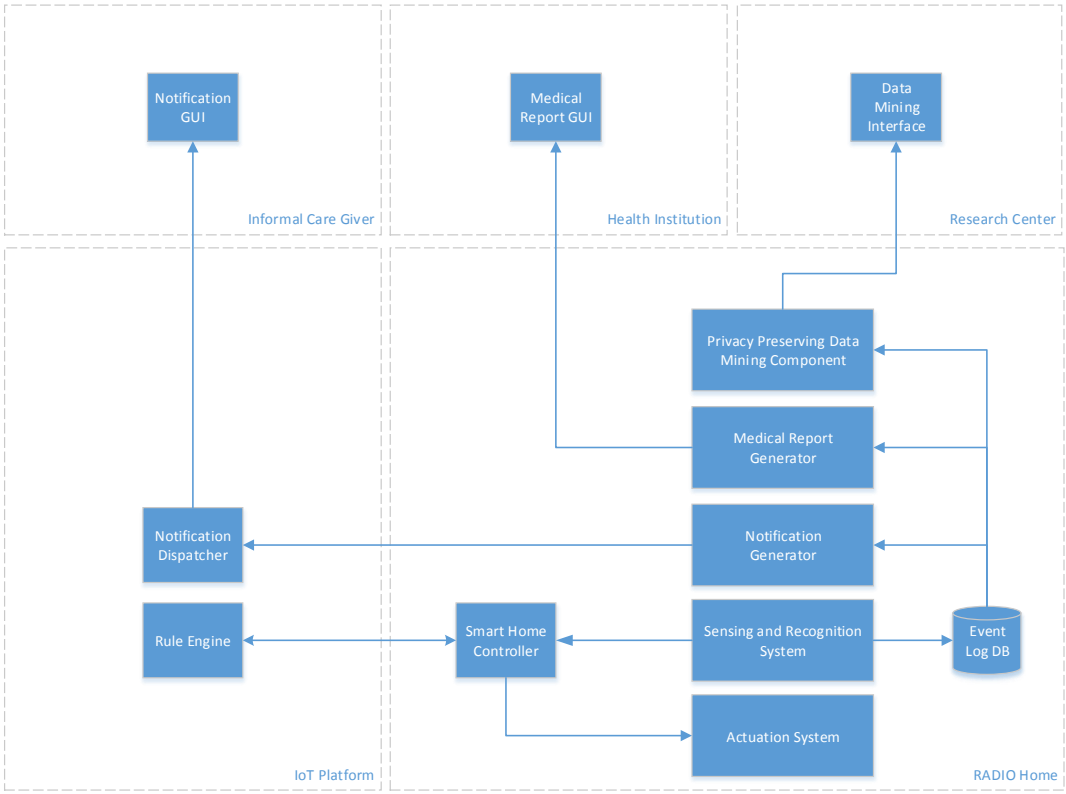

Figure 2: Information exchanged between the components of the RADIO Ecosystem

Privacy Preserving Data Mining Component that participates in and provides partial computations to a distributed computation initiated by the clinical experimentation application.

3.2.2 IoT Platform

The IoT Platform is a collection of Smart Home services outside the boundaries of the RADIO Home that aims to provide more complex automation functionality. The components of the IoT Platform that are relevant to the RADIO Ecosystem are:

Notification Dispatcher that accepts notification from multiple RADIO deployments and forwards them to the respective care givers' application.

Rule Engine that maintains a set of rules for each RADIO deployment and communicate with the respective IoT Gateway of the RADIO Home. The Rule Engine accepts sensor data and produces actuations when certain conditions are satisfied. Its main use is to provide complex automation functionality to the RADIO Home.

3.2.3 Care givers' application

Care givers' application presents individual emergency data such as alerts and notifications to the informal care givers.

3.2.4 Care institution's application

Care institution's application is responsible for presenting the relevant clinical data and emergency notifications produced by the registered RADIO Home deployments. More specifically, the care institution's application:

- connects to a RADIO Home and retrieves relevant clinical data in a secure way;
- retrieves and presents alerts and notifications that are pushed by a registered RADIO Home de-

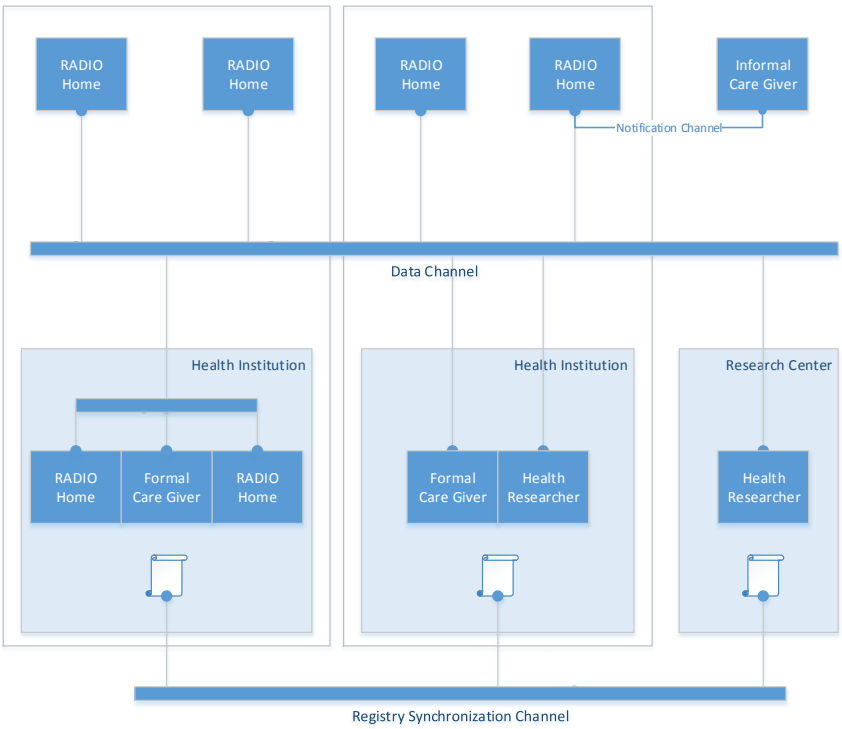

Figure 3: Interconnections and channels of communication between different RADIO deployments

ployment;

- monitors the availability of all the RADIO Home deployments that belong to the institution.

3.2.5 Clinical experimentation application

The main functionality of the clinical experimentation application is to formulate and orchestrate statistical data computations across several RADIO Home deployments in order to retrieve useful aggregated statistics for research purposes. More specifically, the clinical experimentation application

- gives the ability to the researcher to formulate computations that aggregate data across the RADIO Home deployments registered to the RADIO ecosystem;
- contacts the RADIO Home deployments in order to initiate a secure computation in a distributed fashion and retrieves the final result;
- presents the aggregated results to the researcher.

3.3 Topology

In this section we discuss the RADIO ecosystem as a network of RADIO Home deployments and the channels of communication that are required by the architecture.

Figure 3 presents a potential instance of a RADIO ecosystem network topology. In that topology, a collection of RADIO Homes is logically organized as subsidiaries of a health institution. In the physical world, those RADIO Homes can either reside in the health institation or be in a remote location. In either case the health institution is considered to be connected to the RADIO Home. The formal care giver application is considered to be local to the health institution and can access the RADIO Homes that are controlled by the health institute. The health researcher is using the clinical experimentation application to perform data analysis and be either located inside a dedicated research institute or inside the health institute.

3.3.1 Communication Channels

We can distinguish the following communication channels.

Data Channel that is the main channel of the network. This channel is used by the components of the ecosystem in order to provide or consume data. All the components must be able to connect to the data channel. The data security requirements are discussed separately in section 3.4

Notification Channel that is the channel where the notifications flow. It is mainly used to provide notifications to the informal care giver's application and therefore, in principle, the only requirement is the connectivity between the informal care giver's device and the corresponding RADIO Home deployment.

Synchronization Channel is the channel that connects the health institutions and the research centres of the ecosystem in order to sychronize there local registry data. The maintainance of local registries is for discoverability purposes that will be presented in Section 3.3.2. It should be noted that in practice the data and the synchronization channels might share the same physical medium of transportation. However, conceptually there are two separate channels that have different connectivity requirements.

3.3.2 Discoverability

The organization of the RADIO ecosystem assumes that deployed RADIO Homes must be assosiated with exactly one *health institution*. In other words, RADIO Homes cannot exist without an associated health institution, that is most probably the institution which is affiliated with the formal care giver.

The health institution also provides a discovery service to the other components of the RADIO ecosystem. More specifically, the care institution application and the clinical experimentation application can use the discovery service in order to locate the appropriate RADIO Homes to contact. In order to provide that kind of service, the health institutes maintain a local registry of the registered RADIO Homes. Each health institute synchronized its registry with the other health institutes in the RADIO ecosystem via the synchronization channel. This synchronization yields a global registry that can be used to discover components across the ecosystem.

3.3.3 Changes in the topology

During the lifecycle of the RADIO ecosystem, it is natural to expect that RADIO Homes will be deployed or removed from the ecosystem, health institutes and research centers will join. We distinguish three different procedures that must be followed when a new site is added to the topology.

Assume first the deployment of a new RADIO Home. As mentioned previously each RADIO Home is affiliated with a health institution.

1. After the installation of the physical devices (e.g. Smart Home sensors, RADIO robot) in the RADIO Home, the RADIO Home (specifically the Smart Home Controller) is registered to the IoT Platform. This includes the deployment of a signed digital certificate (see Section 6).
2. The health institution registers the newly deployed RADIO Home to its local registry. The registration procedure produces VPN credentials for the RADIO Home that are transfered in a secure, off-band, way and deployed in the RADIO Home in order to establish a secure connection with the affiliated health institution (see Section 3.4).
3. Authorization and authentication of the appropriate medical personel is defined by the administrator of the health institution.
4. Informal care-givers that will be notified for urgent events are also defined for the specific RADIO Home.

Assume now that the health institution decides to join the ecosystem. This requires the following steps.

1. Deploy the site-to-site VPN with other health institutions (see Section 3.4) that requires a valid

signed digital certificate.

2. Install the registry component and initiate the first synchronization with the rest of the health institutions. This requires to known at least one an institution or research center that has already joined the RADIO ecosystem.
3. Start deploying the RADIO Homes following the steps described previously.

Lastly, the procedure for a reserch center is similar to the health institution with the difference that the registry is read only. More specifically, the procedure comprises of the following steps.

1. Deploy the site-to-site VPN with other health institutions (see Section 3.4) that requires a valid signed digital certificate.
2. Install the registry component and initiate the first synchronization with the rest of the health institutions. This requires to known at least one an institution or research center that has already joined the RADIO ecosystem.
3. Define the researchers that are authorized to access the data analysis module of the RADIO ecosystem.

3.4 Security

The overall communication architecture between the Care Giver's Environment and the Care Institution Environment, as well as between the Care Institution Environments themselves has to take into account the sensitivity of personal data that must be exchanged between the above-mentioned entities.

The use of Virtual Private Network (VPN) technology, which extends a private network across a public network, such as the Internet, is a necessity in order to seamlessly achieve the communication objectives of the RADIO project in a secure way. The proposed protocol suite to be used for the implementation of the VPNs in the RADIO project is the Internet Protocol Security (IPSec).

IPSec is an open standard protocol suite for secure Internet protocol (IP) communications by authenticating and encrypting each IP packet of a communication session. IPSec includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to be used during the session. IPSec can be used in protecting data flows between a pair of hosts (host-to-host), between a pair of security gateways (network-to-network), or between a security gateway and a host (network-to-host). It uses cryptographic security services to protect communications over IP networks. IPSec also supports network-level peer authentication, data origin authentication, data integrity, data confidentiality (encryption), and replay protection. It is an end-to-end security scheme operating in the Internet Layer of the Internet Protocol Suite, in contrast with other widespread Internet security systems, which operate in the upper layers at the Application Layer. This unique feature allows user applications to be automatically secured by IPSec at the IP layer.

The Care Institution Environments may be interconnected by deploying IPSec between their respective security gateways (network-to-network), also known as Site-to-Site IPSec VPN, by means of either a partial mesh or a full mesh topology. The Care Giver's Environment may be interconnected with the respective Care Institution Environment by deploying IPSec between the security gateway of the Care Institution Environment and the Workstation and/or mobile device of the Care Giver's Environment (network-to-host), also known as Remote Access IPSec VPN, by means of a Hub and Spoke topology where the Care Institution Environments are considered Hubs and the Care Giver's Environments Spokes. In either case a secure IPSec tunnel, which provides the secure transmission of sensitive personal data in a transparent to the application way, is created between the respective endpoints.

IPSec uses the following protocols to perform various functions:

- Authentication Headers (AH) provide connectionless data integrity and data origin authentication for IP datagrams and provides protection against replay attacks. AH is embedded in the data to be protected.

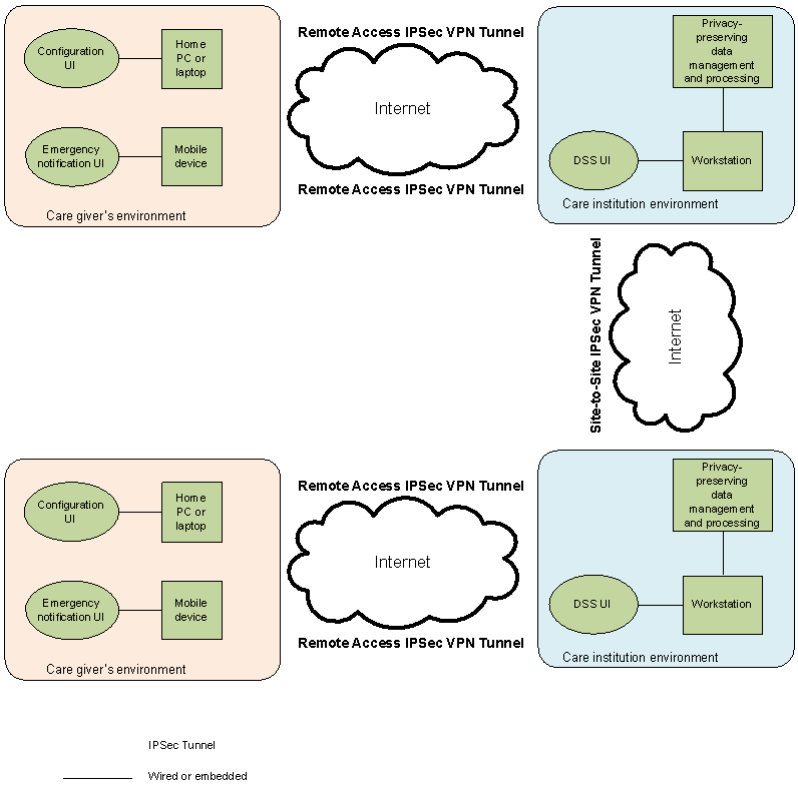

Figure 4: Secure entity interconnection over public networks in RADIO ecosystem

- Encapsulating Security Payloads (ESP) provide confidentiality, data-origin authentication, connectionless integrity, an anti-replay service, and a limited traffic flow confidentiality. ESP encapsulates the data to be protected.
- Security Associations (SA) provide the bundle of algorithms and data that provide the parameters necessary for AH and/or ESP operations. The Internet Security and Key Management Protocol (ISAKMP) provides a framework for authentication and key exchange, with actual keying material provided either by manual configuration with pre-shared keys or Internet Key Exchange (IKE and IKE v2).

Cryptographic algorithms defined for use with IPSec include:

- HMAC-SHA1/SHA2 for integrity protection and authenticity
- TripleDES - CBC for confidentiality
- AES-CBC for confidentiality
- AES-GCM providing confidentiality and authentication together efficiently

The deployment of IPSec VPNs between the RADIO project entities, that need to exchange sensitive data, is a scalable standardized solution that builds secure data channels on top of the Internet which is considered an “untrusted” network. The overall design allows also the creation of multiple connections (using Site-to-Site VPNs or Remote Access VPNs) between entities, which provide a level of redundancy in case of communication network failures. The “extension” of the several private networks over the VPN connections facilitates the overall network monitoring of the various devices that are located in disparate networks.

4 CARE INSTITUTION APPLICATION

4.1 Overview

Events produced at the RADIO Home level should be processed and transformed into clinical reports to be consumed by the medical personel that is responsible for the health of the primary subject that is monitored. Clinical reports are generated from the Medical Report Generator and stored locally at the RADIO Home. Only authorized personel will be able to access those clinical reports by using the care institution application.

4.2 Use Cases

The main use case regarding the access of the RADIO Home clinical reports can be formulated in the following steps.

1. The formal care giver (i.e. medical personel) uses the Formal care giver's interface to view the clinical reports of a specific RADIO Home. In order to do that she must authenticate herself to the system by typing her credentials.
2. The user interface contacts the registry of the health institution to locate the RADIO Home's physical address and then communicates with the Medical Report generator to retrieve the requested clinical report.
3. The Medical Report generator authenticates the user and if the credentials are valid and the user is authorized, the generator retrieves the requested clinical report or generates it on request by retrieving the appropriate event logs from the Event Log database.
4. The clinical report is send back to the user interface.

4.3 Authentication and Authorization

As in every aspect of the RADIO Ecosystem, data management must be controlled and protected from unauthorized use. To this end, the medical personel must authenticate themselves to the RADIO Ecosystem and gain access only to the information that have clearance.

The most common authentication scheme is the use of a combination of a username and password that must be provided by the medical personel when they want to access the clinical reports.

Authorization, on the other hand, requires that each information item has attached a list of permissions for each user. The most common authorization scheme used is the role-based hierachical authorization where information items are organized in a tree-like hierarchy and permissions can be attached in every node of the hierarchy. Permissions are cascaded to the children of the hierarchy.

4.4 Interfaces

4.4.1 Formal Care Giver interface

The user interface to be used by the formal care givers should provide the means for user authentication. Moreover, it must be able to connect to the RADIO Home and project the clinical reports in a user friendly way. The connection between the RADIO Home and the user interface must cover the transmission security described in Section 3.4.

5 Privacy Preserving Data Mining

5.1 Overview

The RADIO ecosystem architecture foresees the statistical analysis of the data produced by the RADIO Home deployments in a way that no requirements (as described in Section 2) are violated. In particular, as mentioned in Section 3, a health researcher should be able to use the clinical experimentation application in order to pose statistical queries against the collection of the data that reside to the RADIO ecosystem. However, the data produced in each RADIO Home deployment are considered private and as such the privacy preserving data mining (PPDM) component must respect the privacy requirements for managing and processing such data.

The proposed architectural approach is reduced into two main ideas:

- The set of the valid statistical queries that are allowed by the system must be appropriately restricted in order to avoid exposing individual data points but only aggregated data. A wide range of existing statistical method depends only in aggregation of data, and therefore can also be formalized in that restricted query set.
- Instead of fetching raw data in order to compute the aggregation, the computations will be performed local to the data and only the result will be trasmitted. In order words, the RADIO Home deployment will contain the processing units in order to perform partial computations on its private data. Moreover, multiple RADIO Homes must collaborate in such a way that can produce the final result of the computation and at the same time will not expose any of their private datapoints.

5.2 Use Cases

Figure 5 demonstrates a simplified version of a typical workflow in order to perform a distributed data mining computation.

1. A health researcher writes a statistical method using the experimentation user interface. The only computations that are allowed to write are the ones that are provided by the PPDM in order to ensure privacy preservation. The analysis is compiled into a set of statistical queries that will be performed by the PPDM components of all the RADIO Home deployments.
2. The experimentation application contacts the synchronized registry (see Section 3.3.2) to locate the health institutions of the RADIO ecosystem and then send the computations to be executed.
3. The care institutions maintain a list of all the registered RADIO Homes and forward the computations to the PPDM component of each RADIO Home.
4. The PPDM component accesses the EventLog database, obscures the results and sends back the obscured result to the health institution.
5. The health institution then combines the obscured results and sends back the unobscured aggregated result to the application of the health researcher.

5.3 Privacy Protection

The privacy protection of the individual data points depends, of course, on the obscuring function that each RADIO Home applies to the raw data. This function should possess certain properties that, on one hand efficiently obscure the raw data from malicious users, and on the other hand enables the aggregations to be performed despite the fact that the constituent data are not available.

As a simplified example consider the Figure 5 and assume that the aggregation that must be collaboratively computed is the summation of all the data points of the RADIO Home nodes. The computation

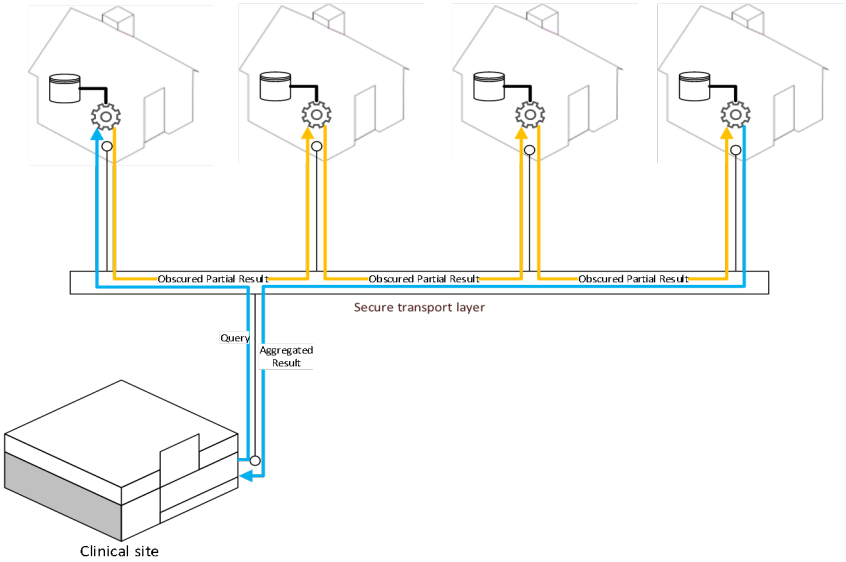

Figure 5: A simplified example of an initiated distributed secure computation

starts without loss of generality by a random node and that nodes are organized in a circle. The starting node adds a random number to its real data and passes the obscured results to the node at the right. The next node adds to this value its own data with its own random number and passes the result to the next node. The last node will eventually have the aggregation of all the nodes. Moreover, the starting node also passes the negation of its random number to its left node and the negative summation of all the random number is performed in the same fashion but in the opposite direction. Now, the desired result is the summation of the two sub aggregations. Specific methods and techniques will be introduced in Task 5.4.

5.4 Interfaces

5.4.1 Programming Language

Traditionally, data analysts use scripting languages that provide the relevant abstractions and libraries to enable their data analysis. It seems natural that the interface of the health researcher to the privacy preserving data mining functionality to a similar programming language or a provided library or domain specific language on top of an existing mainstream programming language (e.g. R, S, Matlab, Python).

In any case, the facilities that such a programming language should provide is the aggregation operations that the privacy-preserving data mining component supports, the common operations on scalar and vector data, and selection of attributes of the eventdatabase to be aggregated. It should also forbit certain data operations from execution that are not supported due to privacy protection.

6 AUTOMATION AND NOTIFICATIONS

6.1 Overview

The Smart Home service is comprised of a *Smart Home Controller* deployed in the Smart Home and the *IoT platform*, that provides additional functionality that cannot be supported by the Smart Home Controller. The IoT platform provides all the ICT resources needed in order for the Smart Home service to scale for millions of Smart Home owners. For a general overview of the S&C Smart Home service, the reader is redirected to Section 3 of D3.1 that provides a good background about the service.

RADIO Home system is providing an automation mechanism in order to perform automatic actuation over the (Z-Wave) Smart Home devices without the intervention of the home owner. To this end, the RADIO Home deployment provides two automation mechanisms. On one hand there is a *24x7 scheduler* integrated inside the Smart Home Controller that allow the Smart Home user (or the service provider) to schedule and perform actions over the Smart Home devices. On the other hand, there is the *Rule Engine* that is deployed in the IoT Plaform and is a service that extends the time scheduling actuation by triggering automation actions based on more complex rules and conditions authored by the user.

The following sections provide a brief presentation of the relevant provided services of the IoT Platform and discuss the functionality of the Smart Home Controller.

6.2 IoT Platform

In the context of RADIO, the Smart Home service and the IoT platform is viewed as a single solution, however the IoT platform per se has ICT resources that can be exploited aside from the smart home service, and hence by other ICT resources of the RADIO solution. Those resources are exposed as a RESTful API. The core services currently provided by the IoT platform are

Sensor Service that is mainly responsible for managing the data and the status of the sensors and actuators of the respective deployment;

Event Service that provides real time communication between the IoT platform and the Smart Home.

RADIO Ecosystem services of automation and notifications will built upon those services, thus, in the following, we provide some context of how those services currently operate.

6.2.1 Sensor Service

The first bunch of functions applicable for RADIO are mainly devoted to manage data and status of sensors/actuators.

1. Upload Sensor Values
2. Upload Sensor/Actuator Status
3. Download Sensor Values
4. Download Sensor/Actuator Status
5. Download Sensors of an Installation

Currently, IoT supports a notification box for a given installation. The notifications are being generated by interaction of user with the graphical user interface (for instance, arm or disarm the security function), by a sensor pushing data/status (for instance, when door opens or close), or by internal computations of sensor data uploaded (for instance, the energy consumption is higher compared to the previous period). There are five categories; (i) comfort, (ii) security, (iii) energy, (iv) control and (v) custom.

For the aforementioned API functions, the following code provides examples on how to access and use

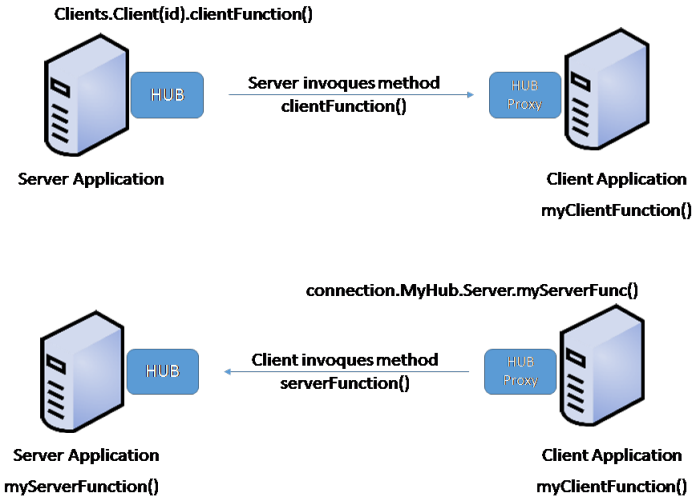

Figure 6: Client-server method invocation

them. Developers will get further information about these functions once the setup of the environment is done and the code is loaded into their favourite integrated development environment (IDE).

6.2.2 Event Service

The Event Service API provides means of real-time communication of events between the IoT platform and clients, either web or mobile phone ones. The Event Service API relies on ASP.NET SignalR technology.

ASP.NET SignalR is a library for ASP.NET developers that simplifies the process of adding real-time web functionality to applications. Real-time web functionality is the ability to have server code push content to connected clients instantly as it becomes available, rather than having the server wait for a client to request new data. SignalR is used to add any sort of "real-time" web functionality to your ASP.NET application ("chat" is often used as an example). Any time a user refreshes a web page to see new data, or the page implements long polling to retrieve new data, it is a candidate for using SignalR. Examples include dashboards and monitoring applications, collaborative applications (such as simultaneous editing of documents), job progress updates, and real-time forms. S&Cs Event Service handles connection management automatically, and lets you broadcast messages to all connected clients simultaneously or send messages to specific clients. The connection between the client and server is persistent, unlike a classic HTTP connection, which is re-established for each communication.

S&Cs Event Service supports "server push" functionality, in which server code can call out to client code in the browser using Remote Procedure Calls (RPC), rather than the request-response model. The Event Service uses the WebSocket transport where available, and falls back to older transports where necessary. While someone could certainly write your application using WebSocket directly, using the service means that a lot of the extra functionality should be implemented. Most importantly, this means that enables the coder to develop an application to take advantage of WebSocket without having to worry about creating a separate code path for older clients.

The SignalR API contains two models for communicating between clients and servers: (i) Persistent Connections and (ii) Hubs.

A *Connection* represents a simple endpoint for sending single-recipient, grouped, or broadcast messages. The Persistent Connection API (represented in .NET code by the PersistentConnection class) gives the developer direct access to the low-level communication protocol that SignalR exposes. Using the Connections communication model will be familiar to developers who have used connection-based APIs such as Windows Communication Foundation.

A *Hub* is a more high-level pipeline built upon the Connection API that allows your client and server to call methods on each other directly (as schematised in Figure 6). SignalR handles the dispatching across machine boundaries as if by magic, allowing clients to call methods on the server as easily as local methods, and vice versa. Using the Hubs communication model will be familiar to developers who have used remote invocation APIs such as .NET Remoting. Using a Hub also allows to pass strongly typed parameters to methods, enabling model binding. S&C Event Service implements this model.

The list of the supported Hubs is the following:

Gateway Hub that provides live events related to the gateway. For instance, when a backup is created, when a sensor has been configured, or when a sensor has been linked or unlinked to the gateway.

Event Hub that provides live events about global notifications arrived to user's notification box.

Installation Hub that provides live events related to installation. For instance, change on security state, comfort state, average temperature, instantaneous electricity consumption.

Sensor Hub that provides live events about new sensor/actuator data and status.

Further details about the methods on each Hub can be found in the code snipsets in Appendix A.2.

6.2.3 Security

The communication with the IoT uses industry standard security mechanisms to interchange information from deployed sensors at home and clients (like web or mobile phone applications). The IoT platform deployment for RADIO uses self-signed certificates (i.e. generated by S&C) and HTTPS for communications.

In order to access the IoT resources and the Smart Home user interfaces the users have to authenticate to the IoT Platform. The IoT Platform supports the standard user password authentication.

6.3 Smart Home Controller

The Smart Home Controller lives locally to the RADIO Home, integrated the Smart Home sensors and communicates with the IoT Platform services. The Smart Home Controller contains also an 24x7 scheduler responsible for executing actions in a scheduled manner.

The Smart Home Controller has been upgraded to offer local accessibility to the different devices that forms the smart home deployment without the need of the use of the IoT Platform services. Of course, not all available functions of the IoT Platform API are possible to be supported locally, so the Smart Home Controller provides only a subset of the IoT API functions. At a glance, the Smart Home Controller API allows to:

1. Inquire information about smart home devices, like
   (a) Name
   (b) Area (where is located at home)
   (c) Type (magnet, presence, thermostat, ...)
   (d) Current value (like current temperature)
   (e) Current status
2. Perform actuation actions, like
   (a) Switch on/off smart plugs
   (b) Switch on/off lights
   (c) Dimming of lights
   (d) Thermostat target temperature

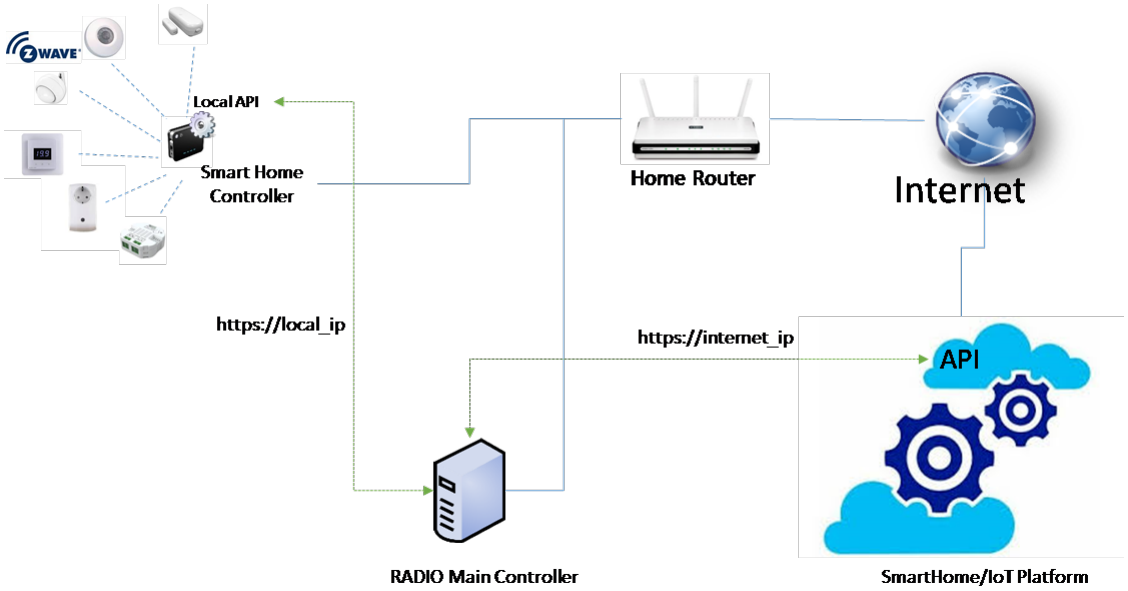

Figure 7: Smart Home Controller API

(e) Open/Close doors, shadows, curtains, ...

3. Manage the security status of the system
4. Inquire the security status of the system and security devices

A local client, like "RADIO Main Controller" in Figure 7, means a device that operates within the RADIO home local area network. The client would be able to operate with smart home devices by using the two APIs, the one exposed by the Smart Home Controller (local API) and the one exposed by the IoT platform. Which one to be used is out of the scope of this section, and will depend on the final target to be implemented by the local client.

The most important API difference between the local API and the IoT API for the client application is that to use local one, the client application should implement a polling mechanism in order to detect changes in the information provided by the smart home controller. While for the IoT API, as explained in Section 6.2.2, the client gets automatic updates through the different HUBs supported, and perform specific actions upon reception of specific events.

The local API is complemented with a Java SDK. An example snipset that illustrates the basic functionality of the Smart Home Controller local API can be found in Appendix A.3.

6.3.1 24x7 Scheduler

In current Smart Home solution, as described in D3.1, the 24x7 scheduler is ready at IoT level, but not at Smart Home level. RADIO Home scheduler allows the home owner, or the service provider (care company for instance) to perform actions over the smart home divided in three different categories:

Comfort for instance setting of thermostat temperature

Security setting security level at home, mimic presence at home

Energy switch on/off appliances, heavy duty energy consumers, switch on/off lights, create different light ambient with dimmers, etc

6.3.2 Security

The local API uses same authentication credentials as the IoT API, and those credentials are synchronised automatically by the IoT management processes.

6.4 Notifications

The Hubs supported by S&Cs smart home have been described in Section 6.2.2. When a (software) client login into the IoT, its credentials will allow it to receive only the events related to any installation that have been previously assigned to that user id.

There are two possibilities to extend the notification mechanism in order to integrate ADL recognition system (WP3) to report an emergency situation.

- First, ADL recognition has to be mapped into a sensor device or group of sensors. For instance, if the complexity of the ADL system is low, like a "FALL detection", the ADL system can be mapped as a safety sensor, and can use the notification message already define by the smart home service. If the ADL system has to be mapped as a group of sensors by the IoT, the rule engine can generate a notification (using the already existing Hubs) if the condition can be computed.
- Alternatively, the ADL system can be virtualised, that is the ADL is not directly mapped to a real sensor. This case can also be supported but will imply the development of a piece of software code that will take into account all the information contained in the virtualised ADL, and produce an output.

6.4.1 Notification Dispatcher

If the notification goes through existing Hubs, any user associated to that installation will receive the notification, including the home owner through the smart home GUI and app. This would be fine to forward notifications to caregivers (for instance families), but may not be the optimal way for the institution taken care remotely of the elder person. In such a particular case, and when there should be notification messages that only care institution should support, it is desirable to create additional Hubs.

A new Hub created will use implicitly IoT API for communication, thus ensuring the notifications can only be consumed by the intended clients (in this case the care institution). The care institution software client, will know the installation, the sensor and the time the notification is created. And have the freedom to encapsulate in the content of the message any additional information need.

6.4.2 Informal Care Giver's Interface

The informal care giver's interface is a simple interface that must present a real-time feed of the produced notifications. This can be implemented by subscribing to the appropriate hub that dispatch the notifications. The interface should also provide means of authenticating as a valid user to the IoT Platform.

6.5 Rule Engine

RADIO Home Rule Engine is a step forward beyond the 24x7 scheduler. The rule engine extends the actuation based on time by triggering automation actions over the actuator devices based on activity reported by the different sensors at home. As examples that benefit from such a rule engine are the automatic door open/close, the automatic ventilation when the quality of air is degraded, automatic switch on/off of climate system if door/window is opened or closed to prevent waste of energy resources and high bills from energy retailer).

Moreover, an interesting feature is the ability to send notifications, which is interesting for early notification of anomalous situations (no activity from presence sensors over a period of time may indicate a potential problem with elder person for instance, no consumption on electrical kitchen means the elder person is not having hot meals).

6.5.1 Information Flow

Figure 8 show the way data will flow from/to smart home to IoT for the execution of a rule.

1. A device reports a change to the controller (door is open for instance). The IoT gateway forwards the data to the IoT cloud by using the API.

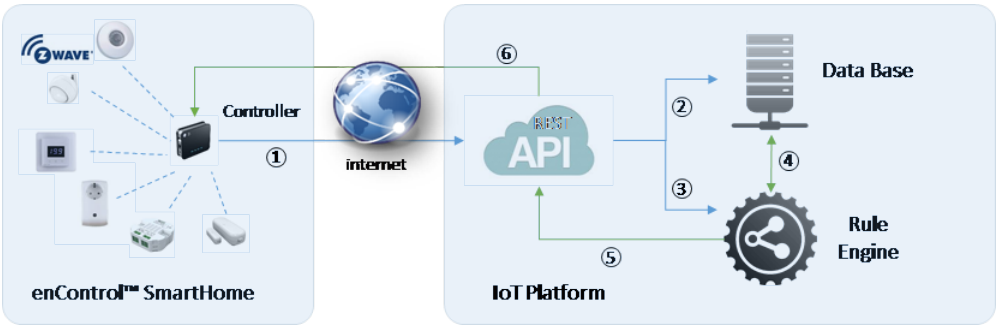

Figure 8: Rule Engine Information flow

2. The IoT stores then information in the local data base
3. The IoT pushes the event to rule engine component.
4. The Rule Engine component applies the event to the specified rules, it may require to know the status or data from other devices, so will inquire the data base.
5. If the condition is met, the rule engine executes the action using the IoT API
6. The IoT platform communicate to the controller the action to be executed (for instance, shut down climate system).

6.5.2 Rules

The rules of the engine are configured by the user. There are simple condition statements using the "if-this-then-that" form. When the *condition* of the "if" statement is satisfied then the *action* in the "then" statement is executed.

The condition and actions are abstract expressions that involve the current or previous states of the sensors in the Smart Home. Specifically, supported conditions of the rule engine are based on:

- Status of a device
  - security device (ok, detect, alarm, offline, locked, unlocked)
  - other device (on, off, locked, unlocked)
- Status of security (Arm away, arm stay, disarm and alarm)
- Target temperature of thermostat
- Value captured by sensors device (for instance temperature)
- A value reported by a device is compared with constant
- A value reported by a device is compared with a value reported by other device

Moreover, conditions also support expressions based on temporal parameters. For example, the rule engine supports:

- A condition occurred during a period of time
- A condition occurred during a specific time
- A condition occurred during a specific date
- A condition occurred during a specific day of a week

As mentioned before automation actions are in general commands to the Smart Home actuators, thus the supported actions depend on the deployed sensors in the Smart Home. An indicative list of supported actions contains the following actions

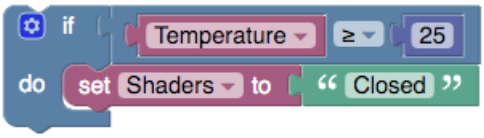

Figure 9: A “blockly” visual representation of a rule definition

- Set security status
- Set thermostat target temperature
- Restore thermostat temperature
- Switch on/off devices
- Set dimmer value
- Set position of curtains
- Send notifications
- Execute the action after a period of time

The definition of the active rules regarding a RADIO Home deployment will be authored by the user through an authoring user interface. The graphical user interface will use a visual representation of the rules and the definition can be completed by a non technical user. Figure 9 depicted the envisaged visual representation of a rule.

Besides the definition of the rules, there are no other direct user interactions with the Rule Engine. All interactions (execution of a given rule) are based on push events towards the IoT platform. The execution of a rule must be immediate upon reception of the event from the Smart Home.

7 Conclusion

In this report we described the overall architecture of the RADIO ecosystem and established the interconnections of the various components, entities and sites.

In the proposed architecture, special care has been taken regarding data protection requirements. Sensitive data is only accessible by authenticated and authorized personel. Transmission of raw data points is avoided, while trasmission of derived abstract data is encrypted when leaving the boundaries of a RADIO Home. Moreover, privacy preservation in data analysis is enabled from the architecture by enforcing the data to remain local and allow only certain aggregation to be performed.

A SMART HOME APIS

A.1 Sensor Service API

A.1.1 C# Example

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Linq;
using System.Net;
using nassist.ServiceInterface.Services;
using nassist.ServiceModel;
using nassist.Shared.SensorData;
using ServiceStack;

namespace Example
{
    public class Program
    {
        private static IJsonServiceClient client;
        private static AuthenticateResponse credentials;

        private const string BASE_URL = "http://base_url/api";

        private const string CAMERA_ID = "XXXX";
        private const string SENSOR_ID = "XXXX";
        private const string INSTALLATION_ID = "XXXX";

        private const string USERNAME = "XXXXX";
        private const string PASSWORD = "XXXXX";

        public static void Main(string[] args)
        {
            client = new JsonServiceClient(BASE_URL);
            credentials = client.Post(new Authenticate { UserName = USERNAME, Password = PASSWORD, RememberMe = true });

            uploadSensorValues();
            uploadSensorStatuses();
            getSensorValues();
            getSensorStatuses();
            getSensorsForInstallation();
            getNotificationsByType();
            createCustomNotification();
            Console.ReadKey();
        }

        public static void uploadSensorValues()
        {
            try
            {
                client.Post(new SensorValues
                {
                    Id = SENSOR_ID,
                    DataPoints = new List<DataPoint> { new DataPoint { Date = DateTime.Now, Value = new
                        Random().NextDouble() } }
                });

                Console.WriteLine("Data upload success!");
            }
            catch (WebServiceException ex)
            {
                Console.WriteLine("Error uploading data: " + ex);
            }
        }

        public static void uploadSensorStatuses()
        {
            try
            {
                client.Post(new SensorStatuses
                {
                    Id = SENSOR_ID,
                    StatusPoints = new List<StatusPoint> { new StatusPoint { Date = DateTime.Now, Status = "ok", Trigger =
                        credentials.UserId, TriggerName = credentials.UserName } }
                });

                Console.WriteLine("Status upload success!");
            }
            catch (WebServiceException ex)
            {
                Console.WriteLine("Error uploading data: " + ex);
            }
        }

        public static void getSensorValues()
        {
            var valuesResponse = client.Get(new SensorValues { Id = SENSOR_ID });

            foreach (var dataValue in valuesResponse.Values)
            {
                Console.WriteLine("Date: " + dataValue.Date + " Value: " + dataValue.Value);
            }
        }

        public static void getSensorStatuses()
        {
            var statusesResponse = client.Get(new SensorStatuses { Id = SENSOR_ID});

            foreach (var dataValue in statusesResponse.Statuses)
            {
                Console.WriteLine("Date: " + dataValue.Date + " Status: " + dataValue.Status);
            }
        }

        public static void getSensorsForInstallation()
        {
            var installationSensors = client.Get(new InstallationSensors { Id = new Guid(INSTALLATION_ID) });
```

```
            foreach (var sensor in installationSensors.Sensors)
            {
                Console.WriteLine(sensor.Id + " " + sensor.Name + " " + sensor.Type);
            }
        }

        public static void getNotificationsByType()
        {
            var eventsResponse = client.Get(new EventsBatch { UserId = credentials.UserId, Type = "security" });

            foreach (var evt in eventsResponse.Events)
            {
                Console.WriteLine(evt.Subtype + " " + evt.TranslatedDescription);
            }
        }

        #region Helpers

        public static string ImageToBase64(string fileName)
        {
            Image i = Image.FromFile(fileName);

            using (var stream = new MemoryStream())
            {
                i.Save(stream, ImageFormat.Jpeg);

                return Convert.ToBase64String(stream.ToArray());
            }
        }

       public static void createCustomNotification()
        {
            try
            {
                client.Post(new Events
                {
                    Event = new AzureEvent
                    {
                        Date = DateTime.UtcNow,
                        Type = "custom",
                        Subtype = "custom",
                        InstallationId = INSTALLATION_ID,
                        Installation = installationDetails.Name,
                        Description = "My custom event text",
                        Pending = true
                    },
                    UserIds = new List<int> { installationDetails.OwnerId.Value }
                });

                Console.WriteLine("Event created successfully");
            }
            catch (Exception e)
            {
                Console.WriteLine("Error creating event: " + e);
            }
        }

        #endregion
    }
}
```

A.1.2 Java Example

```
package nassist.api.client.example;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.IOException;
import java.net.HttpURLConnection;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.Random;
import java.util.UUID;

import org.apache.commons.codec.binary.Base64;
import org.apache.commons.io.FileUtils;
import org.joda.time.DateTime;

import nassist.api.client.example.dto.Authenticate;
import nassist.api.client.example.dto.AuthenticateResponse;
import nassist.api.client.example.dto.AzureEvent;
import nassist.api.client.example.dto.CameraPhoto;
import nassist.api.client.example.dto.CameraPhotoResponse;
import nassist.api.client.example.dto.DataPoint;
import nassist.api.client.example.dto.EventsBatch;
import nassist.api.client.example.dto.EventsBatchResponse;
import nassist.api.client.example.dto.InstallationSensors;
import nassist.api.client.example.dto.InstallationSensorsResponse;
import nassist.api.client.example.dto.Photo;
import nassist.api.client.example.dto.Sensor;
import nassist.api.client.example.dto.SensorStatuses;
import nassist.api.client.example.dto.SensorStatusesResponse;
import nassist.api.client.example.dto.SensorValues;
import nassist.api.client.example.dto.SensorValuesResponse;
import nassist.api.client.example.dto.StatusPoint;
import net.servicestack.client.JsonServiceClient;
import net.servicestack.client.Utils;
import net.servicestack.client.WebServiceException;

public class Program {

        private static String USERNAME = "XXXX";
        private static String PASSWORD = "XXXX";

        private static String BASE_URL = "https://base_url/api";

        private static String INSTALLATION_ID =   XXXXXXXXX ;
        private static String SENSOR_ID =   XXXXXXXX  ;

        private static JsonServiceClient client;
        private static AuthenticateResponse authDetails;

        private static SimpleDateFormat dateFormat = new SimpleDateFormat("yy-MM-dd");

        public static void main (String[] args){
                Authenticate auth = new Authenticate();
                auth.setUserName(USERNAME);
                auth.setPassword(PASSWORD);
```

```
        client = new JsonServiceClient(BASE_URL);
        authDetails = client.post(auth);
        uploadSensorValues();
        uploadSensorStatuses();
        getSensorValues();
        getSensorStatuses();
        getSensorsForInstallation();
        getNotificationsByType();
createCustomNotification();
}

public static void uploadSensorValues(){
        SensorValues uploadValuesRequest = new SensorValues();
        uploadValuesRequest.setId(SENSOR_ID);

        ArrayList<DataPoint> dataPoints = new ArrayList<DataPoint>();

        DataPoint data = new DataPoint();
        data.setDate(new Date());
        data.setValue(new Random().nextDouble());

        dataPoints.add(data);

        uploadValuesRequest.setDataPoints(dataPoints);

        try{
                client.post(uploadValuesRequest);
                System.out.println("Data upload success!");
        } catch (WebServiceException e){
                System.out.println("Error uploading data: " + e.getErrorCode() + " " + e.getErrorMessage());
        }
}

public static void uploadSensorStatuses(){
        SensorStatuses uploadStatusesRequest = new SensorStatuses();
        uploadStatusesRequest.setId(SENSOR_ID
                        );
        ArrayList<StatusPoint> statusPoints = new ArrayList<StatusPoint>();

        StatusPoint status = new StatusPoint();
        status.setDate(new Date());
        status.setStatus("ok");
        status.setTrigger(authDetails.getUserId().toString());
        status.setTriggerName(authDetails.getUserName());

        statusPoints.add(status);

        uploadStatusesRequest.setStatusPoints(statusPoints);

        try{
                client.post(uploadStatusesRequest);
                System.out.println("Status upload success!");
        } catch (WebServiceException e){
                System.out.println("Error uploading statuses: " + e.getErrorCode() + " " + e.getErrorMessage());
        }
}

public static void getSensorValues(){
        SensorValues valuesRequest = new SensorValues();
        valuesRequest.setId(SENSOR_ID);

        SensorValuesResponse valuesResponse = client.get(valuesRequest);

        for(int i = 0; i < valuesResponse.Values.size(); i++){
                System.out.println(valuesResponse.Values.get(i).Value);
        }
}

public static void getSensorStatuses(){
        SensorStatuses statusesRequest = new SensorStatuses();
        statusesRequest.setId(SENSOR_ID);

        SensorStatusesResponse statusesResponse = client.get(statusesRequest);

        for(int i= 0; i < statusesResponse.Statuses.size(); i++){
                System.out.println(statusesResponse.Statuses.get(i).Status);
        }
}

public static void getSensorsForInstallation(){
        InstallationSensors allSensorsRequest = new InstallationSensors();
        allSensorsRequest.Id = UUID.fromString(INSTALLATION_ID);

        InstallationSensorsResponse installationSensors = client.get(allSensorsRequest);

        for(Sensor s : installationSensors.Sensors){
                System.out.println(s.getId() + " " + s.getName() + " " + s.getType());
        }
}

public static void getNotificationsByType(){
        EventsBatch eventsRequest = new EventsBatch();
        eventsRequest.setUserId(authDetails.getUserId().toString());
        eventsRequest.setType("security");

        EventsBatchResponse eventsResponse = client.get(eventsRequest);

        for(AzureEvent event : eventsResponse.Events){
                System.out.println(event.Subtype + " " + event.TranslatedDescription);
        }
}

private static void createCustomNotification() {
        AzureEvent event = new AzureEvent();
        event.setDate(new Date());
        event.setType("custom");
        event.setSubtype("custom");
        event.setDescription("My custom event text");
        event.setInstallationId(INSTALLATION_ID);
        event.setInstallation(installationDetails.getName());
        event.setPending(true);

        Events request = new Events();
        request.setEvent(event);

        ArrayList<Integer> usersToNotify = new ArrayList<Integer>();
        usersToNotify.add(installationDetails.OwnerId);

        request.setUserIds(usersToNotify);

        try{
                client.post(request);
                System.out.println("Event created successfully");
        } catch (Exception e){
                System.out.println("Error creating an event");
```

```
                    e.printStackTrace();
                }
        }
        // Helpers
        public static String imageToBase64(String fileName) throws IOException
        {
                return Base64.encodeBase64String(FileUtils.readFileToByteArray(new
                      File(Program.class.getResource(fileName).getFile())));
        }
}
```

A.2 Event Service API

A.2.1 C# Example

```
using System;
using Microsoft.AspNet.SignalR.Client;
using ServiceStack;

namespace LiveNotifications
{
    public class Program
    {
        private static HubConnection con;
        private static IHubProxy installationHub;
        private static IHubProxy sensorHub;

        private const string NOTIFICATIONS_URL = "http://base-url";
        private const string INSTALLATION_SECURITY_EVENT = "receiveNewInstallationSecurityStatus";
        private const string SENSOR_NEW_VALUE_EVENT = "receiveNewSensorData";

        private const string INSTALLATION_ID = "XXXX";
        private const string SENSOR_ID = "XXXX";

        private const string UserName = "XXXX";
        private const string Password = "XXXX";

        private static readonly JsonServiceClient Client = new JsonServiceClient(NOTIFICATIONS_URL + "/api");

        public static void Main(string[] args)
        {
            Console.WriteLine("Press any key to start and a second time to exit the program");

            Console.ReadKey();

            // Authenticate on the platform
            Client.Post(new Authenticate { UserName = UserName, Password = Password });

            // Create a connection against the nofications system and configure it to use the obtained credentials
            con = new HubConnection(NOTIFICATIONS_URL)
            {
                CookieContainer = Client.CookieContainer
            };

            con.StateChanged += change =>
            {
                switch (change.NewState)
                {
                    case ConnectionState.Connected:
                        Console.WriteLine("Connected!");

                        // Methods to call on server must be camel case
                        installationHub.Invoke("joinGroup", INSTALLATION_ID);
                        sensorHub.Invoke("joinGroup", SENSOR_ID);

                        break;

                    case ConnectionState.Disconnected:
                        Console.WriteLine("Disconnected!");
                        break;
                }
            };

            // Hub names must be camel case
            installationHub = con.CreateHubProxy("installationHub");
            sensorHub = con.CreateHubProxy("sensorHub");

            // Subscribe to desired events on each Hub
            installationHub.On<string, string, string, string>(INSTALLATION_SECURITY_EVENT, (installationId, date, status,
                  trigger) =>
            {
                Console.WriteLine("Received Security Status change notification");
                Console.WriteLine("Installation Id: " + installationId);
                Console.WriteLine("Date: " + date);
                Console.WriteLine("Status: " + status);
                Console.WriteLine("Trigger: " + trigger);
            });

            sensorHub.On<string, string, string>(SENSOR_NEW_VALUE_EVENT, (sensorId, date, value) =>
             {
                 Console.WriteLine("Received New Sensor value notification");
                 Console.WriteLine("Sensor Id: " + sensorId);
                 Console.WriteLine("Date: " + date);
                 Console.WriteLine("Value: " + value);
             });

            con.Start();

            Console.ReadKey();

            con.Stop();
        }
    }
}
```

A.2.2 Java Example

```
package nassist.api.examples.livenotifications;

import java.net.CookieHandler;
import java.net.CookieManager;
import java.net.HttpCookie;
import java.util.Scanner;
import microsoft.aspnet.signalr.client.ConnectionState;
import microsoft.aspnet.signalr.client.LogLevel;
```

```
import microsoft.aspnet.signalr.client.Logger;
import microsoft.aspnet.signalr.client.StateChangedCallback;
import microsoft.aspnet.signalr.client.http.CookieCredentials;
import microsoft.aspnet.signalr.client.hubs.HubConnection;
import microsoft.aspnet.signalr.client.hubs.HubProxy;
import microsoft.aspnet.signalr.client.hubs.SubscriptionHandler3;
import microsoft.aspnet.signalr.client.hubs.SubscriptionHandler4;
import nassist.api.examples.livenotifications.notificationmodel.dto.Authenticate;
import net.servicestack.client.JsonServiceClient;

public class Program {

        private static HubConnection con = null;
        private static HubProxy installationHub = null;
        private static HubProxy sensorHub = null;

        private static final String NOTIFICATIONS_URL = "https://base-url.com";

        private static final String INSTALLATION_SECURITY_EVENT = "receiveNewInstallationSecurityStatus";
        private static final String SENSOR_NEW_VALUE_EVENT = "receiveNewSensorData";

        private static final String INSTALLATION_ID = "XXXX";
        private static final String SENSOR_ID = "XXXX";

        private static final String UserName = "XXXXX";
        private static final String Password = "XXXX";

        private static final JsonServiceClient client = new JsonServiceClient(NOTIFICATIONS_URL + "/api");

        public static void main(String[] args){
                System.out.println("Press any key to start and a second time to exit the program");

                Scanner sc = new Scanner(System.in);

                sc.nextLine();

                // Obtain credentials
                CookieManager cmanager = new CookieManager();
                CookieHandler.setDefault(cmanager);

                Authenticate auth = new Authenticate();
                auth.setUserName(UserName);
                auth.setPassword(Password);
                client.post(auth);

                Logger logger = new Logger() {
                        @Override
                        public void log(String message, LogLevel level) {
                                System.out.println(message);
                        }
                };

                // Create connection with notification server
                con = new HubConnection(NOTIFICATIONS_URL, "", true, logger);

                CookieCredentials credentials = new CookieCredentials();

                for(HttpCookie cookie: cmanager.getCookieStore().getCookies()){
                        credentials.addCookie(cookie.getName(), cookie.getValue());
                }

                con.setCredentials(credentials);

                con.stateChanged(new StateChangedCallback() {
                        @Override
                        public void stateChanged(ConnectionState connectionStateOld, ConnectionState connectionStateNew) {
                                switch(connectionStateNew){
                                case Connected:
                                        System.out.println("Connected!");
                                        // Method names must be camel case
                                        installationHub.invoke("joinGroup", INSTALLATION_ID);
                                        sensorHub.invoke("joinGroup", SENSOR_ID);

                                        break;

                                case Disconnected:
                                        System.out.println("Disconnected!!");
                                        break;

                                default:
                                        break;
                                }
                        }
                });

                // Hub names must be camel case
                installationHub = con.createHubProxy("installationHub");
                sensorHub = con.createHubProxy("sensorHub");

                installationHub.on(INSTALLATION_SECURITY_EVENT, new SubscriptionHandler4<String, String, String, String>() {
                        @Override
                        public void run(String installationId, String date, String status, String trigger) {
                                System.out.println("Received Security Status change notification");
                                System.out.println("Installation Id: " + installationId);
                                System.out.println("Date: " + date);
                                System.out.println("Status: " + status);
                                System.out.println("Trigger: " + trigger);
                        }
                }, String.class, String.class, String.class, String.class);

                sensorHub.on(SENSOR_NEW_VALUE_EVENT, new SubscriptionHandler3<String, String, String>(){
            @Override
            public void run(String sensorId, String date, String value) {
                System.out.println("Received New Sensor value notification");
                                System.out.println("Sensor Id: " + sensorId);
                                System.out.println("Date: " + date);
                                System.out.println("Value: " + value);
            }
        }, String.class, String.class, String.class);

                con.start();

                sc.nextLine();

                sc.close();

                con.stop();
        }
}
```

A.3 Smart Controller API

A.3.1 Java Example

```
package com.sensingcontrol.gateway.localclient;
import net.servicestack.client.JsonServiceClient;
import com.sensingcontrol.gateway.localclient.model.dto.Authenticate;
import com.sensingcontrol.gateway.localclient.model.dto.AuthenticateResponse;
import com.sensingcontrol.gateway.localclient.model.dto.InstallationGatewaySecurityStatusResponse;
import com.sensingcontrol.gateway.localclient.model.dto.InstallationSecurityStatus;
import com.sensingcontrol.gateway.localclient.model.dto.InstallationSecurityStatusResponse;
import com.sensingcontrol.gateway.localclient.model.dto.InstallationSensorsActuableResponse;
import com.sensingcontrol.gateway.localclient.model.dto.InstallationSensorsComfortResponse;
import com.sensingcontrol.gateway.localclient.model.dto.InstallationSensorsSecurityResponse;
import com.sensingcontrol.gateway.localclient.model.dto.InstallationSensorsWithAreaIdAndNameResponse;
import com.sensingcontrol.gateway.localclient.model.dto.SensorDimmable;
import com.sensingcontrol.gateway.localclient.model.dto.SensorDimmableResponse;
import com.sensingcontrol.gateway.localclient.model.dto.SensorPowerToggle;
import com.sensingcontrol.gateway.localclient.model.dto.SensorPowerToggleResponse;
import com.sensingcontrol.gateway.localclient.model.dto.SensorThermostatFanMode;
import com.sensingcontrol.gateway.localclient.model.dto.SensorThermostatMode;
import com.sensingcontrol.gateway.localclient.model.dto.SensorThermostatSetPoint;
import com.sensingcontrol.gateway.localclient.model.dto.SensorThermostatSetPointResponse;
import com.sensingcontrol.gateway.localclient.model.dto.SensorThermostatUpdateFanMode;
import com.sensingcontrol.gateway.localclient.model.dto.SensorThermostatUpdateFanModeResponse;
import com.sensingcontrol.gateway.localclient.model.dto.SensorThermostatUpdateMode;
import com.sensingcontrol.gateway.localclient.model.dto.SensorThermostatUpdateModeResponse;
import com.sensingcontrol.gateway.localclient.model.dto.SensorWithAreaIdAndName;
import com.sensingcontrol.gateway.localclient.model.dto.SensorWithAreaName;

public class App
{
  private static String USERNAME = "YOUR USERNAME";
  private static String PASSWORD = "YOUR PASSWORD";
  private static String BASE_URL;

  private static String INSTALLATION_ID =     ";
  private static String SENSOR_ACTUABLE_ID = "";
  private static String SENSOR_DIMMABLE_ID = "";
  private static String THERMOSTAT_ID = "";

  private static JsonServiceClient client;

  public static void main (String[] args){
    BASE_URL = "https://" + new AutoDiscovery().waitForIP(INSTALLATION_ID);
    Authenticate auth = new Authenticate();
    auth.setUserName(USERNAME);
    auth.setPassword(PASSWORD);

    client = new JsonServiceClient(BASE_URL);
    client.post("/authenticate", auth, AuthenticateResponse.class);

    getSecurityStatus();
    setSecurityStatus();
    getSensors();
    getActuableSensors();
    toggleActuatorSensor();
    toggleDimmableSensor();
    getComfortSensors();
    setThermostatSetPoint();
    setThermostatMode();
    setThermostatFanMode();
    getSecuritySensors();
  }

  private static void getSecurityStatus() {
    System.out.println("\nGetting security status");
    InstallationSecurityStatusResponse response = client.get("/installations/" + INSTALLATION_ID + "/securitystatus",
          InstallationSecurityStatusResponse.class);
    System.out.println("Security status obtained successfully: " + response.SecurityStatus);
  }

  private static void setSecurityStatus() {
    System.out.println("\nSetting security status");
    InstallationSecurityStatus request = new InstallationSecurityStatus();
    request.setSecurityStatus("disarm");
    client.put("installations/" + INSTALLATION_ID + "/gatewaysecuritystatus", request,
          InstallationGatewaySecurityStatusResponse.class);
    System.out.println("Security status changed successfully");
  }

  private static void getSensors() {
    System.out.println("\nGetting all sensors - Name(id) Area Value Status");
    System.out.println("------------------------------------------------");
    InstallationSensorsWithAreaIdAndNameResponse response = client.get("/installations/" + INSTALLATION_ID +
          "/sensors/area", InstallationSensorsWithAreaIdAndNameResponse.class);
    for(SensorWithAreaIdAndName s : response.Sensors){
      if (s.Name != null) System.out.println(s.Name + "(" + s.Id + ") " + s.AreaName + " " + s.Value + " " + s.Status);
    }
  }

  private static void getActuableSensors() {
    System.out.println("\nGetting actuable sensors");
    System.out.println("------------------");
    InstallationSensorsActuableResponse response = client.get("/installations/" + INSTALLATION_ID + "/sensors/actuable",
          InstallationSensorsActuableResponse.class);

    for(SensorWithAreaName s : response.Sensors){
      System.out.println(s.AreaName + " " + s.Name + " " + s.Value + " " + s.Status);
      }
  }

  private static void toggleActuatorSensor() {
    System.out.println("\nToggling Actuator Sensor");
    SensorPowerToggle request = new SensorPowerToggle();
    request.setValue("true");
    client.post("/sensors/" + SENSOR_ACTUABLE_ID + "/toggle", request, SensorPowerToggleResponse.class);
    System.out.println("Actuator changed to target value successfully!");
  }

  private static void toggleDimmableSensor() {
    System.out.println("\nSetting Dimmable Level");
    SensorDimmable request = new SensorDimmable();
    request.setValue(50d);
    client.post("/sensors/" + SENSOR_DIMMABLE_ID + "/dimmable", request, SensorDimmableResponse.class);
    System.out.println("Dimmable changed to target value successfully!");
  }
```

```
  private static void getComfortSensors() {
    System.out.println("\nGetting comfort sensors");
    System.out.println("-------------------------");
    InstallationSensorsComfortResponse response = client.get("/installations/" + INSTALLATION_ID + "/sensors/comfort",
          InstallationSensorsComfortResponse.class);
    for(SensorWithAreaName s : response.Sensors){
      System.out.println(s.AreaName + " " + s.Name + " " + s.Value + " " + s.Status + " " + s.SetPoint + " " + s.Mode);
    }
  }

  private static void setThermostatSetPoint() {
    System.out.println("\nSetting Thermostat Set Point");
    SensorThermostatSetPoint request = new SensorThermostatSetPoint();
    request.setSetPoint(20d);
    request.setIsCelsius(true);
    SensorThermostatSetPointResponse response = client.put("/thermostats/" + THERMOSTAT_ID + "/update/setpoint", request,
          SensorThermostatSetPointResponse.class);
    if(response.isSetPointSuccess()){
      System.out.println("Thermostat Set Point set successfully!");
    }
  }

  private static void setThermostatMode() {
    System.out.println("\nSetting Thermostat Mode");
    SensorThermostatUpdateMode request = new SensorThermostatUpdateMode();
    request.setMode(SensorThermostatMode.Cool);
    SensorThermostatUpdateModeResponse response = client.put("/thermostats/" + THERMOSTAT_ID + "/update/mode", request,
          SensorThermostatUpdateModeResponse.class);
    if(response.isModeSuccess()){
      System.out.println("Thermostat Mode set successfully!");
    }
  }

  private static void setThermostatFanMode() {
    System.out.println("\nSetting Thermostat Fan Mode");
    SensorThermostatUpdateFanMode request = new SensorThermostatUpdateFanMode();
    request.setFanMode(SensorThermostatFanMode.On);
    SensorThermostatUpdateFanModeResponse response = client.put("/thermostats/" + THERMOSTAT_ID + "/update/fanmode",
          request, SensorThermostatUpdateFanModeResponse.class);
    if(response.isFanModeSuccess()){
      System.out.println("Thermostat Fan Mode set successfully!");
    }
  }

  private static void getSecuritySensors() {
    System.out.println("\nGetting security sensors");
    System.out.println("------------------------");
    InstallationSensorsSecurityResponse response = client.get("/installations/" + INSTALLATION_ID + "/sensors/security",
          InstallationSensorsSecurityResponse.class);
    for(SensorWithAreaName s : response.Sensors){
      System.out.println(s.AreaName + " " + s.Name + " " + s.Value + " " + s.Status);
    }
  }
}
```