D1.3

Initial Component, Architecture, and Intercommunication Design

WP1 – Distributed System Architecture

SIFIS-Home

Secure Interoperable Full-Stack Internet of Things for Smart Home

Due date of deliverable: 30/09/2021
Actual submission date: 30/09/2021

Responsible partner: FSEC
Editor: Marko Komssi;
E-mail address: marko.komssi@f-secure.com

29/09/2021
Version 1.2

Project co-funded by the European Commission within the Horizon 2020 Framework Programme		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

The SIFIS-Home Project is supported by funding under the Horizon 2020 Framework Program of the European Commission SU-ICT-02-2020 GA 952652

Authors: Riccardo Coppola (POL), Luca Ardito (POL), Andrea Saracino (CNR), Wisam Alabbasi (CNR), Domenico De Guglielmo (MIND), Marko Komssi (FSEC), Jukka Aittakumpu (RIO)

Approved by: Luca Ardito (POL), Göran Selander (ERI)

Revision History

Version	Date	Name	Partner	Section Affected Comments
0.1	19/05/2021	Defined ToC	FSEC, CNR, POL	All
0.2	30/06/2021	Defined High Level Architecture and Stakeholders	FSEC, CNR, POL, RIO, SEN, MIND, RISE, LUM	All
0.3	20/07/2021	Defined internal component architecture	FSEC, CNR, POL, RIO, SEN, MIND, RISE, CEN, LUM	Section 3
0.4	12/08/2021	Defined APIs and Workflows	FSEC, CNR, POL, RIO, SEN, MIND, RISE, ERI, CEN, LUM	Sections 4 and 5
1.0	30/08/2021	Ready for review	FSEC, CNR, POL	All
1.1	16/09/2021	Comments addressed and Second Internal Review	FSEC, CNR, POL	All
1.2	29/09/2021	Ready to Submit	FSEC, CNR, POL	All

Executive Summary

This deliverable reports the preliminary design of the SIFIS-Home architecture and the SIFIS-Home framework. The SIFIS-Home architecture is the logical representation of the SIFIS-Home aware devices and their interaction in the smart home settings. The SIFIS-Home framework is instead the software architecture installed in the devices of the SIFIS-home architecture, used to provide services and manage safety, security and privacy aspects in the smart home. The deliverable will present the SIFIS-Home framework at three levels of detail, discussing the functionality of each component and the intended interaction. A first set of APIs and workflows will also be introduced.

Table of contents

1 Introduction

To manage security, privacy and safety in the smart home environment, the SIFIS-Home project deploys a software framework intended to run on the smart home devices, which can be customized by installing third party applications. According to the requirements elicited in the deliverables D1.1 and D1.2, the SIFIS-Home framework must be resilient, which implies the replication of functionalities and fault tolerant communications. Thus, the core of SIFIS-Home is based on a peer-to-peer (P2P) architecture, still a smart-home is a heterogeneous environment, where some devices that can be customized interact with a set of devices with limited possibilities of customization and functionalities. To represent such heterogeneity and the interaction between the devices and users, we also define the SIFIS-Home architecture, defining all the actors and their interactions.

In this deliverable we report at first the description of the SIFIS-Home architecture, also by presenting the concept of the Smart Home cyber-perimeter and some background information on the DHT model that will be used to implement the SIFIS-Home communication protocol. Afterward, we will describe in detail the preliminary architecture of the SIFIS-Home framework, describing at first a high-level view of the architecture, followed by a detailed view of the components and their interactions. Thus, we will describe in details all the architectural components. Furthermore, the deliverable will report examples of the SIFIS-Home APIs. The SIFIS-Home framework has been designed following a top-down approach, as discussed in D1.1. The design pattern followed is the microservices design pattern, which favours both flexibility and modularity.

Both the SIFIS-Home framework and the SIFIS-Home architecture are input for the activities of WP5, which will implement and deploy the first testbed of the SIFIS-Home architecture. Feedbacks from the implementation and deployment of the SIFIS-Home framework and architecture will be reported in D1.4, together with a finalized list of the implemented APIs.

The deliverable is organized as follows. Section 2 of this deliverable introduces the SIFIS-Home architecture, while Section 3 introduces the SIFIS-Home framework. Section 4 presents the initial set of APIs with examples. Section 5 introduces a number of operative workflows of key SIFIS-Home operations and Section 6 concludes the deliverable.

2 SIFIS-Home Architecture

The *SIFIS Home architecture* is the representation of the devices and actors interacting with the *SIFIS-Home Framework.* More in details the architecture depicts at a logical level the devices that are present in a SIFIS-Home aware smart home, their interconnection and interactions.

2.1 *The Smart Home Cyber-Perimeter*

Protecting the Smart Home and its users from unintended disclosure of sensitive information requires defining a logical distinction between the outside and inside of the Smart Home. A concept that we are defining in the scope of SIFIS-Home, which will be relevant to define the SIFIS-Home architecture and for defining data privacy policy, is the Smart Home *Cyber-Perimeter*. The concept is illustrated in Figure 2.

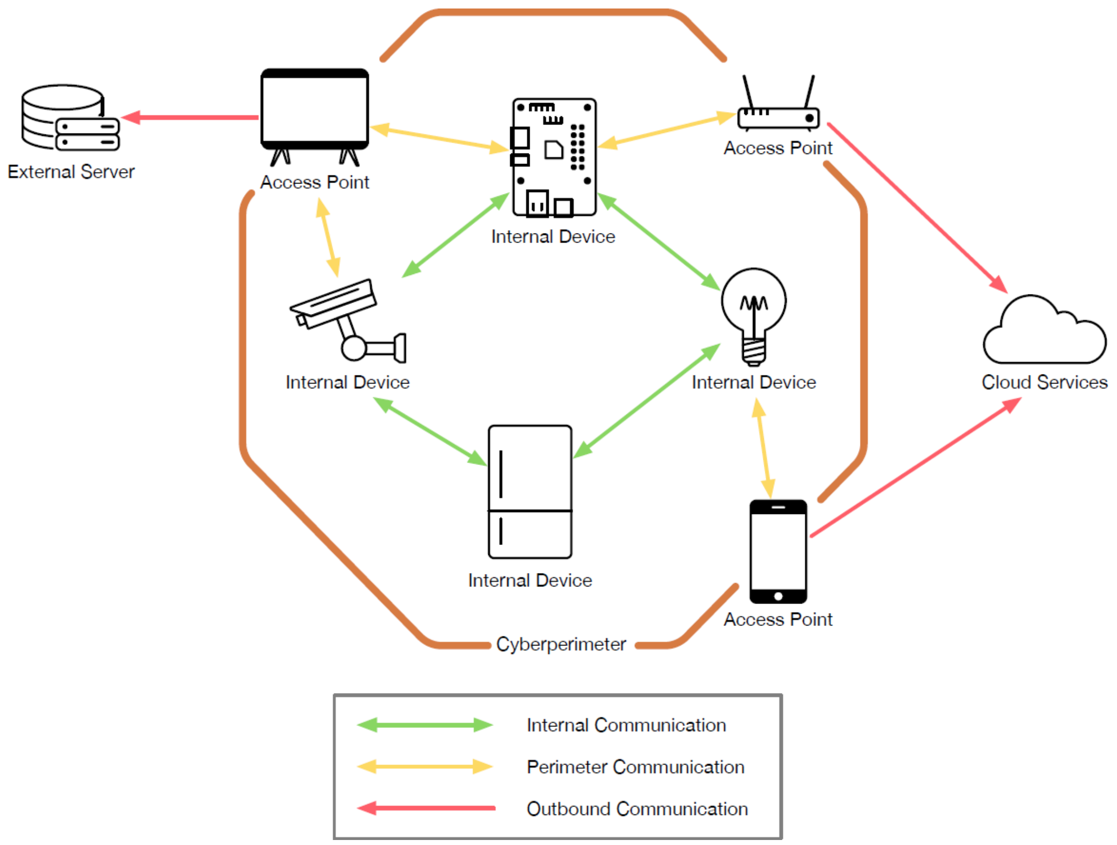

Figure 1: Concept of Smart Home Cyber-Perimeter

The separation between these two domains is what we define as the Smart Home Cyber-Perimeter. In particular, the cyber-perimeter is a logical barrier, which identifies the elements (i.e., devices and application) of the Smart Home, which can be used to receive and send data toward entities that are not part of the Smart Home (i.e., external entities). For example, a smart speaker with an embedded voice assistant exploiting a cloud service to process voice commands is both an access (since it receives instructions from the cloud) and exit (since it sends data to the cloud) point of the Smart Home cyber-perimeter. Though even inside the Smart Home cyber-perimeter, there might be specific privacy constraints, a violation of the Smart Home privacy is performed when sensitive information leaves the Smart Home cyber-perimeter.

We base this definition of privacy violation on the worst possible case: once a piece of data leaves the Smart Home cyber-perimeter, the users potentially lose control of that data piece, which can thus be re-used and redistributed indefinitely. We derive that data can be safely exchanged among devices and services inside the Smart Home cyber-perimeter. The rationale behind this distinction is in the trade-off between ensured privacy and needed accuracy for data analysis algorithms, which are essential to provide smart services. Thus, inside the Smart Home cyber-perimeter, data can be exchanged and processed without applying privacy-enhancing techniques to maximize data analysis accuracy, providing the best service level. In fact, inside the cyber-perimeter, data cannot be shared with external entities and remains only available to the data owner, i.e., the Smart Home residents. The validity of this assumption depends on the devices and applications behaviour on the cyber-perimeter, which act as access points to the Smart Home. Their behaviours should be monitored and certified, when possible, to ensure that when they have access to sensitive information, they are not going to send them outside out of the cyber-perimeter. When this cannot be ensured, or it is known that a data piece is bound to leave the perimeter, it should be processed through specific privacy-enhancing techniques to avoid disclosing sensitive information.

2.2 *Components and Actors of the SIFIS-Home architecture*

The main components of the SIFIS-Home architecture are the following:

- *Smart Devices:* These devices are characterized by a relatively good computational capability; they are based on general purpose computational hardware and their functionalities are managed through an Operative System (OS). Smart devices can be customized by installing third party software and have the capability of directly communicating among them, autonomously exchanging information. This intercommunication enables a Peer-to-Peer (P2P) logical model, which is easily represented by means of a distributed hash table (DHT). Example of Smart Devices are Smart TVs, Smart Refrigerators, Laptops/Desktops, Family Hubs.
    - *Internet Connected Smart Devices*: This is a subset of the Smart Devices which are characterized by the presence of a network interface which enable connectivity outside of the smart home. Example of these devices are smart routers (connected to optical fiber or DSL), smartphones and tablets (with 4G/5G connectivity). Internet Connected Smart Devices are in general on the Smart Home cyber-perimeter, as they have a direct interface to send network traffic out of the cyber perimeter.
- *Not So Smart Devices (NSSD):* This set is made by those devices which present smart functionalities and present a network interface, yet they still have very limited computational power, and only present a firmware instead of a fully-fledged OS. For this reason, the NSSDs cannot be customized by installing third party software or applications. Examples of NSSDs are smart sensors, smart cameras, smart lights, smart speakers, smart locks. NSSDs can have Internet capabilities, but generally the Internet connection will be forced to happen through their responsible Smart Devices.

Figure 2 illustrates the communication and interaction between components and actors of SIFIS-Home architecture. The Smart Devices are at the core of the SIFIS-Home architecture, since they will be the only devices installing the SIFIS-Home framework. Thus, the minimal *instance* of a SIFIS-Home architecture is the one made by a single smart device. When more than one smart device is added to a SIFIS-Home architecture instance, we consider them as logically interconnected. The interconnection and network communication are handled at application level by a DHT protocol. In SIFIS-Home we are using the Kademlia protocol as DHT, which is detailed in the next subsection. The DHT abstracts from the actual network implementation, i.e., the smart devices can be physically connected to the same Wi-Fi network, be connected to different Wi-Fi hot spots, or use ad hoc Wi-Fi routing protocols such as AODV [Perkins, 1999]. The SIFIS-Home architecture is, thus, oblivious of the actual network technology and topology, which makes it adaptable to any network configuration. The DHT allows to view the SIFIS-Home architecture from outside as a single entity, accepting requests from external services and applications as if it was a monolithic server. Following the DHT protocol, Smart Devices handle in a distributed way computational task, data storage, message forwarding, attribute retrieval and data analysis, implementing thus the services of the SIFIS-Home framework.

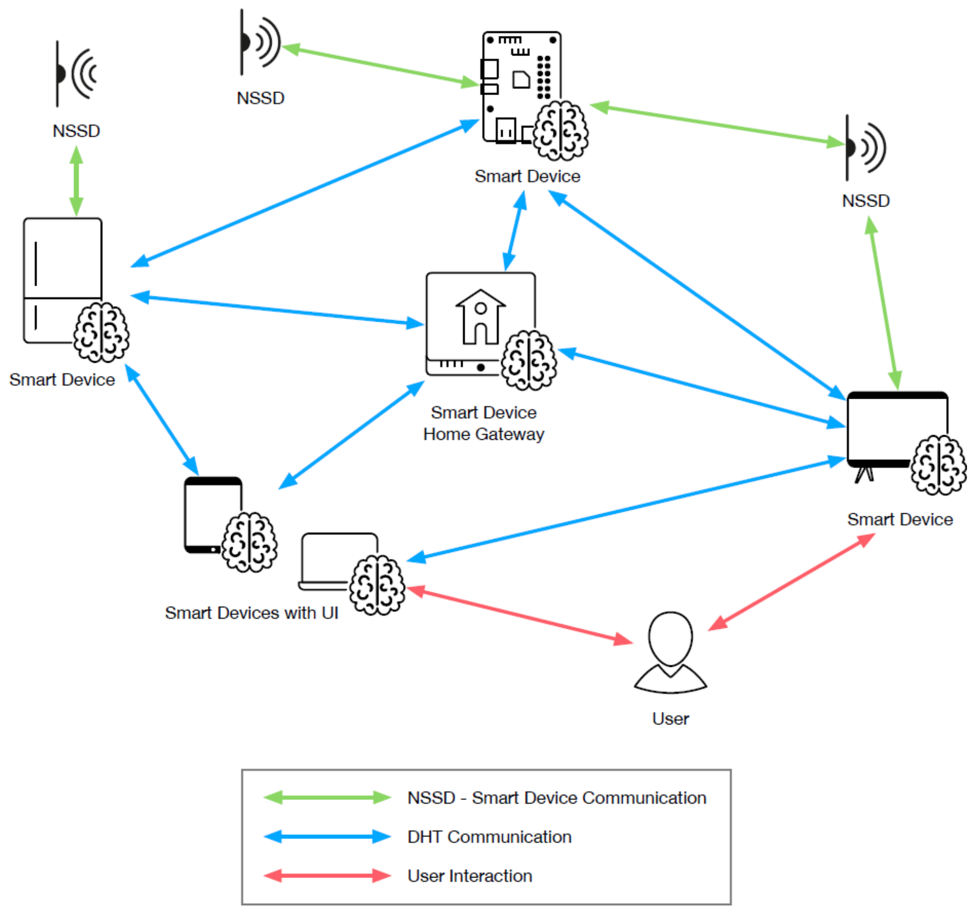

Figure 2: Communication and interaction between components and actors of SIFIS-Home architecture

Whilst smart devices manage framework operations and message exchange, the NSSDs provide the SIFIS-Home architecture with additional capabilities to read physical measures through sensing mechanisms, and to actively interact with the physical world through actuators. NSSDs are thus considered as peripheral devices under the direct control of one or more smart devices. When registered, a smart device connects via Wi-Fi, Bluetooth, or 802.15.4 protocols to one or more smart devices (number and topology will be selected by the SIFIS-Home administrator) and will only accept messages and commands from these devices. Each NSSD offers thus a set of APIs to receive queries on sensed data, or to perform operations, which are issued by the responsible smart devices. A very preliminary implementation of a P2P architecture of Smart Devices in a smart home environment has been presented in [La Marra et. al, 2017].

The actors we have defined for the SIFIS-Home architecture are the following:

- *SIFIS-Home Administrator*: The administrator is a human user who is the owner of an instance of the SIFIS-Home architecture. Generally, this person coincides with the smart home owner. The administrator defines usage policies, assign roles to other tenants or guests of the home, sets up restrictions, application preferences, smart home routines and possible configuration. Since SIFIS-Home is human centred, the administrator is considered the highest authority, who can supersede at any time the SIFIS-Home framework decisions. The administrator can install third party applications on the framework and remove or change the authorizations to applications installed by other users.
- *SIFIS-Home Tenant*: The SIFIS-Home tenant is the standard user of the smart home. The tenant is a resident of the smart home and the main target of the SIFIS-Home system. This user

can set up preferences and configurations which should not be in contrast with those set up by the administrator. The tenant can give command to the smart home devices via voice command or via a user interface on devices like smartphone, tablet or PC, and he can install applications on the smart devices which are not in contrast with the administrator policies. The user can remove the applications he installed. Each tenant will have his or her own *profile*, describing usage preferences and configuration, which cannot be modified by other tenants. Only the administrator can modify the profile of another tenant. Generally, the administrator is also a tenant. Moreover, each tenant can define and configure one or more *usage modes* (e.g., “do not disturb”), which can be switched several times during the usage of the platform.

- *SIFIS-Home Maintainer:* The maintainer is an entity external to the smart home which is trusted by the administrator to correctly configure the smart home security, privacy and safety policies. In a commercial model, the maintainer can be the provider of the SIFIS-Home framework, holding the same right of the administrator. The maintainer can also install applications for handling specific management services, and they can remove applications installed from any user. The maintainer is not a mandatory actor, but when present has to be considered a trusted party.
- *SIFIS-Home Tenant with restrictions*: This user is a smart home tenant with restrictions on the functionalities he can use. The restrictions are needed to avoid possible damage or hazard to the tenant or to the home devices. Typical example of these tenants are children, who can still interact with the smart home and beneficiate from services, still they cannot use dangerous functionalities (e.g., turning on a stove). As the other tenants, they can install applications, however the set of available applications can be limited according to specific safety policies. As the other tenants, they can have a personal profile and their own usage policies. However, these policies are generally set by the administrator.
- *Guest*: A guest is a smart home user who is not resident in that smart home but is accessing and using the premises for a limited amount of time, upon authorization of the administrator or another tenant. Guests do not have profiles, nor they can set up policies, still they can use a subset of the house functionality. The services and functionality available are set by the administrator for all guests. Guests cannot install applications.
- *External Operator*: The external operator could be a technician, a plumber, gardener, or house maid, accessing the house for a limited amount of time, with the authorization of a tenant. Differently from guests, the operators will not use the smart home functionalities, still they might have specific policies needed to protect their privacy.

The SIFIS-Home actors are the entities defined and recognized by the SIFIS-Home framework. Thus, they can be used as subjects for the usage, security, privacy and safety policies,

2.3 *The Distributed Hash Table Kademlia*

The SIFIS-Home architecture is based on a peer-to-peer (P2P) approach exploiting a Distributed Hash Table (DHT) indexing scheme to organize the smart home network nodes. The DHT permits a decentralized distributed system that provides a lookup service similar to a hash table. The (key, value) pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key. Keys are unique identifiers that map to particular values, which can be anything, addresses to documents, or arbitrary data.

Kademlia advantages

The Distributed Hash Table (DHT) adopted in the SIFIS-Home architecture is based on Kademlia [Maymounkov et al. 2002]. The protocol allows to store data and provide communication between nodes. Kademlia provides many desirable features that any other DHT does not simultaneously offer.

The main advantages in using Kademlia are described in the following:

- Kademlia minimizes the number of inter-node introduction messages.
- Configuration information, such as nodes on the network and neighbouring nodes, spreads automatically as a side effect of key lookups.
- Nodes are knowledgeable of other nodes. This allows routing queries through low latency paths.
- Kademlia uses parallel and asynchronous queries, which avoid timeout delays from failed nodes.

Kademlia nodes and data

Kademlia nodes are represented in the form of a binary tree, where nodes are the leaves of the binary tree, as shown in Figure 3. The NodeId of each node can be found by tracing the bit value of the edges to that node, and the highlighted node in the tree would have a NodeId 0011. From the perspective of a single node, the tree is divided into subtrees. From the node's location in the tree traversing downwards, the subtrees are successive lower subtrees that don't contain that node. The highest subtree consists of half of the binary tree not containing the node. The next one consists of half of the remaining tree not containing the node. The Kademlia protocol ensures that every node knows at least one node in each of its subtrees if that subtree includes a node.

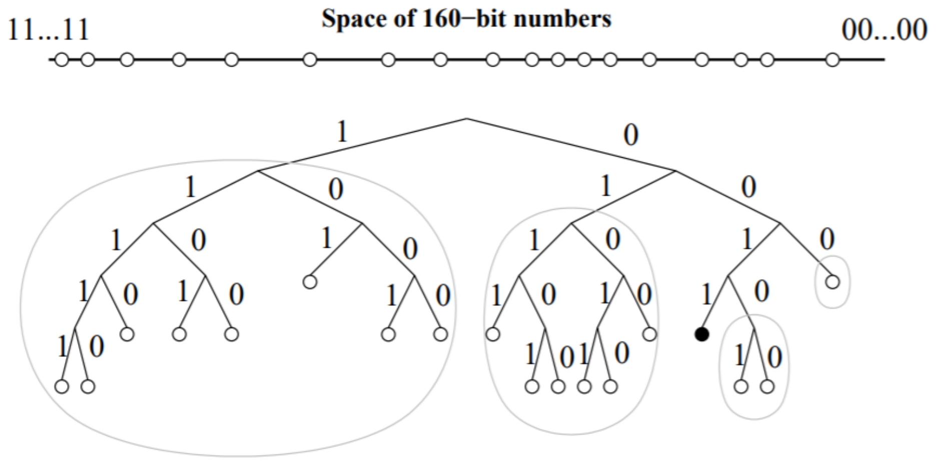

Figure 3: Illustration of Kademlia nodes in the form of a binary tree

Kademlia uses keys to identify both nodes (NodeId) and data on the Kademlia network. The keys are opaque, 160-bit quantities. Since Kademlia stores content in (key, value) pairs, both data and node on the DHT are uniquely identified by a key in the 160-bit key-space.

To decide which node a (key, value) pair should be stored at, Kademlia uses the notion of distance between two identifiers. Given two 160-bit identifiers, x, and y, Kademlia defines the distance between them as their bitwise eXclusive OR (XOR) interpreted as an integer: $d(x, y) = x \oplus y$. XOR captures the notion of distance implicit in the binary tree sketch of the system. In a fully populated binary tree of 160-bit IDs, the magnitude of the distance between two IDs is the height of the smallest subtree containing both. When a tree is not fully populated, the closest leaf to an ID x is the leaf whose ID shares the longest common prefix to x.

In Kademlia, nodes store contact information about each other to route query messages: for each $0 <$ i

< 160, every node keeps a list of triplets (IP Address, UDP Port, NodeId), for nodes of the distance between 2i and 2i + 1 from itself, these lists are called k-buckets. Each Kademlia node also has a Routing Table: a binary tree whose leaves are k-buckets. Moreover, Kademlia uses a replication parameter k which specifies how many nodes a data should be replicated and the size of a node's routing table.

Protocol messages
The Kademlia protocol consists of four remote procedure calls (RPCs):

- PING: probes a node to see if it is online.
- STORE: instructs a node to store a (key, value) pair for later retrieval.
- FIND NODE: takes a 160-bit key as an argument. The recipient of the RPC returns information about the k nodes closest to the target id.
- FIND VALUE: behaves like FIND NODE returning the k nodes closest to the target Identifier with one exception: if RPC recipient has received a
- STORE for the given key returns the stored value.

3 SIFIS-Home Framework

The design of the SIFIS-Home framework has been based on the *microservices* design pattern. In fact, to design the SIFIS-Home framework we have taken in consideration the requirements described in D1.1 and D1.2. To this end, we followed a top-down approach to derive the software architecture of the SIFIS-Home framework. By using microservices is possible to define a modular architecture where each component offers a specific set of functionalities, which can be invoked either by other architectural components, or externally. In the following we provide a view of the SIFIS-Home architecture as a whole and we will then analyze the functionalities of each subcomponent.

The high-level architecture of the SIFIS-Home framework is reported in Figure 4, and comprises six principal building blocks:

- **SIFIS-Home API Gateway**: this component includes a set of high-level APIs and has the purpose of interfacing the SIFIS-Home framework with the external components. The SIFIS-Home APIs are also used as interface toward *Web of Things (WoT)* [WoT, 2020]and *Fiware* [FIWARE, 2021]standards for representation of resources, mapping the SIFIS-Home functionalities with the already existing APIs for smart home devices.
- **User Interface:** this component is in charge of providing the Graphical User Interfaces to all different typologies of users that will utilize the SIFIS-Home Framework.
- **Secure Lifecycle Manager**: this component handles the standard workflow of the SIFIS-Home framework. It is hence responsible of the bootstrap of the framework, and of the registration/deregistration of the devices. The component can notify events for the registered devices to handle them, and can also force operations (e.g., a device can be forced to leave the SIFIS-Home network). The secure lifecycle manager mainly receives inputs from the other hardware and software components of the SIFIS-Home network, to which it exposes specific APIs (e.g., device registration, application installation, request management).
- **Secure Communication Layer**: this component is responsible of the communication between the devices of the SIFIS-Home architecture. It implements the distributed Hashtable of the network, and transforms the registration requests from new devices into actual updates to the distributed Hashtable architecture. The secure communication layer handles all means of

communication between devices inside the SIFIS-Home network (either SD or NSSD). The usage of the DHT is the primary mean of retrieving any specific kind of information in the SIFIS-Home architecture. The component also implements aspects of the communication that are related to security, e.g., security protocols at application and network layer, implementation of security libraries, functionality replication.

- **Application toolboxes**: this component collects related and interconnected sub-components that are all services inside the SIFIS-Home infrastructure. Example components are: the privacy-preserving data analysis toolbox, the provided end-point for the access to the SIFIS-Home marketplace, the configuration panel for the SIFIS-Home system, the mechanisms to handle intrusion detection and to perform policy enforcement. All those instruments are at lower level than the previously described components and are supposed to be invoked by the others in the SIFIS-Home infrastructure.
- **Proactive security management layer**: This component is responsible of maintaining the security aspects of the SIFIS-Home infrastructure in advance. It includes a series of monitors that are used to verify the security of the APIs of the system, of the distributed hashtable, of the network and system calls. It includes also instruments for manifest checking, distributed trust and self-healing.

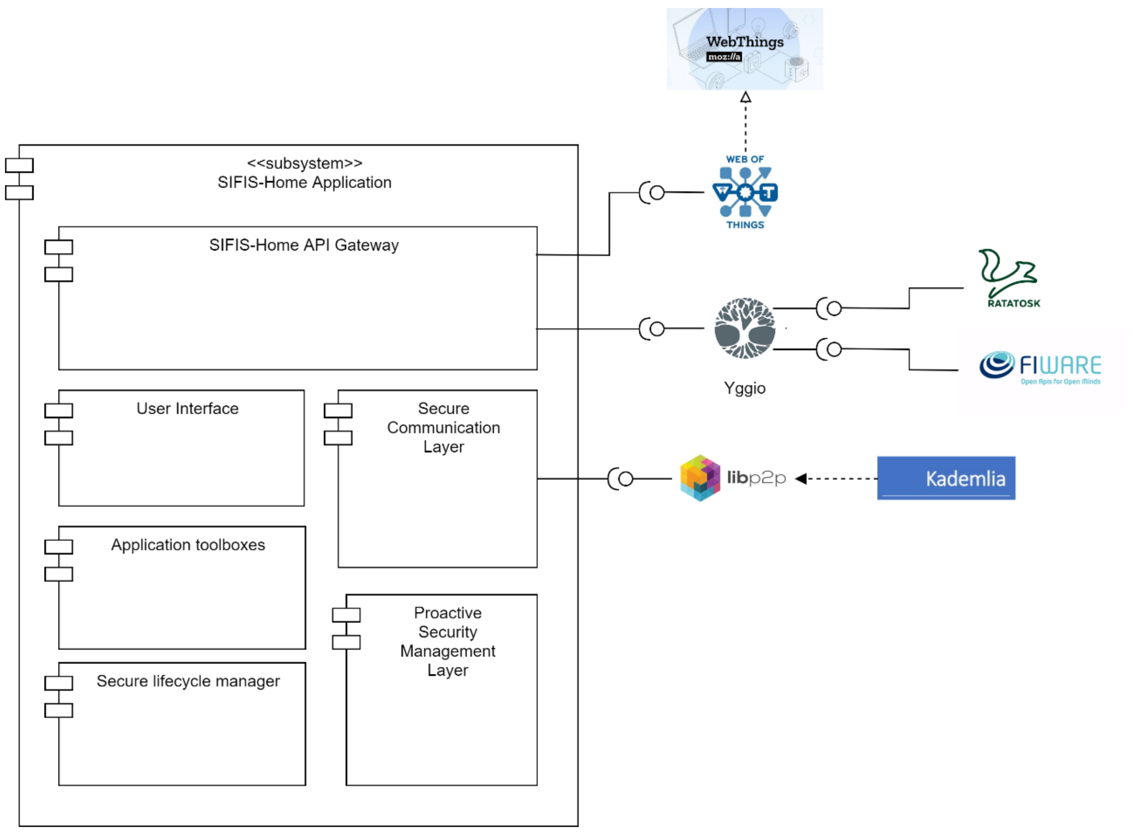

Figure 4: High-level architecture of the SIFIS-Home framework

Figure 5 provides a more detailed architecture and illustrates the main components of each of the six building blocks. The following subsections expand the previous component diagrams by describing the internal components in which each of the six high-level modules is divided. Higher-granularity architectural details are provided for more complex sub-components that can be decomposed furtherly.

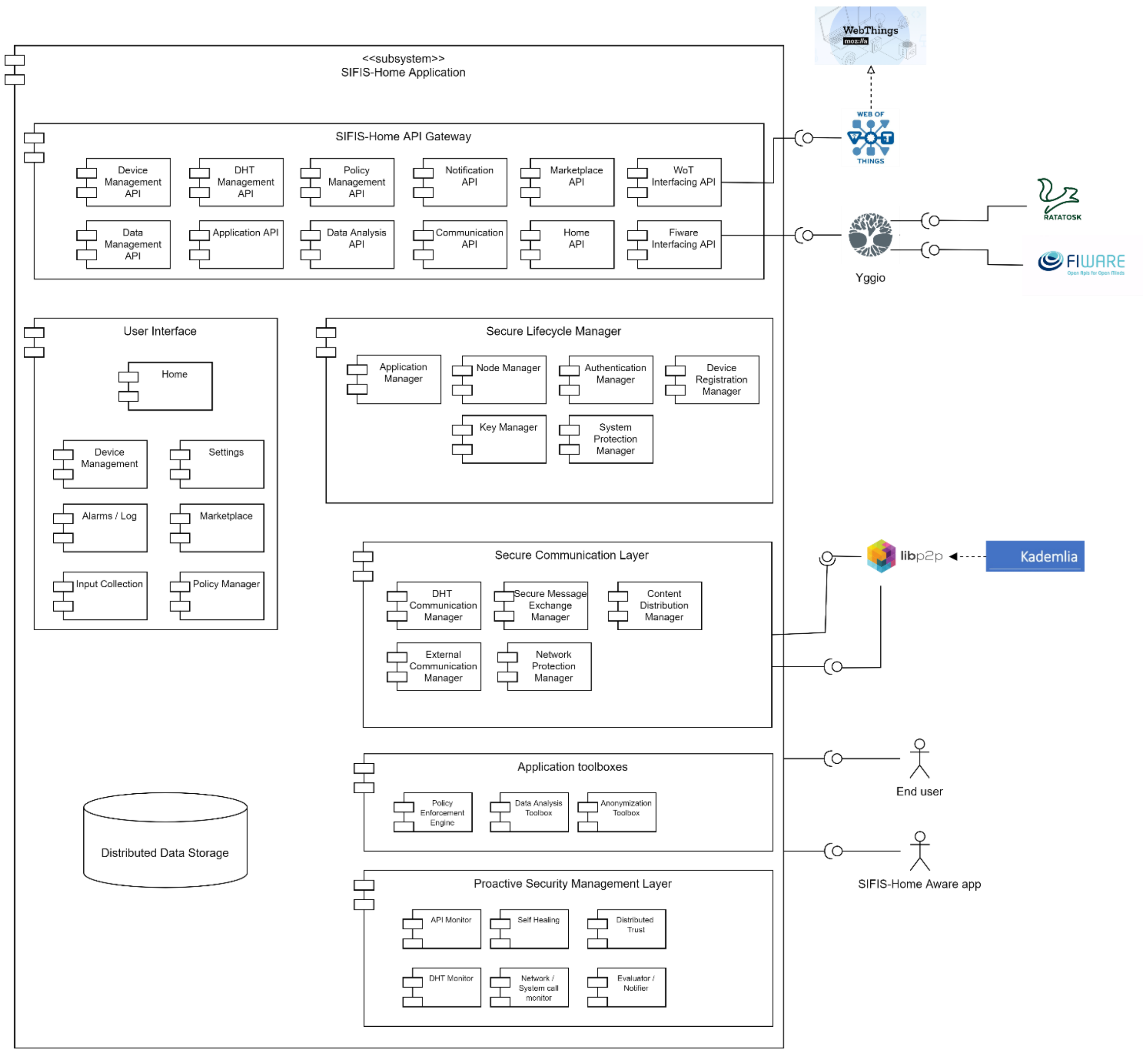

Figure 5: Components of each six building blocks of the SIFIS-Home framework

3.1 *SIFIS-Home API Gateway*

The SIFIS-Home API gateway is the logical endpoint exposing the services of the SIFIS-Home framework. This component collects the interfaces to the functionalities offered by the SIFIS-Home framework components, both as stand-alone and in cooperation, for complex operations.

Figure 6 introduces the API groups in SIFIS-Home API Gateway that is used to abstract the various functionalities offered by the SIFIS-Home framework. The APIs can be either public, which means they can be invoked also by authorized components outside of the SIFIS-Home framework (e.g., the maintainer or an external device), or private, meaning they can be invoked only by components of the SIFIS-Home framework.

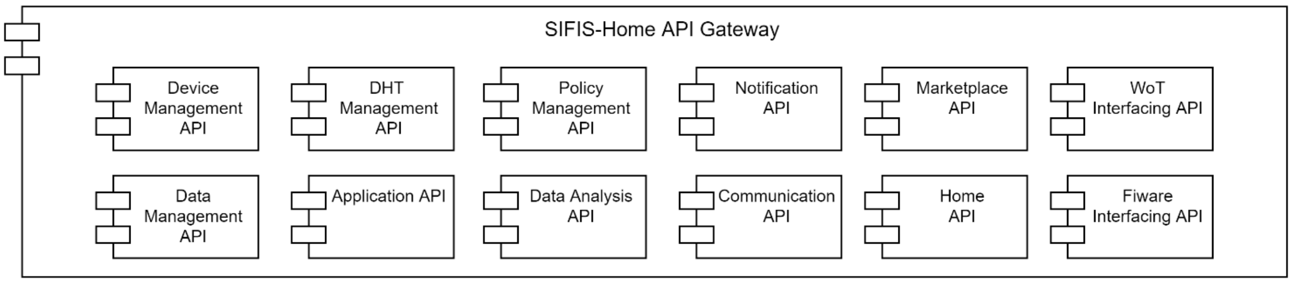

Figure 6: The SIFIS-Home API gateway

The SIFIS-Home gateway also acts as communication and integration point between the SIFIS-Home framework and existing framework for representation of IoT resources and capabilities. Currently, we are considering two of these frameworks, namely FIWARE and Web of Things. We will detail in the following the interaction with the two frameworks. We will consider in the meanwhile possibility of integrations with other developing frameworks such as the Zig Bee Alliance's Matter, which is under definition and the OpenHAB framework, following suggestions from the advisory board.

3.1.1 Interaction with Ratatosk, the FIWARE NGSI v2 compliant Context Broker

FIWARE NGSI [FIWARE, 2021] is an abstraction software framework used to simplify the interaction among different IoT devices. In the SIFIS-Home project we are building upon FIWARE and WoT for high level representation of services and resources. In particular, for FIWARE we rely on the *Ratatosk* implementation which is described in the following and its integration with the Yggio [YGGIO, 2021] management system.

3.1.1.1 Background

Ratatosk in SIFIS Home is written in modern node.js Javascript instead to make it easy to understand and re-useable. Ratatosk also enables an entirely new security level, which is a non-negotiable, mandatory requirement for a SIFIS Home context broker by having an architecture that enables developers to add a plug-in to enforce access and identity management. When accessing Ratatosk with enabled security, one must provide an authorized access token in the request header, otherwise the request will get rejected. Further while Ratatosk being completely stateless to enable horizontal performance scaling of the SIFIS Home system to run several Ratatosk instances in parallel it is also fully supports multi tenancy to ensure that entities, attributes and subscriptions that belongs to one user is completely invisible and inaccessible for all other users that log into the system.

Like the FIWARE Orion context broker, Ratatosk can create, manage, and subscribe to context elements, called entities, that can be devices or any type of resources that can be described in a structured way and assigned attributes with values. Since SIFIS Home can connect to many different types of entities, Ratatosk will then grant access to all entities, and only those entities, that a provided user access token authorizes.

Except being used to maintain the state of the resources, entities, of the different authorized users in a SIFIS Home system Ratatosk context broker will also be used to maintain state and information for many internal services that are needed used to fulfil the requirement specification specified by D1.2. Further Ratatosk will implement parts of the SIFIS Home API the SIFIS Home applications, like device management, will be dependent on to visualize, interact and manage the SIFIS Home system.

3.1.1.2 Ratatosk Components

These are the main components used to implement Ratatosk and all are implemented to be completely stateless to enable horizontal performance scaling

- NGSI Client API: This component implements the NGSI v2 API end point including all syntax check and validation of input data. It is divided into 4 main categories:
    - Entities
    - Bulk
    - Registrations
    - Subscriptions

- NGSI Routes: Exposes the middleware that implements NGSI v2. This includes
    - NGSI subscription manager: Manage subscriptions
    - NGSI global registration manager: Manage registration of subscriptions
    - NGSI global updates manager: Manage global updates
    - NGSI v2 routes: Exposes the routes for the end points for entities, bulk and subscriptions.
- NGSI Server: The server that listens to all incoming events and decides what to do with them mainly via NGSI Routes. It is the main entry point to Ratatosk.
- NGSI Access Manager: Enables developers to add a security plugin that handles access and identity management via a call back.
    - This is a key difference between the standard FIWARE Orion Context broker the SIFIS Home Ratatosk context broker. Ratatosk is completely secure and will not allow access to any resource unless an authorized token is provided.
- NGSI Storage: The database storage. The API end points here turn into either read or write from the central database that stores all information. It implements 4 different kinds of end points:
    - Entities
    - Bulk
    - Notifications
    - Subscriptions
- NGSI Utilities: A set of tools to support the services listed about.

3.1.1.3 Ratatosk Architecture

The diagram in Figure 7 shows the detailed architecture of Ratatosk. The database storage and the security plug in for access and identity management is outside of the actual Ratatosk code. NGSI subscribers that subscribes to context data changes handled by Ratatosk will be outside of the architecture diagram.

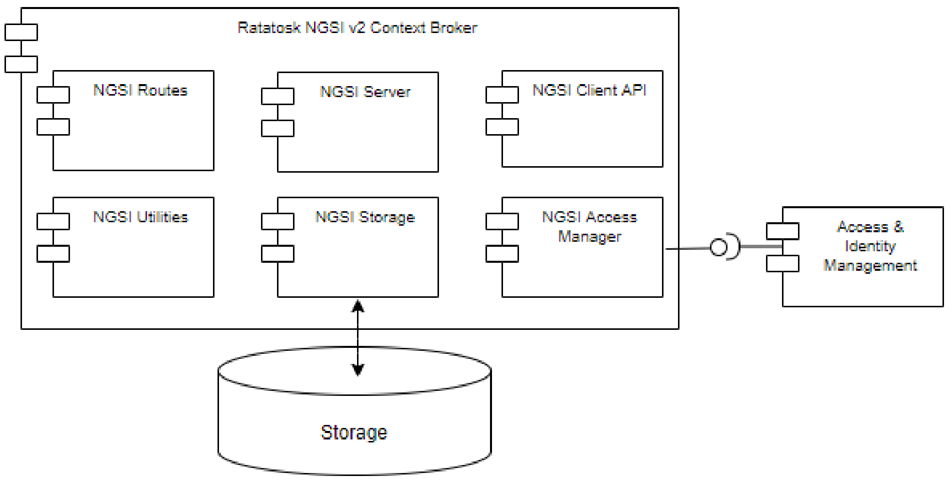

Figure 7: Detailed architecture of Ratatosk

3.1.1.4 FIWARE NGSI v2 API

Ratatosk implements the FIWARE NGSI v2 API specification as specified in the following Swagger: https://swagger.lab.fiware.org but with the very important add on that an authorization token is required in the header to get access to a user's available resources.

This is a GIT Bash Curl example for reference that retrieves all available entities that is visible for the authorized users access token:

```
$ curl -X GET https://..../v2/entities -H "accept: application/json" -H "Authorization: Bearer
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJYSEpqNlRVWXo4OHBBeXpQZDY
yX2RVTnJ2NU9JU2lyb2hGb2pOTXFURDhRIn0.eyJleHAiOjE2MzA5NjA5MzIsImlhdCI6MTY
zMDkzOTMzMiwiYXV0aF90aW1lIjowLCJqdGkiOiJjYzMxOGM5ZC04YjA2LTQ2ZTgtYTI3N
y04ZGZhYWJjZTI4NjEiLCJpc3MiOiJodHRwczovL2tleWNsb2FrLnlnZ2lvMy1iZXRhLnNlbnN
hdGl2ZS5uZXQvYXV0aC9yZWFsbXMveWdnaW8iLCJhdWQiOiJ5Z2dpby1zZXJ2aWNlcyIsIn
N1YiI6ImQ2NTg5YTE5LTc1N2MtNDNkMC04OTE2LWJmN2M1OTI0NTlmMCIsInR5cCI6Ik
lEIiwiYXpwIjoieWdnaW8tc2VydmljZXMiLCJzZXNzaW9uX3N0YXRlIjoiN2Y0Y2FhMGUtNz
Y1My00ZGM3LTg2NTItZjEzOGY0MzFhNWU4IiwiYXRfaGFzaCI6IlhVaXU1cDEyaUl6Y1Zs
blV5eVljZHciLCJhY3IiOiIxIiwiZW1haWxfdmVyaWZpZWQiOnRydWUsInByZWZlcnJlZF91c2
VybmFtZSI6InlnZ2lvLXRyYWluaW5nIiwibG9jYWxlIjoic3YifQ.DJ2UKa3prvmlpviKhgcDgpM
sJC-etB2O8tZQP-
zjCw6kxXkYTp4jUct98KKFBeeN5YkuaygzBHGBztuvkFb6sZO0FjzgPGD0K00B4asrJepkDJK
YapR6puWRMjkFSzkeMWtwCTyMswXie-
TolFzO4OiLhvf0_Vld1hTp2L9AGv4VWCTNNJRf6cFBycrITEDoDVZYsWZl7CjEPXBr8XUp
wEJp63nejzzEhJoKUDN279mD8gaw5PM8Kwhn529t_QbvuPKb4yOSPtYrdEizc5tTOcoZSBAvr
Ecca8DTdA8rESD1-qmocbiNLGakAeD3rgD2Ewv-oGBA5AL7b4VGiT_AdA"
```

The NGSI v2 Rest API end points that are available:

- Entry point
  - GET /v2
- Entities
  - GET /v2/entities - List entities
  - POST /v2/entities - Create entity
  - GET /v2/entities/{entityId} - Retrieve entity

  - DELETE /v2/entities/{entityId} - Delete entity
  - GET /v2/entities/{entityId}/attr - Retrieve entity attributes
  - PUT/v2/entities/{entityId}/attr - Replace all entity attributes
  - POST /v2/entities/{entityId}/attr - Update or append all entity attributes
  - PATCH /v2/entities/{entityId}/attr - Update existing entity attributes
- Attributes
  - GET /v2/entities/{entityId}/attr/{attrName} - Get attribute data
  - POST /v2/entities/{entityId}/attr/{attrName} - Update attribute data
  - DELETE /v2/entities/{entityId}/attr/{attrName} - Remove a single attribute
- Attribute value
  - GET /v2/entities/{entityId}/attr/{attrName}/value - Get attribute value
  - POST /v2/entities/{entityId}/attr/{attrName}/value - Update attribute value
- Types
  - GET /v2/types - List entity types
  - GET /v2/types/{entityType} - Retrive entity type
- Subscriptions
  - GET /v2/subscriptions - List subscriptions
  - POST /v2/subscriptions - Create subscription
  - GET /v2/subscriptions/{subscriptionId} - Retrieve subscription
  - DELETE /v2/subscriptions/{subscriptionId} - Remove subscription
  - PATCH/v2/subscriptions/{subscriptionId} - Update subscription
- Registrations
  - GET /v2/registrations - List registrations
  - POST /v2/registrations - Create registrations
  - GET /v2/registrations/{registrationId} - Retrieve registration
  - DELETE /v2/registrations/{registrationId} - Remove registration
  - PATCH /v2/registrations/{registrationId} - Update registration
- Batch operations
  - POST/v2/op/update - Update
  - POST /v2/op/query - Query
  - POST /v2/op/notify - Notify

All of the end points have several available parameters to be able to query the right entity or subscription. Some of the most noteworthy ones are:

- type: The type of the entity, this could be anything from a device, an IoT node, a room, a shelf, some setting, a car, a shop, another SIFIS Home device, a smart device, a not so smart device, etc. Example:
  - “type": "Room",
- attr: The attribute list that belongs to and describe the entity. Example:
  - "temperature": { "value": 22.3, "type": "Number" },
  - "lux": { "value": 1200, "type": "Number" },
  - "location": { "value": "40.3745726, 3.1764575", "type": "geo:point",}
  - "setting": { "Installed Apps": "34", "type": "number",}

- metaData: Extra information that can be used to describe an attribute. Example:
  - "location": { " value": "40.3745726, 3.1764575",, "type": "geo:point", "metadata": { "crs": { "value": "WGS84", "type": "Text" }} }
- q: A query that can be used to retrieve all entities that fulfill a certain expression. Example
  - /v2/entities?q=humidity>60
- mq: A query that can be used to retrieve attributes those metaData fulfill a certain expression. Example
  - /v2/entities?mq=Location=”Building1”

3.1.1.5 Limitations of the NGSI v2 API

Some key limitation in the FIWARE NGSI v2 specification that is used in current version of Ratatosk is the lack of API support to manage users, login, intrusion detection, application life cycles (Market Place), time series data as well as streaming data. All of them like are vital requirement of SIFIS Home and needs to be handled outside of the current version of Ratatosk. Ratatosk builds around the key limitation in NGSI v2’s lack of user support by adding the security plug in for access token authentication but other APIs in SIFIS Home are required to log into the system and get the access token in the first place.

Another limitation of current version of Ratatosk is that it can only handle a flat structure of context data so dependencies between entities cannot be managed well. Within SIFIS Home we are planning to add support for NGSI LD – Linked Data in Ratatosk which will significantly expand the usability of Ratatosk in a SIFIS Home system by enabling entities to connect to each similar to a network. With linked data support Ratatosk will be able to describe a hierarchy of entities and how they relate to each other which will enable the full SIFIS Home vision of different type of SIFIS Home compatible devices connecting to and be aware of each other.

3.1.2 Interaction with WebThings

WebThings enables direct control of smart home devices over the web by giving them URLs, making them discoverable and linkable, also defining a standard data model and APIs to make the devices interoperable and to exchange data between devices and systems. WebThings implementation includes two main items:

- WebThings Gateway: allows the user to monitor and manage smart home devices and infrastructure over the web, it has also a rules engine to automate some functiones based on a predefined set of rules.

- WebThings Framework: consists of software components developed to directly interact with the WebThings API and use smart home services.

The SIFIS-Home developers APIs build upon WebThings model, which is used to abstract from the specific producer-based implementation of functionalities used to provide generic services, such as “Switch on Light”, “Open Lock”, “Increase Temperature”, etc. Following the Web of Things terminology, we can name these services “Capabilities”. The Capabilities help developers of third-party applications to provide applications able to invoke these generic services, without having to be worried about the actual implementation which in general is device specific. To clarify, let us suppose, for example, that two device manufacturers provide for their device “Refrigerator”, two different implementations of the “lowerFridgeTemp()” API, to decrease the current temperature in the refrigerator by 1 °C. To offer this API to third party developers, not having to foresee two distinct invocations, one for Manufacturer 1 and one for Manufacturer 2, the manufacturers describe the API as a capability “lowerFridgeTemp()”, exposed by SIFIS-Home. In such a way, the developer shall simply invoke the capability and the actual implementation, or the specific refrigerator present in a home, are fully transparent to him.

3.1.2.1 WebThings Architecture

The diagram in next Figure shows the architecture of WebThings that is is mainly composed of:

- **Things:** a physical or virtual entity in the smart home environment abstracted and integrated in the SIFIS-Home architecture. Things are thus representation of a resource in a IoT environment offering services and functionalities.
- **Thing Description:** a structured description of general metadata, domain-specific metadata, supported Protocol Bindings and links. These metadata are self-descriptive, so that consumers are able to identify Thing capabilities and how to use them based on the affordances concept.
- **Consumers:** entities that are responsible for interaction with Things and the processing of Thing Description. They can be either other devices, applications or web services.
- **Intermediaries:** entities that sits between Things and Consumers and are indistinguishable from Things for Consumers.

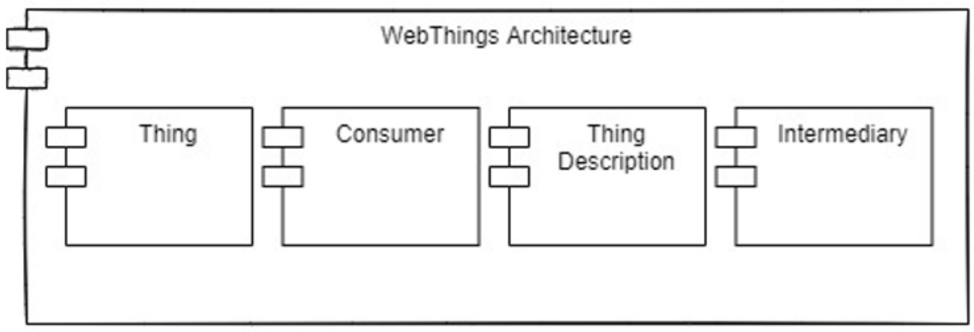

Figure 8: WoT Thing Architecture

3.1.2.2 WebThings Components

When WebThings architecture components are implemented as a software stack to take the role of representing an Exposed Thing and making the WoT interface available to the Thing Consumers or representing a Consumed Thing and making this Thing available to the applications running on the servient and need to process Thing Descriptions to interact with Things. The resulted software stack is called a servient. WebThings servient model has two different implementations, either based on the WoT scripting language or based on native language implementation, as shown in and the components are:

- **Behavior Implementation:** represents the overall application logic of the servient in terms of servient autonomous behavior, interaction affordances, Consumer behavior, Intermediary behavior, and the definition of Things, Consumers, and Intermediaries that are hosted by the Thing.
- **WoT Scripting API:** is the interface between Behavior Implementation and a scripting-based WoT Runtime. Where the WoT Runtime is a Scripting Runtime system that manages WoT-specific aspects, interprets and executes application scripts.
- **Native WoT Runtime:** a Servient implementation without the WoT Scripting API. Any programming language may be used for the WoT Runtime application-facing API. But usually, it is the native language of the Servient software stack.
- **Private Security Data:** security data such as the secret key used for interaction with Thing are managed by WoT Runtime but not directly accessible by the application. This data is stored in a separate isolated memory and an abstract of a group of operations are made accessible.
- **Protocol Stack Implementation:** implements the WoT interface used by Consumers to use remote Things via Exposed Things and produces protocol messages for interactions over the network, where multiple protocols may be implemented supported by protocol bindings for different IoT platforms.

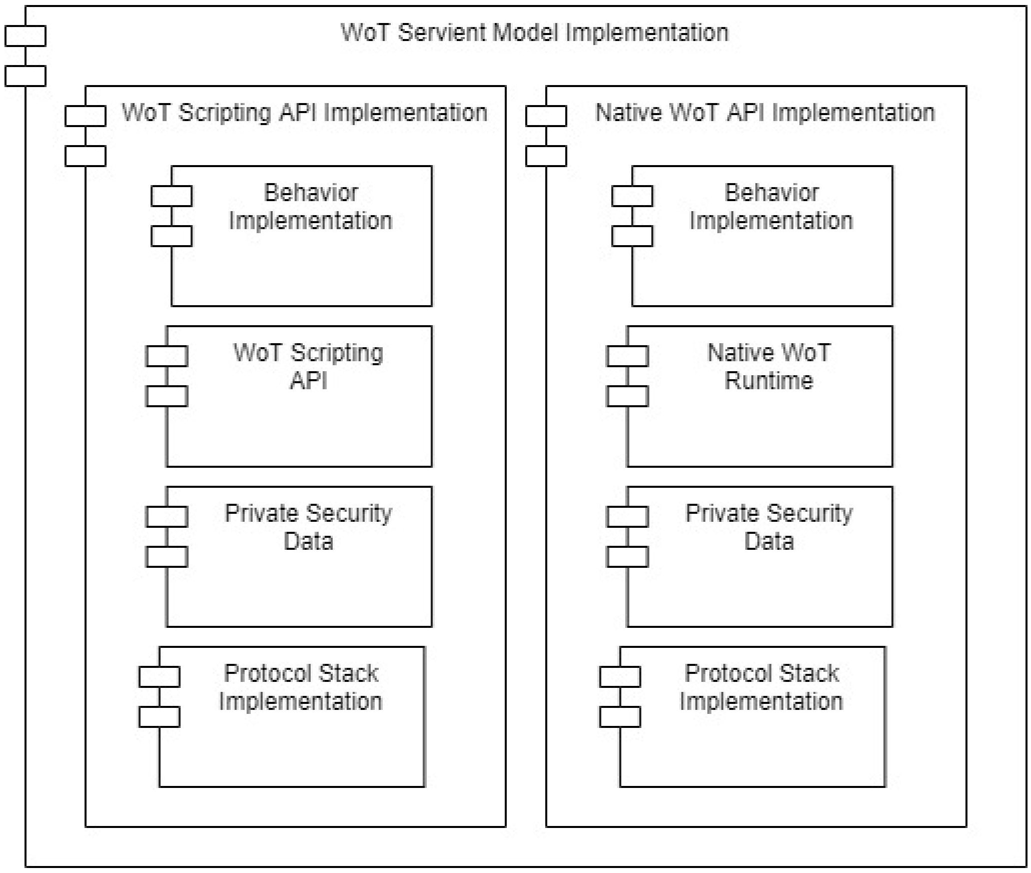
WoT Servient Model Implementation
WoT Scripting API Implementation
Behavior Implementation
WoT Scripting API
Private Security Data
Protocol Stack Implementation
Native WoT API Implementation
Behavior Implementation
Native WoT Runtime
Private Security Data
Protocol Stack Implementation

Figure 9: Architecture of the WoT servient

3.2 *User Interface*

The user interface is the main interaction element between the SIFIS-Home framework and the tenants and other users (e.g. administrator, maintainer…). The user interface is used to configure user preferences, interact with GUI capable applications, install and remove applications, set-up usage, safety and security policies.

The diagram in Figure 10 shows the structure of the User Interface component. The User Interface module includes the following components that are further elaborated in six sub-sections:

- **Home:** the principal component of the User Interface, containing commands that lead the resident user of the SIFIS-Home system to the lead features of the infrastructure.
- **Device management:** components for the configuration of the devices in the SIFIS-Home network.
- **Alarms/Log:** a component including features to show alarms to the user, and to gather logs of the functioning of the SIFIS-Home infrastructure.
- **Settings:** the component provides user interfaces for the configuration of the SIFIS-Home infrastructure. Different interfaces are provided to different actors of the SIFIS-Home system.
- **Marketplace:** the component provides graphical user interfaces from which the user can visualize available applications to be installed on the SIFIS-Home system, download, install and update them if new versions are available.
- **Input Collection:** modules in charge of providing the facility of collecting the inputs from the

user, in all the forms that are allowed by the system.

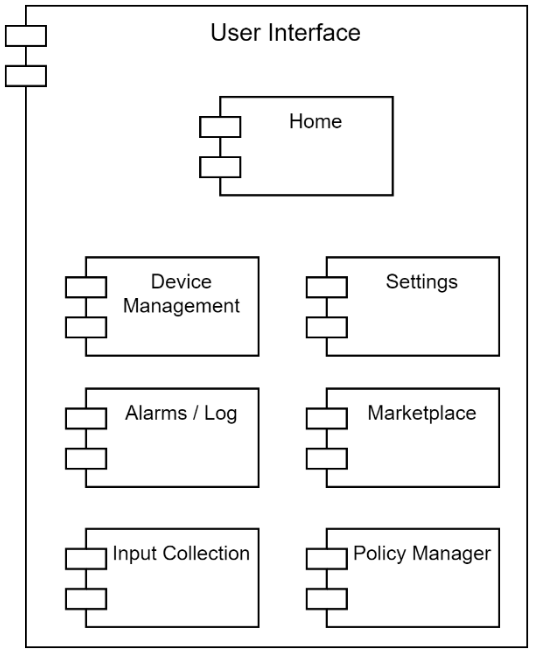

Figure 10: Components of the User Interface module

3.2.1 Home

Figure 11 illustrates the Home module that consists of the most important features of the devices. An aim of the module is that users can see the overall status of their Smart Home at a glance, without having to navigate further.

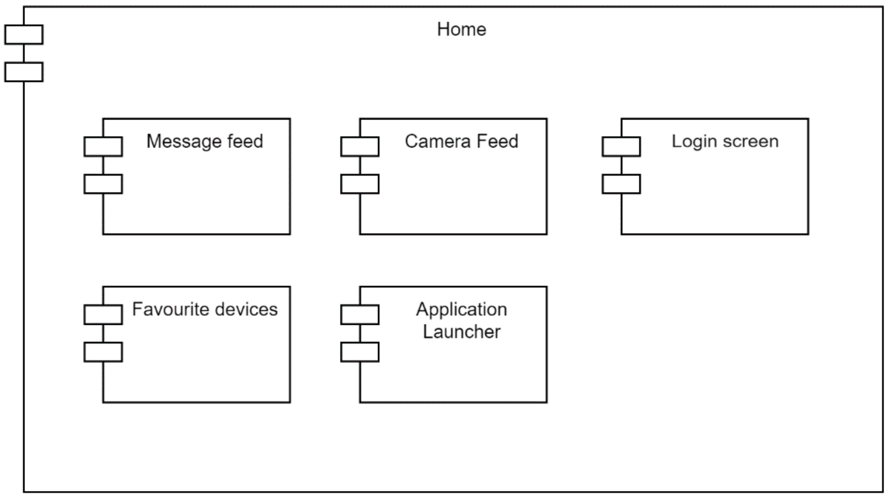

Figure 11: Components of the Home module

The module has five detailed components as follows

- **Message Feed:** This component stores and shows to the users a list of messages received from the smart devices of the SIFIS-Home architecture. Message Feed may contain crucial alerts, a link to the smart device they come from and call to action-buttons depending on the type of the message (e.g., show more-buttons, shut down an alarm and/or feature).
- **Camera Feed:** Shows a live camera feed and an easy way to browse through different

cameras if there are more than one. In event of an alarm, Camera Feed should have a recording of the source of the alarm if there are any cameras pointing at the location or (using facial recognition) a picture of an intruder.

- **Login Screen:** Users may choose to log-in using their biometric data or an access code. Login Screen provides the means for a quick and easy login, in addition to using Voice Commands,.
- **Favourite devices:** Showcases favourited (pinned) devices and their data saving time from having to navigate further down in the User Interface.
- **Application Launcher:** the software that allows the selection of the application installed on the current SIFIS-architecture and to execute them. The application launcher provides an interface to organize and uninstall the applications that are installed in the SIFIS-home architecture.

3.2.2 Policy Manager

The policy manager is responsible for the creation, review and deployment of policies across SIFIS-Home connected devices and APIs. Policies are defined and distributed to monitor access to the SIFIS-Home Architecture and APIs, so that operations or service requests are either authorized or denied. The policy manager Manages all security settings centrally, develop group or individual security profiles and support concurrent admin sessions. The policy definition and evaluation follow an access control model based on ABAC (Attribute Based Access Control), which also considers mutable attributes for decisions that might change over time. The model exploited by the SIFIS-Home framework is the Usage Control (UCON), which extends the classical ABAC by including the possibility to revoke previously granted authorizations. The Policy Manager is thus an interface for defining those policies whose enforcement is handled by the Policy Enforcement Engine.

3.2.3 Device Management

The Device Management module provides methods and settings to control SIFIS-Home connected devices in the network. They allow Resident User to use device features, view activity, share user rights and more.

The module consists of seven detailed components as follows

- **Device Sharing:** If resident user has to give another person access to their SIFIS-Home framework, they are able to share selected devices or features.
- **Device Configuration:** Shows to the user a list of all the devices and their statuses and actions to manage Device selection e.g., adding or removing a device. Selecting a device leads user to the device's main settings including
  - Renaming the device
  - Editing the share rights
  - Adding and editing biometric identification
- **Device History:** Shows to the user the device's message feed which lists all the actions it has taken and graphs showcasing the data from a selected period of time (on a daily, weekly, monthly and yearly basis).
- **Device Features:** It is a graphical interface offering to the user a panel for adjust device settings, such as:

- Add/remove notifications (Activity Log and Mobile Notifications)
- Create and edit automated actions (e.g., switch off light bulbs at night, lock doors after User leaves premises)
- Adjust temperatures and other device specific features

- **Device Rights:** Defines which rights do certain users have. Can be divided in three categories:
  - View
    Users with View rights can only view a shared device without permission to modify or use it.
  - View and edit
    View and edit rights give user permission, in addition of reading its data, to edit its features (e.g., setting alarms, adjusting room temperature, turn on an oven) and settings (sharing the device to another user).
  - Owner
    Has access to every setting related to the device in addition of viewing and editing.
- **Device Alerts:** Allows user to select events which cause an alarm. Different type of data means different needs for alerts (when values reach a specific point (e.g., minimum, maximum, on/off). Each device has its own alert features, for example a camera could alert in event of an intruder or a resident collapsing due to a medical condition. Smart kitchen equipment could alert when fridge’s temperature gets too low or an oven has been heated to ideal temperature.
  - Visual presentation
    Both SIFIS-Home and a related app show visual cue or a presentation of the alarm when it goes off.
  - Auditive presentation
    SIFIS-Home can be set to inform user(s) using an alarm sound.

3.2.4 Alarms / Log

Figure 12 illustrates the Alarms / Log module that is used to show, visualize and set off alarms. Alarms are logged in an Activity Log and shown in both SIFIS-Home system and SIFIS-Home app in order to reach user wherever they are.

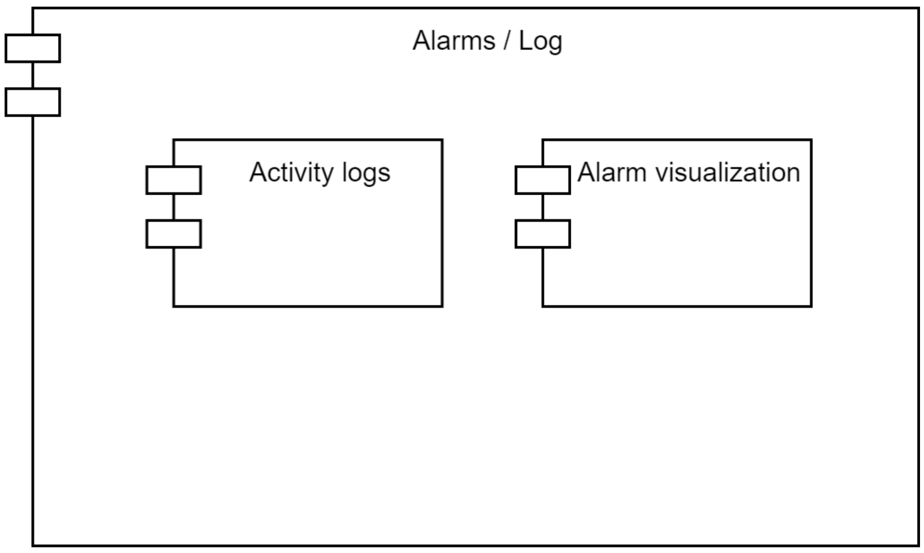

Figure 12: Components of the Alarms / Log module

The Alarms / Log module includes two components as follows

- **Activity Logs:** A sortable list of all the activity from the SIFIS-Home connected devices. Can be divided into *important events* and an *overview* of the events of the last day.
- **Alarm visualization:** A visual presentation of the alarm on SIFIS-Home and a mobile app. Can include both auditive and visual presentations.

3.2.5 Marketplace

Figure 13 illustrates the marketplace module that can been seen as a place for users to pick up and developers to produce features and devices.

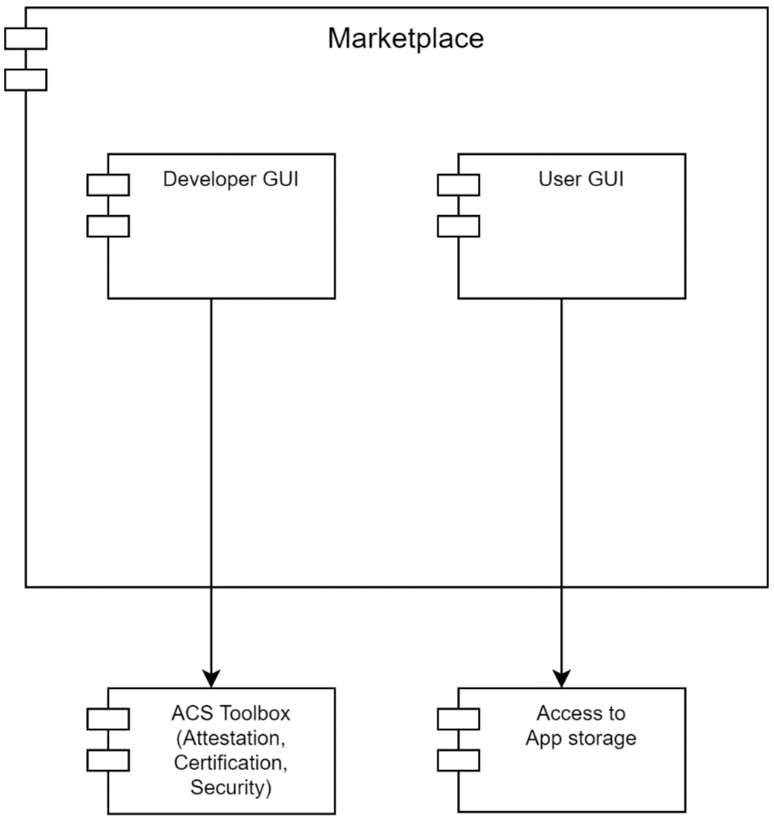

Figure 13: Structure and components of the Marketplace module

The Marketplace module includes two components as follows

- **Developer GUI:** Allows developers to create and add new devices and features for both existing and new devices. Once a new device or feature has been certified as secure and functional, they are presented in the User Marketplace.
- **User GUI:** Presents all the available devices which can be installed as a part of SIFIS-Home. Includes devices and features from major Smart Device manufacturers.

3.2.6 Settings

Figure 14 presents the Settings module that is divided to User Settings, which controls the logged-in user account, and System Settings, which controls the SIFIS-Home settings that are related to system level administration.

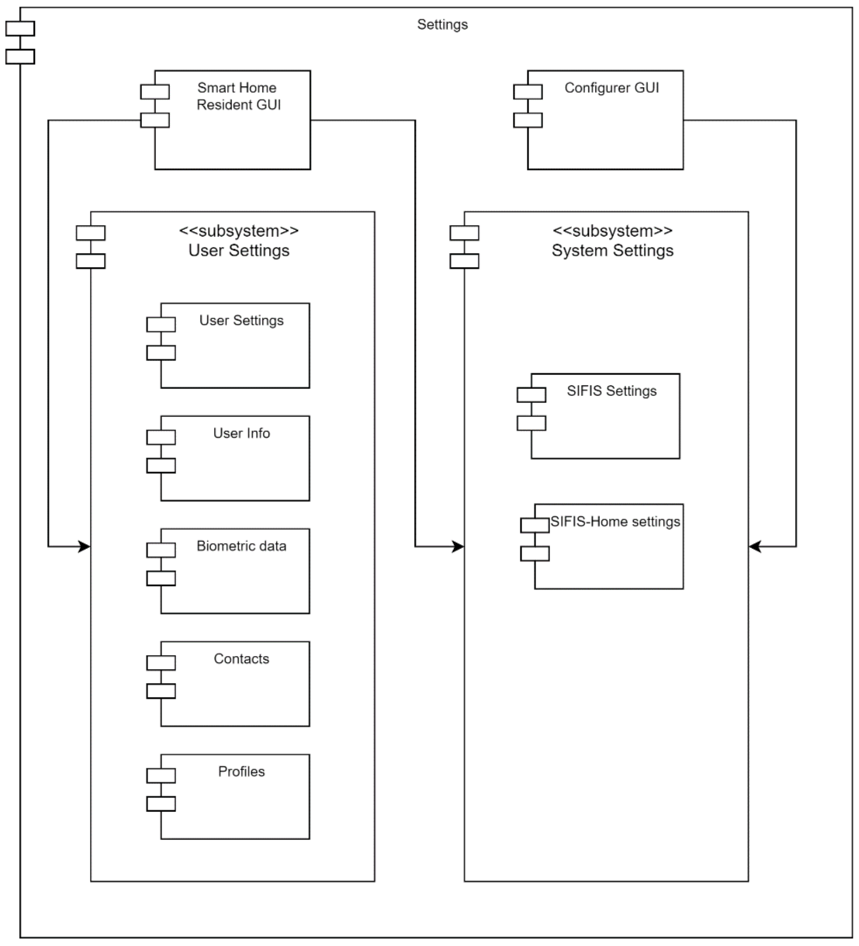

Figure 14: Structure and components of the Settings module

The Settings module components are elaborated as follows

- **Smart Home Resident GUI:**
  - **User Settings:**
    - **User Settings:** General viewable and editable settings for certain features. Contains also settings for adjusting notifications and alarm functionality.
    - **User Info:** General information about the user including contact information.
    - **Biometric data:** Editing and viewing biometric data. Users are able to edit the Touch ID, Face ID and Voice settings used to control certain devices.
    - **Contacts:** List of user's contacts which are used to share device rights and alarms.
    - **Profiles:** Administrative level setting to view, edit and add user profiles to the SIFIS-Home system.
      - Editing user's rights to the network or a device
      - The features users are permitted to access
      - Turning off/on user's event log
  - **System Settings:** General system-wide settings to control SIFIS-Home.

- **SIFIS Home settings**
  - Overview of the network and its connected devices
  - Event log presenting all the activities recorded by the devices
  - Edit saved Wi-Fi networks
  - Remove saved data

- **Configurer GUI:**
  - **System Settings:** General system-wide settings to control SIFIS-Home.
    - **SIFIS Home settings**
      - Overview of the network and its connected devices
      - Event log presenting all the activities recorded by the devices
      - Edit saved Wi-Fi networks
      - Remove saved data

3.2.7 Input collection

Figure 15 presents the Input collection module that is used to collect various type of input data from the SIFIS-Home tenants and quests.

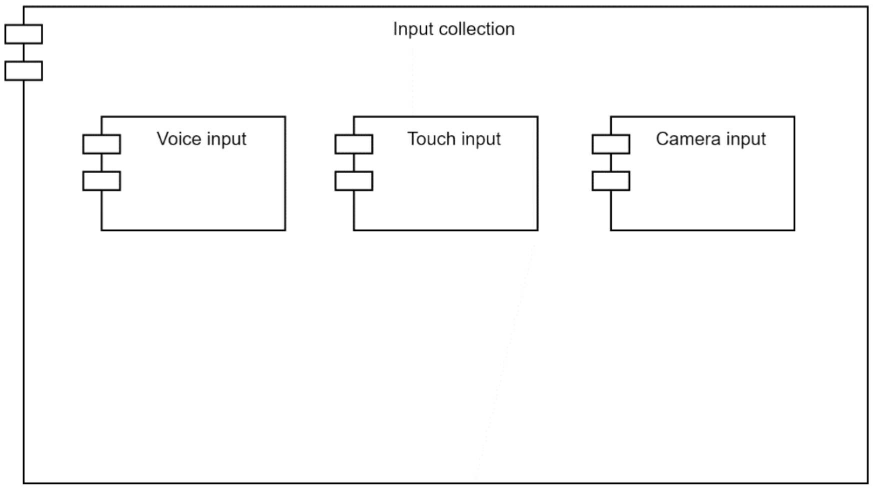

Figure 15: Components of the Input collection module

The Input collection module consists of the following components

- **Voice Input:** Used to recognize User by their voice. Certain keywords trigger an action which is noted by the system (e.g., logging in on the system, using device features). SIFIS-Home should have a collection of preconfigured actions matching certain words or sentences and Users should be able to create their own actions for certain features.

- **Touch Input:** Used when logging in on SIFIS-Home or to receive commands from touch-based device, such as smartphone of tablet. The presence of touch inputs also reflects an active interaction with a user, which is used as a feature for anomaly detection.

- **Camera Input:** Interface to collect gesture based commands and to provide identification and authorization services based on facial recognition. Provides also features to identify possible misbehaviours and dangerous situations (e.g. smoke in the house).

3.3 *Application Toolboxes*

The diagram in Figure 16 shows the structure of the Application Toolboxes module.

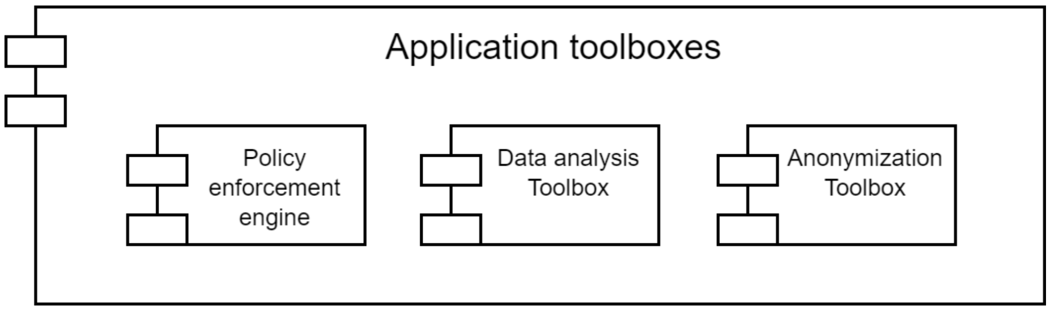

Figure 16: Components of the Application Toolboxes module

The Application Toolboxes module includes the following components that further elaborated in sub-sections:

- **Policy Enforcement Engine:** The policy enforcement engine is responsible of evaluating requests of actions and services in the SIFIS-Home framework, against the policies of usage, privacy and security defined by the administrator or maintainer.
- **Data Analysis Toolbox:** The data analysis toolbox includes all the mechanisms to perform statistical-based and machine learning data analysis. The analysis toolbox provides thus the main building blocks to analyze textual data, tabular data and multimedia data in order to perform predictions, analyze voice and gesture commands, detect intrusions and misbehaviors, as well as providing advanced smart services to the smart home users.
- **Anonymization Toolbox:** The anonymization toolbox contains software tools to preserve privacy of data during analysis. Depending from the data type and the desired level of privacy, the anonymization toolbox can generalize or suppress data information, as well as supporting differential privacy for privacy preserving data analysis.

3.3.1 Policy Enforcement Engine

Figure 17 illustrates the sub-components of the Policy Enforcement Engine component. More in details whenever an actor (subject) or component of the SIFIS-Home Architecture is willing to perform an operation, or requesting a service which might have security, safety or privacy implications, a request is issued and sent to the policy enforcement engine. The request is matched with the relevant policies, being thus either authorized or denied. The decision is based on a number of conditions depending from the subject performing the request, the operation requested and the context in which the operation is performed. The policy definition and evaluation follow an access control model based on ABAC (Attribute Based Access Control), which also considers mutable attributes for decisions that might change over time. The model exploited by the SIFIS-Home framework is the Usage Control (UCON), which extends the classical ABAC by including the possibility to revoke previously granted authorizations.

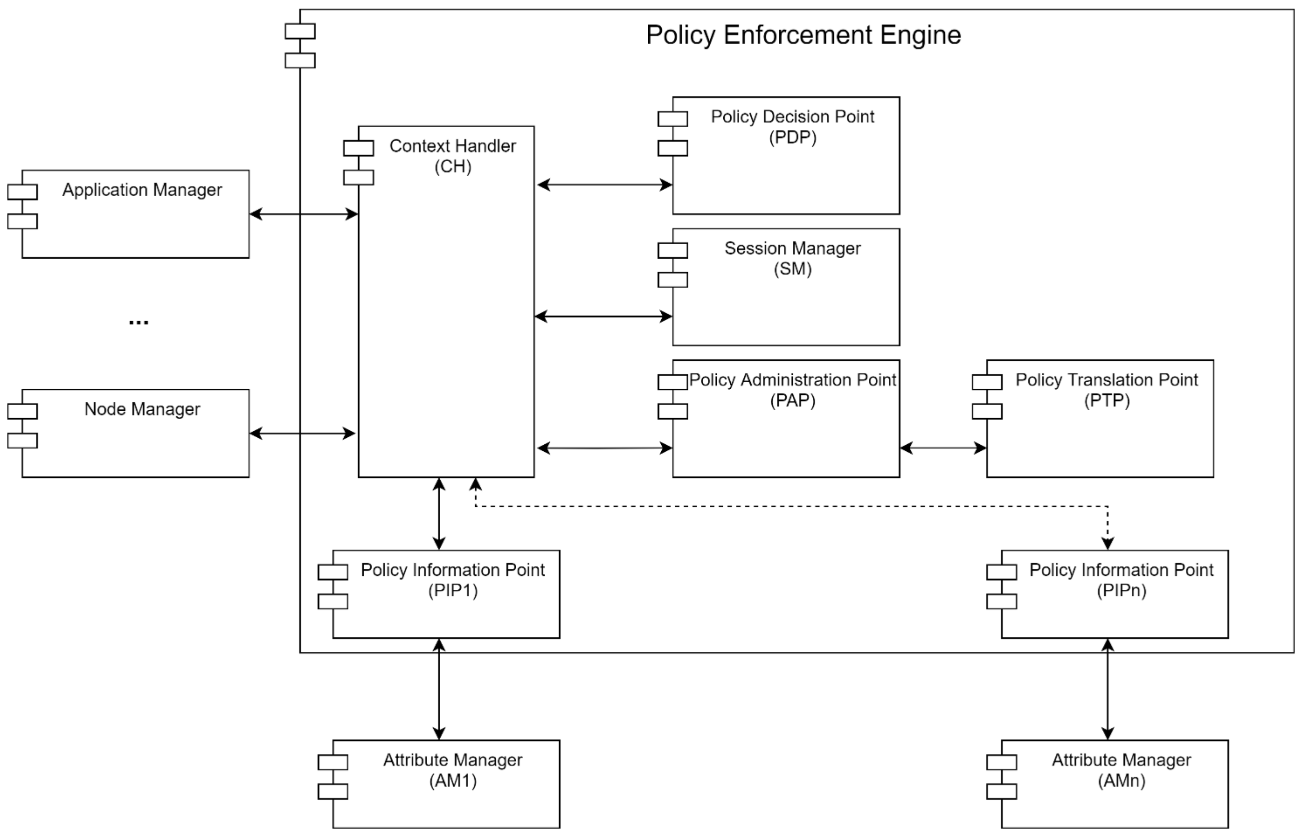
Policy Enforcement Engine
Policy Decision Point (PDP)
Context Handler (CH)
Application Manager
Session Manager (SM)
...
Node Manager
Policy Administration Point (PAP)
Policy Translation Point (PTP)
Policy Information Point (PIP1)
Policy Information Point (PIPn)
Attribute Manager (AM1)
Attribute Manager (AMn)

Figure 17: Sub-components of the Policy Enforcement Engine component

In the following we will give a description of the components of the policy enforcement engine:

- **PEP**: *Policy Enforcement Point*. This component acts as interface to the policy enforcement engine, sending and receiving messages from it. In particular, the PEPs intercept the issuing of a critical operation and prepares the request to be evaluated. Afterward, the PEP receives the decision on the policy evaluation and enforces the policy effectively granting, denying or revoking the access and usage to a resource, operation or service. In the SIFIS-Home framework, the PEPs are the various subcomponent of the framework issuing operations that will directly control both the running applications and devices, both Smart and NSSD.
- **PDP**: *Policy Decision Point*. This component is the actual policy evaluation engine. It takes as input an *access (usage) request* and an *access (usage) policy* returning one of three possible decisions: (i) PERMIT, (ii) DENY, (iii) UNDETERMINED. Policies and requests are expressed in XACML language [XACML, 2013].
- **CH:** *Context Handler.* This component is mainly a message dispatcher. It is at the core of the Policy Enforcement Engine and is responsible of forwarding the access requests first to the various PIPs for attribute retrieval, then forwards the complete access request to the PDP, finally returning the decision to the PEP. The CH also receives notification from PIPs for attribute value changes, forwarding the new value to the PDP for policy reevaluation, eventually notifying the PEP for a triggered access revocation. The CH is responsible of handling concurrency issues, also in applications considering several PEP and PIP.
- **PIP:** *Policy Information Point.* This component retrieves attributes related to the *subject* (i.e. the entity requesting a action or resource), *object* (i.e. the protected resource or action), or *environment* (i.e. the description of the context) of received access requests. Each PIP act as the interface of the Policy Enforcement Engine with a specific Attribute Manager. The

implementation of each PIP is custom for the specific application, Attribute Manager and kind of attribute that should be retrieved.

- **SM:** *Session Manager.* This component is a database which stores all the active sessions, with the necessary information to perform policy reevaluations.
- **PAP:** *Policy Administration Point.* This component stores the policies for the PDP and automatically retrieves the correct policy(es) to be matched with the requests sent by the PEPs.
- **PTP:** *Policy Translation Point.* The PTP modules enables translation between high level human readable policies (which will be described in deliverables of WP2 and WP4) and the enforceable XACML policies. PTP module firstly checks if there are conflicts between the available high-level policies. If this is the case, the Evaluator/Notifier and Alarms/Log modules are used to send an alarm to the user, e.g., in the form of a notifications. If there are no conflicts, instead, the policies are translated in the corresponding XACML policies.
- **AM:** *Attribute Manager:* This component is the interface between a PIP and the attribute environment. In the SIFIS-Home framework AMs are the various monitors provided by the Proactive Security Manager and the Distributed Storage itself.

3.3.2 Data Analysis Toolbox

Figure 18 presents the sub-components of the Data Analysis Toolbox component. It includes four sub-components to carry out statistical-based and machine learning data analysis.

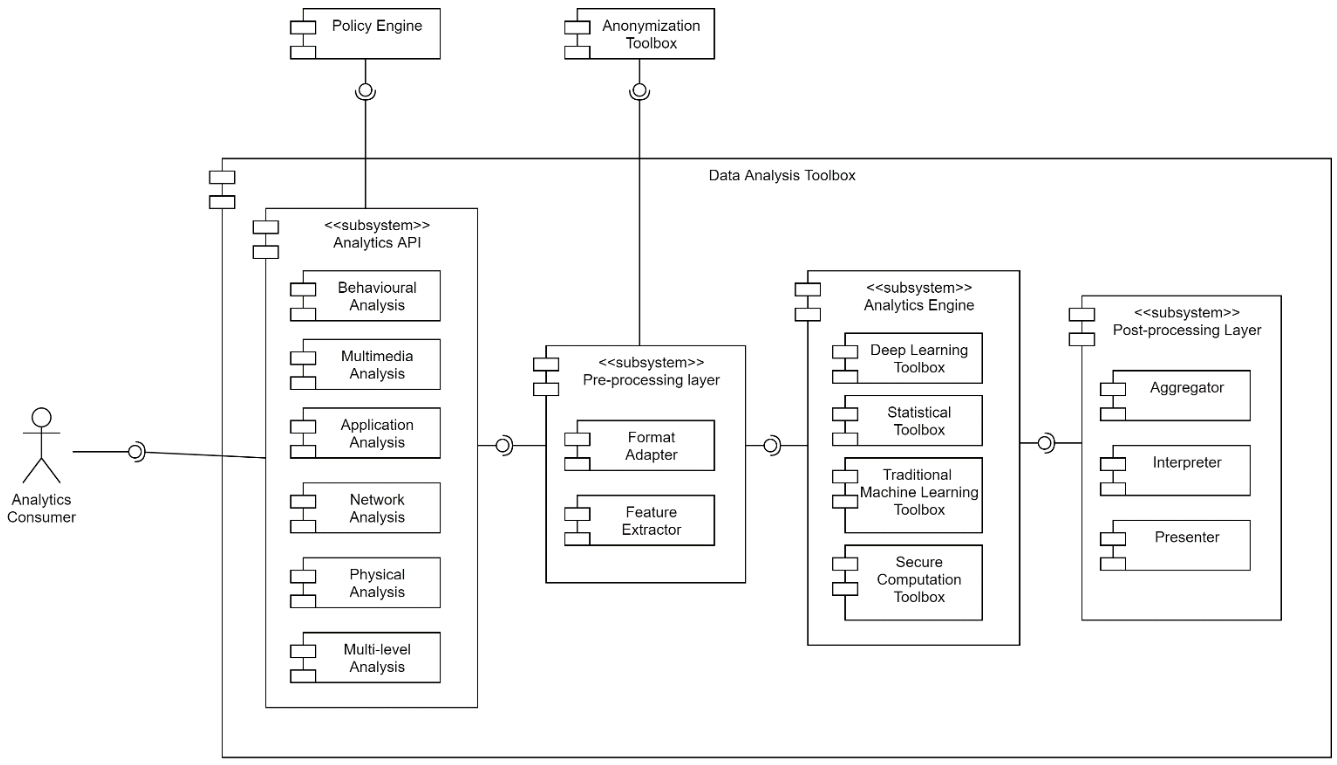

Figure 18: Sub-components of the Data Analysis Toolbox component

The building blocks of the data analysis toolbox are the following:

- The Analytics API: this component provides key data sources and enable data integration and visualization. It is responsible to grant access to data analysis mechanisms, provide the functionality to request these mechanisms, pass required parameters and datasets for them to be performed, and define the way of how the system should respond to the user and visualize the analysis results. This component has six analysis methods that enable real-time predictions using datasets, models, clusters, and anomaly detectors. Data collected from SIFIS Home environment are analyzed and then produce results to answer questions like whether a software intrusion has been detected, is there a person in a dangerous situation. It allows to extract important insights, and execute data manipulations, like converting speech to text and perform person recognition and identification.

- The pre-processing layer: this component is responsible for the data preparation functions in terms of cleaning, formatting, normalization, transformation into an understandable format, and feature extraction. It involves cleaning missing and noisy data, transforming data into an appropriate and unified format, normalizing data according to the range of values, and reducing data dimensionality by selecting or combining variables into features. The product of this component is the final dataset presented in a unified format ready to be analyzed by the analytics engine.
- The Analytics Engine: this component receives the output dataset from the pre-processing layer after being anonymized as an input and provides methods and models to describe and analyze data based on machine learning, deep learning, and statistical approaches with secure computation. This component invokes the mechanisms to handle intrusion detection, identity recognition, object detection, parental control, multi-level intrusion detection, and software intrusion detection. After data analysis is performed, the results are sent to the last component to be aggregated and presented to the user.
- The Post Processing Layer: this component is responsible for the aggregation of the results obtained from the analytics engine and prepare them to be displayed either on-screen or on hardcopy to the user. It also interprets the results and communicate with the alerting and monitoring component in case an intrusion or an anomaly has been detected.

3.3.3 Anonymization Toolbox

Figure 19 presents the sub-components of the Anonymization Toolbox component. This toolbox is responsible for data anonymization to protect the privacy of sensitive data by performing multiple operations:

- The generalization component modifies original data to be in a more general version.
- The suppression component is used to delete or replace the values that disclose identifying information about the user with NULL value.
- The perturbation component replaces the original data by adding noise to dataset elements in a random manner.

After the anonymization operations are performed on the original dataset, the anonymized dataset is forwarded to the pre-processing layed in order to be shared with the Analytic Engine.

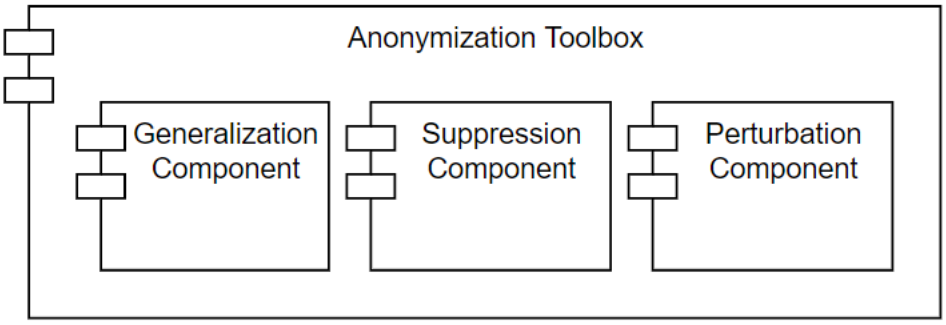

Figure 19: Sub-components of the Anonymization Toolbox component

3.4 *Secure Lifecycle Manager*

Figure 20 presents Secure Lifecycle Manager that is a core module of the SIFIS-Home framework. This module acts as orchestrator of the framework lifecycle, regulating presence and behaviour of both Smart Devices and applications. In particular, the Secure Lifecycle Manager enables and handles new device registration, as well as un-registration. Moreover, it manages installation and removal of third-party application, according to the received instructions from the user or from the policy engine and the intrusion detection system.

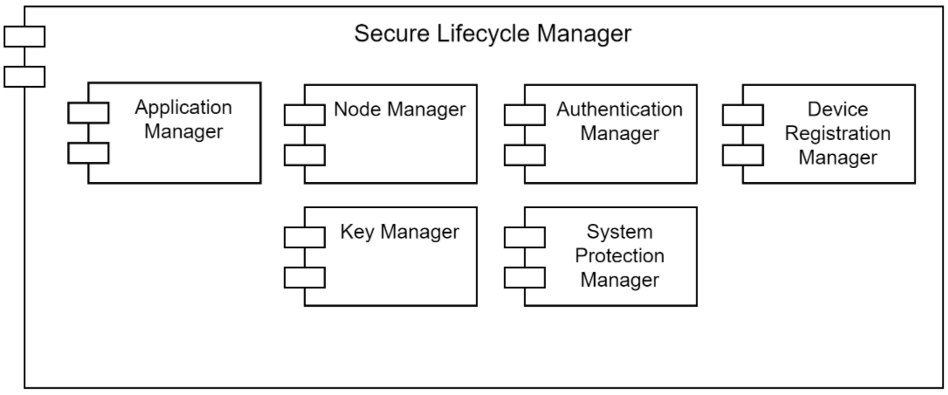

Figure 20: Architecture of the Secure Lifecycle Manager

The Secure Lifecycle Manager module includes the following components:

- Application Manager – The application manager is a component which is in charge of monitoring and controlling the behaviour of the third-party applications installed in the SIFIS-Home architecture. This component has the needed functionalities to install and remove applications, as well as interrupting their execution. Moreover, the Application Manager collects information on the application behaviour, related to their execution, the API they are invoking, the used resources and the invoked system calls. The application manager interacts with the Application Launcher to effectively launch an application execution, with the Marketplace to download and install desired applications, the Policy Enforcement Engine to verify and enforce policies related to access of resources by applications, and with the System Protection Manager, providing information related to applications and receiving commands to halt and remove misbehaving applications.

- Node Manager – The node manager is the manager of the SIFIS-Home DHT. It controls and manages the behaviour of the Smart Devices at DHT level. The Node Manager controls thus the Smart Devices that are allowed to be part of the SIFIS-Home architecture, in fact it effectively registers and unregisters devices in the DHT allowing them to participate to the communication among the Smart Devices. A device is enabled to write and read from the DHT only if it is a registered node, all other devices are unable to read the SIFIS-Home information and to exchange messages that can alter the behaviour of the SIFIS-Home architecture. The Node Manager interacts with the Device Registration Manager to handle the registrations of smart devices and with the System Protection Manager to provide information on the DHT activities, or to remove nodes which are misbehaving.
- Authentication Manager – This component provides functionalities for requesting, updating, validating, consuming and managing security credentials used to enforce authentication, authorization and access/usage control of resources and services in the SIFIS-Home network.
  This component is expected to interact with the component "Key Manager", as well as with the components "Secure Message Exchange Manager" and "Content Distribution Manager" of the "Secure and Dependable Communication Manager".
- Key Manager – This component provides functionalities for requesting, establishing, updating and managing security material and credentials used to enforce secure communications and protection of message exchanges in the SIFIS-Home network.
  This component is expected to interact with the components "Authentication Manager" and "Device Registration Manager", as well as with the components "Secure Message Exchange Manager", "Content Distribution Manager" and "Network Protection Manager" of the "Secure and Dependable Communication Manager".
- Device Registration Manager – This component provides functionalities for the enrolment, registration and de-registration of devices in the SIFIS-Home network, as well as for the discovery of devices, auxiliary entities, services and resources in the SIFIS-Home network.
  This component is expected to interact with the components "Authentication Manager" and "Key Manager", as well as with the components "Secure Message Exchange Manager" and "Content Distribution Manager" of the "Secure and Dependable Communication Manager".
- System Protection Manager – This component acts as interface between the Proactive Security Layer and the other components of the secure lifecycle manager. More in details, the System Protection Manager regulates the collection of behavioural data from nodes and applications and forwards them for intrusion detection analysis. Thus, it receives commands which are dispatched to the application manager and node manager to block suspicious behaviours of both nodes and applications.

3.5 *Secure Communication Layer*

The Secure Communication Layer includes tools and functionalities for handling secure message exchange between the components of the SIFIS-Home architecture. The Secure Communication Layer takes care of message encryption, in particular for communications going outside of the smart home premises.
The diagram in Figure 21 shows the structure of the Secure and Dependable Communication Manager module.

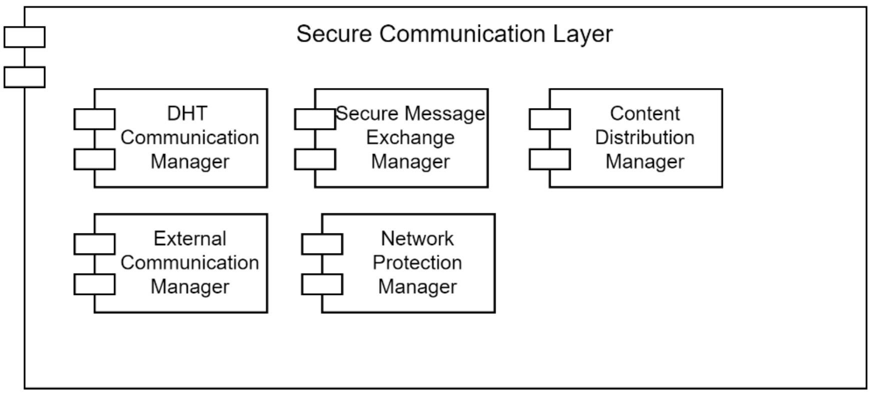
Secure Communication Layer
DHT Communication Manager
Secure Message Exchange Manager
Content Distribution Manager
External Communication Manager
Network Protection Manager

Figure 21: Architecture of the Secure Communication Layer

The Secure Communication Layer module includes the following components.

- DHT Communication Manager – This component exploits the DHT protocol to define the routing of messages and commands among the smart devices. The routing is done at application level, providing for each intended recipient, the list of DHT nodes that shall be traversed in order to deliver the message. The component interacts with the Secure Message Exchange Manager, which will handle the message transmission at the lower protocol levels.
- Secure Message Exchange Manager – This component provides functionalities to ensure (end-to-end) protection of communications and message exchanges, between the devices in the SIFIS-Home network as well as between such devices and auxiliary entities providing security/administrative supporting services in the network.
  This component is expected to interact with the component "Content Distribution Manager", as well as with the components "Authentication Manager", "Key Manager" and "Device Registration Manager" of the "Secure Lifecycle Manager".
- Content Distribution Manager – This component provides functionalities to ensure that (protected) data and control messages are practically exchanged in the SIFIS-Home network infrastructure, between the devices in the SIFIS-Home network as well as between such devices and auxiliary entities providing security/administrative supporting services in the network. Support is provided for different message exchange models, such as one-to-one, one-to-many (group communication), and publish-subscribe.
  This component is expected to interact with the component "Secure Message Exchange Manager", as well as with the components "Authentication Manager", "Key Manager", "Device Registration Manager" of the Secure Lifecycle Manager.
- External Communication Manager – This component handles communication with components external to the SIFIS-Home framework. Through this component, it becomes possible to control the SIFIS-Home remotely, through a smartphone or tablet for example. The External Communication Manager ensures interacts with the Secure Message Exchange Handler to ensure the security of the communication with devices outside of the smart home Cyberperimeter.
- Network Protection Manager – This component provides functionalities to take and enforce pre-emptive as well as reactive actions against different types of security attacks, including Denial of Service (DoS). Such methods can possibly co-exist and/or complement each other. This component is expected to interact with the components "Secure Message Exchange Manager" and "Content Distribution Manager", as well as with the components "Key

Manager" and "System Protection Manager" of the Secure Lifecycle Manager.

3.6 *Proactive Security Manager*

The Proactive Security Manager identifies misbehaviours, security and safety threat in a instance of the SIFIS-Home architecture. In SIFIS-Home we consider the following misbehaviour types:

- Physical Intrusion: An unauthorized person enters the house premises, or a Guest or Operator performs unauthorized operations (e.g. when not in the presence of the tenants, begins looking in the drawers, or they perform aggressive behaviours).
- Software Intrusion: An application performs intentionally behaviours aimed at either violating privacy, taking control of the SIFIS-Home architecture, hijacking communications between devices, escalate privileges, violating data integrity, causing physical damage to devices, the home or tenants, undermining the expected functionality of the smart home services and devices.
- Device or service misfunction: A device or a service unintentionally misbehaviour, due to a technical issue (e.g. broken sensor or actuator), or programming mistake.

The Proactive Security Manager, thus, extracts/receives features and data from daemons constantly monitoring all the layers of the SIFIS-Home architecture and framework. Features include information on operations on the DHT, readings from physical sensors, cameras and microphones, API monitors and System Calls. The received information are continuously analysed by the Proactive Security Manager, exploiting the classifiers of the Data Analysis Toolbox. Afterward, in case a misbehaviour is identified, the Proactive Security Manager interacts with the Application Manager and the Node Manager in the Secure Lifecycle Manager to tackle the misbehaviour according to a set of pre-defined actions. The Proactive Security Manager also cooperates in both directions with the Distributed Trust Manager, to improve detection and prevention of misbehaviours coming from Smart Devices.

The diagram in Figure 22 shows the structure of the Proactive Security Management Layer module.

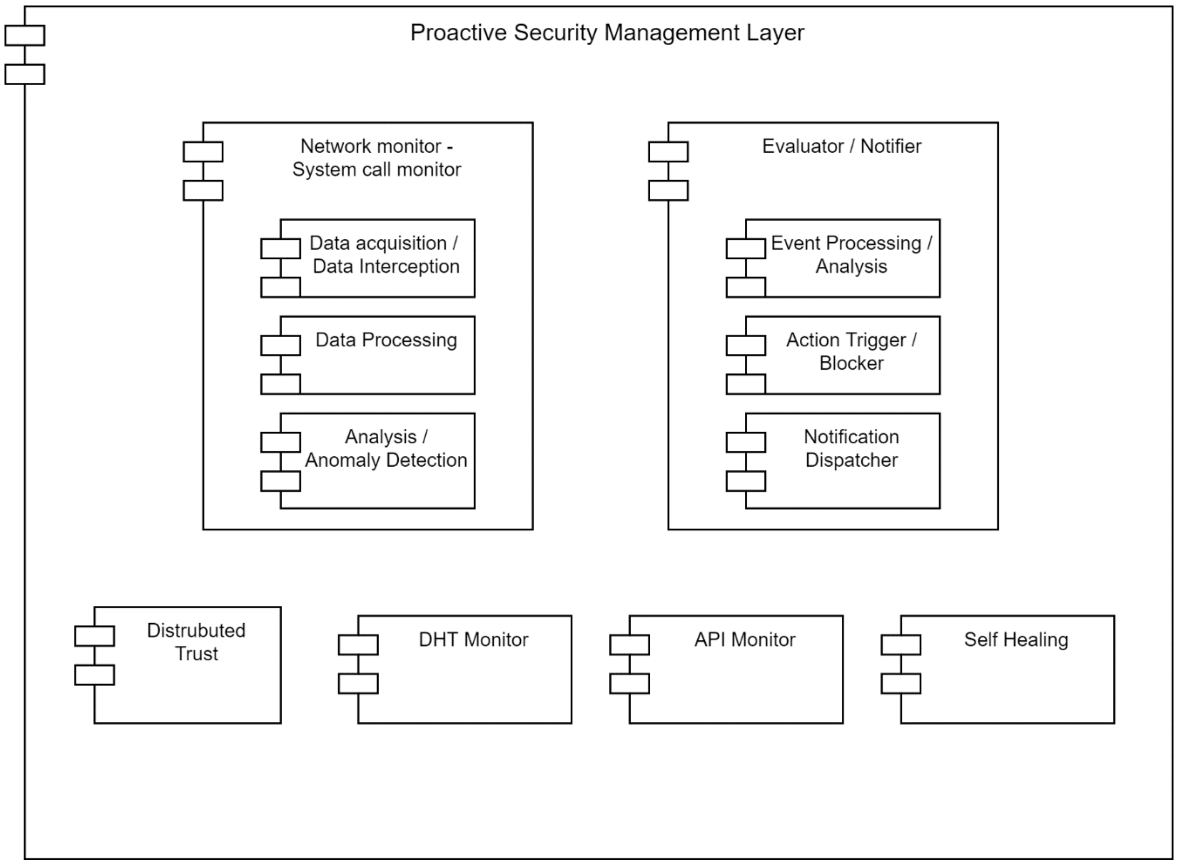
Proactive Security Management Layer
Network monitor - System call monitor
Data acquisition / Data Interception
Data Processing
Analysis / Anomaly Detection
Evaluator / Notifier
Event Processing / Analysis
Action Trigger / Blocker
Notification Dispatcher
Distrubuted Trust
DHT Monitor
API Monitor
Self Healing

Figure 22: Proactive Security Management Layer

The Proactive Security Management Layer module includes the following components:

- Network monitor / System call monitor – The primary task of this component is monitoring of network traffic. In order to proactively detect harmful activity a monitoring entity needs to be present. The acquired data is then processed and analysed for anomalies. Upon detecting anomalies, the event is further communicated to "Evaluator / Notifier" component.
- Evaluator / Notifier – Evaluator processes incoming events and performs actions based on specified policies. Examples of typical actions are blocking of malicious traffic, and/or dispatching a notification to an end-user interface.
- API Monitor – This component controls the number of times security critical APIs are invoked and monitors the invocation parameters. These information set is provided to the Proactive Security Manager to identify possible misbehaviours.
- DHT Monitor – This component controls the number of operations performed on the DHT, in particular reading and writing operations, together with registration and un-registration of nodes.
- Distributed Trust – The distributed trust manager is a component in support of the anomaly detection. Being in a fully distributed peer-to-peer environment, a root of trust is lacking and in the event of having one or more devices (both smart devices and NSSDs) compromised, identification of the compromised device(s) will be based on a collective decision where every Smart Device participates. The collective decision is based on a weighted voting procedure on identifying the misbehaving device(s). Elements to this decision are features and readings on physical and software measures, coming from sensors and the various monitors offered by the Proactive Security Management Layer. The distributed trust manager exploits these readings from each device to constantly update a trustworthiness value, which is based on three

parameters: belief, disbelief and uncertainty. When a voting procedure is performed, the vote from each device is weighted by its trustworthiness value. Moreover, nodes with a low trustworthiness level are immediately considered either as corrupted or damaged. Through the node manager, these devices are prevented to further participate to the architecture activities, unless the self-healing engine manages to fix the issue. Further details, together with a first implementation of this distributed trust mechanism can be found in [Faiella et al. 2016].

- Self-Healing – This component automatically detects, analyses, and fixes network and devices failures with minimal human intervention to ensure the property of being self-tolerant and dynamically adapt to device failures and network requirements and changes. This is performed using the 'Secure and Dependable Communication Manager' for network communication between devices and distributed Hashtable architecture updates when a devise is registered/deregistered. And the 'Secure Lifecycle Manager' for framework bootstrapping, registration/deregistration of the devices and forcing these operations when needed

3.7 *Distributed Data Storage*

The distributed data storage is a distributed database on which are stored data used by the SIFIS-Home framework. This component is implemented through the DHT itself, which provides a structured data space where all nodes (smart devices) can store information and retrieve information. The data storage is replicated with a configurable replication grade. Also, no device physically stores all the database. This ensures fault tolerance and system dependability.

4 Examples of SIFIS-Home API

In the following sub-sections, we provide an initial set of defined APIs for each block of the SIFIS-Home gateway. They provide an example to further design and develop the final list of APIs that will be reported in deliverable D1.4.

4.1 *Home API*

Registers used inputs when logging in on the system.

- **Message Feed** consists of data from *Alarms / Activity Log*
- **Camera Feed** requires access to devices that are able to record or stream Videos
- **Login Screen** needs access to *Input Collection* and *Biometric Data* from *User Settings* to provide different ways of logging in
- **Favourite Devices** accesses *Device Management* in order to display device data
- **Application Launcher** accesses *Marketplace* to display installed Applications

4.2 *Device Management API*

Accesses devices connected to SIFIS-Home.

- **Request authorization credential**, as required by a subject to perform certain operations at certain resources of certain target network nodes.
    - Name: ***Request Auth Credential***
    - Return type: Boolean [*true:* success, *false*: failure], String [authorization credential]
    - Parameters: [credential_target (s), credential_issuer (s), scope (s),

{security_specific_parameters (s), profile_specific_parameters (s)}];

- Description: The API relates to the “Authentication manager component”. It receives the credential target, issuer, scope of the authorization, and security and profile-specific parameters, and returns the operational result (Boolean) and the authorization credential (if successful).

- **Upload authorization credential**, as required by a subject to perform certain operations at certain resources of certain target network nodes.
  - Name: ***Upload Auth Credential***
  - Return type: Boolean [*true:* success, *false*: failure], String [parameters]
  - Parameters: [credential_target (s), authorization_credential (s),
    {security_specific_parameters (s), profile_specific_parameters (s)}];
  - Description: The API relates to the “Authentication manager component”. It receives the credential target, the authorization credential, and security and profile-specific parameters, and returns the operational result (Boolean) and the security and profile specific related parameters. Note that an access request is automatically checked against the authorization credential at the credential target, when later receiving a request for resource access from the subject.

- **Check status of authorization credential**, to perform certain operations at certain resources of certain target network nodes.
  - Name: ***Check Auth Credential Status***
  - Return type: Boolean [*true:* success, *false*: failure], String [parameters]
  - Parameters: [authorization_credential (s) / authorization_identifier (s), filter (s)];
  - Description: The API relates to the “Authentication manager component”. It receives an authorization credential, the identifier of authorization credential and filter criteria. It returns as output the operation result, and security and profile specific parameters.

- **Perform the exchange of a security secret**, to establish an (end-to-end) security association with another network node.
  - Name: ***Exchange Security Secret***
  - Return type: Boolean [*true:* success, *false*: failure], [String [security secret], String [information derived from the security secret], String [external authorization data]]
  - Parameters: [target_device (s), identity_credential (s), establishment_parameters (s), [external_authorization_data (s)], [wrapping_container]];
  - Description: The API relates to the “Key Manager” component. It receives as input the target device, an identity credential, and establishment-specific parameters. It receives optionally external authorization data. As output, it returns the operation result, the security secret, and optionally information derived from the security secret and external authorization data.

- **Perform an update of pairwise key material shared with another network node**, .

- Name: ***Update Pairwise Key Material***
- Return type: Boolean [*true:* success, *false*: failure]
- Parameters: [target_device (s)];
- Description: The API relates to the “Key Manager” component. It receives as input the target device, identified with a string parameter, and returns as output the operational result.

- **Join a security group**, retrieving the group key material and related parameters; **further material/information retrieval** as group member; **leaving the security group**.
  - Name: ***Join Security Group***
  - Return type: Boolean [*true:* success, *false*: failure], String [group key]
  - Parameters: [group_manager (s), group_name (s), roles (s), public_key (s), [required_info (s)]];
  - Description: The API relates to the “Key Manager” component. It receives as input the Group Manager of the group, name of the group, roles wished in the group, [own public key], [interest for other group information]. It produces as output the operation result (Boolean), group key material, [group members' public keys], [additional group information].
- **Discovery of network nodes and their resources**, e.g. through a Resource Directory.
  - Name: ***Discovery Network Nodes***
  - Return type: List<String>
  - Parameters: [target_discovery_request(s), [target_attributes (s)]];
  - Description: The API relates to the “Key Manager” component. It receives the target of the discovery request and optional attributes for the request. It returns the list of the links to the discovered resources
- **Registration of network nodes and their resources**, e.g. at a Resource Directory
  - Name: ***Register Network Nodes***
  - Return type: Boolean [*true:* success, *false*: failure]
  - Parameters: [target_registration_request(s), links (list<s>)];
  - Description: The API relates to the “Key Manager” component. It receives as parameters the target of registration request, and list of links to register. It returns as output the operation result (a Boolean value).
- **Bootstrapping** of a network node, e.g. at a LwM2M Bootstrap Server
  - Name: ***Bootstrap Network Node***
  - Return type: Boolean [*true:* success, *false*: failure], String, String
  - Parameters: [target_bootstrap_server (s), [security-mode-specific-information (s)]];

    - Description: The API relates to the “Key Manager” component. It receives as input the target bootstrap server, and [security-mode specific information]. It returns as output the operational result, the target device manager server, and security mode specific information.

- **Registration of** a network node, e.g. at a LwM2M Device Manager Server
    - Name: ***Register Network Node***
    - Return type: Boolean [*true:* success, *false*: failure], String [configuration parameters]
    - Parameters: [target_device_management_server(s), security_mode_information (s)];
    - Description: The API relates to the “Key Manager” component. It receives as parameters the target Device Manager Server, and [security-mode specific information]. It gives as output the operation result, and configuration parameters.
- **Registration of** a new smart device in the SIFIS-Home architecture
    - Name: ***Register New Smart Device***
    - Return Type: Boolean [*true:* success, *false*: failure]
    - Parameters: [digest, metadata]
    - Description: This API is the high-level interface which triggers the joining of a new Smart Device to the SIFIS-Home architecture. The digest parameter is used to verify the integrity of the SIFIS-Home framework instance installed on the device, whilst the metadata provide information on the device type and functionalities.

4.3 *WoT Interfacing API*

The WoT interfacing API work as end point o query the devices exposing their functionalities through a WoT interface. Examples of APIs are the following:

- Name: ***Call WoT Turn Light On***
    - Return type: Boolean [*true*: light turned on, *false*: error]
    - Parameters: [{light-ID (f), room-ID (f)}, intensity (f), colour (f)];
    - Description: It calls the WoT implementation of the API to turn on a specific light bulb, identified by the light-id parameter or all the lights in a specific room. If the first parameter is missing all available lights will be turned on. Intensity and colour can be used for configurable light bulbs and are ignored if the light bulb is not configurable. The invoked API returns *true* if the light has been turned on, or *false* if the light is already on, is not reachable or reported as not working.

4.4 *Communication API*

This API allows to request the transmission of messages to other network nodes. Corresponding Responses/Acknowledgments/Reset messages are handled by the communication stack.

Note: arguments in brackets are optional to specify

- Name: ***Send message***
    - Return type: Boolean [*true:* success, *false*: failure]
    - Parameters: [data (s), transfer_parameters (s), security_protocol_parameters (s), target_recipient (s)];
    - Description: This API relates to the "Content Distribution Manager" component and the "Secure Message Exchange Manager" component. The API is used to send a message to a recipient. It receives as parameters the data to send, transfer- and security-protocol specific parameters, the target recipient(s). It gives as output the operation results (a Boolean value).
- Name: ***Abort Message***
    - Return type: Boolean [*true:* success, *false*: failure]
    - Parameters: [sent_message (s)];
    - Description: This API relates to the "Content Distribution Manager" component and the "Secure Message Exchange Manager" component. The API is used to abort a message sending. It receives as parameters the pointer to a sent message or the corresponding transmission. It gives as output the operation results (a Boolean value).

4.5 *Application API*

The Application API provides operations to manage applications and/or assignments to users or groups.

- Name: ***Add application***
    - Return type: Boolean [*true:* success, *false*: failure]
    - Parameters: [activate (b), applicationId (s), app (s), [settings (s)]];
    - Description: The API is used to add a new application to the solution. It receives as parameters: activate, a Boolean value (executes the activation operation when creating the app), the application Id (identifier for the app to be added), the app (app-specific name) and app-specific settings. It returns a boolean value with the operation result.
- Name: ***Delete Application***
    - Return type: Boolean [*true:* success, *false*: failure]
    - Parameters: [applicationId (s)];
    - Description: The API is used to remove an inactive application from the solution. It receives as parameter the applicationId (identifier for the app to be delted). It returns a boolean value with the operation result.

4.6 *Notification API*

Creates notifications based on ready-made or custom conditions or events of the devices connected to SIFIS-Home.

Notifications are created from these events:

- Object: An object is detected by camera
    - An intruder tries to get in the house: camera detects the intruder and requests access for sound alarm or to make lights blink repeatedly
    - A familiar face is detected: front door gets unlocked and a push notification is sent "Person N.N. has entered the house."
- DeviceState: a certain state is detected by the device
    - Temperature reaches 35°C: device requests access for SMS/Push notification
    - Outdoot temperature gets below –20°C: heating will be adjusted to prevent indoor air from getting too cold and/or a notification will be sent as a push notification to alert tenants.

Notification types:

- Email, SMS, Push notification, Device actions (lights, sounds)

Requires access to Device Management and Home component:

- Device Alerts to select event which cause an alarm (when a value reached a specific point or state)
- Device Features to turn on and off notifications
- Camera Feed to detect Objects

5 Operative Workflows of main SIFIS-Home Operations

This section reports a preliminary set of operative workflows related to few operation of the SIFIS-Home framework. The full set of workflows will be reported in D1.4 as the definition of a larger set of operations will benefit from the relation and feedbacks of the implementation activities in WP5.

5.1 *Node Registration Workflow*

The "Node Registration" is the operation that is used to include a new Smart Device in the SIFIS-Home architecture. This operation is thus going to be issued every time the tenants buy a new smart device and install the SIFIS-Home framework on it. The workflow is shown in Figure 23. At each registration a specific Smart Device, already part of the SIFIS-Home architecture is selected to be the entry point for the new joining device and will take care of handling the registration process. The Registration procedure is started by the invocation of the *RegisterNewSmartDevice()* API from the Device Management API component of the joining node. The device sends in this way a digest of the SIFIS-Home framework installed on it and a set of metadata describing the device type and functionalities. Thus, the request issued by the joining device is handled by the Authentication Manager of the entry point Smart Node. The Authentication Manager will verify the credentials of the joining device, to verify that a legitimate instance of the SIFIS-Home framework is installed on it, then it will issue a request to the Policy Enforcement Engine to verify that there are no policies opposing to the joining of this specific device. If a PERMIT decision is returned the Device Registration Manager will release a Node ID, while the Key Manager will release a set of keys for the joining device to communicate inside the SIFIS-Home architecture. Finally, the node manager officially registers the joining node by updating the DHT and refactoring the application level networking, if needed.

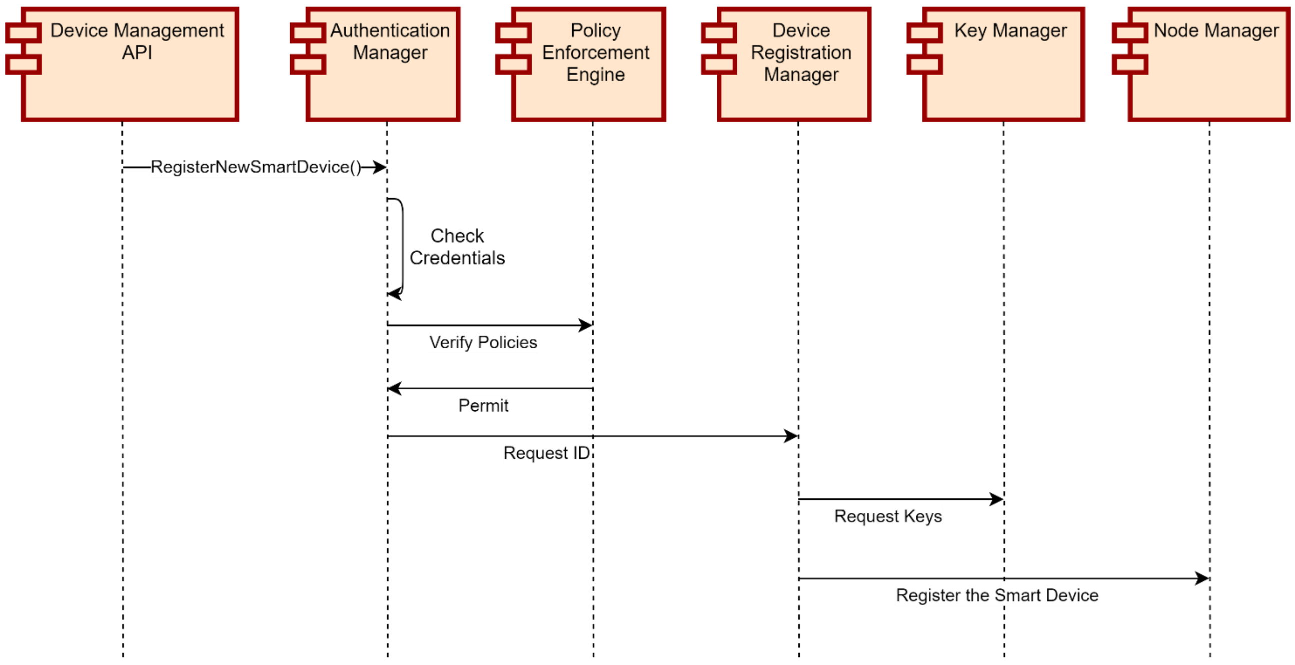

Figure 23: Node Registration Workflow

5.2 *Anomaly detection analytic workflow*

This workflow, which will be detailed in D4.2, shows the sequence of events in the execution of the anomaly detection analytic. As shown in Figure 24 the first part of the workflow consists in collecting data from the monitors and input interfaces in the GUI and Proactive Security Manager. The collection is done through a polling mechanism, where periodically the information are stored in the Distributed Storage through the DHT communication manager. The Application Manager in the Secure Lifecycle Manager will handle the classification procedure through a dedicated service that will invoke the Policy Enforcement Engine, as shown in Figure 25. Data are anonymized before the analysis through the anonymization toolbox and then processed through a specific analytic. The output of the analytic is then processed by the Evaluator/Notifier which might either notify the user or trigger corrective actions through the Application Manager or the Node Manager (not shown in Picture).

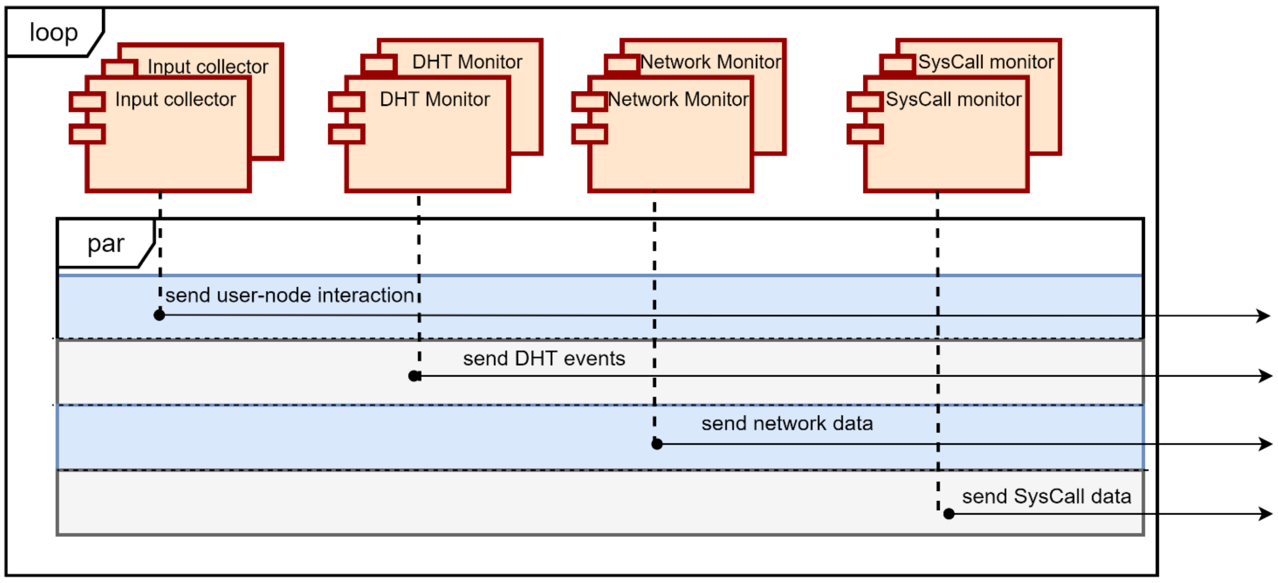

Figure 24: First part of the anomaly detection workflow.

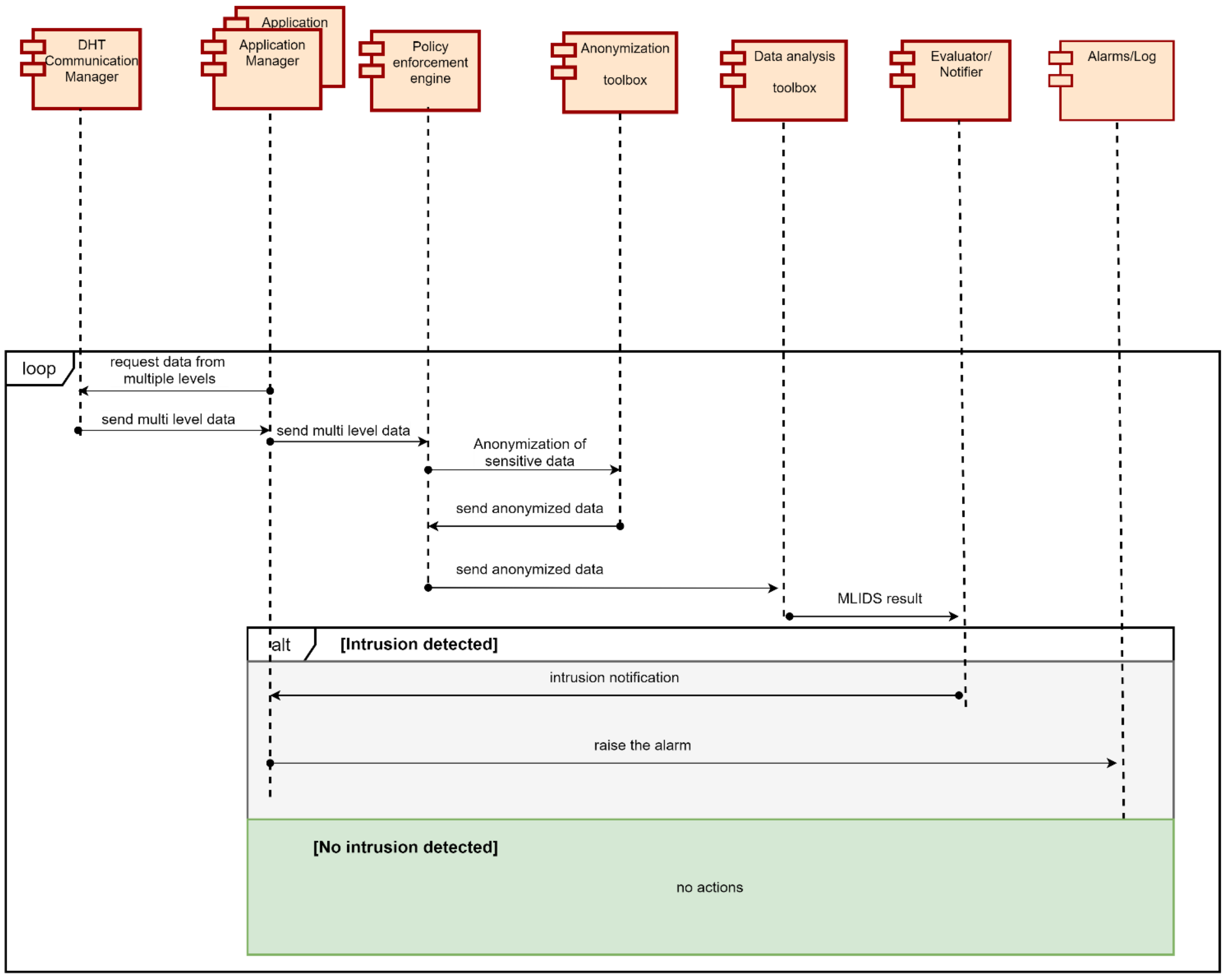

Figure 25: Second part of the anomaly detection workflow

5.3 *Policy Translation Workflow*

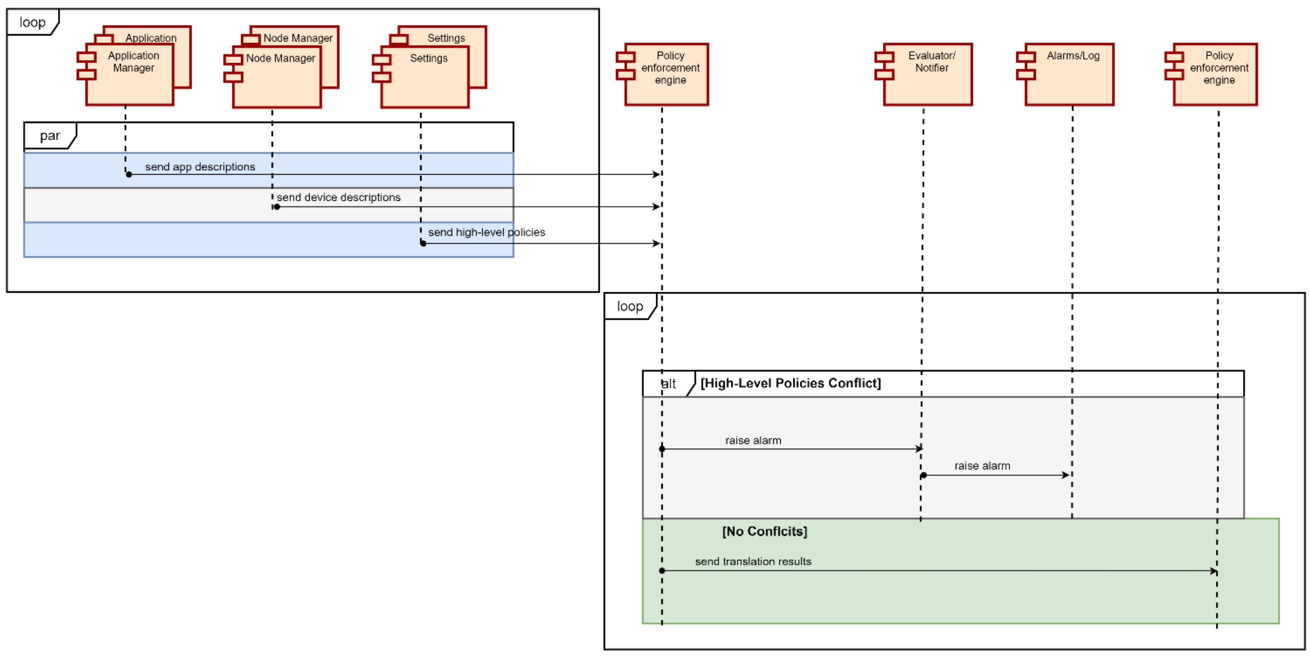

Figure 26: Policy Translation Workflow

The workflow depicted in Figure 26 represents the operation performed by the Policy Translation Point (PTP) subcomponent of the Policy Enforcement Engine. The goal of this workflow is to translate high-level policies like "*WHEN evening AND in living room THEN deny audio registration*" into a set of low-level policies, preferably in a XACML-like formalism, that will be then used by the SIFIS-HOME system to control the behavior of the devices and applications installed in the smart home. In parallel, another goal of the task is to detect potential conflicts between high-level policies.

These processes will happen through a reasoning process that will analyze contextual information as well as the capabilities of the devices and applications installed in the home. To this end, we will design a custom ontology, and we will exploit the tools and software provided by the Semantic Web framework.

In the SIFIS-HOME architecture, the ontology and the developed software are included in the **Policy Enforcement Engine** module, within a specific sub-module named **Policy Translation Point (PTP)**. To implement the translation of high-level conflicts and the conflicts detection, we continuously get the following inputs from other modules:

- From the **Application Manager** module, we get information about the applications installed in the home, e.g., their capabilities, their current statuses, etc.
- From the **Node Manager** module, we get information about the devices installed in the home, e.g., their capabilities, their current statuses, etc. In parallel, we also get information about the home itself, e.g., the rooms, the position of the various devices, etc.
- From the **Settings** module, we retrieve the list of high-level policies to be checked and translated.

From the collected input, the PTP module firstly checks if there are conflicts between the available high-level policies. If this is the case, the **Evaluator/Notifier** and **Alarms/Log** modules are used to send an alarm to the user, e.g., in the form of a notifications. If there are no conflicts, instead, the policies are translated in the corresponding XACML policies. Such policies are then sent to the same **Policy Enforcement Engine** module, which is in charge of enforcing them on the right devices and applications.

6 Conclusion

In this deliverable we have presented the preliminary design of the SIFIS-Home Architecture and the SIFIS-Home framework. The SIFIS-Home architecture helps in providing a formal definition of SIFIS-Home actors, users and devices, which will be used to define security and safety policies, based on their rights to perform specific actions in specific contexts. The SIFIS-Home framework defines instead all the components which have been used to map the needed functionalities addressing the requirements specified in D1.1 and D1.2. We recall that the architecture of the SIFIS-Home framework presented in this document is preliminary, and though it has followed a monitored bottom-up approach, we are aware that some modifications might occur following the feedbacks from the implementation and deployment phase.

This deliverable will provide inputs to deliverable D1.4 and to the activities of WP5.

7 References

[Maymounkov et al. 2002] P. Maymounkov, D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric", Peer-to-Peer Systems. IPTPS 2002. Lecture Notes in Computer Science, vol 2429. Springer, Berlin, Heidelberg

[Faiella et. al 2016] Mario Faiella, Fabio Martinelli, Paolo Mori, Andrea Saracino, Mina Sheikhalishahi:
Collaborative Attribute Retrieval in Environment with Faulty Attribute Managers. ARES 2016: 296-303

[WoT, 2020] Web Of Things (WoT) Architecture, W3C recommendation 9 April 2020, https://www.w3.org/TR/wot-architecture/

[FIWARE, 2021] What is FIWARE?, https://www.fiware.org/developers/

[YGGIO, 2021] Yggio DiMS, Digitalization infrastructure Management System, https://sensative.com/yggio/

[La Marra et al, 2017] Antonio La Marra, Fabio Martinelli, Paolo Mori, Andrea Saracino:
Implementing Usage Control in Internet of Things: A Smart Home Use Case. TrustCom/BigDataSE/ICESS 2017: 1056-1063

[Facchini et al, 2020] Simone Facchini, Giacomo Giorgi, Andrea Saracino, Gianluca Dini:
Multi-level Distributed Intrusion Detection System for an IoT based Smart Home Environment. ICISSP 2020: 705-712

[Saracino et al, 2021] M Sheikhalishahi, A Saracino, F Martinelli, A La Marra, Privacy preserving data sharing and analysis for edge-based architectures, International Journal of Information Security, 1-23

[XACML, 2013] eXtensible Access Control Markup Language (XACML) Version 3.0, Oasis Standard, http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[Perkins, 1999] C.E. Perkins, E.M. Royer, Ad-hoc on-demand distance vector routing, Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications

Glossary

Acronym	Definition
DHT	Distributed Hash Table
FR	Functional Requirements
NFR	Non-functional requirement
OS	Operative System
P2P	Peer to Peer
SIFIS-Home	Secure Interoperable Full Stack Internet of Things for Smart Home
UC	Use case
US	User story
SD	Smart Device
NSSD	Not So Smart Device
XACML	eXtensible Access Control Markup Language
PEP	Policy Enforcement Point
PIP	Policy Information Point
PDP	Policy Decision Point
PAP	Policy Administration Point
PTP	Policy Translation Point
CH	Context Handler
AODV	Ad-hoc On-demand Distance Vector