WP2 – Concept, Requirements & Specification

D2.4: Architecture Definition and Functional Specification

Deliverable Lead: ASC

Contributing Partners: ATOS, WORLD, CHA, AITEX, TUDA, NFE, TALK, ESE, TIE, IESE

Delivery Date: 06/2014

Dissemination Level: Public

Version 1.0

Based on the requirements identified in D2.3, this deliverable introduces the global architecture of the whole ALFRED system. This architecture defines the components and their interaction between each other in detail. The architecture is the foundation of the functional specification in this document. This specification will define all functionalities provided by each component. The sequence of actions and the involved subcomponents will be explained in detail for each functionality.

Document Status	
Deliverable Lead	Michael Krummen, ASC
Internal Reviewer 1	Nina van der Vaart, NFE
Internal Reviewer 2	Tim Dutz, TUDA
Type	Deliverable
Work Package	WP2: Concept, Requirements & Specification
ID	D2.4: Architecture Definition and Functional Specification
Due Date	31.05.2014
Delivery Date	30.05.2014
Status	For Approval

Document History	
Contributions	V0.1, ASC, 26.03.2014 V0.2, ASC, 16.04.2014 V0.3, ASC, TALK, WORLD, TIE, AITEX, 23.04.2014 V0.4, ASC, TALK, WORLD, TIE, AITEX, TUDA, 30.04.2014 V0.5, ASC, TALK, WORLD, TIE, AITEX, TUDA, 09.05.2014 V0.6, ASC, TALK, WORLD, TIE, AITEX, TUDA, 21.05.2014 V0.7, ASC, 23.05.2014 V0.8, ASC, NFE, TUDA, 28.05.2014
Final Version	V1.0, ASC, 30.05.2014

Note

This deliverable is subject to final acceptance by the European Commission.

Disclaimer

Project Partners

Ascora GmbH, Germany	Atos Spain sau, Spain
Worldline, Spain	Charité - Universitätsmedizin Berlin - Department of Geriatrics, Germany
Asociacion de Investigacion de la Industria Textil, Spain	Technische Universität Darmstadt, Germany
National Foundation for the Elderly, The Netherlands	Talkamatic AB, Sweden
E-Seniors, France	TIE Nederland N.V., The Netherlands
IESE Business School, Spain	

Executive Summary

The main objective of this deliverable is to provide a detailed view of the whole ALFRED system. The deliverable is divided into two main parts: The global architecture definition and the functional specification. The global architecture definition shows all involved components of the ALFRED system. Each identified component is described in detail, providing insights to the encapsulated business logic. Each component as well as each subcomponent is described thoroughly. In the functional specification the services of each identified subcomponent of the ALFRED system is described.

The whole AFRED system is divided in eight components; each component provides encapsulated functionality to the rest of the system. The main functionalities of each component are as follows:

- The **Personal Assistant** component will be realised as the end user application of the ALFRED system. This component will be virtual butler ALFRED running on a mobile device. In this Personal Assistant component the functionalities of all other components will be merged. It will not provide actual end user functionalities, but it acts as a runtime environment and a mediator between ALFRED apps and the functionalities of the ALFRED system.
- The **Health Monitor** component is a distributed system. It encapsulates on the one hand the runtime environment for wearable sensors as well the accumulation of the sensor data on the mobile device of the end users. On the other hand it will collect and process the data to health information on a server.
- The **Knowledge and Information Storage** component is an abstraction of all required databases in the ALFRED system. It enables the other components to store any data save and secure.
- The **Context-aware Speech Recognition** component handles all voice based end user input. It is a distributed system with a client and a server side. To achieve more accurate voice recognition, information from the Personalisation Manager and the ALFREDO Marketplace are utilized.
- The **Personalisation Manager** component will be the central point for all user profile information. It will be able to reason about provided data, to generate information about the user. The information is then provided as a service to the other components of the ALFRED system.
- The **Event Manager** component gathers event information. The event information will be automatically collected by crawling specific domains in the internet and manually from user input in a web portal.
- The **Game Manager** component will provide and manage the serious game of the ALFRED system. It uses information for the Personalisation Manager and the ALFREDO Marketplace in order to suggest games to the end user of the virtual butler. In addition, it will monitor the end user by analysing data from the Health Monitor to adapt the serious games to the needs of the end user.
- The **ALFREDO Marketplace** component will be the repository for all ALFRED apps. It will be a distributed system with a server and client side, providing all functionalities to manage the deployment lifecycle of the ALFRED apps.

Table of Contents

List of Figures

1 Introduction

ALFRED – Personal Interactive Assistant for Independent Living and Active Ageing – is a project funded by the Seventh Framework Programme of the European Commission under Grant Agreement No. 611218. It will allow older people to live longer at their own homes with the possibility to act independently and to actively participate in society by providing the technological foundation for an ecosystem, consisting of four pillars:

- **User-Driven Interaction Assistant** to allow older people to talk to ALFRED and to ask questions or define commands in order to solve day-to-day problems.
- **Personalized Social Inclusion** by suggesting social events to older people, taking into account their interests and their social environment.
- A more **Effective & Personalized Care** by allowing medical staff and caretakers to access the vital signs of older people monitored by (wearable) sensors.
- **Physical & Cognitive Impairments Prevention** by way of serious games that help the users to maintain and possibly even improve their physical and cognitive capabilities.

1.1 ALFRED Project Overview

One of the main problems of western societies is the increasing isolation of older people, who do not actively participate in society either because of missing social interactions or because of age-related impairments (physical or cognitive). The outcomes of the ALFRED project will help to overcome this problem with an interactive virtual butler (a smartphone application also called ALFRED) for older people, which is fully voice controlled.

The ALFRED project is wrapped around the following main objectives:

- To empower older people to live independently for longer by delivering a virtual butler with seamless support for tasks in and outside the home. This virtual butler (the ALFRED app) aims for a very high end-user acceptance by using a fully voice controlled and non-technical user interface.
- To prevent age-related physical and cognitive impairments with the help of personalized serious games.
- To foster active participation in society for the ageing population by suggesting and managing events and social contacts.
- And finally, to improve caring by offering direct access to vital signs for carers and other medical staff as well as alerting in case of emergencies. The data is collected by unobtrusive wearable sensors monitoring the vital signs of ALFRED's users.

To achieve its goals, the project ALFRED conducts original research from a user centred perspective and applies technologies from the fields of Ubiquitous Computing, Big Data, Serious Gaming, the Semantic Web, Cyber Physical Systems, the Internet of Things, the Internet of Services, and Human-Computer Interaction. For more information, please refer to the project website at http://www.alfred.eu.

1.2 Deliverable Purpose, Scope and Context

The purpose of this deliverable is to act as a clear guidance for the technical implementation during the whole course of the project. It gives not only an overview of the overall architecture of the ALFRED system but also provides deep insights in the mechanisms, functionalities and responsibilities of each part of the ALFRED system. This allows the parallel implementation of components and subcomponents while ensuring a seamless integration of all components at a later stage. The Architecture Definition and Functional Specification deliverable is also the basis for the work on the technical specification (with Deliverable D2.5).

The descriptions in the document are high level and explicit without concrete technological selections. This abstraction level will allow for an accurate and detailed common view between all partners of the ALFRED project; enabling the examination of the state-of-the-art with the same perspective afterwards during the technical specification.

1.3 Document Status and Target Audience

This document is listed in the Description-of-Work (DoW) as “public”, as it provides general information about the goals and scope of the ALFRED project and can therefore be used by external parties in order to get according insight into the project activities.

While the document mainly aims at the project’s contributing partners, this public deliverable can also be useful for the wider scientific and industrial community. This includes other publicly funded research and development projects, which may be interested in collaboration.

1.4 Abbreviations and Glossary

A definition of common terms and roles related to the realization of the ALFRED project as well as a list of abbreviations is available in the supplementary document “Supplement: Abbreviations and Glossary”, which is provided in addition to this deliverable. Further information can be found at http://www.alfred.eu.

1.5 Document Structure

This deliverable is broken down into the following sections:

- Chapter 1 provides an introduction for this deliverable including a general overview of the project, and outlines the purpose, scope, context, status, and target audience.
- In chapter 2 all components of the ALFRED system are defined. This is done by first providing an overview of the whole ALFRED system. Then the components are

explained in detail, by showing the internal structure of the components and describing the responsibilities of the containing subcomponents.

- Chapter 3 uses the same structure as chapter 2 in order to define all provided functionalities of the components of the ALFRED system. With the help of high level UML sequence diagrams all involved subcomponents for any given functionality are shown.
- Chapter 4 closes the deliverable with a summary and the next steps for the ALFRED project based on the outcomes of the task behind this deliverable.

2 Global Architecture Definition

2.1 Overview

The global architecture of the ALFRED system shows the interaction and the responsibilities of all components, subcomponents and modules, of which the ALFRED system as a whole is made up of. This chapter introduces the seven main components of the ALFRED system, as depicted in Figure 1.

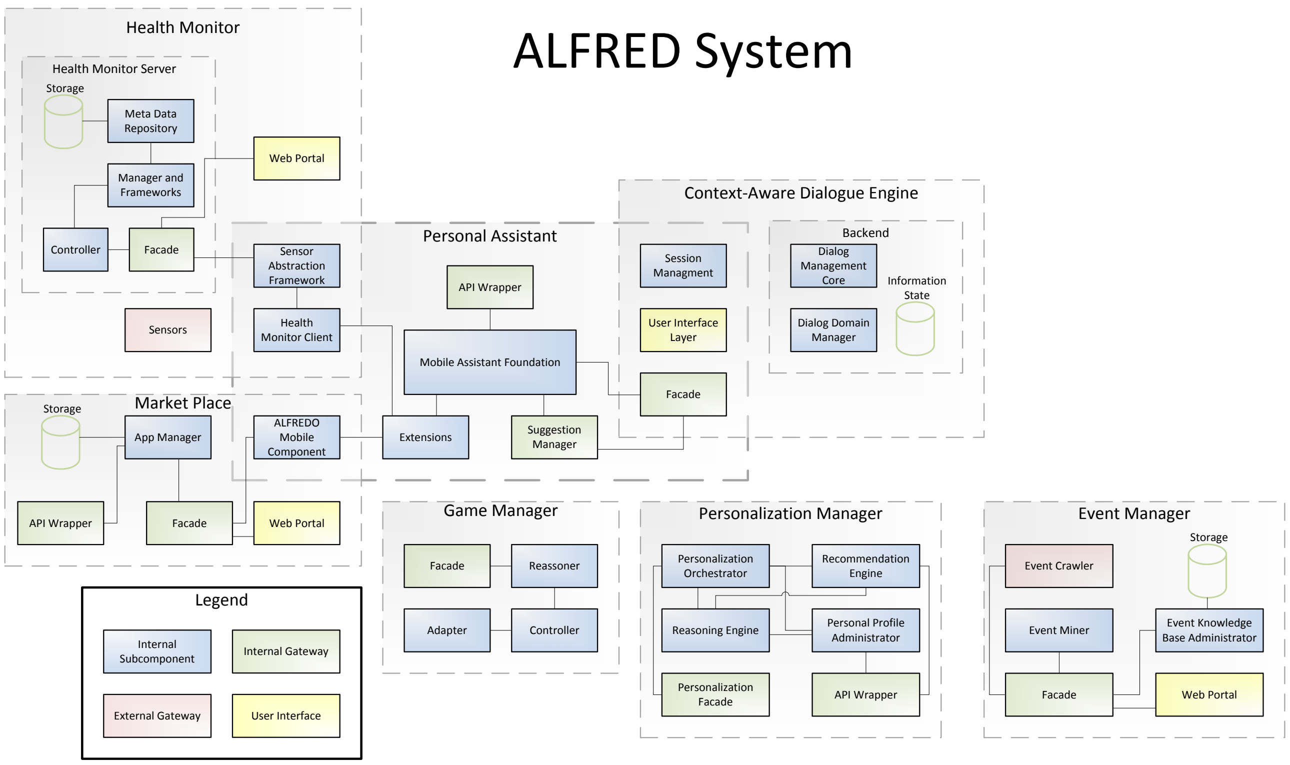

Figure 1: Infographic - Overview of the Architecture of the ALFRED System

Each component of the ALFRED system provides encapsulated functionalities to the rest of the system. The most visual component will be the **Personal Assistant (PA)** component, which will provide the main interaction point to the end user of the ALFRED system and which will run as an app on a mobile device (the “ALFRED device”). It is this component that what will be perceived by the user as the “virtual butler ALFRED”. The Personal Assistant component will merge all functionalities of the other seven components and their subcomponents and it will provide some of these functionalities to third-party apps and games that are associated with the ALFRED system, the so-called “extensions” to ALFRED.

The other seven components can be divided into two categories. The first category is made up of those components that have both an online subcomponent, and a mobile subcomponent running locally on the ALFRED device. There are four of these components, and they are:

- The **Health Monitor (HM)** component has a mobile subcomponent running locally on the ALFRED device that encapsulates the framework which provides access to the external sensors and wearables that are part of the ALFRED solution. The online subcomponent of the HM collects the raw sensor data and processes it to health information.
- The **Context-aware Speech Recognition (CADE)** component handles all voice based end user input. It interacts with the end user via the mobile subcomponent. To achieve more accurate voice recognition, information from the Personalisation Manager (see below) and the ALFREDO Marketplace (see below) are utilized in the more sophisticated interpretation of the user input by the online, server-based subcomponent of CADE.
- The **ALFREDO Marketplace (AM)** component will be the repository for all ALFRED extensions (= apps). On the ALFRED device, the respective subcomponent provides management functionalities to the end user, such as finding, installing, updating and deleting an ALFRED extension. On the server side, the subcomponent of the AM provides functionalities required to manage ALFRED extensions from a provider's perspective.
- The **Game Manager (GM)** component will provide and manage the serious game of the ALFRED system. It uses information for the Personalisation Manager and the ALFREDO Marketplace in order to suggest games to the end user of the virtual butler. In addition, it will monitor the end user by analysing data from the Health Monitor to adapt the serious games to the needs of the end user.

The second category of components includes subsystems with only a single online, server-sided component:

- The **Personalisation Manager (PM)** component will be the central hub for all user profile information. It will be able to reason on the provided raw data in order to acquire new, more complex knowledge about the user. All this information is then provided as a service to the other components of the ALFRED system.
- The **Event Manager (EM)** component gathers event information, which is automatically collected by crawling specific domains in the Internet and manually through user input in a web portal. Not depicted in Figure 1 is the **Knowledge and Information Storage (KIS)** component. It acts as a provider of various types of databases to the components of the ALFRED system. It thus enables the other components to store all data save and secure, ensuring data sustainability and privacy.

The following subchapters provide a brief overview for each of ALFRED's eight main components.

2.2 Personal Assistant

2.2.1 Overview

The Personal Assistant (PA) provides a unified access point to the functionalities of the other ALFRED components. It's running locally on the ALFED mobile device and consists of the following subcomponents:

- **Extensions:** all kinds of third-party applications and games not part of the ALFRED system (as described in this document), but rather building upon the functionalities provided by it.
- **User Interaction:** contains modules which handle user interaction.
- **API Wrapper:** this subcomponent contains modules that allow extensions to access functionalities provided by other ALFRED components.
- **Mobility Assistant Foundation:** the core of the Personal Assistant and as such, of the ALFRED system as a whole. Connects and manages all components and extensions.

Figure 2 depicts the architecture of the Personal Assistant in detail.

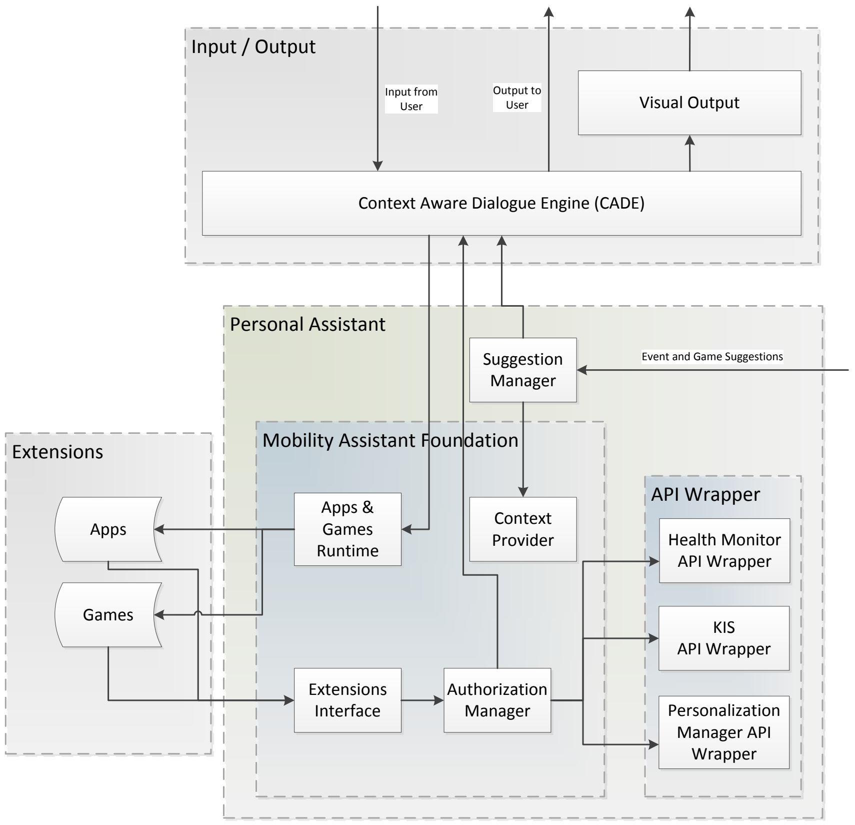

Figure 2: Infographic - Architecture of the Personal Assistant

2.2.2 Extensions

Extensions are all types of third-party applications, including (serious) games, which are based on the ALFRED system. Such extensions are developed within the ALFRED project, but the project team also aims to establish a community of developers that will provide additional extensions after the project itself has run out. Extensions will be managed and handled by runtime modules, residing within the Personal Assistant core component.

2.2.3 User Interaction

When the user presses ALFRED’s main (and only) button, she initiates the CADE component (see chapter 2.4), which will then listen to user’s voice, try to interpret what it has received and eventually, if a command has been understood, trigger the respective function in the ALFRED system. In some cases, however, it may also be necessary to involve another modality, such as a graphical user interface. This, for example, could happen when the user wants to call a person by saying her name, but when CADE is not able to understand that name and thus requires further clarification. In these situations, CADE could provide a number of options, or even a virtual keypad to the user.

2.2.4 API Wrapper

In order to enable extensions to access the system’s components via a unified interface, it is meaningful to encapsulate the calling mechanisms to these components and to provide API wrappers instead. These wrappers offer a standardized interface and hide the distribution of the ALFRED components across the local device and the cloud.

2.2.5 Mobility Assistant Foundation

The Mobility Assistant Foundation is the core subcomponent of the Personal Assistant component and as such, of the entire ALFRED application. It handles extensions, forwards user interaction to the respective components, method calls between components, and so forth.

Requests from extensions to the internal API wrappers will be handled by the Extensions Interface module. This additional interface will hide the structure of the internal components from external ones and assures to keep control over the extensions’ activities.

Extensions may need additional information about the user (i.e., where she is at and what she is doing) or the device itself (e.g., which apps are currently active). Such information will be accessible through the Context Provider module, which is part of the Mobility Assistant Foundation subcomponent.

2.3 Health Monitor

2.3.1 Overview

The Health Monitor (HM) environment involves four main pieces of software. The general layout is described in the following infographic (see Figure 3):

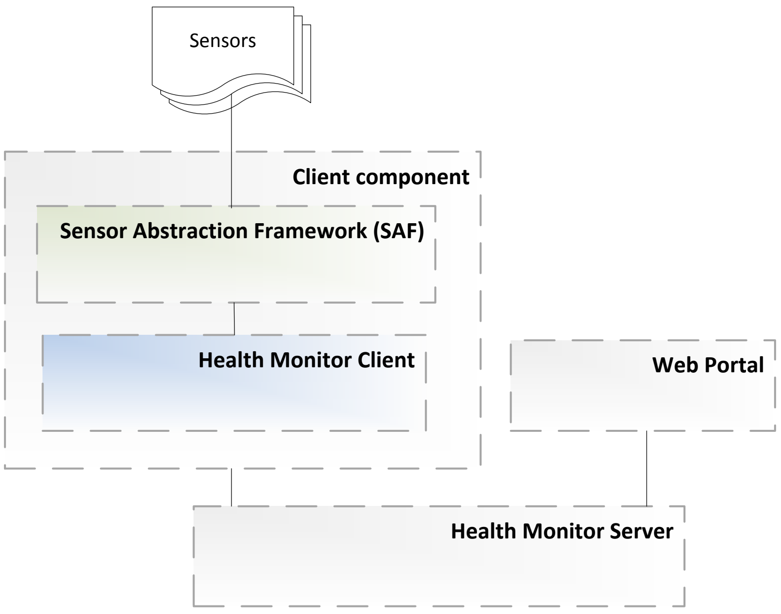

Figure 3: Infographic - Overview of the Architecture of the Health Monitor

The four main subcomponents of the HM component are:

- The **Sensor Abstraction Framework** (SAF) provides access to sensor measurements through an open API called the SAF API.
- The **Health Monitor Client** is the subcomponent installed locally on the ALFRED device. It receives information from the sensors via the SAF and is in constant communication with the Health Monitor Server.
- The **Health Monitor Server** is a cloud-based subcomponent that receives and interprets information from the Health Monitor Client.
- The **Web Portal** provides web-based access to the HM services.

A more detailed overview of these components is given in the sections that follow.

2.3.2 Sensor Abstraction Framework

The Sensor Abstraction Framework (SAF) is an extensible independent subcomponent running locally on the ALFRED device, which can provide access to various kinds of sensors (especially those within wearable devices). The following figure depicts the architecture of this subcomponent (see Figure 4):

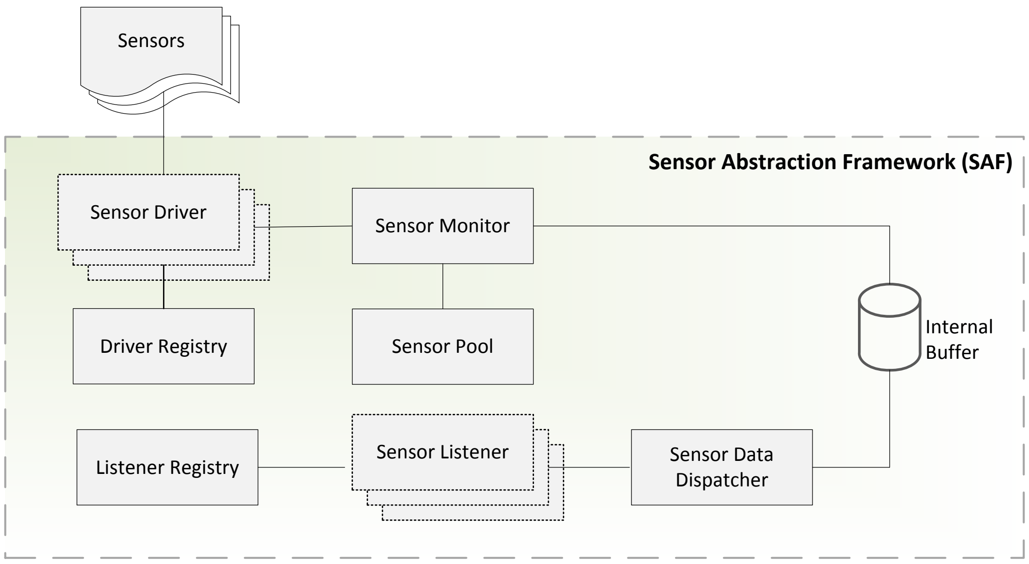

Figure 4: Infographic - Architecture of the Sensor Abstraction Framework

Access to sensors is realized by registering in the Driver Registry, so that different Sensor Drivers know how to communicate with different sensor types. All Sensor Drivers provide the same interface to interact with all types of sensors. For example, one driver may recognize, configure and capture data from a particular type of temperature sensor.

The Sensor Pool maintains information about all active sensors, that is, sensors from which measurements must be taken. Typically, a sensor is active when there is at least one listener registered for it.

The Sensor Monitor receives all sensor measurements, coming from all active sensors in the system, that is, the sensors included in the Sensor Pool, and stores them into an internal buffer. This private storage uncouples the Sensor Monitor from the Sensor Data Dispatcher, effectively implementing the producer-consumer pattern and supporting significant speed differences between both modules.

The Sensor Data Dispatcher module dispatches sensor measurements among all the registered Sensor Listeners. Each listener will be able to listen to measurements coming from one or more sensors.

The Listener Registry maintains information about all registered Sensor Listeners.

The Health Monitor Client (see section 2.3.3) is registered in the SAF just like any other listener and thus receives sensor measurements from a set of preselected (health-related) sensors on a regular basis.

Two main design choices drive the architecture of the SAF subcomponent: independence and extensibility. The SAF is an independent subcomponent that can work stand alone and does not depend on any other components of the ALFRED system. The framework can be easily extended by registering new drivers that receive information from new sensor types. This feature enables third parties to integrate new sensors with new capacities along with new extensions that make use of such capacities, providing extra value to the overall ALFREDO ecosystem.

2.3.3 Health Monitor Client

The Health Monitor Client represents the bridge between the user and the Health Monitor service and it is responsible for two major functionalities:

- Providing easy access to the user health profile.
- Receiving information from sensors and transmitting such information 'conveniently pre-processed' to the Health Monitor Server.

The following figure (Figure 5) describes the architecture that supports these services.

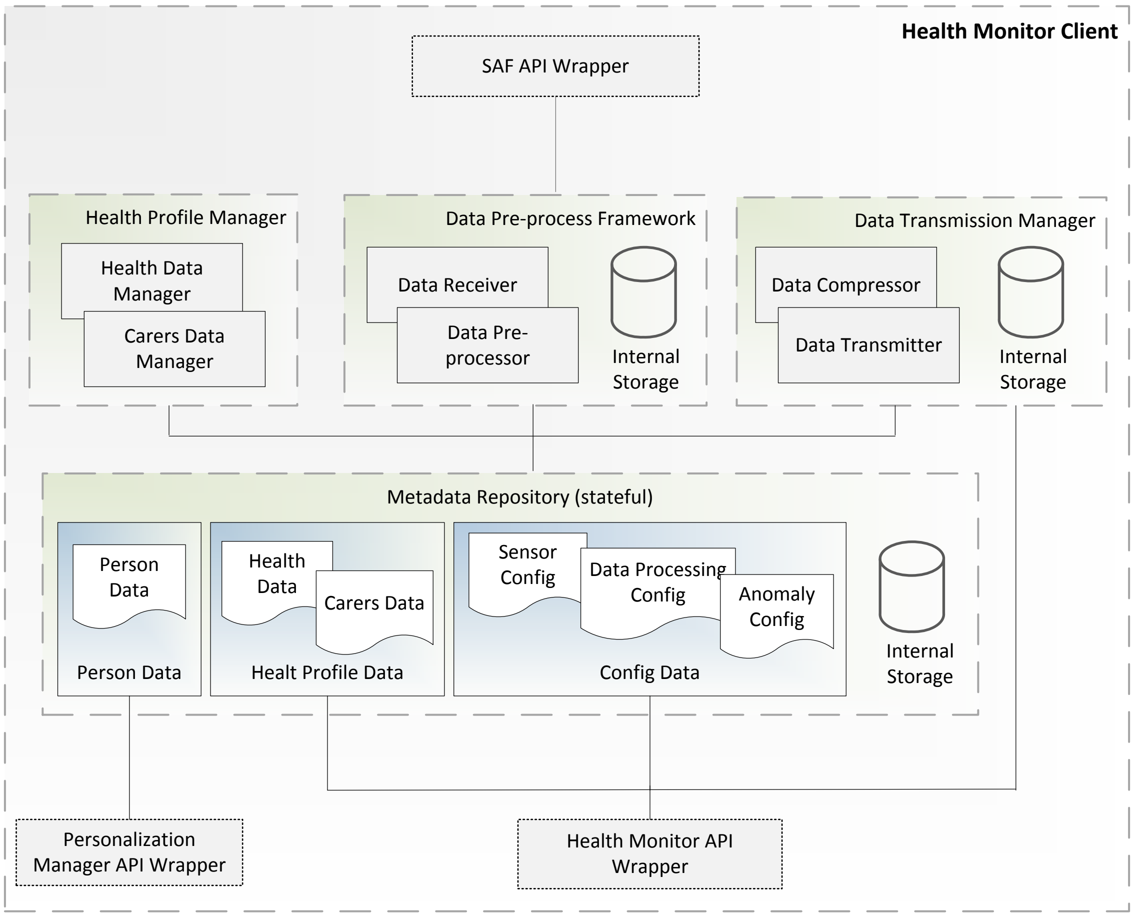

Figure 5: Infographic - Architecture of the Health Monitor Client

The four main modules of the Health Monitor Client subcomponent are:

- The Health Profile Manager that provides access to the user's health profile.
- The Data Pre-process Framework that receives all data coming from the sensors and pre-processes it for further transmission.
- The Data Transmission Manager is responsible for transferring all data to the Health Monitor Server.
- The Metadata Repository provides support to the other three modules. It contains all the metadata the other modules require to perform their activities.

The first functionality (access to the user profile) is realized by the Health Profile Manager, whereas the second functionality (transmitting sensor data) is realized by the Data Pre-process Framework and the Data Transmission Manager.

2.3.3.1 The Health Profile Manager

This module provides easy access to the user's health profile. The Health Data Manager submodule involves access to the user's health data. The Carers Data Manager submodule includes granting and revoking the appropriate permissions to a caregiver, presumably by specifying the carer identifier, the carer relationship and by confirming the operation. Both submodules depend on the Metadata Repository module to retrieve and manipulate the required information (see section 2.3.3.4).

2.3.3.2 The Data Pre-process Framework

This module is registered in the SAF as a listener and receives measurements coming from a collection of previously configured sensors. All measurements are received through the Data Receiver subcomponent.

All measurements are pre-processed and prepared for further transmission by the Data Pre-processor. The type of pre-processing depends on the sensor type, the measurement data and the associated configuration. The configuration can be obtained from the Metadata Repository (see section 2.3.3.4). Part of the configuration is sensor-measurement dependent and another part must be defined by the medical staff.

As an example of a sensor-measurement dependent configuration, consider detecting changes in the position of the user. In this example, measurements coming from an accelerometer may be discarded if they match (with some tolerance) previous measurements, so there is no need to process, transmit and store those values. Only significant changes in the values are interesting and will be processed

As an example of a configuration defined by the medical-staff consider measurements taken only at particular moments in the patient's daily life, taking maximum, minimum or average values.

The Data Pre-process Framework may require Internal Storage for calculations. In the first example presented before, temporary storage between successive measurements is required for storing previous values of the accelerometer. In the second example, temporary storage between successive measurements is required for calculating maximum, minimum and average values.

The Data Pre-process Framework must work in combination with the Data Post-process Framework included in the Health Monitor Server, since the latter is the consumer of the pre-processed data.

2.3.3.3 The Data Transmission Manager

This module obtains data pre-processed by the Data Pre-process Framework and efficiently transmits it to the Health Monitor Server.

Before transmitting data, it may be compressed by the Data Compressor in order to obtain a better communication performance and optimize resource utilisation.

The data is finally transmitted to the Health Monitor Server, making use of the Health Monitor API.

An Internal Storage is required in order to uncouple pre-processing from transmission and to support changes in the speed of both activities. This storage is also required to temporarily store all data, during communication failures, congestion or disconnection periods, for later retransmission.

2.3.3.4 The Metadata Repository

This module provides homogeneous access to all the metadata that the other modules depend on. In particular, we are considering the following metadata collections:

- Person data that provides information about the user.
- Health Profile data that provides information about the user's health, carers, user-carer relationships, and granted permissions.
- Configuration data that consists of the following categories:
  - Sensor configuration: information about the sensors available for the user, as well as their proper configuration. This is required for configuring SAF-Health Monitor Client connection.
  - Data Processing configuration: information about how sensor measurements must be pre- and post-processed. This is required for configuring the Data Pre-process Framework.
  - Anomaly configuration: information about anomaly detection. Necessary for early-detection of anomalies in the client-side or for complementing Data Processing configuration in order to perform appropriate pre-processing activities for enabling easy detection of anomalies in the server-side.

The task of this module is twofold. On the one hand, it uncouples the Health Monitor Client subcomponent from external sources under a common repository interface. On the other hand, it functions as a cache, so that the same data will not be retrieved twice, improving performance and yielding failover capabilities. To that end, an Internal Storage module is provided.

The Metadata Repository is an intermediate storage space. All the information will ultimately be obtained from either the Personalisation Manager or the Health Monitor Server.

2.3.4 Health Monitor Server

This subcomponent is a stateless (process-only) cloud based element, due to the fact that flexible and fast scalability is required with increasing load. As more and more devices are expected to join the ecosystem, resource consumption will boost and the only way to guarantee reasonable performance is to rely on flexible cloud platforms.

The Health Monitor Server represents the core of the HM environment and performs two main roles:

- It is the counterpart of the Health Monitor Client subcomponent and maintains a permanent connection with it, receiving sensor data and conveniently storing them.
- It publishes a collection of health-related services consumable by other components of the ALFRED system, and makes them available through the Health Monitor API.

To achieve these functionalities the Health Monitor Server uses the following architecture (see Figure 6).

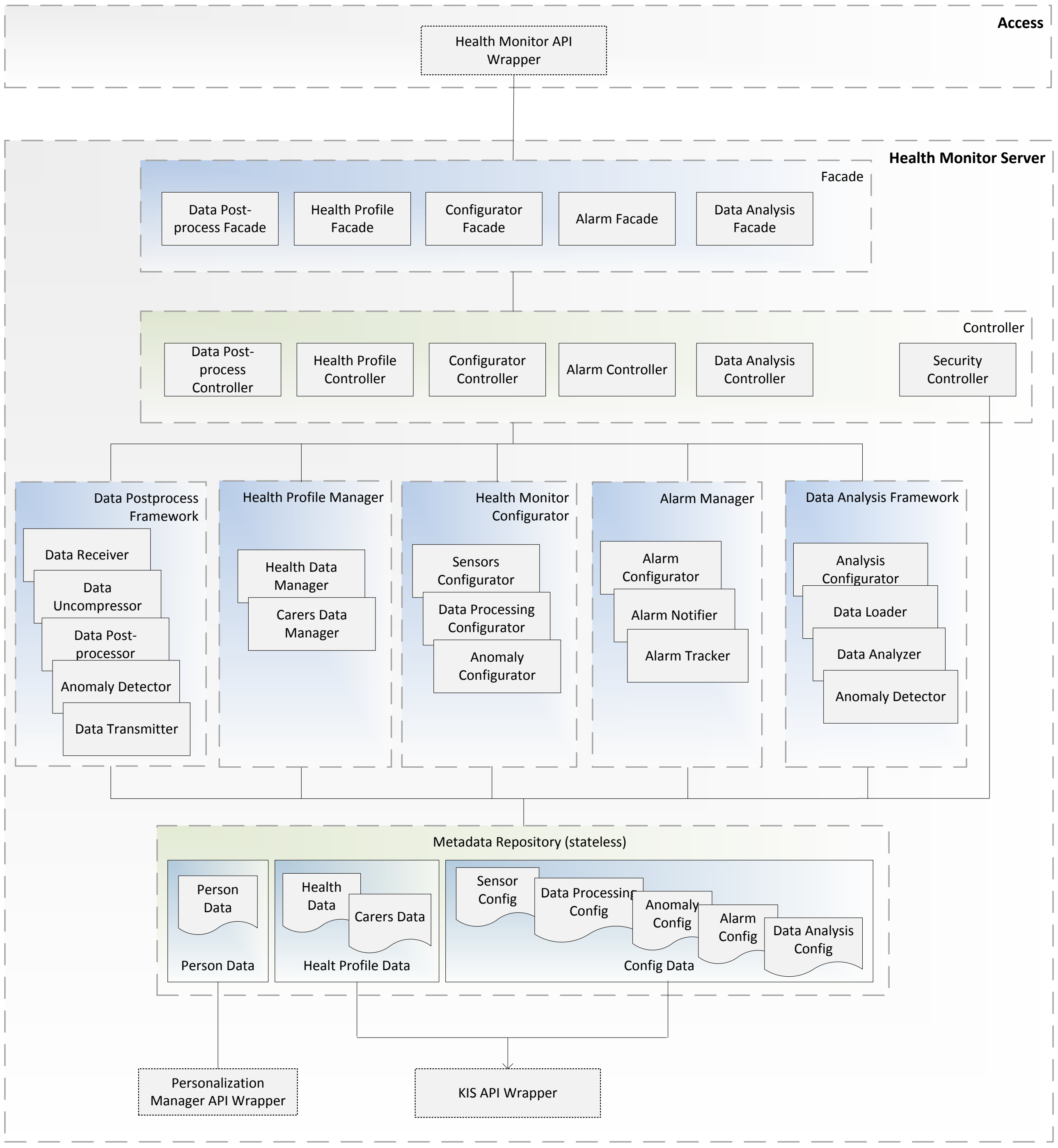

Figure 6: Infographic - Architecture of the Health Monitor Server

These are the main modules of the Health Monitor Server:

- The **Facade** publishes the Health Monitor API.
- The **Controller** controls the main workflow that will follow after a request.
- The **Data Post-process Framework** is the module responsible for receiving sensor data coming from the Health Monitor Client, post-processing it, and storing it.

- The **Health Profile Manager** deals with all user health profile related operations, including health data and carers data manipulation.
- The **Health Monitor Configurator** includes all the HM configurations required for keeping track of user health.
- The **Alarm Manager** is the module responsible for managing alarms (configuration, notification, etc.) due to detected anomalies.
- The **Data Analysis Framework** is the module that provides access to previously recorded sensor data, as well as appropriate tools for analysing them.

In the next sections more details about these modules are provided.

2.3.4.1 The Facade

The Facade is the public interface of the Health Monitor Server, and encapsulates the Health Monitor API as a single point of communication. The provided services are grouped into four categories:

- Data Post-process Facade: transfers pre-processed measurement data from the Health Monitor Client.
- Health Profile Facade: accesses and manipulates the user health profile.
- Configurator Facade: manages HM configurations
- Alarm Facade: manages alarms.
- Data Analysis Facade: retrieves and analyses measurement data.

The Facade component relies on the Controller in order to realise the published functionalities. For each inbound request, the appropriate Facade submodule will be activated. After checking that the request is authenticated and authorized, the Facade submodule will forward the request to its Controller counterpart, as explained in the next section.

2.3.4.2 The Controller

The Controller module captures all valid requests and begins the workflow, calling the appropriate components and orchestrating processing. To that end, it relies on the following internal submodule:

- The Data Post-process Controller, responsible for managing all operations related to measurement data reception and post-processing.
- The Health Profile Controller, responsible for managing all operations related to health data management.
- The Configurator Controller, responsible for managing all operations related to the HM configurations.
- The Alarm Controller, responsible for managing all operations related to alarms.
- The Data Analysis Controller, responsible for managing all operations related to data retrieval and analysis.
- The Security Controller, responsible for checking that the operation is performed by a valid user with appropriate permissions. This submodule is consulted by the Facade after receiving the inbound request and before proceeding with further processing.

2.3.4.3 Data Post-process Framework

This module receives pre-processed sensor measurements coming from the Health Monitor Client through the Data Receiver subcomponent.

Data may be compressed by the Health Monitor Client (see section 2.3.3.3). If this is the case, then it must be uncompressed before proceeding. The Data Uncompressor is responsible for this task.

Data may be post-processed in order to obtain extra measurements for further decision taking. The type of post-processing depends on the measurement data and the associated configuration. The configuration can be obtained from the Metadata Repository (see section 2.3.4.8).

The Data Post-process Framework must work in combination with the Data Pre-process Framework included in the Health Monitor Client (see section 2.3.3.2 for further information).

Post-processed data is checked in order to find anomalies that might trigger alarms. This is the goal of the Anomaly Detector submodule. Information about which anomalies must be checked and how, is available under the Metadata Repository (see section 2.3.4.8).

Finally, data must be transmitted to the final storage system, incarnated by KIS. This is the Data Transmitter submodule's purpose.

2.3.4.4 Health Profile Manager

This module provides access to all operations related to health profile management, that is, all operations related to health and carers data. The following submodules in charge of the different portions of these data can be identified:

- The Health Data Manager submodule provides tools for managing the user’s medical records (e.g. patient diseases, analysis results, history, etc.).
- The Carers Data Manager submodule publishes operations to manage care information, define user-caregiver relationships and define access levels of carers on user data.

Irrespective of the concrete submodule, the managed metadata must be obtained/stored, using the Metadata Repository (see section 2.3.4.8).

2.3.4.5 Health Monitor Configurator

The Health Monitor requires some specific configurations to keep track of the user’s health. This module provides access to all operations related to user health configuration management. In particular, we can identify the following submodules:

- The Sensors Configurator submodule is responsible for managing the configuration of the different sensors. An example of a sensor configuration parameter may be the accuracy level, or the sampling period used to capture measurements. This data could be different depending on the user’s medical history.
- The Data Processing Configurator submodule provides tools for configuring sensor data processing. Again, depending on the user, this data may vary. For example, for some users a comprehensive monitoring process would be required; for other users just average or aggregated values would suffice.

- The Anomaly Configurator submodule is responsible for managing anomaly configurations. The carer should define what is an anomaly based on sensor measurements. This submodule does not include how to notify anomalies. This is the purpose of the Alarm Configurator subcomponent included in the Alarm Manager component (see section 2.3.4.6).

Irrespective of the concrete submodule, the managed metadata must be obtained/stored using the Metadata Repository (see section 2.3.4.8).

2.3.4.6 Alarm Manager

This module is responsible for managing all aspects regarding health-related alarms in the system. On the one hand, the Alarm Configurator submodule provides all operations required to set up and configure alarms (adding, configuring and removing) to the end-user. Alarm configurations include information about the anomaly to be detected, as well as information about how to notify the incidence. All the information about alarm configuration must be stored using the Metadata Repository (see section 2.3.4.8). On the other hand, the Alarm Notifier and Alarm Tracker realise the alarm processing subsystem. The Alarm Notifier receives alerts coming from the Anomaly Detector included within the Data Post-process Framework (see 2.3.4.3) and notifies the incidences to all parties involved. Different notification channels may be used (e.g. e-mail or PUSH notifications), depending on the particular alarm configuration. The Alarm Tracker is an optional submodule that will be included in case the system needs to register the result of notifications (i.e. whether the notification has successfully reached its destination). As stated before, all the information related to alarms, including the information about who must be notified and how, is available under the Metadata Repository (see section 2.3.4.8).

2.3.4.7 Data Analysis Framework

This module supplies appropriate tools for retrieving and analysing the historical sensor measurements, managed by the Data Post-process Framework.

The Analysis Configurator provides tools for setting user preferences useful for the Data Analysis Framework. These preferences are finally obtained and stored in the Metadata Repository (see section 2.3.4.8).

The Data Loader is responsible for loading the data from the persistent store (aka KIS) in an efficient way.

The Data Analyser submodule analyses data according to user preferences, obtaining maximum, minimum and average values as demanded, as well as other required statistical parameters.

The Anomaly Detector identifies a posteriori the anomalies included in the analysed data (presumably detected and successfully notified in real time). With some heuristics, the anomaly detector can recognize patterns in historical data for the early detection of serious diseases.

All results generated by these submodules are prepared for consumption by external components (for example the Web Portal, see section 2.3.5).

2.3.4.8 Metadata Repository

This module provides access to all the metadata the other modules depend on in a homogeneous way. In particular, we are considering the following metadata collections:

- Person data: information about the user.
- Health Profile data: information about the user's health, his/her caregiver, the relationships and granted permissions.
- Configuration data: considering at least the following subcategories:
  - Sensor configuration: information about the sensors available for the user, as well as their proper configuration. This is required for configuring SAF-Health Monitor Client connection.
  - Data Processing configuration: information about how sensor measurements must be pre- and post-processed. This is required for configuring the Data Pre-process Framework included the Health Monitor Client and the Data Post-process Framework included in the Health Monitor Server.
  - Anomaly configuration: information about anomaly detection. This is necessary for detecting anomalies in the Data Post-process Framework.
  - Alarm configuration: information about alarms.
  - Data Analysis configuration: information about how to analyse and present results to users through the Data Analysis Framework.

It should be noted that the metadata available in the Metadata Repository of the server side does not exactly match the metadata available in the Metadata Repository of the client side (see section 2.3.3.4). Each Metadata Repository will include strictly necessary metadata to achieve its consumers' functionalities.

Again, the task of this module is twofold. On the one hand, it uncouples the Health Monitor Server modules from external sources under a common repository interface, hiding the complexities of external APIs under a comfortable homemade interface. On the other hand it provides symmetry with the Health Monitor Client, combining into the same component a collection of data semantically related.

It should be noted that the Metadata Repository included in the Health Monitor Server is stateless (unlike the Metadata Repository included in the Health Monitor Client, see section 2.3.3.4 for further information). This means that it does not provide cache capabilities; it only represents means to access metadata in a homogeneous way.

All the information will ultimately be obtained from either the Personalisation Manager or the KIS.

2.3.5 Web Portal

The Web Portal provides web-based access to the HM services. The architecture is depicted in the following figure (see Figure 7).

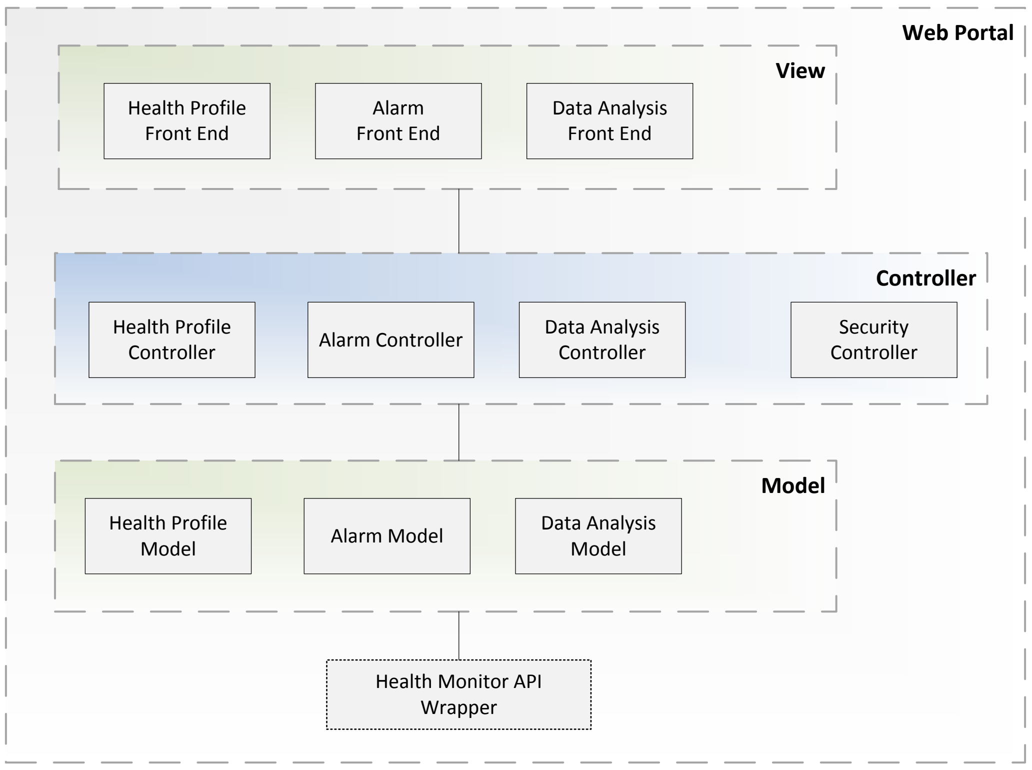

Figure 7: Infographic - Architecture of the Web Portal of the HM

A MVC (Model View Controller) pattern is used for the Web Portal architecture:

- The **View** provides access to the web application, receiving user inputs and rendering appropriate outputs
- The **Controller** takes control of the request and begins the workflow in order to generate an adequate response to each request
- The **Model** encapsulates the actual data/services requested by the user. In our case, all data/services are finally implemented in the Health Monitor Server, and interpreted by this component

In the next sections, these modules are further described.

2.3.5.1 The View

The View is in direct contact with the final user. It manages the inputs and displays the outputs received from the other modules.

The View can be divided into three submodules:

- The Health Profile Front End is the submodule responsible for receiving and displaying information related to the user's health data. This includes tools for medical caregivers to define and review patient medical records, as well as configurations for controlling its health.
- The Alarm Front End is the submodule responsible for receiving and displaying information related to the alarms. This includes tools for carers to define and configure anomaly detection rules and alarms.
- The Data Analysis Front End is the submodule responsible for receiving and displaying information related to the measurement data and its analysis. This

includes tools for caregivers to define and configure the data processing rules. It also includes tools for real time monitoring as well as historical data mining.

To achieve these functionalities, the View is in direct communication with the Controller.

2.3.5.2 The Controller

The Controller orchestrates the process that takes place for every inbound request received through the view. We can identify the following submodules:

- The Health Profile Controller is responsible for controlling all health data related requests.
- The Alarm Controller is responsible for controlling all alarm related requests.
- The Data Analysis Controller is responsible for controlling all data analysis related requests.
- The Security Controller is responsible for guaranteeing that requests are valid (essentially, that they come from authenticated users).

For each request, the Controller will have to implement a particular workflow, that is, a sequence of steps, most of which will involve interacting/manipulating the Model in some way.

2.3.5.3 The Model

The Model encapsulates the atomic services provided by the HM in a way that allows simple interaction from the Controller component. To that end, it is internally composed of three submodules:

- The Health Profile Model includes the atomic services related to health data management.
- The Alarm Model includes the atomic services related with alarm management.
- The Data Analysis Controller includes the atomic services related to data analysis management.

2.4 Context-aware Speech Recognition

2.4.1 Overview

The Context-Aware Dialogue Engine (CADE) is responsible for spoken interaction between ALFRED and the end user. It allows the user to give spoken queries and commands, interprets the utterances and provides verbal feedback and responses. It also identifies ALFRED apps corresponding to the user's intention. CADE interfaces with other ALFRED components in order to fetch app resources, request information and trigger activities. CADE consists of the following subcomponents:

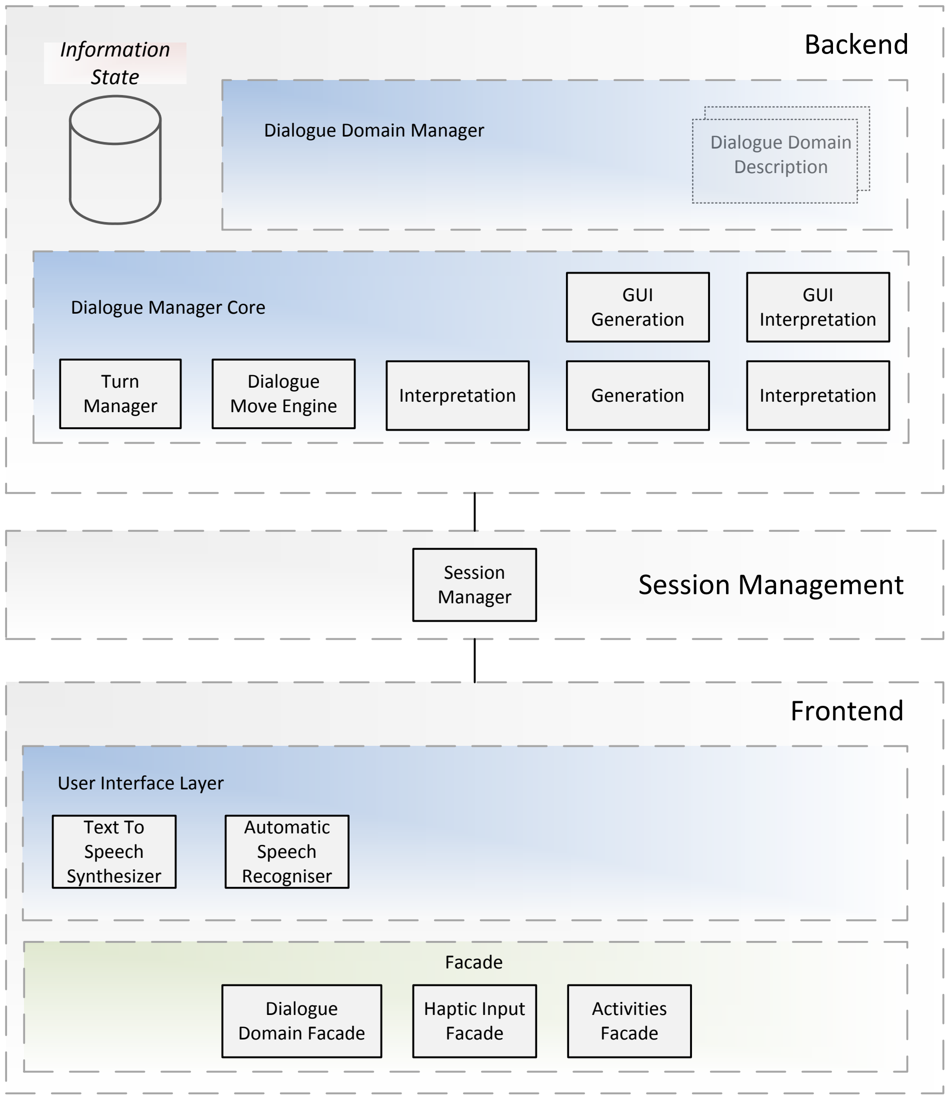

Figure 8: Infographic - Architecture of the Context-Aware Dialogue Engine

- **Frontend:** mainly consisting of an automatic speech recognizer and a text-to-speech synthesizer. It also provides the facades to other ALFRED components.
- **Backend:** consisting of dialogue move engine, natural-language interpreter, etc.
- **Session Manager:** providing each frontend with access to a backend, and routing communication between frontend and backend.

The subsequent subsections describe the CADE subcomponents in more detail.

2.4.2 Frontend

The CADE Frontend provides the user interface layer of the spoken dialogue support in ALFRED. It runs locally on the ALFRED device and consists of the following parts:

- Automatic Speech Recognizer (ASR)
- Text-To-Speech Synthesizer (TTS)
- Dialogue Facade (see subsection 2.4.2.1)

The ASR listens to the microphone and returns a textual hypothesis of what the user has said, along with confidence values. Inversely, the TTS receives text to be spoken and outputs it via the mobile device's speakers.

Within the ALFRED project, neither ASR nor TTS engines will be developed. Instead, already existing speech engines will be adopted and wrapped within the frontend subcomponents.

2.4.2.1 Dialogue Facade

The CADE Frontend provides the dialogue facade, enabling other ALRED components to trigger dialogues. For example, the Event Manager can invoke a service in the facade in order to trigger ALFRED to ask the user about participating in an upcoming event. The facade also receives notifications from the graphical user interface, e.g. when the user has clicked the push-to-talk button.

2.4.3 Backend

The CADE Backend contains the main logic governing spoken dialogue interaction between ALFRED and the user. It typically runs on a server, and consists of the following parts:

- **Information State:** Contains dialogue related information such as dialogue history, issues currently under discussion, and beliefs shared by the user and the system.
- **Interpretation:** Produces semantic interpretations of the user utterances recognised by the ASR.
- **Dialogue Move Engine (DME):** Uses dialogue plans defined in Dialogue Domain Descriptions (see below) in order to identify the user's intention and find corresponding apps. The DME may ask follow-up questions to the user and request information from other ALFRED components. It may also request to start or stop activities in other ALFRED components.
- **Generation:** Produces textual realizations in natural language from semantic representations of dialogue moves that the DME has selected.
- **Turn Manager:** Distributes the "turn" (the right and opportunity to speak) between the user and the system.
- **Dialogue Domain Description**: Describes the ontology, the plans and the grammar of a particular dialogue domain. The ontology defines concepts, entities and actions that the user and the system may reference in questions, answers and requests. The dialogue plans describe how actions are carried out and how questions are answered. Plans also describe what information is needed in order to

carry out the actions or answer the questions. The grammar defines mappings between linguistic surface forms and semantic entities. In ALFRED, there will be a Core Domain Description, describing the dialogue domain for the core functionality of the assistant, as well as app-specific domain descriptions. The Core Domain Description will be relatively small, as most functionality in the assistant will be provided by the apps. The app-specific domain descriptions will be stored in the ALFREDO Marketplace and will be defined by app developers in a high-level format such as XML.

- **Dialogue Domain Manager:** Contains Dialogue Descriptions for specific ALFRED apps.
- **GUI Interpretation:** Produces semantic representations of input received from the GUI.
- **GUI Generation:** Produces graphical descriptions from semantic representations of dialogue moves that the DME has selected. In ALFRED, this may be used for generating clarification buttons when the system is uncertain what the user said.

2.4.4 Session Manager

The frontend and the backend of CADE communicate via the Session Manager. When a new frontend connects to the Session Manager, it first ensures that there is an available backend that can serve it. It then associates them to each other and routes subsequent messages in both directions

2.5 ALFREDO – Marketplace

2.5.1 Overview

One of the main achievements of the ALFRED project will be the delivery of new functionalities for older end users. Those new functionalities will be provided by installing new applications, called extensions, on the ALFRED system. The place where end users can find those extensions will be the ALFREDO Marketplace.

Additionally, the Marketplace will provide for a web based user interface for developers of ALFRED extensions. This web application will be usable from any modern web browser and will allow developers to register and login to manage the extensions that they have uploaded to the ALFREDO Marketplace. The web application will also provide testers with tools to review applications in order to accept or reject their publication in the ALFREDO marketplace.

The ALFREDO Marketplace is divided into two subcomponents (see figure 9):

- The **Backend** is responsible for the business logic of the ALFREDO Marketplace and will provide a façade to communicate with other ALFRED components. It consists of the modules API Component, APP Manager and Web façade.
- The **Frontend** is divided into two modules Web Component and Mobile Component. It will be responsible for offering the façade between end user and the ALFREDO system

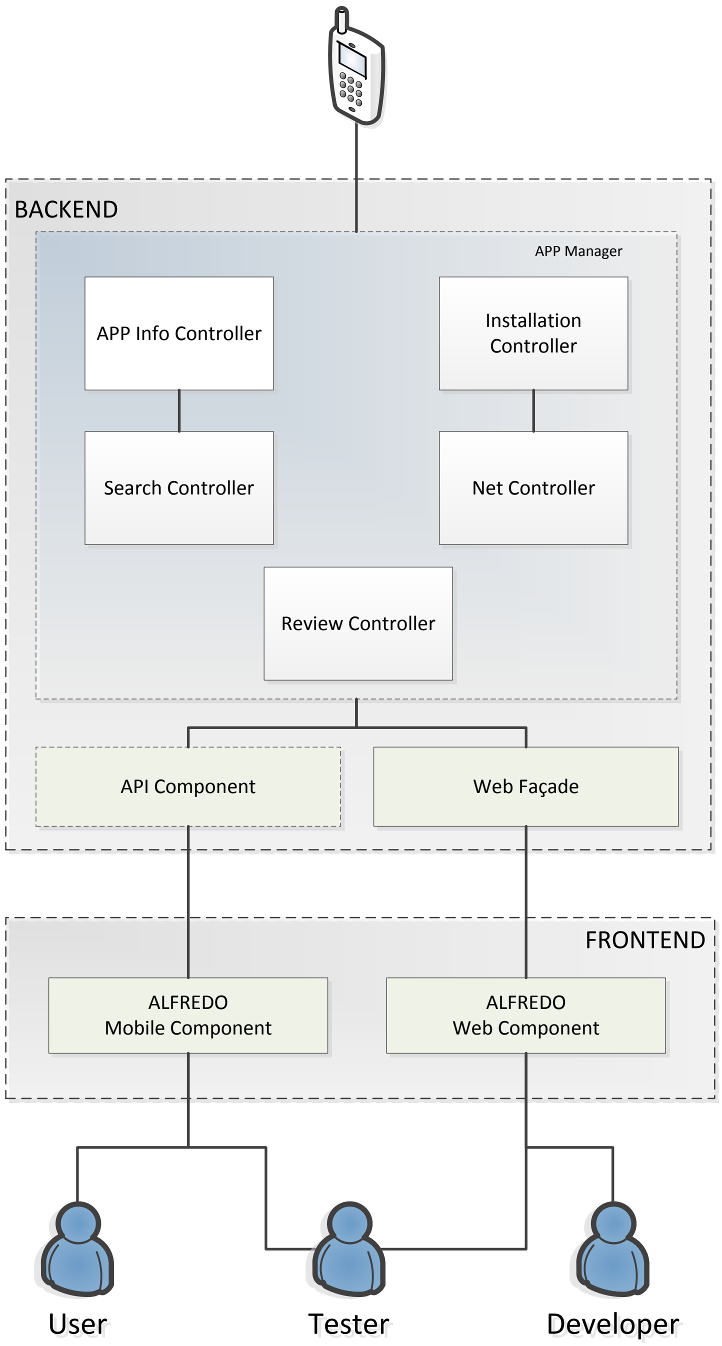

Figure 9: Infographic – Architecture of the ALFREDO Marketplace

In the following subsections, the two segments Backend and Frontend will be discussed in detail.

2.5.2 Backend

The backend encapsulates the business logic for application management and the communication with other ALFRED components. This description covers mainly these two functions.

The business logic will be represented by the **APP Manager**. It will act as the core of the Marketplace, responsible for the operations requested by other components of the ALFRED system, such as the Game Manager or the Context-aware Dialogue Engine. The App Manager is composed of the following subcomponents:

- **APP Info Controller** is responsible for requesting application information from the device and from the Knowledge and Information Storage (KIS) and provide it to other ALFRED components.
- **Search Controller** is a key component which will manage the search of applications on the ALFRED system. This controller will forward the search query to the KIS component.
- **Net Controller**, responsible for uploading and downloading applications from the net to the ALFRED device
- **Installation Controller**, responsible for installing and uninstalling the applications on the ALFRED device
- **Review Controller**, responsible for the review process of applications. Only accessed via web by testers

The backend is also responsible for the communication with other ALFRED components. To provide easy application management access, the backend will encapsulate the logic of the operations in an **API Component.** The main responsibility of this component will be to handle all the requests from other ALFRED components and allocate the required tasks to the APP Manager. Vice versa, the API Component will handle the responses from tasks executed by the APP Manager, returning the right feedback to the ALFRED component which has performed the call.

Finally the backend encapsulates all the requests from the web interface. The **Web façade** is responsible for handling web requests, in a similar manner to the API component. The Web façade will provide a public interface to expose the methods of the ALFREDO **Web Component** (see section 2.5.3.1)

2.5.3 Frontend

The frontend segment will provide the user interface layer. .

The Marketplace will be accessible through a web based user interface and a mobile user interface.

Developers and testers will be able to access the Marketplace via web. . Developers will be able to upload, update or remove their own applications. Testers will be able to pass test cases to the uploaded applications in order to accept or reject them. These two user roles will be described in the next section.

The older end users will be able to access the Marketplace via their ALFRED device. They will be able to install, uninstall, upgrade and search for applications.

2.5.3.1 ALFREDO Web component

The Web component is the web interface for ALFREDO marketplace.

From a technical perspective, this component will be realized as a web application providing a web based UI for service providers and consumers. It will target technical experts (developers and testers) and will offer tools to submit or review new applications for older people.

- The component provides for two roles: The Tester is responsible for testing and subsequently approving or rejecting the submitted applications by developers
- The Developer is the application publisher. He will be able to upload a new application or a new version of an already published application

2.5.3.2 ALFREDO Mobile component

The Mobile component will run on the ALFRED device and will communicate with the backend through the API Component.

As mentioned before, end users will be able to perform the operations to install, uninstall, update, rate and search applications within their ALFRED device.

This mobile component will be integrated within the ALFRED system providing a mobile based user interface. This will allow end users to discover, buy, install, upgrade and uninstall applications via this component.

2.6 Personalization Manager

The Personalisation Manager will provide the older end user with a personalized experience . This component will host specific information of users (i.e. older person) and contribute to the privacy assurance. The gathered information is relatively static, including, personal information of the older person (e.g., age, marital status, gender, nationality, family members, contact details, etc.),health related information (e.g., allergies, medical history, known conditions or medications, etc.), and personal preferences for applications and the installed ALFRED applications. These persisted data will be available via the Personalisation Manager using web services, while safeguarding the privacy of the older person. The privacy assurance will be achieved by examining the privacy access levels of requesters and the access level of the requested information. The requesters can be individuals related in some manner with the older person such as carers, family members, doctors, nurses, etc.

Moreover, this component will provide personalized suggestions for events that can be of interest for the older person. The recommendations can work both pro-actively (e.g., provide recommendations every 12 hours) and re-actively (e.g., respond to inputs such as “Find me a bingo game in Berlin for tomorrow”). The resulted recommendations will be based on information such as:

- The personal profile of the older person (e.g., recommend events close to the home address of the user).
- Health related information (e.g., would not recommend physically demanding events to a user with a heart condition).
- Social networks context (e.g., events that friends of the older person are participating in).
- A knowledge base of events (e.g., social, athletic, musical, etc.).
- And possible user input when he/she asks for recommended events.

The Personalisation Manager will consist of four main building blocks as is depicted on the high-level architecture infographic below (see Figure 10).

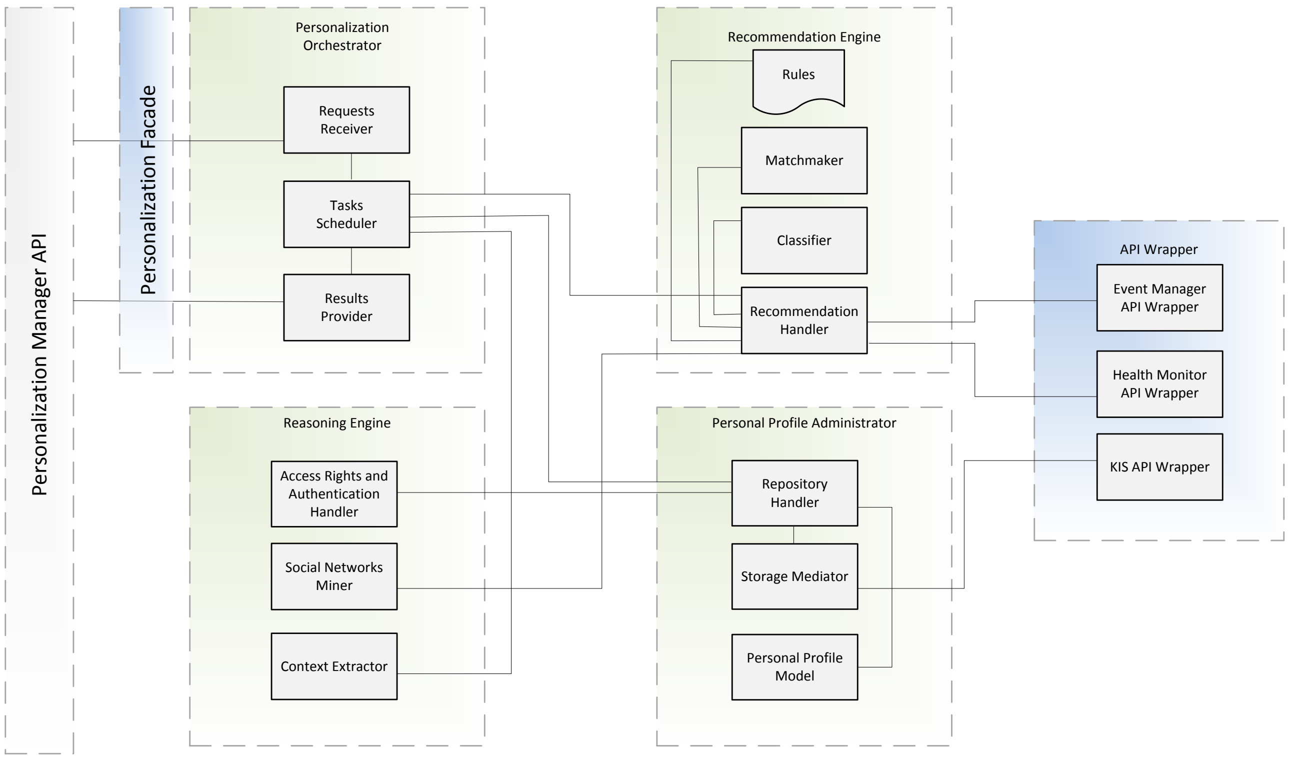

Figure 10: Infographic - Architecture of the Personalization Manager

A short description of the sub-components is presented below and more in depth description follows in the next sections (2.6.1 – 2.6.4).

- The **Personalization Orchestrator** will be the interface of the Personalisation Manager component with the other components and will be responsible for the allocation of specific tasks to the appropriate sub-component (e.g., Recommendation Engine, Personalization Profile Administrator, Reasoning Engine)
- The **Recommendation Engine** will provide recommendations for events which are of interest to the older person based on his profile, preferences history, social and location context, health condition, etc.
- The **Personalization Profile Administrator** will provide the appropriate models/information schemas to store profile information of the older person such as name, family status, contact details, blood type, known significant medical issues, etc. It will store the actual information in the KIS component and will perform CRUD (short for "Create, Read, Update, Delete") operations on them. The Personalization Profile Administrator will be responsible for the delivery of information to other components, with the help of the Reasoning Engine.
- The **Reasoning Engine** will be responsible for using the semantically rich information stored in KIS via the Personalization Profile Administration. The engine will assist both sub-components of the Personalisation Manager and external components (such as CADE) to take "intelligent decisions" and actions through intelligent reasoning.

2.6.1 Personalization Orchestrator

The Personalization Orchestrator will act as an interface between the internal sub-components of the Personalisation Manager (Personalization Profile Administrator, Data Mediator and the Recommendation Engine) and the other components of the ALFRED system such as the HM, the Knowledge and Information Storage and the Personal Assistant. The Personalization Orchestrator will obtain inputs and requests through the Personalisation Manager API and be responsible for the orchestration and allocation of the required tasks. Any associated inputs to the appropriate sub-component will be provided through the Personalization Orchestrator subcomponent. It will provide back to the requester the results.

2.6.2 Recommendation Engine

This component is going to enhance the personalized ALFRED experience with recommendations tailored to the personal profile of the user (older person) and his/her context, based on parameters that include social, locational and health information. It will work pro-actively, providing for recommendations in intervals or on specific locations, according to the user settings. Additionally it will react on specific commands (e.g., provide recommendations based on user commands such as “find me swimming courses in Berlin”). The recommendations will be triggered in both cases by the Personalization Orchestrator and the results will be transferred through the Personalization Orchestrator to the Personal Assistant (or another app that acts as a mediator),presented through its Suggestion sub-component.

The recommendation engine will include the results of extensive mining of social networks, to provide social context aware recommendations. This is motivated by the fact that both the ALFRED objectives as well as the defined use cases consider the social inclusion extremely important for its users (older persons). The mining of social networks will be performed by the Reasoning Engine. Health related aspects will be taken into account when suggesting events and activities, as well as mobility levels and accessibility.. Last but not least, user preferences will be used to recommend activities and based on previous choices that results will be refined. These criteria will be defined in forms of rules and will be used in statistical models and matchmaking algorithms that will be applied on the knowledge base/data which includes available events, user inputs, personal profile and historical data, social and locational context.

2.6.3 Personalisation Profile Administrator

This component will be critical for the personalized experience of the user. It will handle all information related to the older person based on rich models/schemas explicitly representing personal information, contact information and significant medical information (e.g., allergies, serious conditions, medications the older person is using, etc.). Additionally, users (or else, stakeholders) related to the older person such as family members and carers will be handled through the Personalization Profile Administrator as well as their specific privileges (data access levels) associated to the older person.

User settings and preferences concerning the behaviour of applications and functionalities of the ALFRED system will be handled through the Personalization Profile Administrator along with the interaction data of the user with the system.

Both internal processes in the Personalisation manager as well as other ALFRED components (HM) will consult the Personalization Profile Administrator. For example, internally the Recommendation Engine will use personal profile information to provide recommendations tailored to the user. On the other hand, through the Personalization Orchestrator, a query for medical history of an older person may be received for examining if indications of increased heart rate should trigger an alarm.

The Personalization Profile Administrator sub-component acts as a mediator to the data which are handled in the KIS. Through the KIS API it will perform CRUD operations on the data, provided that with the help of the Reasoning Engine it is determined that the requester of the CRUD operation has the access rights for the operation.

2.6.4 Reasoning Engine

The Reasoning Engine subcomponent will act as a decision-making mechanism, based on personal information. It contributes to the functionalities of the Personalization Manager. This description covers mainly three processes that are performed thanks to the Reasoning Engine.

- First, it contributes to the privacy assurance by determining the access rights level of data requesters. The personal profile of older persons includes very sensitive information (e.g., medical history) as well as less sensitive (e.g., personal settings and preferences). These data will be “characterized” by different levels of access privileges. Also, users related to the older person will have access levels specified. So when a user requests information about an older person, the Reasoning Engine will be in charge of comparing the user access level to the one of the information he/she requested, and decide if the information should be provided or not.
- Second, it helps with the determination of context in certain older person “commands” to ALFRED. For example, the older person asks ALFRED to call her daughter with the following verbal input “*Call my daughter*”. This input to the CADE component (see section 2.5) will be treated with the Personalisation Manager by retrieving the personal information of Helen. The Reasoning Engine will determine who Helen’s daughter is and what her contact details are. This depends heavily on the personal profile model description requiring rich semantic information to allow such reasoning.

Finally, the Reasoning Engine will capitalize on social networks of the older person that are defined in his/her social networks account (for which ALFRED will need certain access rights) to provide social context for the user. For example by mining “social networks friends” of the older person and extracting events they published, the Reasoning Engine can the Recommendation Engine with events to suggest to the user (e.g., “Helen your friend Emma is going to a bingo game that is taking place 2 km from your home”).

2.7 Event Manager

The Event Manager is responsible for building and maintaining a Knowledge Base[1] of events (e.g., social, athletic, musical, and educational, etc.). This Events Knowledge Base will be used by the Recommendation Engine of the Personalisation Manager (see section 2.6) to provide personalized selections of events that the older person could be interested in. This will help older persons to stay engaged and active. The population of the Events Knowledge Base with events can be manual or semi-automatic.

Manual population concerns cases where users enter events through a web portal. Semi-automatic population concerns the use of web crawlers, data processing and user evaluation of the extracted events in order to populate the Knowledge Base (the interaction also takes place through a web portal).

In the context of the Event Manager, we consider three types of users.

The Personalisation Manager which communicates with the Event Manager through its API and can only read events.

The other two users are the “simple users” and the “reviewers” – both interacting with the Event Manager through its GUI sub-component. The simple users can enter events, or mine events in the web and select which should be used in the Events Knowledge Base. However, the events that will actually be considered “valid” and can be used for recommendations of events to older persons will be only those that are examined and accepted by the reviewers. This reviewing process is performed to protect the Events Knowledge Base from simple users that would attempt to enter malformed events in it. Also the users that act as reviewers, have administrative rights on the Web locations that will be crawled when mining for events and can perform all CRUD operations in the Events Knowledge Base. The Event Manager is composed by 4 main building blocks as is depicted in the figure below:

[1] A Knowledge Base is a centralized repository for information

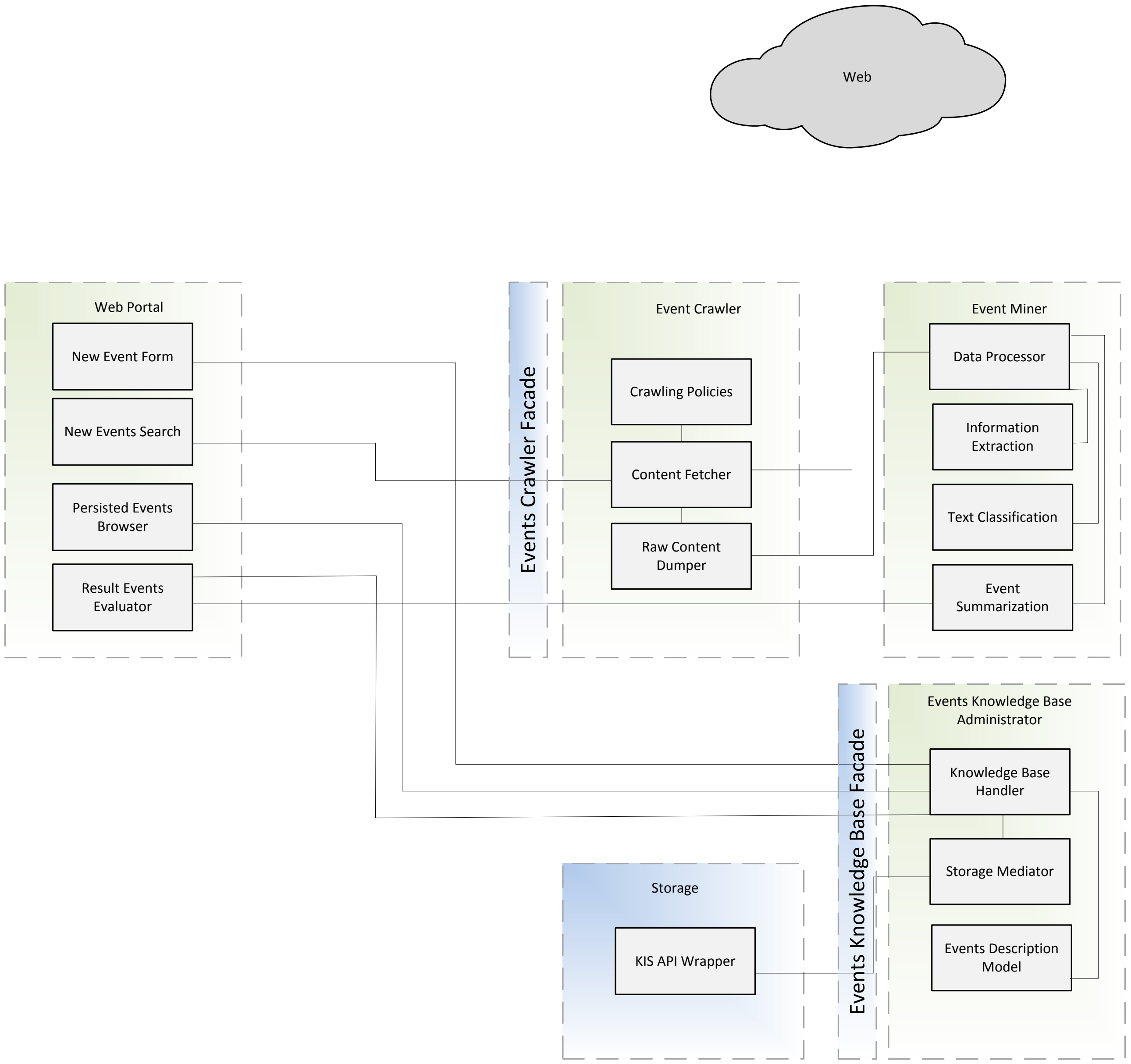

Figure 11: Infographic - Architecture of the Event Manager

The main building blocks of the Event Manager component are briefly described below with a more detailed description following in the next sections (2.7.1 -2.7.4).

- The **Web Portal** provides a GUI for users to enter events or search for events based on various criteria (e.g., location, type of event, date, etc.) and then select results that they would like to save in the Events Knowledge Base. Moreover users can browse for events already stored in the Events Knowledge Base.
- The **Web Crawler** fetches content (in form of text) from specified locations on the Web related to events. The fetched content is passed for further processing to the Event Mining sub-component

- The **Event Miner** processes the fetched contents to extract event descriptions and provides a summary to the user. It parses the description in the appropriate form for the Events Knowledge Base.
- The **Events Knowledge Base Administrator** defines the information model of the event data and is responsible for CRUD operations to the events

2.7.1 Web Portal

The Web Portal sub-component will provide a user interface for users to populate the Events Knowledge Base with specific events, or trigger the crawling of events based on one or more criteria (such as type of event, location, and time period, etc.). Events originating from non-administrator users have to be evaluated first by reviewers before the events are actually considered valid, while events confirmed by reviewers are stored in the Events Knowledge Base. In the case of manually inserting the event, the form will include all the necessary information to populate the model for the description of events. In the case of semi-automatic extraction of events from the web, the resulted events will be presented to the user and she will decide, which of them should be added to the Events Knowledge Base. Finally through the Web portal all users (both regular and reviewers) can browse the events that are stored in the Events Knowledge Base. The reviewers have the administrative rights in the Events Manager, including the rights to evaluate events that are proposed by regular users and the rights to perform CRUD operations upon the Events Knowledge Base. In essence, reviewers are in charge of determining which events will be sent to/used by the Personalization Manager.

2.7.2 Web Crawler

The Web Crawler subcomponent will fetch the contents from a pre-defined set of web locations (as defined by the selection policy). This process is initiated by a user via the Event Manager’s Web Portal. The user will also be able to arrange an interval policy to regularly check and fetch the contents for a set of selected URLs. The Web Crawler subcomponent has the purpose to retrieve contents and to pass the returned contents to the Data Mining subcomponent for further processing. It’s worth mentioning that the Web Crawler will be designed to respect politeness policies which aim to ensure that the crawler does not overload websites with its requests.

2.7.3 Event Miner

The Event Miner subcomponent is the most sophisticated part of the Event Manager component and its tasks start when raw content is passed on from the Web Crawler. The Event Miner parses the raw content (mainly text) and processes it. It is out of this document’s scope to get into the specific details of this processing, but on high level it will perform text classification to identify the type of events and information extraction to identify locations, dates, times, etc. At this point the user’s entries and criteria can be used to filter the results to those that are matching to the user’s inputs (e.g., events that take place in Berlin only). Finally, the extracted event is summarized and presented to the user. If the user indicates the event as relevant, it will be persisted in the Events Knowledge Base.

2.7.4 Events Knowledge Base Administrator

The Events Knowledge Base Administrator sub-component is responsible for creating, retrieving, updating and deleting data from the KIS component where the events are stored. These events are made available through the Events Knowledge Base Administrator in 2 cases.

- The persisted events are available through the Web Portal for users that potentially want to review the data already saved in the Knowledge Base.
- The Events Knowledge Base Administrator will be available as a service for the Events Manager API. It will be used by the Personalisation Manager as a client, searching events from the knowledge base and matching with other parameters such as personal profiles of the older persons (among other parameters). Finally it recommends events tailored to the likings and profile of the user.

2.8 Game Manager

The Game Manager (GM) component is responsible for managing a specific subset of the application extensions to the ALFRED personal assistant, namely the (serious) games. Depending on the criteria applied, these games can be grouped into various categories, such as single-player and multiplayer games, games for cognitive impairments prevention and games for physical impairments prevention. In the context of this document and for the sake of simplicity, it is sufficient to differentiate between two types of games: games that run entirely on the mobile device and do not require additional hard- or software (mobile games), and games that make use of external devices, such as specialized sensors and TV screens (indoor games).

In a sense, the GM links these games to the ALFRED application, and vice versa. It runs locally on the user's mobile device and interfaces several other components of the Personal Assistant, such as the SAF, the PM, and the SM. This is necessary, as the GM fills multiple roles:

- **Game management:** When a game client is downloaded from the ALFREDO marketplace and installed on the device, it is automatically registered by the GM with a profile describing its specific characteristics (e.g., number of minimum players, certain physical capability requirements for playing the game). Over time, the GM enriches game metadata with additional information, such as gameplay duration, user performance and results. This metadata is used to improve the game configuration settings and to make more sophisticated game suggestions (see below). Some anonymous metadata may also be send back to the MP in order to improve customer satisfaction. For instance, the average usage times of games can be used as a type of user rating that is superior to the custom five-star user ratings as known from conventional online software distribution platforms.
- **Game suggestions:** The GM suggests games to the user depending on the current situation. In its simplest form, this could mean that the GM frequently suggests random games at a specific time of day or when the user is at a specific location. More sophisticated versions of the GM will combine dynamic contextual information on the user's current activity and wellbeing (acquired, for example, via the SAF) with static data such as user preferences and game metadata in order to identify the optimal moment to make a game suggestion.
- **Game control:** The GM's main task lies in configuring and starting games, and in monitoring the user's performance during gameplay. When the user decides that she would like to play a game (either following a suggestion made by the GM or by another user, or because she initiates a game session herself by issuing the corresponding command to the Personal Assistant), the GM configures the game using dynamic context information and static metadata (very similar to the game suggestion process). The corresponding options for this configuration must be provided by the games (for example, by allowing for different text font sizes). If the game in question requires specific external devices to function, the GM is responsible for establishing and maintaining the communication with these, for example by using the mobile device's Bluetooth module. Finally, the GM is responsible for monitoring the user's performance during gameplay and for storing this data (as it is subsequently used for improving the game suggestion and configuration process, see above).

According to the GM's roles as described above, the GM is internally made up of four "modules", and three types of APIs which are used by the modules to communicate with other internal components, external (i.e., cloud-based) components, and external devices. The following Figure 12 shows a logical depiction of these modules and APIs.

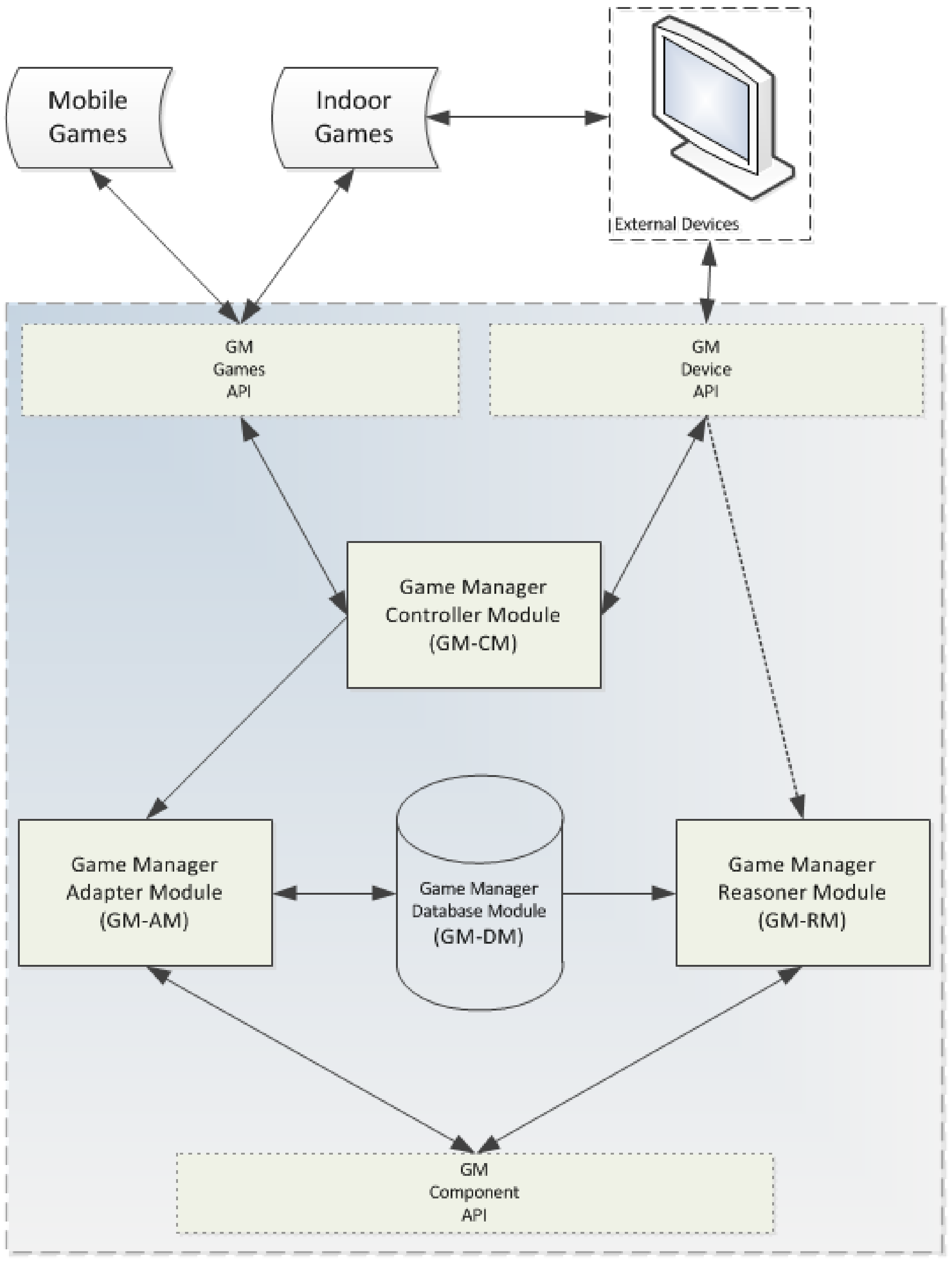
Mobile Games
Indoor Games
External Devices
GM Games API
GM Device API
Game Manager Controller Module (GM-CM)
Game Manager Adapter Module (GM-AM)
Game Manager Database Module (GM-DM)
Game Manager Reasoner Module (GM-RM)
GM Component API

Figure 12: Infographic - Game Manager Modules

- **Reasoner Module (GM-RM)**: The GM's Reasoner Module combines information gathered from the PM, HM, CP and the GM's Database Module (see below) in order to (a) identify situations in which a game should be suggested to the user and (b) to select an appropriate game for the specific situation. If the Reasoner Module detects a suited combination of the contextual situation and the games currently available, it triggers the SM in order to present a suggestion of a game to the user. A more sophisticated version of the Reasoner Module could additionally take into account information acquired from the SAF, thus also reacting to the user's physical wellbeing as determined by vital sensors.
- **Controller Module (GM-CM)**: The Controller Module is the part of the GM that communicates directly with the ALFRED games, both mobile and indoor, by way of the GM Games API. It is responsible for starting and stopping a game on request of the user, and for gathering feedback from the game regarding user performance. This data is forwarded to the Adapter Module, which in turn then pushes it into the Database Module. If required (in the case of indoor games, see below), the Controller Module also establishes a connection to external devices via the GM Device API.
- **Adapter Module (GM-AM)**: The Adapter Module is responsible for using information from the PM, HM, CP, and the GM's Database Module (see below) to adapt/configure/initialize games that the user has started. It also maintains the GM's Database Module and communicates with the MP via the GM Component API and the Game Manager API Client of the ALFREDO Marketplace in order to receive game description profiles (see below) and to make some (anonymous) information, gathered in the user's local database, available to the MP.
- **Database Module (GM-DM)**: The GM's internal Database Module gathers all information about game clients currently installed on the device (both mobile and indoor). The Adapter Module (see above), which is responsible for maintaining the Database Module, associates the “vanilla” game profiles as loaded from the ALFREDO MP during game installation with player usage statistics over time, such as how the player has performed in various game sessions, how she rated the game when asked for, when she accepted and when she declined suggestions to play this specific games, and whom she has played the game with (in case of multiplayer games).

Besides the Game Manager, ALFRED's serious games pillar (pillar IV) of course also includes a set of games. As already pointed out above, these games can be grouped into the categories of mobile games and indoor games. Since indoor games require external devices in order to be playable, these devices can also be considered as being part of ALFRED's game suite (although they are not directly associated to the ALFRED project).

- **Mobile Games:** All games that can run directly on the ALFRED mobile device and that do *not* require additional hardware (including WiFi-connection) are considered “mobile games”. Mobile games can be both single- and multiplayer games, they can be serious games (such as games providing for cognitive decline prevention) or pure entertainment games, and they can be of any genre (puzzle, strategy, reaction, and so forth). Mobile games will make up the bulk of ALFRED's game suite. They can communicate with the ALFRED Game Manager using the GM Games API (more specifically, they communicate with the Game Manager's Controller Module). Some of these games may rely in part or entirely on voice commands, and thus require the CADE to forward such commands to the game. Some mobile games may also be biofeedback games and make use of the player's vital parameters, such as her heart rate, to adapt the game mechanic. To this end, these games will pull such information from the SAF (if the required data is available).

- **Indoor Games:** Contrary to mobile games, indoor games require additional hardware (and possibly software running on that external hardware) to function. As most of such hardware will be non-moveable (such as WiFi routers, TV screens, or bicycle ergometers), the games are essentially limited to the (indoor) locations where these devices are deployed at – hence the name. Just like mobile games, indoor games can receive push messages from the CADE framework to react to voice commands issued by the user, and they may pull sensor data from the SAF to react to a user's changing vital parameters. Additionally, these games will also rely on the GM's Controller Module to establish and maintain a communication with the external devices via the GM Device API.

External Devices: For some types of games, it may be reasonable to rely on additional hardware in order to deliver a satisfying game experience. This holds especially true for various types of exergames, games that require physical activity from their players. Such games usually rely on sensor hardware to track a user's movements (examples for this are camera-based approaches such as Microsoft's Kinect, or sensor mats such as Nintendo's Balance Board). While the ALFRED project will not provide for any such hardware itself, some of the games developed for the ALFRED solution may rely on external, commercially available sensors and actuators. The Controller Module of the Game Manager component uses the GM Device API to communicate with external devices

2.9 Knowledge and Information Storage

2.9.1 Overview

The Knowledge and Information Storage (KIS) will be used for the privacy-aware data management in the ALFRED system. Data will be stored in so called “buckets”. These small data storage units can be used to store data isolated from each other based on the origin. This allows a good privacy management of the data because it enforces that a client of the KIS has only access to data in a specific bucket. This principle is also known as a sandbox in other ICT domains.

The KIS is the central storage point of all data in the ALFRED system. The KIS is segmented into three parts (see Figure 13):

- The **User Data** segment contains all concrete databases, which will store the actual user data. These databases will be based on different data structure concepts, such as NoSQL, semantic, binary.
- The **Cloud Storage** segment contains all components, which will provide the actual business logic for the KIS.
- The **Access** segment contains concrete implementations of API wrappers for different target platforms in order to easily access the functionality of the KIS from a specific platform, e.g. for the target mobile platform.

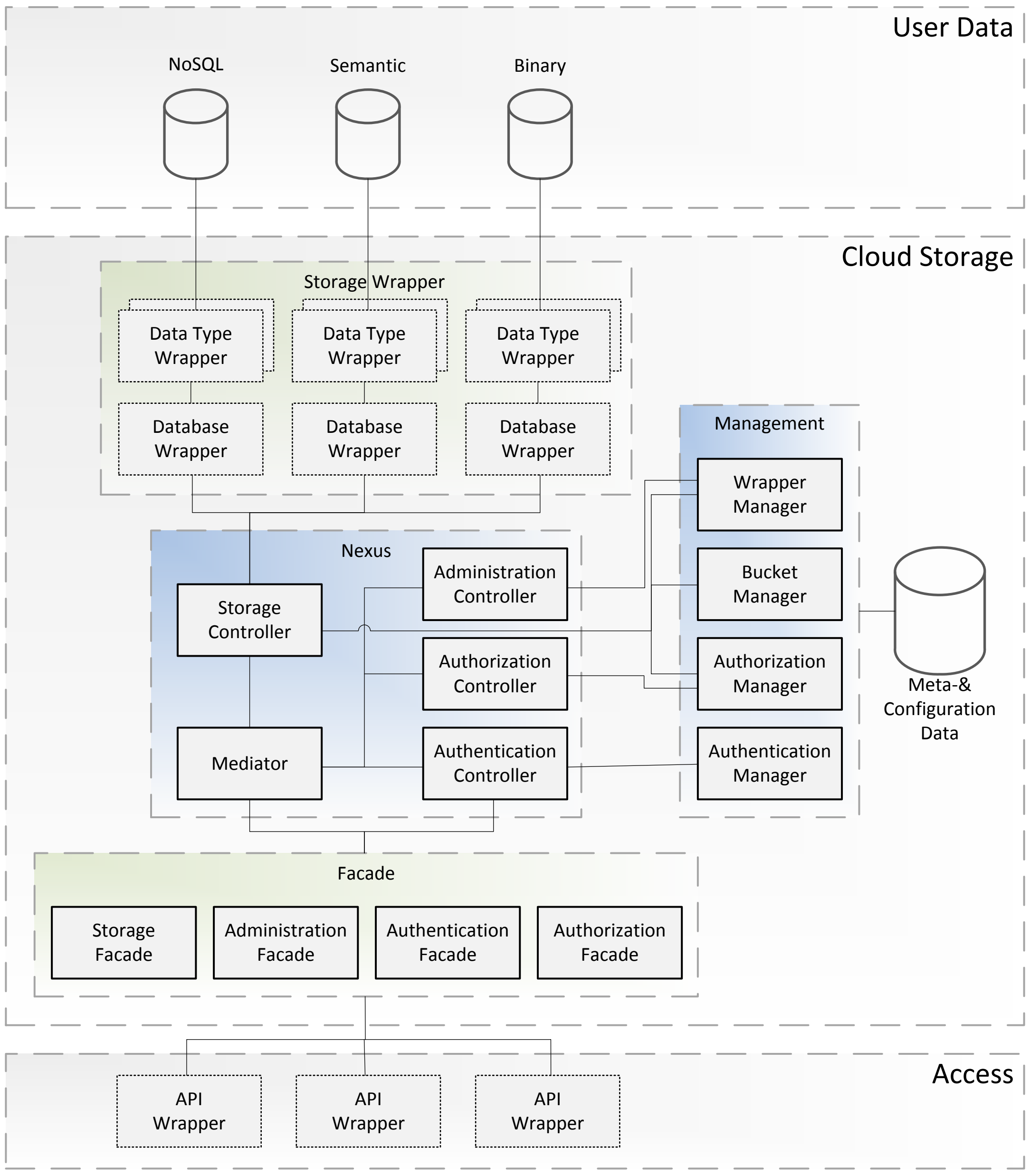

Figure 13: Infographic - Architecture of the Knowledge and Information Storage

In the following subsections the three segments, User Data, API Wrapper and Cloud Storage, will be discussed in detail. For the Cloud Storage segment, its internal segmentation will be used: Storage Wrapper, Nexus, Management and Facade.

2.9.2 User Data

The actual user data will be stored in different databases, selected on basis of the requirements for the data. The concrete databases are not yet selected, but databases on different data structure concepts are foreseen. This is indicated by the three different databases depicted in Figure 13 in the User Data segment:

- **NoSQL** is a rather new[2] approach to overcome drawbacks of relational databases. There are already 150 different databases based on the NoSQL approach. The technics used in these databases are quite different, but they have in common that they are non-relational, distributed and horizontally scalable[3]. Since they are non-relational they do not have a fixed data schema, this is useful to store and query data effective in an environment, where the specific data is unknown at develop time.
- **Semantic** data modelling has lost some attention over the last few years as it was not possible to deliver the breakthrough the ICT domain hoped for. However it is still highly relevant in ICT, as a solution to implement expert systems. In fact with the approach of the semantic web[4], data storages for semantic data are still a highly discussed topic.
- Even so **Binary** data can usually be stored in any database as blob or as an encoded string and are a more effective approach to use databases specialized for this use case. Since different kind of assets are foreseen in the ALFRED project, a binary data storage solution will be integrated in the KIS

The different databases in this segment will also be referred to as "**External Databases**" as counterpart to Internal Database (see section 2.9.6).

2.9.3 Access

The Access segment in Figure 13 indicates different API Wrappers for the KIS. API Wrappers are foreseen in order to provide simple access to the services.. These wrappers will encapsulate the logic of the interface provided by the Cloud Storage – Facade (see section 2.9.4) for a specific target language and platform as a library, e.g. a Java library for Android. In that way, developers who want to use the KIS do not need to know the implementation details for the Cloud Storage – Facade. Neither do they need to have expertise with the chosen technology for the Cloud Storage – Facade.

One Wrapper will be designed to work in the mobile platform environment accessible by the Personal Assistant (see section 2.2). This will allow third party developers to use the specific services provided by the KIS with their ALFREDO App's.

Last but not least, this will also allow shifting to any other storage solution as long as corresponding API Wrappers are provided. This component-based approach will reduce the dependencies of other components to the Cloud Storage component of the KIS.

2.9.4 Cloud Storage – Facade

The Facade is the public interface of the Cloud Storage. All functionalities are provided as a well-defined service. This single point of communication guarantees the integrity of the internal functionality and thereby also the integrity of the stored data. The provided services are grouped in four categories:

- Storage – The provided services are for the storage, retrieving and the manipulation of data in the KIS.

[2] Around 2009
[3] http://nosql-database.org/
[4] http://www.w3.org/standards/semanticweb/

- Administration – The provided services are for the administration of the Cloud Storage, e.g. to add, modify or remove specific database to or from the User Data segment (see Figure 13).
- Authentication – The provided services are for the administration of the clients of the Cloud Storage, e.g. to register or remove a specific entity as a client of the Cloud Storage. Each entity who wants to consume services of the KIS has to be registered as a client of the Cloud Storage.
- Authorization – The provided services are for the administration of clients' rights to use services of the KIS for specific buckets. This allows the owner of a bucket to share the data with other clients.

The Facade is also responsible for the authentication of clients, accessing the services of the Facade.

2.9.5 Cloud Storage – Nexus

The Nexus is the heart of the Cloud Storage; it controls the processes for the data management as well as the management of the Cloud Storage itself.

The services published by the Cloud Storage – Facade (see section 2.9.4) are routed through a mediator subcomponent to the responsible controller subcomponents. These will call internal routines to process the requests made by the before mentioned services. There are four controllers for this purpose:

- Storage – Storage services provided by the Facade will be handled in this controller. It will check the authorization (with the help of the Authorization Manager; see section 2.9.6) of the client before it executes the necessary processes in the context of a specific bucket. If data management for user data is required, the controller requests information for the user data databases from the Wrapper Manager and the Bucket Manager (see section 2.9.6) in order to access the right databases.
- Administration – Administration services provided by the Facade will be handled in this controller. It will access functionalities of the Wrapper Manager in order to add, modify or remove databases to the access segment (see section 2.9.3) for the user data.
- Authentication – Authentication services provided by the Facade will be handled in this controller. It will access functionalities of the Authentication Manager in order to add, modify or remove profiles of clients of the KIS.
- Authorization – Authorization services provided by the Facade will be handled in this controller. It will access functionalities of the Authorization Manager in order to add, modify or remove specific rights for specific clients for specific buckets.

2.9.6 Cloud Storage – Management

The Management of the Cloud Storage encapsulates the business logic for all configurations for the Cloud Storage and the external databases for the user data. The Management component has access to an internal database to ensure the persistency of the configurations. This database will also be referred to as "**Internal Database**" as counterpart to the External Databases (see section 2.9.2).

The functionalities provided by the Management are divided in four subcomponents, so called Managers. Each Manager is specialized for a specific domain:

- The Wrapper Manager manages the External Databases. This includes the control over and the persistent of the configuration of the databases in the Internal Database. The manager chooses and provides for a given storage operation the appropriate Database Wrapper.
- The Bucket Manager manages the buckets used for the data storage in the KIS. The manager stores information of every bucket in the system and provides this information for a given storage operation.
- The Authorization Manager manages the Access Control Lists (ACL). The ACL are used to restrict or grand different levels of access to a specific bucket for a given client of the KIS.
- The Authentication Manager manages a list of the clients of the KIS. This includes some credentials for a specific client, such as a user name and an authentication token. The manager provides the functionality to add or remove clients to the list of clients as well as to authenticate a client, e.g. based on the user name and the authentication token.

2.9.7 Cloud Storage – Wrapper

The Wrapper of the Cloud Storage manages the actual access to the databases where the user data will be stored. For each database a specific Wrapper is needed. A Wrapper consists of a Database Wrapper and any number of Data Type Wrappers. In this constellation the Database Wrapper has different functionalities:

- single point of access for a specific database for the Cloud Storage
- providing of services to access the specific database for the Data Type Wrappers

Each Data Type Wrapper for a specific Database Wrapper will encapsulate the business logic to store, modify and delete data in a specific data format. This allows for a flexible adaption to different requirements and technologies.

3 Functional Specification

3.1 Overview

The functional specification will explain in detail all functionalities provided by each component of the ALFRED system to the other components of the system.

Each component has its own section in this chapter. The interaction of the subcomponents of the components described in the related section of chapter 2 will provide a detailed but still fairly high level view of the functionalities. This will help to obtain an overview of the functionality as well as an understanding of the orchestra of the involved subcomponents.

3.2 Personal Assistant

3.2.1 Overview

The Personal Assistant consists essentially of the mobile application. The application handles user input (usually provided verbally by speaking with the device), performs actions the user wants it to execute, starts and handles third party applications integrated into ALFRED, makes suggestions regarding upcoming events and connects external components.

3.2.2 Core

The core functionality of the mobile ALFRED application is to perform actions when the user presses the main button on the user interface. As the main interaction will take place in a verbal manner, speech recognition is essential.

After the words spoken by the user have been analysed ALFRED will perform actions associated with the words the user said – for example: ALFRED will start an application.

3.2.2.1 Recognizing Input and providing Output

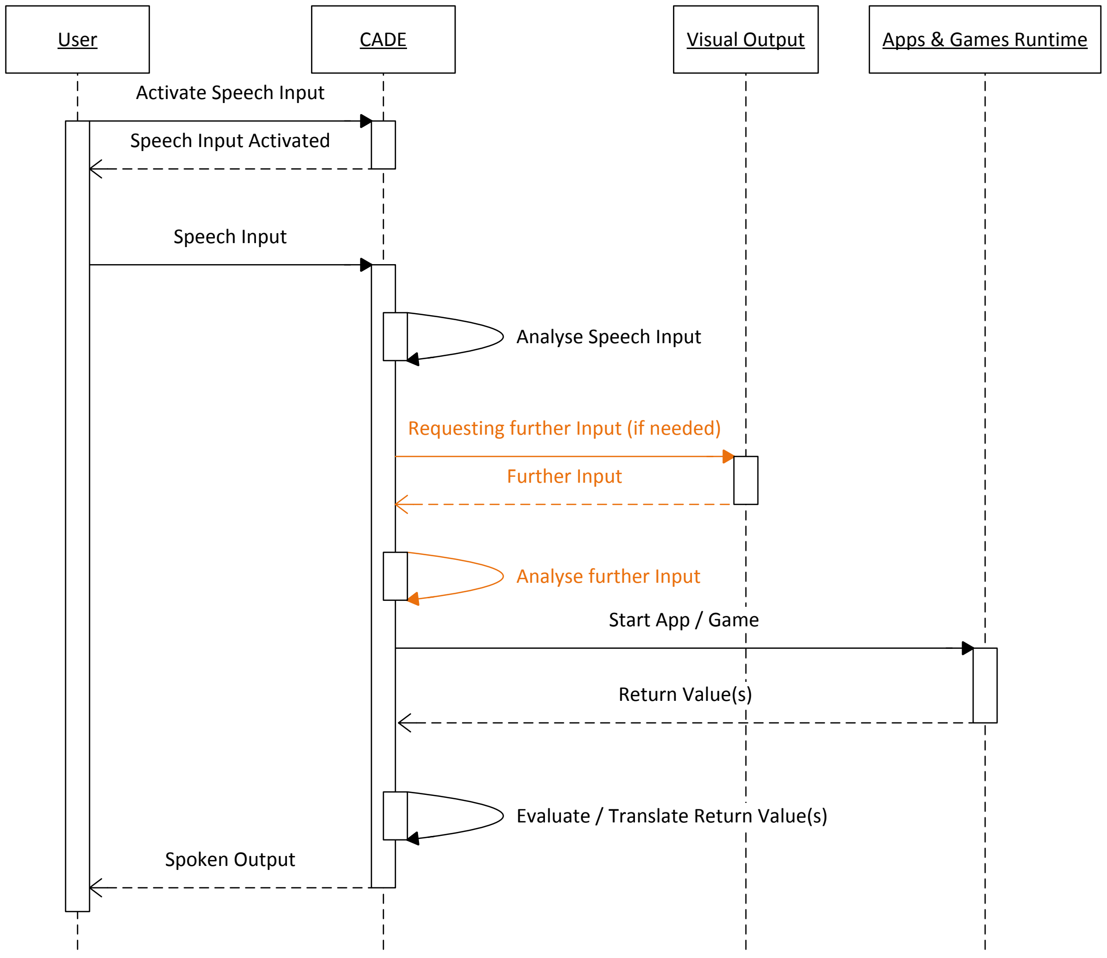

Figure 14: UML Sequence Diagram – PA: Recognizing User Input

The communication between the mobile ALFRED application and its user will take place in a verbal manner. To achieve this, the user has to press a button on the user interface of ALFRED which in turn activates the CADE component. CADE activates the microphone and may provide acoustical (or visual) feedback, so the user knows when to speak.

The audio data recorded by the microphone from now on will be transmitted to the CADE component which analyses the data in order to determine the words the user has spoken.

The messages printed in orange in Figure 14 show the additional user interaction needed, if CADE cannot identify the action(s) to perform. In this case further input is needed before CADE can process the user’s request. Such input can be requested by presenting visual output to the user. This kind of interaction will be useful if, for example, CADE is not able to identify the users spoken input and needs further clarification.

Though the “Visual Output” component is intended for CADE it is also imaginable to provide a public interface used by other components as well.

Output to the user will be provided acoustically. CADE will translate the output (usually provided as textual data) into spoken words by using a Text-To-Speech (TTS) mechanism.

3.2.3 Apps and Games

The extendable feature, integrating additional apps and serious games, requires components to handle those apps. This will be done by a dedicated runtime. Though there are two types of extensions (apps and games) there is only one runtime: the Apps & Games Runtime. The steps of running an application or a game are shown in Figure 15.

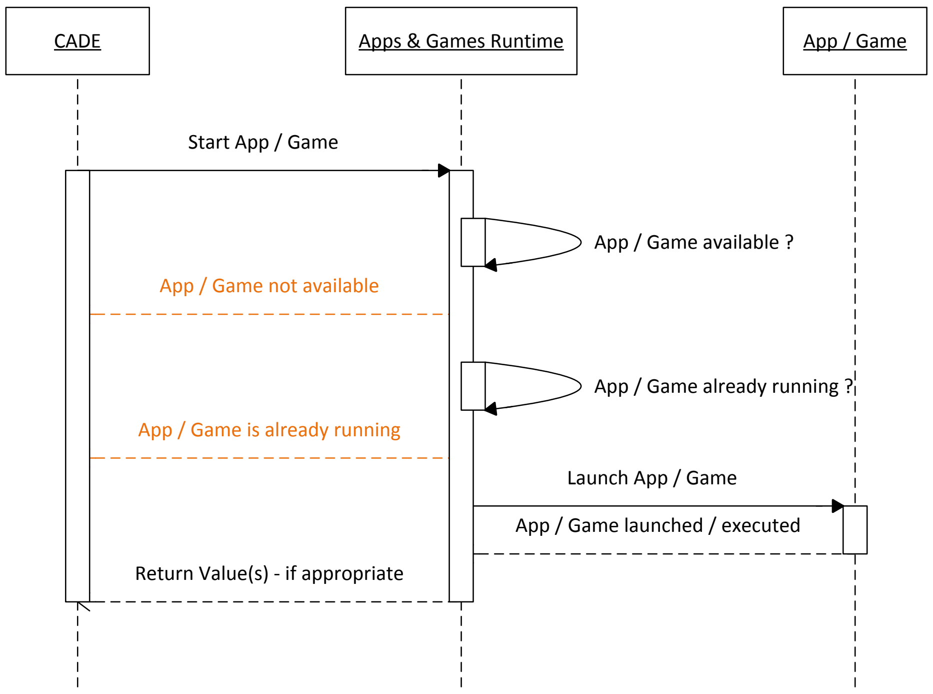
CADE
Apps & Games Runtime
App / Game
Start App / Game
App / Game available ?
App / Game not available
App / Game already running ?
App / Game is already running
Launch App / Game
App / Game launched / executed
Return Value(s) - if appropriate

Figure 15: UML Sequence Diagram – PA: Handling Apps & Games

The Apps & Game Runtime is responsible for launching and monitoring the apps and games the user wants to start. The apps themselves are responsible for requesting permissions and accessing device functions they need, in order to work correctly (as shown in Figure 16).

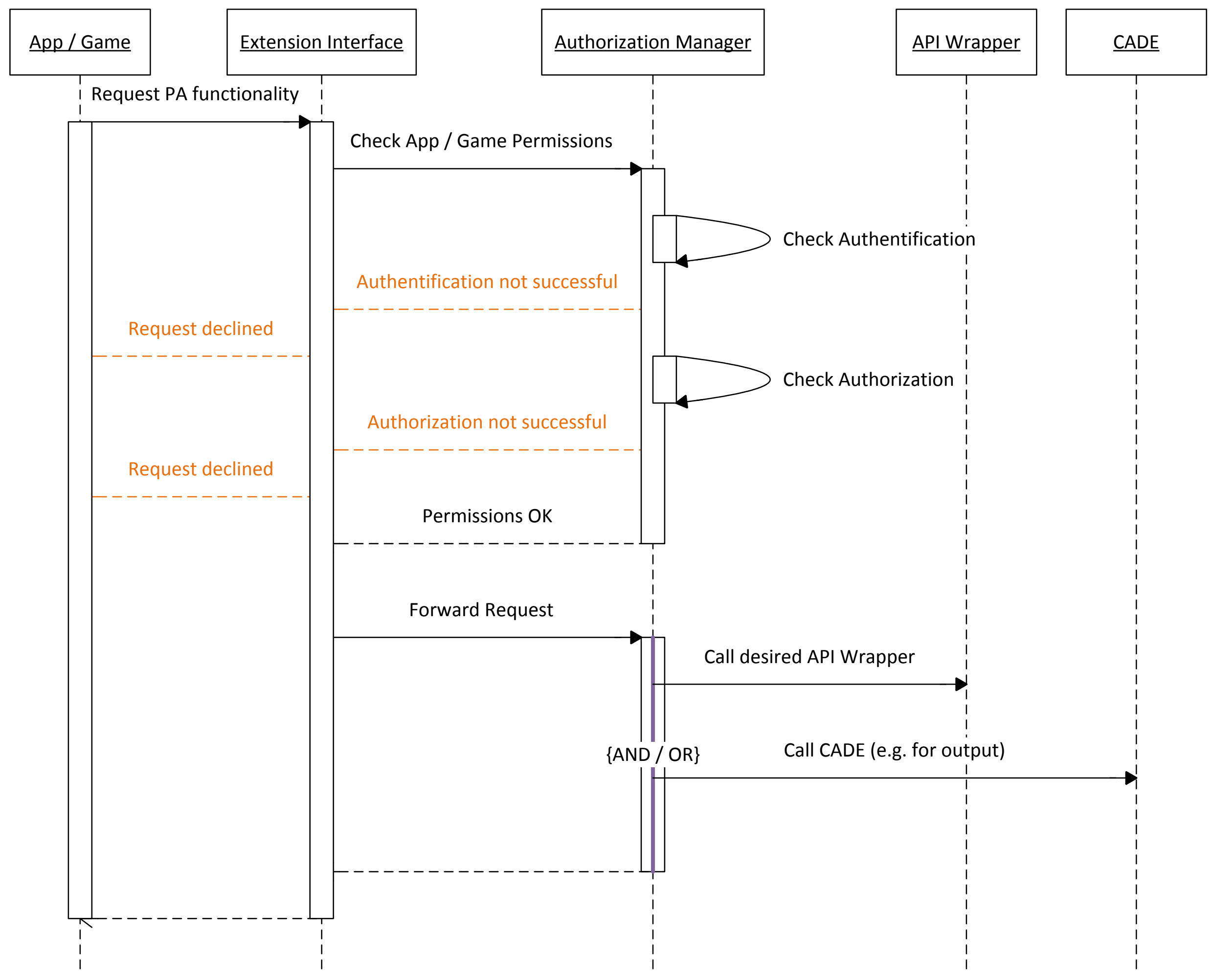

Figure 16: UML Sequence Diagram – PA: Requests to PA Functionality

Apps are able to interact with the ALFRED app through calling methods from an Extension Interface. The Extension Interface offers functionalities which can be used by apps and games to access internal components of ALFRED.

After an app has called the Extension Interface (e.g. for requesting some user data which shall be used within the app) the Extension Interface in turn contacts an Authorization Manager. The Authorization Manager checks the permissions the user has granted to the requesting app and allows the app to perform its desired action or denies access.

As soon as an application has granted access to perform its desired action, this action request will be forwarded to the corresponding API wrapper.

The way of presenting some output to the user is nearly the same as it is when the app wants to perform an action. But instead of forwarding the action request to an API wrapper the Authorization Manager forwards the output to the Output Controller. Outputs can be tagged with a priority. This priority will be used by the Output Controller to present the requested outputs in order.

3.2.4 Suggestions component

There are two types of suggestions which can be presented to the user: events and games.

Suggestions will be created by external components (e.g. an event manager and a game manager), forwarded to the mobile ALFRED application and presented to the user, if appropriate. The process of filtering will be described below (see Figure 17).

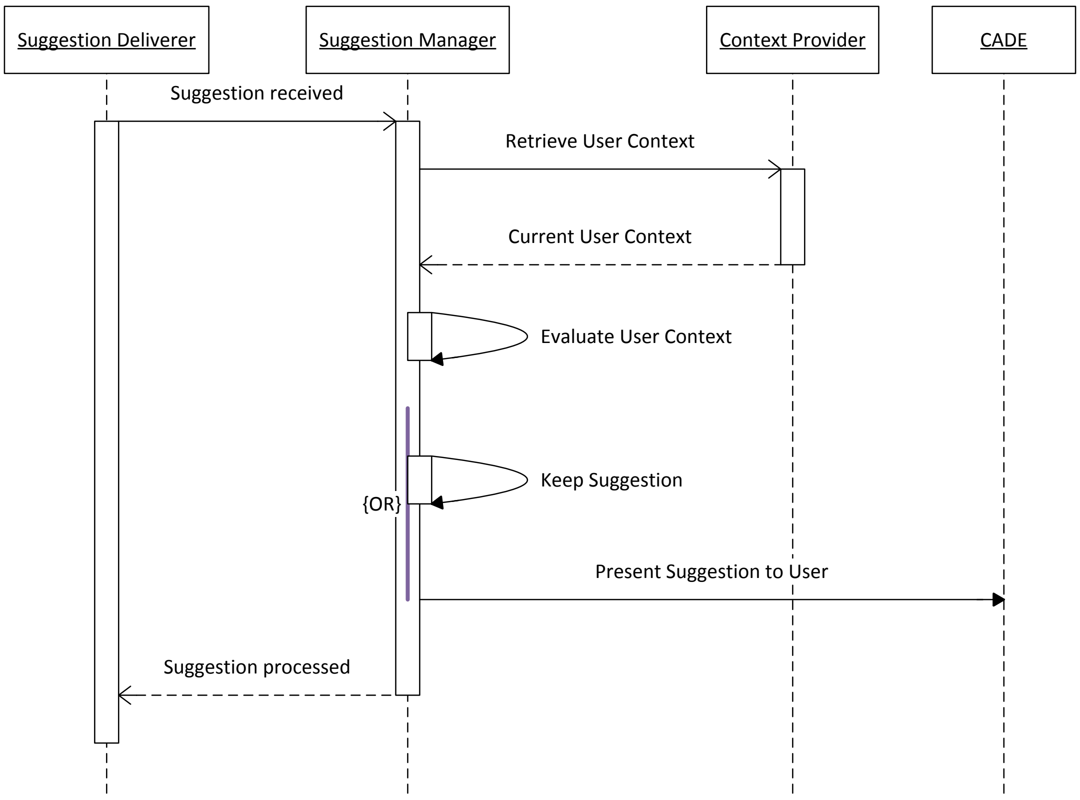

Figure 17: UML Sequence Diagram – PA: Receiving and Filtering Incoming Suggestions

This process starts by receiving or requesting (depends on the mechanism used – push or pull) a suggestion from an external component (e.g. an Event Manager). The Suggestion Manager performs some checks on the incoming information (integrity of data, presence of all information needed depending on the suggestion type, e.g.).

The Suggestion Manager uses another core component in order to retrieve additional context information which is needed to make a decision on whether to suggest this event / game or not: the Context Provider (see chapter 2.2.5).

As soon as the Suggestion Manager has retrieved all context information needed, it decides – depending on the current user context – whether the current suggestion will be presented to the user or kept for a later presentation. If the suggestion meets the current user context the Suggestion Manager calls the CADE component in order to inform the user about the received event or to play the suggested game.

In order to decide whether a suggestion will be presented to the user the Suggestion Manager needs information about the user's current context. Such information can consist of the current location, the current local time, an activity the user is performing at the moment and so on. Information like this will be retrieved by contacting the Context Provider. For example: The Suggestion Manager receives a suggestion for a game that shall be played by the user. To decide whether this game should be suggested right now, the Suggestion Manager calls the Context Provider to return the user's current context. This context information specifies that the user is currently driving a car. The suggestion of playing a game right now is not appropriate, so the Suggestion Manager will decide to either not suggest that game at all or suggest it later.

3.2.5 API Wrapper

The API Wrappers encapsulates the functionality of the other ALFRED components. For each component a distinct wrapper is used. This multiplexes all functionalities of the ALFRED system in on place. Providing access to third party Apps in one point allows the control of the access by the system and thereby by the end user.

3.3 Health Monitor

3.3.1 Overview

As stated in section 2.3, the HM is composed of four main pieces of software: (1) the Sensor Abstraction Framework, (2) the Health Monitor Client, (3) the Health Monitor Server and (4) the Web Portal. In the next sections we will describe the main services that each component publishes, as well as the interaction that takes place internally in order to realize such services.

3.3.2 Sensor Abstraction Framework

SAF is a framework that standardizes access to external sensors by publishing a homogenous interface. All operations are forwarded to SAF using the SAF API Wrapper, which encapsulates SAF functionality for a particular environment (e.g. Android-Java). In the following subsections the main services provided by this framework are described.

3.3.2.1 Register Driver

SAF is open for registering new types of sensors. To receive information from a new type of sensor, a new driver must be registered within SAF. A driver is a piece of software that knows how to communicate with a particular type of sensor. The implementation of the driver is private (vendor-specific), but it must export a public and homogeneous interface so SAF can obtain information from the sensors in a uniform way. This common interface makes it possible to simply plug-in the driver in the framework.

The process is depicted in the following diagram (see Figure 18).

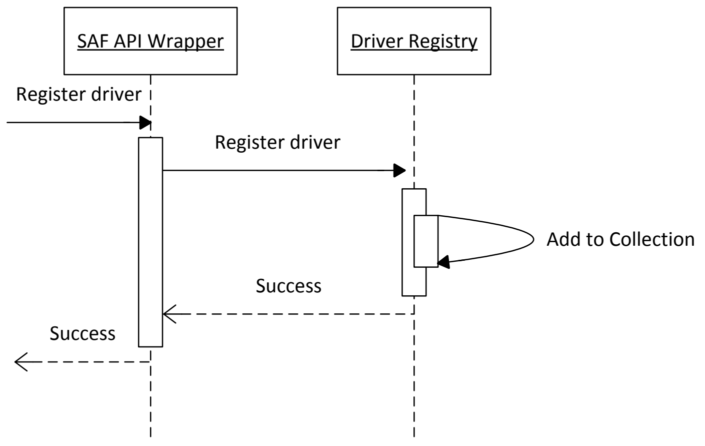

Figure 18: UML Sequence Diagram – SAF: Register Driver

The Driver Registry is responsible for managing sensor drivers. To that end it internally maintains a collection of previously registered drivers. When a new driver is registered, it is internally added to such collection.

Once the driver has been registered, a new type of sensors is available from SAF. To start receiving measurements from a specific sensor of that type, it is necessary to register a *listener* for that particular sensor.

3.3.2.2 Register/Unregister Listener

To begin obtaining information from a concrete sensor supported by SAF a new *listener* must be registered in SAF. A listener is a piece of software that gets activated whenever new data is available in the sensor it is listening to. In this way, the publish/subscribe pattern is implemented in SAF.

In the following diagram a sample registration process is described (see Figure 19).

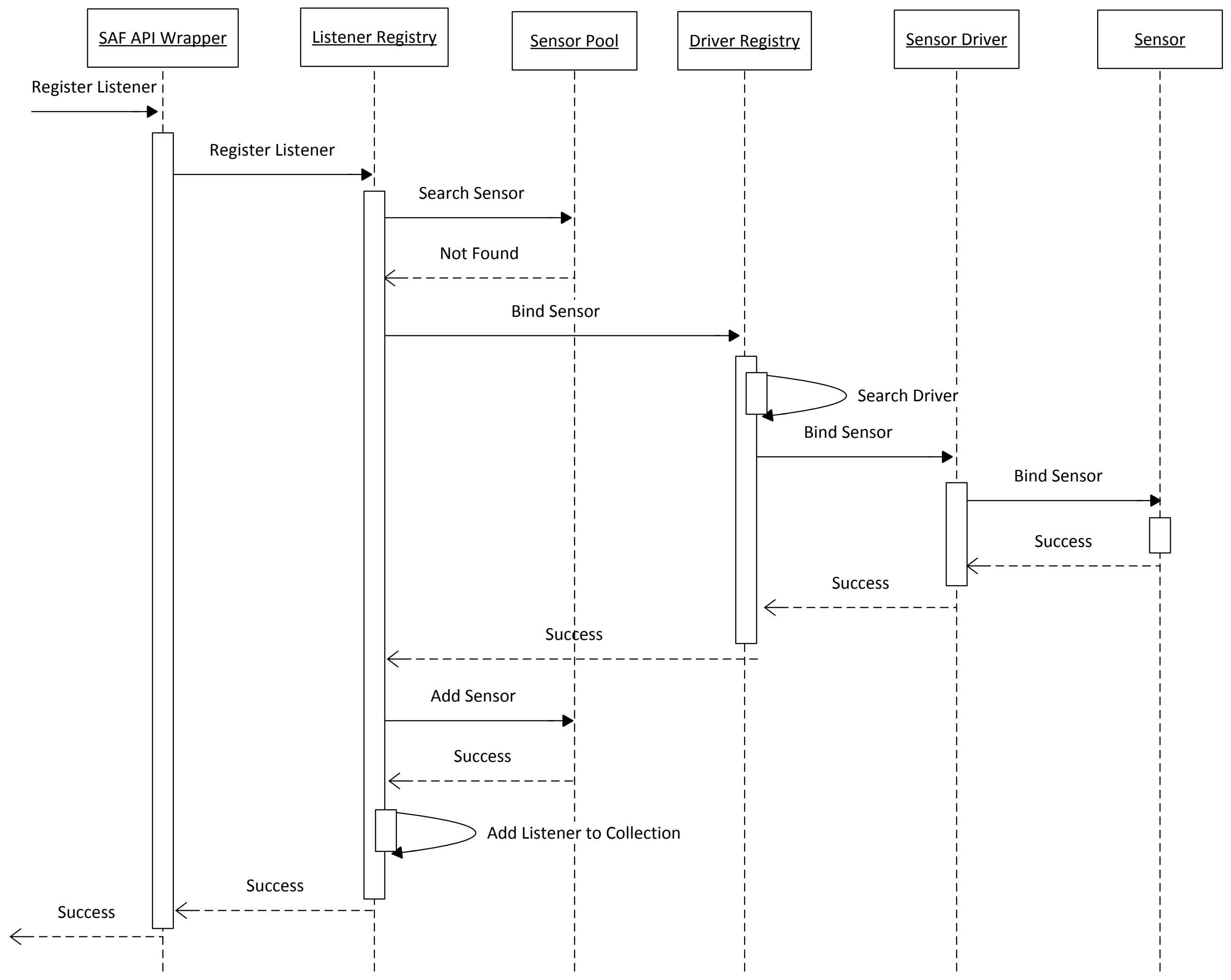

Figure 19: UML Sequence Diagram – SAF: Register Listener

First, the operation is forwarded to the SAF API Wrapper. When registering a new listener, some information must be provided:

(i) An identifier of the sensor.
(ii) The driver that manages the sensor.
(iii) The sensor may emit more than one type of measurement, and the listener must have the capability of specifying the type of measurement to receive (hereafter, for the sake of simplicity, we will assume only one type of measurement per sensor).
(iv) Before receiving information from the sensor, an authentication/attachment/link procedure may take place, so some form of credentials may be required.
(v) Some configuration may be transmitted to the sensor in order to alter its behaviour.

All the information described above could be encoded as an URL (see Listing 1).

Listing 1: SAF – Example: URL Encoding

```
saf:sensordriver:sensorid:measurement?param1=value1&param2=value2...
```

The Listener Registry receives the operation and checks whether the sensor is already active, that is, it checks if the sensor is being monitored because there is another listener already registered for it.

To this end, the Sensor Pool is consulted. If the sensor is included in the Sensor Pool, there is no need to bind the sensor, since it was done before. Otherwise, as depicted in the diagram, the sensor handshake/binding process must take place.

The Driver Registry is asked to perform the binding process. The Driver Registry searches for the appropriate Sensor Driver and forwards the operation to it. The Sensor Driver knows how to communicate with the sensor and accomplishes the connection.

Once the binding process is succeeded, the new Sensor must be actively monitored and therefore it is added to the Sensor Pool.

Finally, the registered listener is added to the collection of already registered listeners.

The next time a listener is registered for the same sensor, neither the binding process nor the addition to the Sensor Pool will take place. However, the listener will be added to the collection of already registered listeners. Therefore, SAF allows multiple listeners per sensor, allowing different apps to register to receive information from the same sensors.

Unregistering a listener is the reverse operation. If it is the only listener attached to the sensor, then the sensor must be removed from the Sensor Pool. Finally de listener must be removed from the collection of registered listeners.

3.3.2.3 Sensor Monitoring

Once a listener has been registered to receive information from a concrete sensor, the Sensor Monitor enables active monitoring on the device.

The monitoring service can be split into two sub-processes that are described in the following paragraphs:

1. The collecting sub-process
2. The dispatching sub-process

The collecting sub-process, consists of the Sensor Monitor that is activated from time to time and collects information from all bound sensor – those sensors included in the Sensor Pool, that is, those sensor that have listeners attached -. But before collecting the information it is necessary to obtain information about the active sensors, and then obtain the Sensor Drivers that enable communication with them. This procedure is described in the following diagram (see Figure 20).

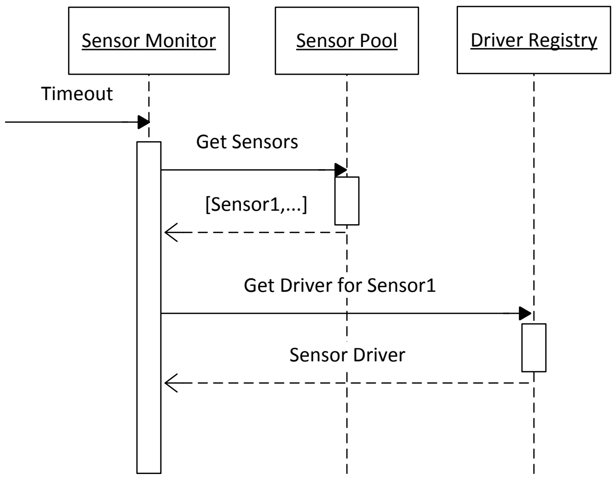

Figure 20: UML Sequence Diagram – HM/SAF: Sensor Monitoring, Search Driver

Once the active sensors are known and their drivers have been located, the monitoring process must go on, as specified in the following diagram (see Figure 21).

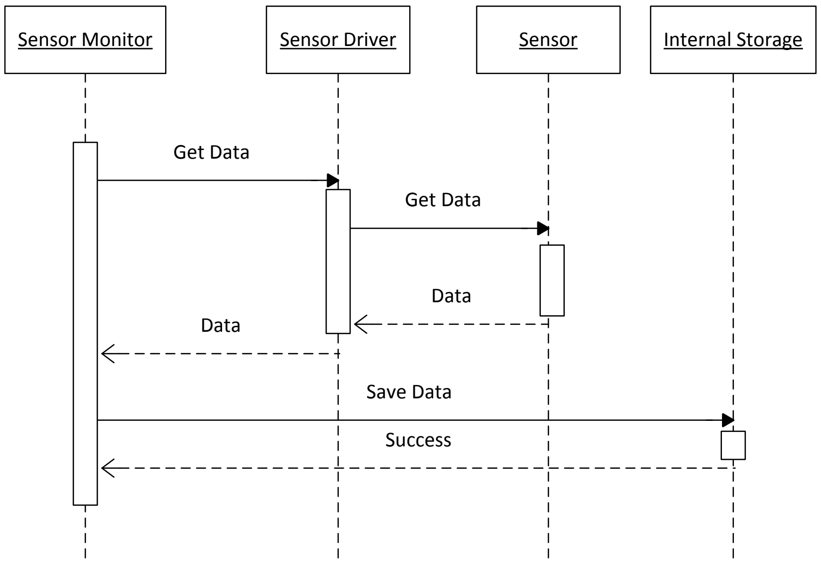

Figure 21: UML Sequence Diagram – HM/SAF: Sensor Monitoring, Retrieve Sensor Data

Data are retrieved from the active Sensor by using its Sensor Driver. After that, data are stored in the Internal Storage, an intermediate buffer/queue/storage useful for uncoupling the monitoring sub-process from the dispatching sub-process and avoid interferences/disturbances/speed differences between both sub-processes.

The sequence of messages described in the previous diagram should be replicated as many times as active sensors exist within the Sensor Pool.

The dispatching sub-process, is initiated when data must be dispatched. The Sensor Data Dispatcher gets activated and reads information from the intermediate memory. Then information is conveniently forwarded to all registered listeners, as described in the following diagram (see Figure 22).

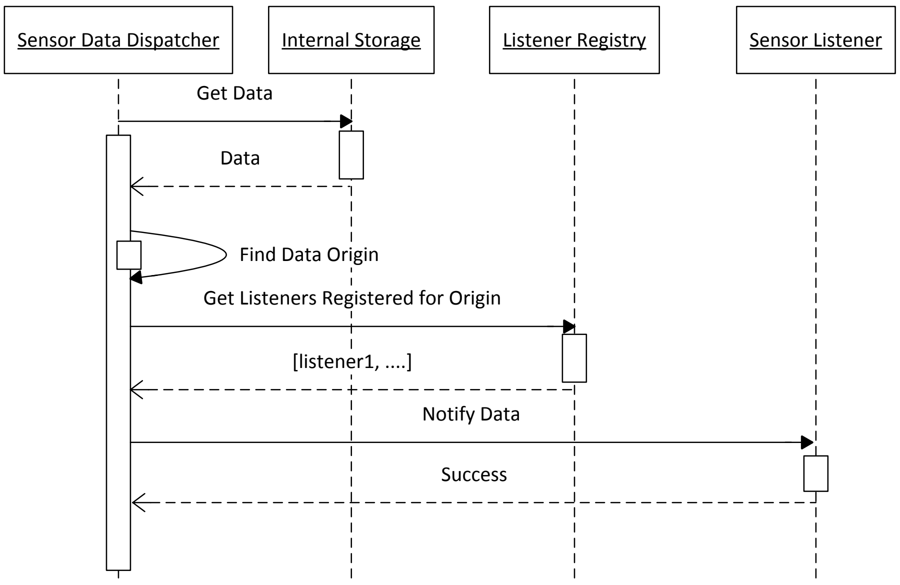

Figure 22: UML Sequence Diagram – HM/SAF: Sensor Monitoring, Dispatch Data

As explained in the diagram, once data has been read from the Internal Storage, it is necessary to detect the origin of such data, that is, the sensor-measurement tuple that generated the information. The next step involves obtaining all the listeners that were registered for the tuple sensor-measurement from the Listener Registry. Finally, all registered listeners must be notified. In the example above only the first listener is notified. The procedure is replicated for all listeners..

3.3.3 Health Monitor Client

In this section we list the main services implemented by the Health Monitor Client. Receiving information from sensors and transmitting it to the Health Monitor Server establish the Health Monitor Client backbone. Also, the Health Monitor Client must provide services to manipulate user-related data, including data about carers as well as user health data.

3.3.3.1 Receive Sensor Information

The Data Receiver subcomponent of the Data Pre-process Framework gets registered in SAF as a listener for certain health-related sensors. The type, identification and configuration of the sensors are obtained from the Metadata Repository and are user-specific. This information is entered by user carers as we will show in next sections (see section 3.3.4).

Once the Data Receiver has been registered as a listener, it begins receiving information from the sensors. The process is further described in the following diagram (see Figure 23).

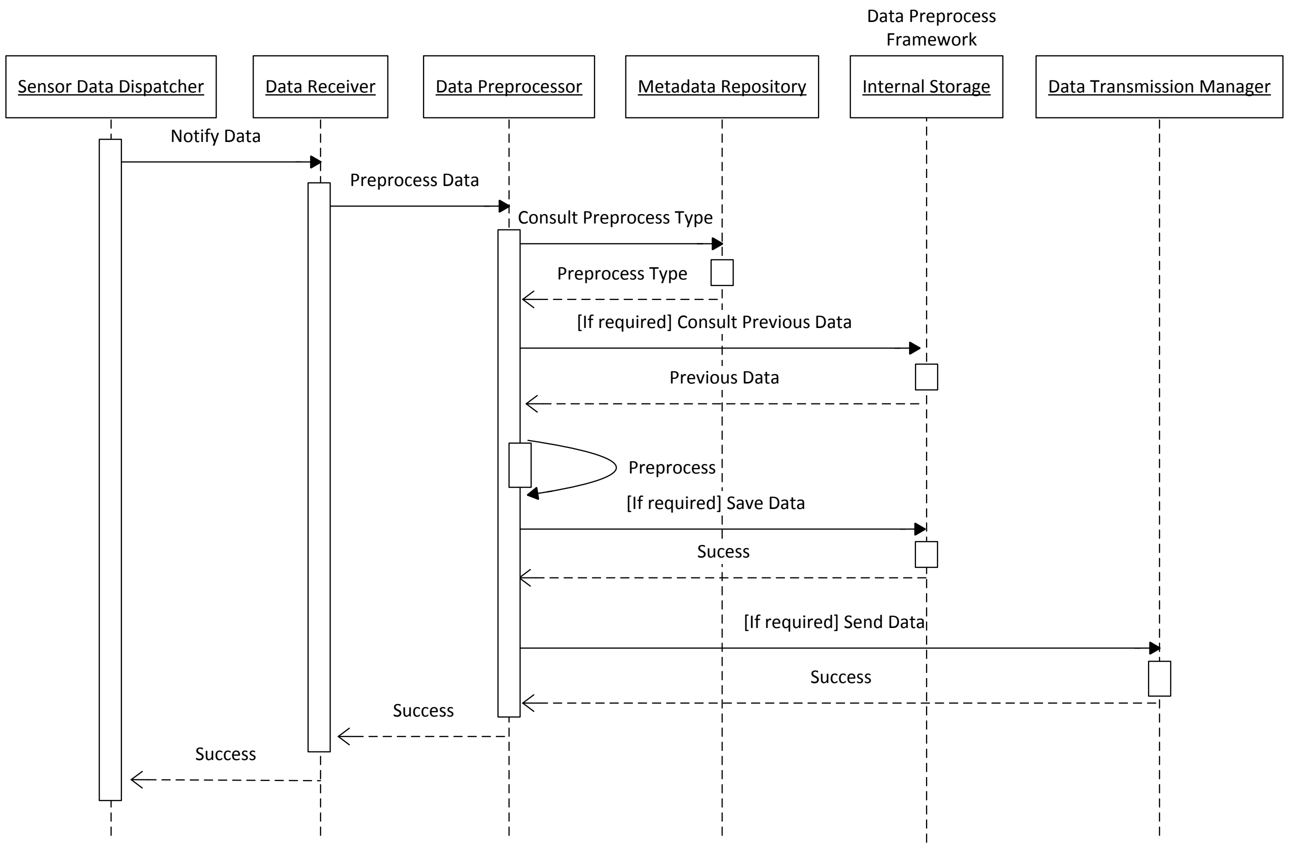

Figure 23: UML Sequence Diagram – HM/Client: Receive Sensor Information

The Sensor Data Dispatcher notifies captured data to the Data Receiver subcomponent, which in turn forwards data to the Data Pre-processor subcomponent. Before pre-processing data, the type of pre-processing must be determined. Such information is available in the Metadata Repository and dictates what to do with the incoming data, which is summarized in the following paragraphs.

Some pre-processing may involve obtaining previously calculated data. If this is the case the Internal Storage is consulted. Then data is pre-processed.. If calculated data must be stored for future references, it is saved in the Internal Storage. Finally, if data must be transmitted to the Health Monitor Server it is sent to the Data Transmission Manager.

The obtained metadata is key to know the exact sequence of steps to undertake. Metadata can be obtained from the Metadata Repository. The Metadata Repository is a common place where all metadata can be accessed. It performs the role of cache and also abstracts the underlying APIs for retrieving/updating the information. In the following diagram this behaviour is illustrated (see Figure 24).

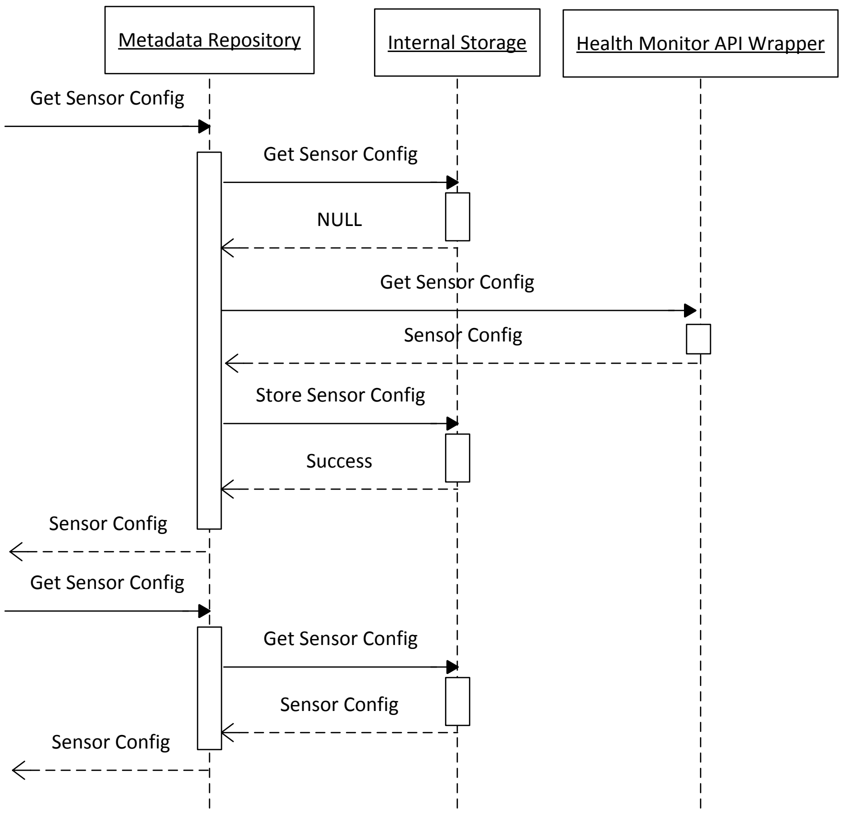

Figure 24: UML Sequence Diagram – HM/Client: Access Metadata Repository

In the diagram, metadata related with sensor configuration is requested. If it is not available locally (which means it has not been retrieved before) the request is forwarded to the Health Monitor Server using its published API. To this end, the Health Monitor API Wrapper is used. Data is then obtained and cached inside the Metadata Repository internal storage. Later on, if the same data is requested it is not necessary to contact the Health Monitor API Wrapper since the data has been cached locally.

The same occurs with metadata related to the user profile. In this case the Personalisation Manager is contacted through the Personalisation Manager API Wrapper instead.

3.3.3.2 Transmit Sensor Information

When the data have been pre-processed they are ready for further transmission to the Health Monitor Server. The following diagram depicts the process (see Figure 25).

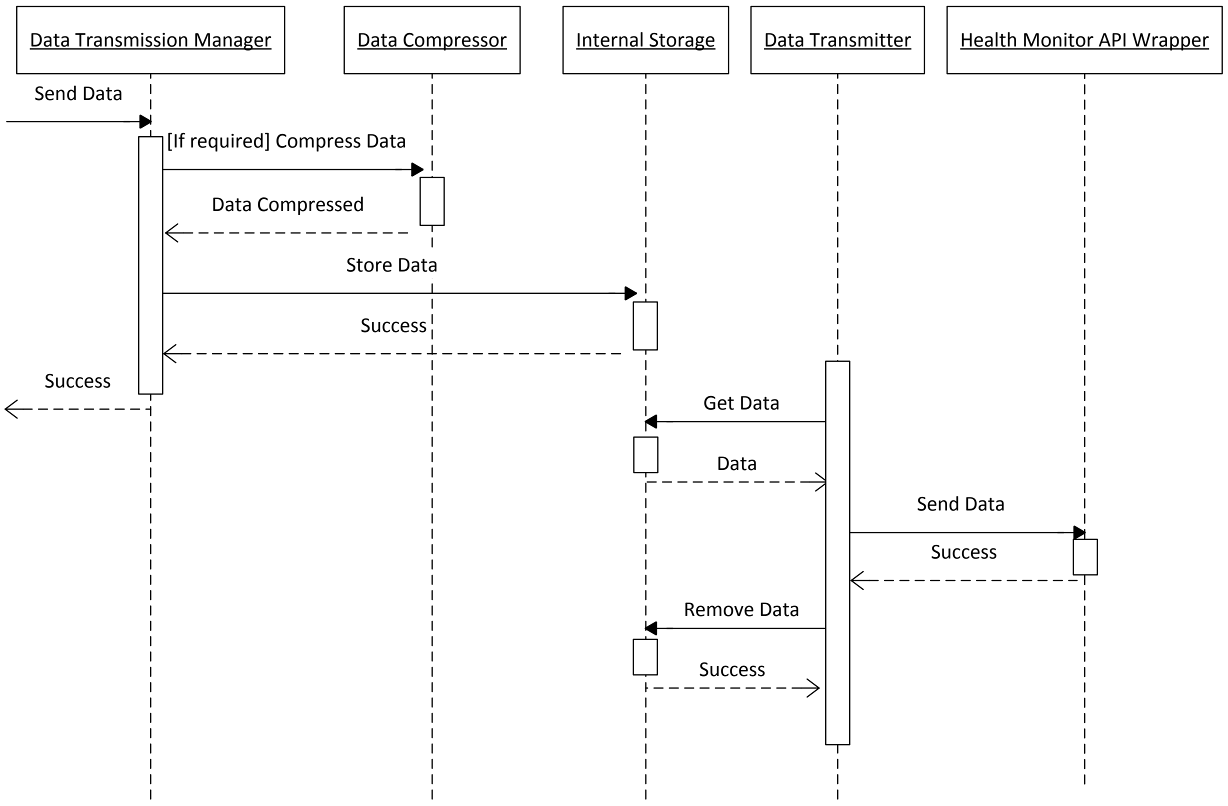

Figure 25: UML Sequence Diagram – HM/Client: Transmit Sensor Information

Before transmitting data, they may be compressed by the Data Compressor subcomponent. After that, data should be stored inside the Transmission Manager Internal Storage. This element uncouples the transmission elements from the rest of the system, maintaining a private buffer. It also provides failover capabilities.

The Data Transmitter subcomponent is activated and data are sent to the Health Monitor Server through the Health Monitor API Wrapper. Finally, if transmission succeeds data are removed from the Internal Storage. Otherwise data are kept until they are successfully retransmitted in future connections.

3.3.3.3 Manage Health Profile Information

The user must be able to manage his health profile at any moment. At least two pieces of information should be accessible in the Health Monitor Client: (1) user health data and (2) carers data:

1. The user will be able to consult and manipulate – where the most basic manipulation operation is ‘delete’ – his health related data.
2. The user will be able to define his carers and grant/revoke different permissions to them.

The gateway to these operations is the Health Profile Manager component, included in the Health Monitor Client. Irrespective of the type of information, the process followed by any operation will look like this (see Figure 26).

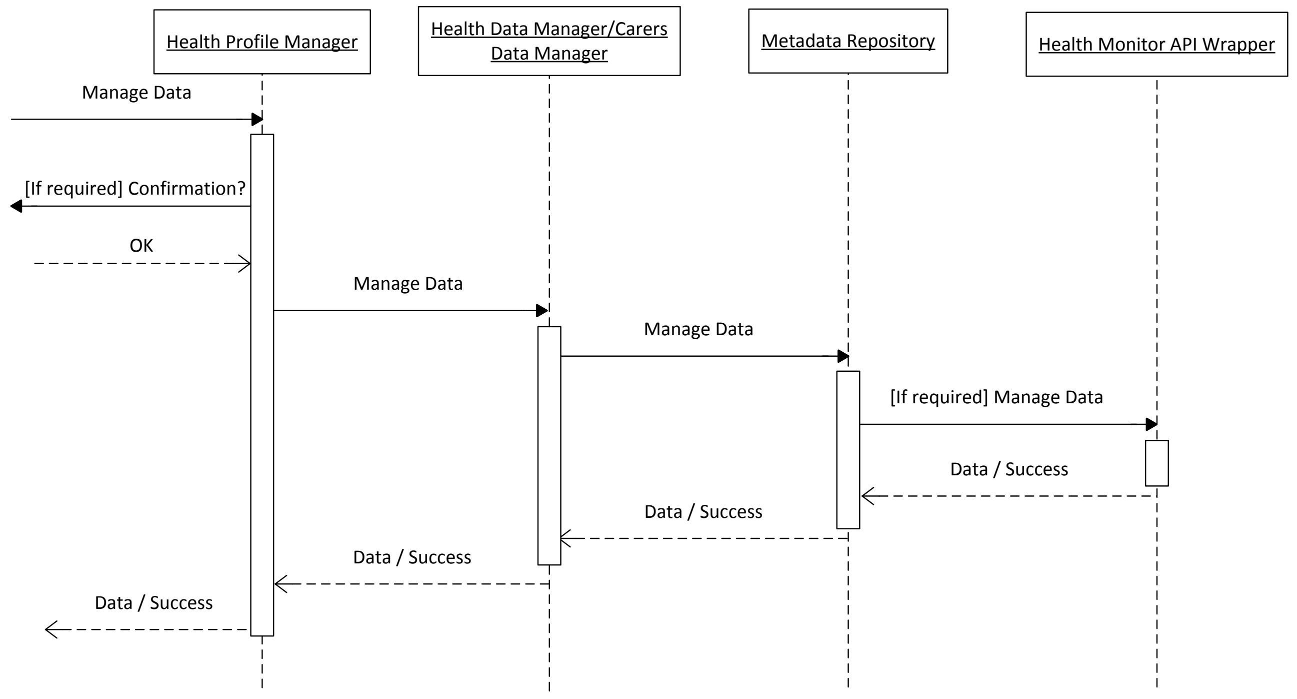

Figure 26: UML Sequence Diagram – HM/ Client: Manage Health Profile Information

First, the command issued by the user reaches the Health Profile Manager, either through the speech recognizer component or through the GUI. If the operation involves managing sensitive data, confirmation from the user may be required.

The operation is then forwarded to the corresponding subcomponent – either the Carers or the Health Data Management - which in turn accesses the Metadata Repository either to obtain the requested information – if the operation is a query – or to update it – if it is a modification operation.

As we have seen in previous sections, accessing the Metadata Repository may involve forwarding the operation to the Health Monitor Server through the Health Monitor API Wrapper. In particular, if the operation is a query operation and the information is not available locally then it must be retrieved from the Health Monitor Server. Also, if the operation is a modification operation, it must always be forwarded to the Health Monitor Server.

3.3.4 Health Monitor Server

This element represents the core of the HM environment and publishes an API to access its services. This API is structured in the form of a Facade, which can be further decomposed into 4 different subcomponents.

All requests are received by the Facade and go through the Controller component. When a request is received by the Facade, its validity is checked– using the Security Controller – and then it is forwarded to the corresponding Controller subcomponent. In the following diagram this behaviour is shown (see Figure 27).

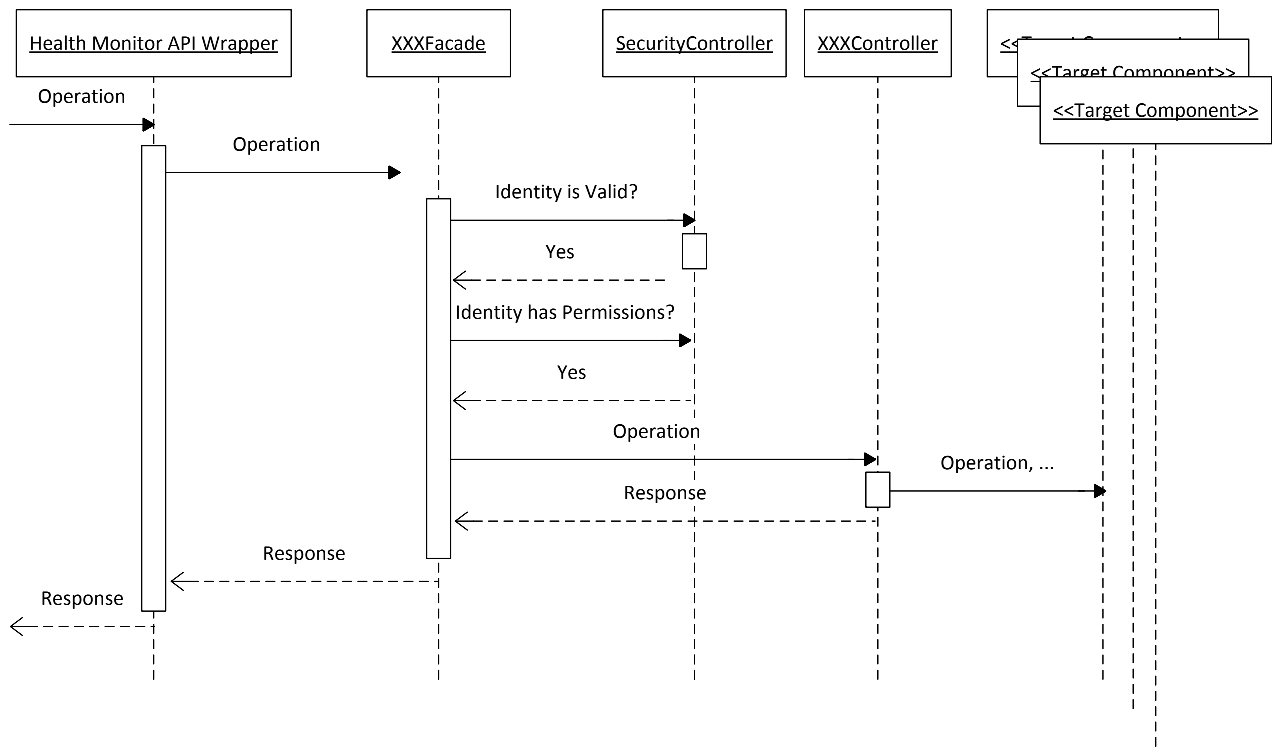

Figure 27: UML Sequence Diagram – HM/Server: Common Behavior to all Requests

First, the identity of the user who issued the command must be verified. To this end, the Security Controller is called. Of course, this operation will depend on the authentication mechanism and user policies implemented in the overall system (e.g. OpenId, oAuth, etc.).

After the identity has been verified, the system must guarantee that the identity has permissions to execute the operation. To this end, the Security Controller is invoked again, now to check whether the user is permitted to execute the service.

Finally, the operation is forwarded to the proper Controller and this one in turn makes as many calls as necessary to other components in order to compose a valid response. In the next sections we will describe the main services published by the Health Monitor Server.

3.3.4.1 Transfer Sensor Data

An essential task of the Health Monitor Server is to receive sensor data from the Health Monitor Client and post-process them, in order to (i) find anomalies and notify alarms and (ii) store permanently the required data in KIS.

The Data Post-process Framework is responsible for such activities, and the process is described in the following diagram (see Figure 28).

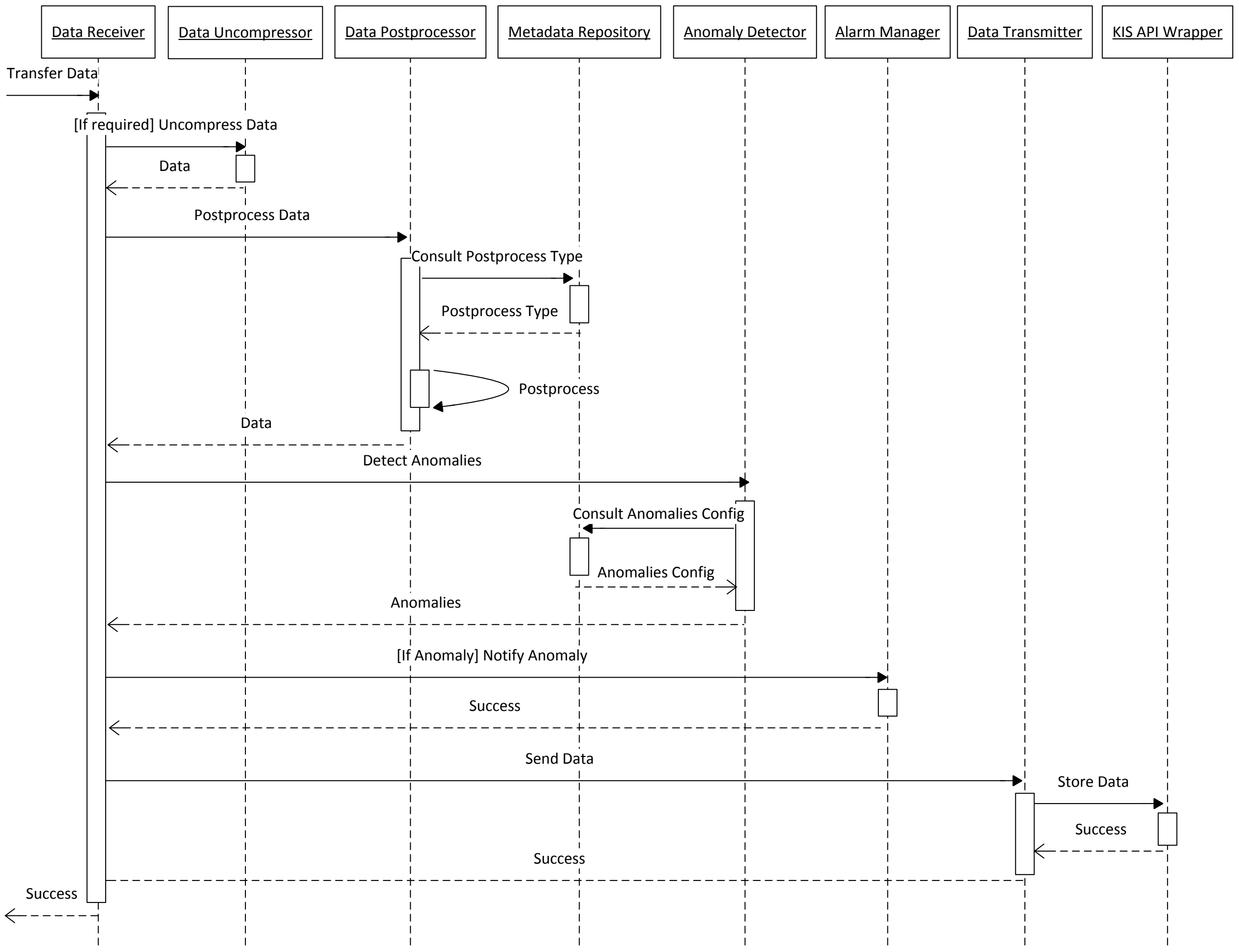

Figure 28: UML Sequence Diagram – HM/Server: Transfer Sensor Data

First, data are received by the Data Receiver subcomponent. If data are compressed, they must be uncompressed by the Data Uncompressor subcomponent. Then, data are forwarded to the Data Postprocessor subcomponent.

The Data Postprocessor subcomponent consults the Metadata Repository in order to find out whether post-processing activities should take place and what kind of post-processing is expected for the data. Afterwards post-processing takes place.

Once data have been post-processed, anomalies must be detected. This is the purpose of the Anomaly Detector subcomponent. In order to detect anomalies, anomaly configuration information must be retrieved from the Metadata Repository.

If anomalies are found, they must be notified to the Alarm Manager component. In the next section we will review the process followed by the Alarm Manager in order to notify alarms. For now we will assume that the job is done.

Finally, data are forwarded to the Data Transmitter subcomponent which in turn transmits them to KIS, using to this end the KIS API Wrapper.

3.3.4.2 Notify Alarm

As we have seen in the previous section, if an anomaly is detected, this incidence is reported to the Alarm Manager component. In particular, the Alarm Notifier subcomponent takes control of the situation and proceeds as described in the following diagram (see Figure 29).

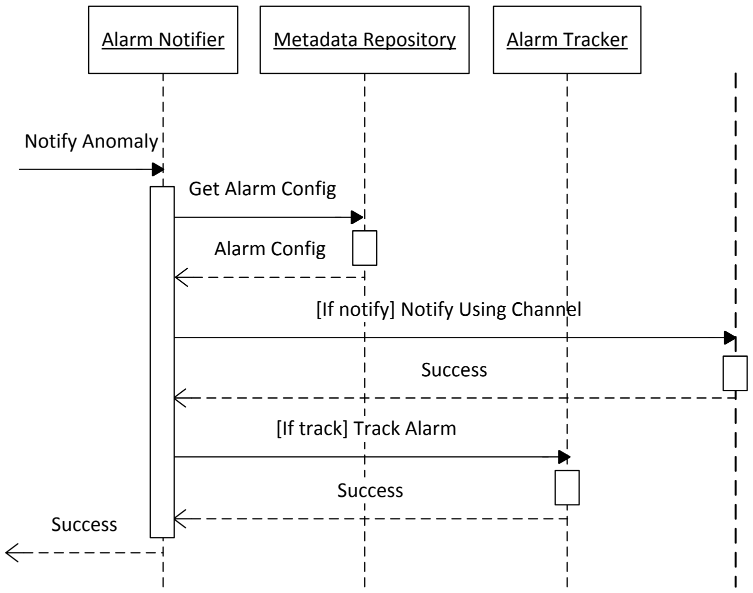

Figure 29: UML Sequence Diagram – HM/Server: Notify Alarm

When the anomaly is received by the Alarm Notifier subcomponent, it accesses the Metadata Repository in order to obtain all the configuration about that anomaly. In particular we are interested in the following configuration data:

1. Whether alarms have been configured for the anomaly
2. Who is/are the receiver/s of such alarm
3. How the alarm/s must be notified, that is, which channels must be used to reach the alarm destination, namely: e-mail, PUSH notification, etc.
4. If the alarm notification must be tracked, in order to reliably detect whether a particular notification has reached its destiny and the destination is aware of the incidence

Once all the information has been obtained, if the anomaly must be notified, the Alarm Notifier proceeds with the specific kind of notification. The user carer is finally reached through a particular channel.

If the alarm notification must be tracked, information about the alarm is transmitted to the Alarm Tracker, which will enable appropriate mechanisms in order to ensure proper reception of the alarm.

3.3.4.3 Manage Health Profile

Managing the Health Profile involves typical CRUD operations on the following collections of data:

- Health Data
- Carers Data

The overall process to access/manipulate data always follows the same steps, irrespective of the particular data type. It is described in the following diagram (see Figure 30).

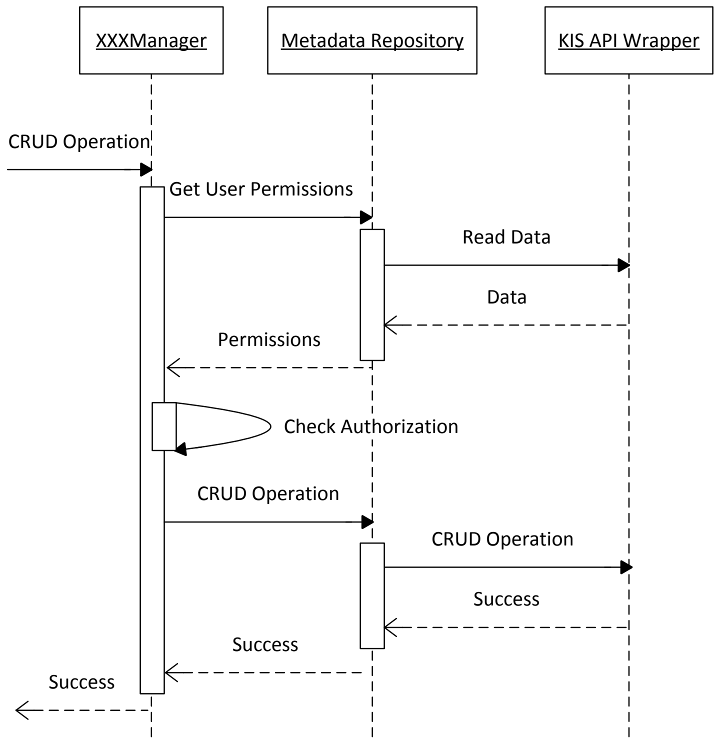

Figure 30: UML Sequence Diagram – HM/Server: Manage Health Profile

The appropriate Manager receives the operation and checks whether the operation issuer has enough permissions to accomplish the operation. To that end, the Metadata Repository is consulted, which in turn consults KIS. Received data are internally interpreted in the Metadata Repository as permissions and these permissions are checked later on in the Manager. If the operation is authorized then the process goes on, forwarding the CRUD operation to the Metadata Repository which in turn forwards the operation to KIS.

3.3.4.4 Configure Health Monitor

In order to keep track of the user's health, the HM requires some configurations regarding:

- Sensors
- Data Processing
- Anomalies

This component handles all operations that deal with these data. The overall process to access/manipulate configurations always follows the same steps, irrespective of the particular configuration type, and is very similar to the process sketched in the previous section. It is described in the following diagram (see Figure 31).

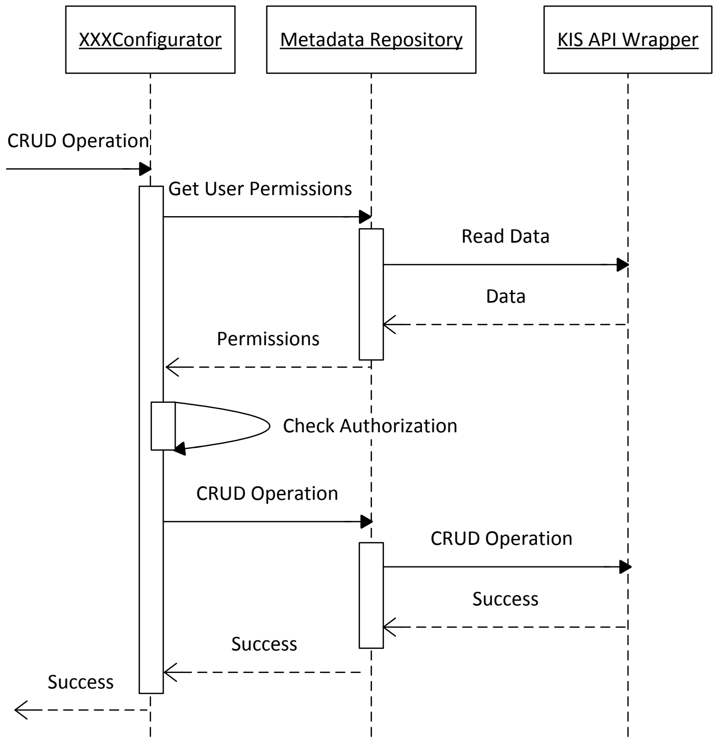

Figure 31: UML Sequence Diagram – HM/Server: Configure HM

3.3.4.5 Manage Alarms

The user must be able to manage alarms, that is, add, configure and remove them. Typically, when an alarm is defined, the following information must be provided:

- An identifier of the anomaly that must be notified
- Who must be notified
- Which channels must be used for notification

In the end, the user must be able to apply CRUD operations to alarms. The overall process to access/manipulate alarms always follows the same steps, and is very similar to the process defined in the previous sections. We refer the reader to Figure 18 for further information.

3.3.4.6 Configure Sensor Analysis

The user configures preferences when sensor data must be analysed. The process followed matches the same pattern presented so far, in order to manage any kind of configuration (see Figure 31).

3.3.4.7 Analyse Sensor Data

This operation involves obtaining measurements information and applying some kind of processing on them. In the simplest case, no processing is done and therefore raw measurements information is obtained.

The processed measurements may belong to a particular time period or to specific time instants. In the last case, if the last measurements are requested then ‘almost’ real-time information is obtained.

The user may define some preferences in order to present results processed in some specific way.

Finally, interesting a posterior data analysis may take place according to user preferences. For example, automatic pattern recognition may be conducted. Also, posterior anomaly detection should be supported.

The general procedure is shown in the next diagram (see Figure 32):

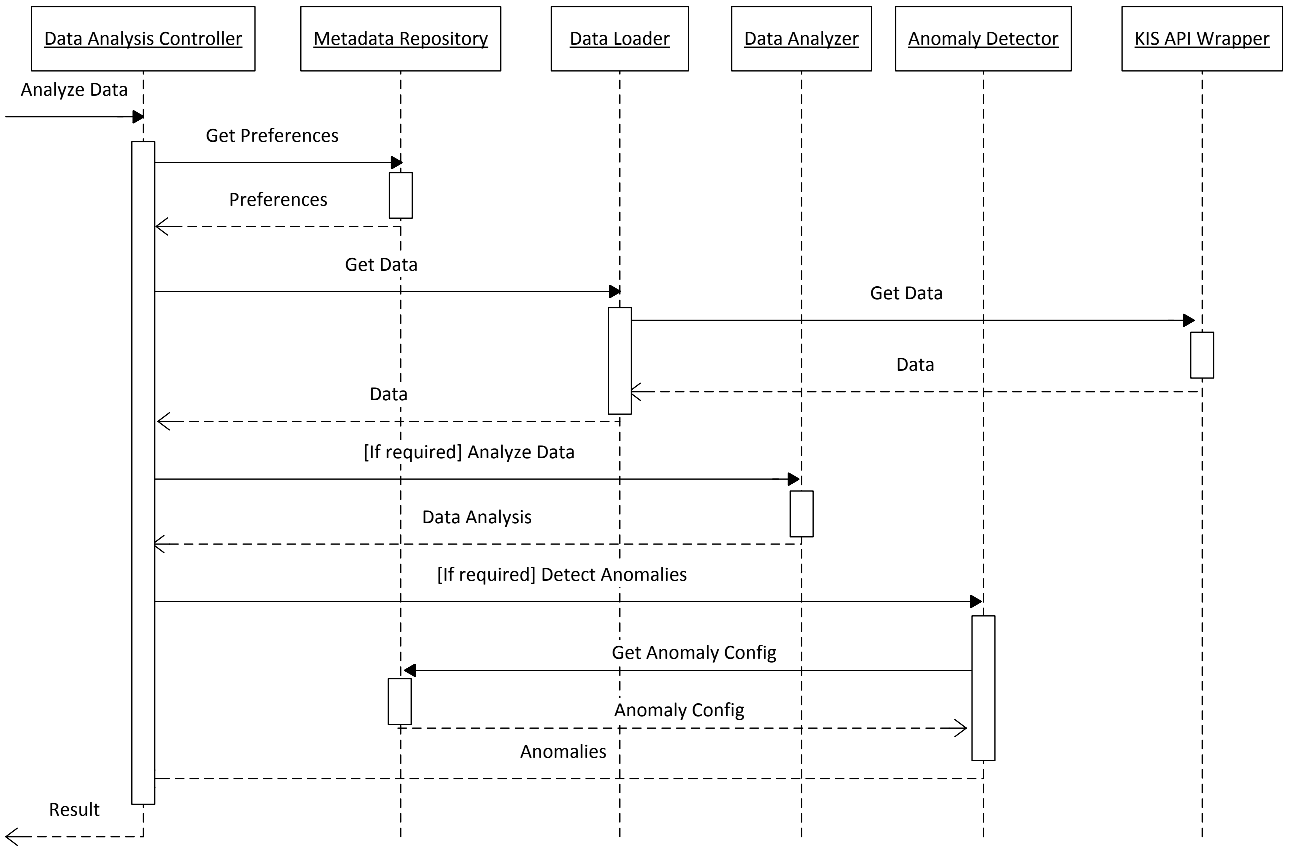

Figure 32: UML Sequence Diagram – HM/Server: Analyse Sensor Data

After the request is processed as described in Figure 15, the Data Analysis Controller takes control and retrieves user preferences from the Metadata Repository. Then data are efficiently retrieved from the Data Loader. If a particular kind of analysis is requested then the Data Analyser gets activated and the analysis results are obtained. If anomaly detection is required then the Anomaly Detector subcomponent is activated and information about the detected anomalies is retrieved. Finally, according to the requested operation, the result is made up and returned

3.4 Context-aware Speech Recognition

3.4.1 Public

This subsection describes functionalities offered by CADE to the rest of the ALFRED framework. The number of such functionalities is quite low. The reason for this is that most aspects of the spoken interaction between the user and ALFRED are handled internally within CADE, and by means of requests from CADE to other components. Nevertheless, other components need to access CADE in certain situations, described in more detail below.

3.4.1.1 Dialogue Domain

Whenever a new ALFRED app has been installed, CADE needs to be informed so that it can load the Dialogue Domain Description for the app and thereby understand spoken input related to the app. For this reason, CADE offers a method for other components to submit notifications that a new app has been installed or enabled.

3.4.1.1.1 Enabled App

This service should be invoked when a new ALFRED app has been installed. It will cause the CADE backend to load the app resources from the Marketplace, enabling it to manage spoken interactions in relation to the app (see Figure 33).

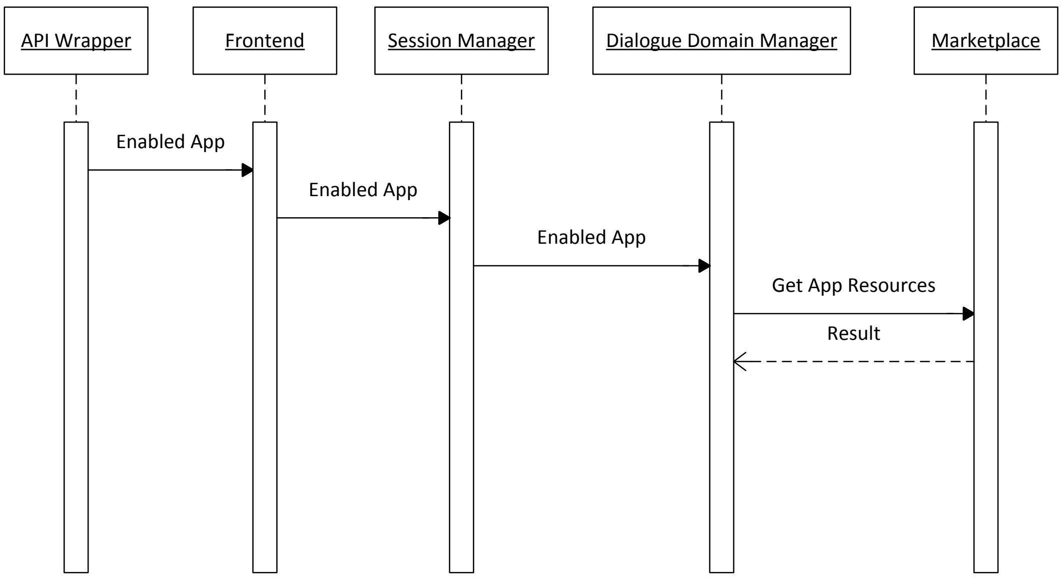

Figure 33: UML Sequence Diagram – CADE: Enabled App.

3.4.1.2 Haptic Input

These services allow other components to provide input from the GUI to the Dialogue Engine, allowing it to update its state and respond appropriately.

3.4.1.2.1 PTT Activated

This service should be invoked when the push-to-talk (PTT) icon in the GUI is clicked. The notification will then be forwarded to the ASR so that the system starts listening to the microphone. (See Figure 34)

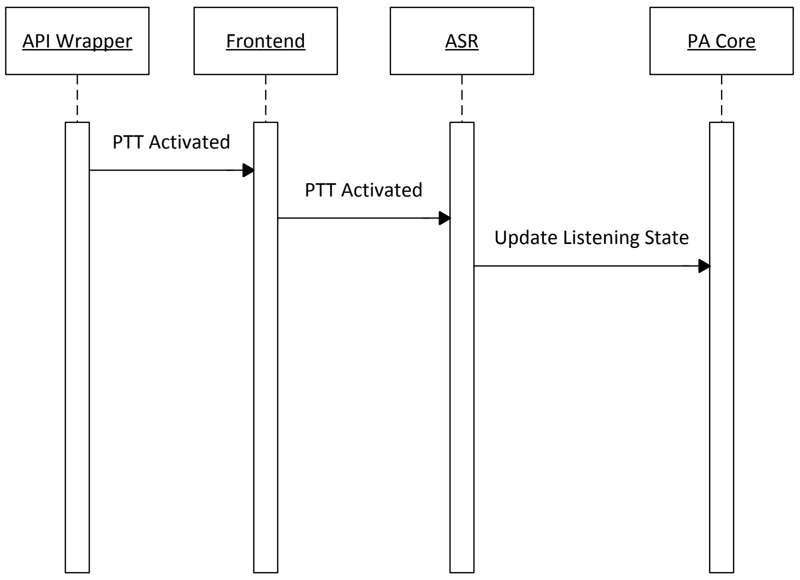

Figure 34: UML Sequence Diagram – CADE: PTT Activated.

3.4.1.2.2 Option Selected

In the case that the GUI is used by the user to clarify what he or she has said, this service is invoked when a choice is clicked. This notification will then be forwarded to the backend, allowing it to update its state and respond.

3.4.1.3 Activities

Whenever a component initiates or completes a speech enabled activity, it should use one of the services below to notify CADE.

3.4.1.3.1 Started Activity

Whenever some component decides to initiate an activity that should be handled verbally by ALFRED, it needs to notify CADE about this. For example, if the Event Manager decides to suggest an event, it notifies CADE that it has started an activity. This allows CADE to find a matching dialogue plan for the activity and thereby to ask the user a question or notify him or her about some information in relation to the activity (see Figure 35).

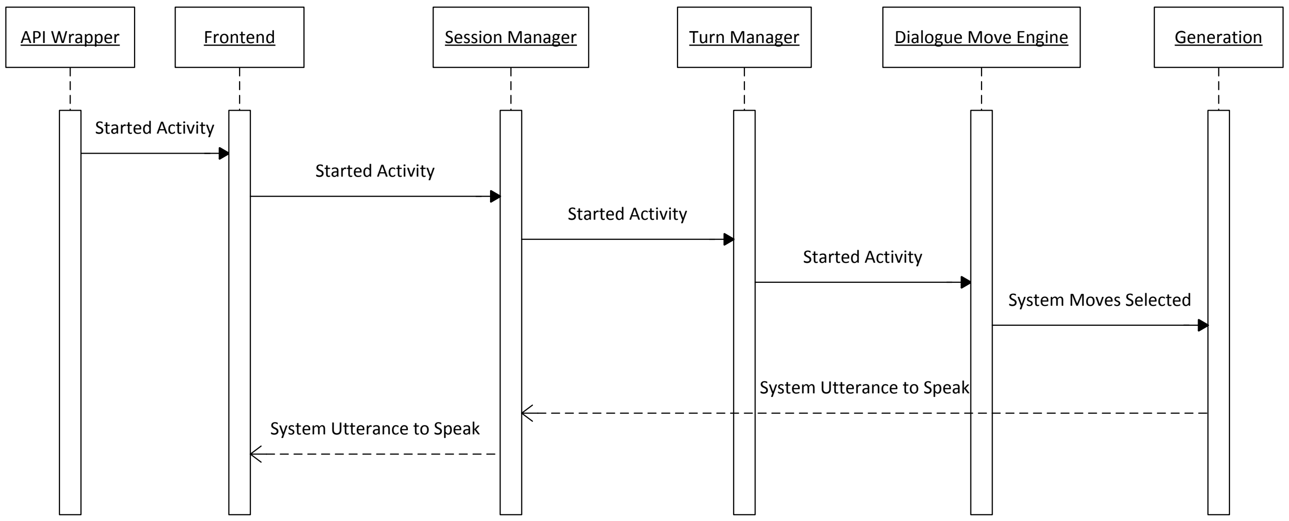

Figure 35: UML Sequence Diagram – CADE: Started Activity

3.4.1.3.1.1 Stopped Activity

When some activities are completed, this fact should be communicated verbally to the user. For example, if an event has been scheduled, the Event Manager notifies CADE that the activity ended. This allows CADE to report the result of the new scheduling to the user and to update its dialogue state accordingly. The sequence depicted in Figure 35 is similar for Stopped Activity.

3.5 ALFREDO - Marketplace

3.5.1 Overview

The Marketplace offers a unique place where end users, developers and testers can find, install, publish and test ALFRED applications.

Applications and games will be offered by the Marketplace in a mobile based UI that will allow end users to discover, buy, install, upgrade and uninstall applications via the Marketplace.

The Marketplace will also offer developers the possibility to publish ALFRED compliant applications. To achieve this, the Marketplace will be accessible through a web based UI, where developers will manage and control their published applications.

At the same time, a specific user role, called tester, will access the marketplace through web or mobile in order to test, accept or deny submitted applications. This will assure that the published applications will fulfil ALFRED requirements.

In the next subsections, we will describe the different sequences of the Marketplace's processes.

3.5.2 Public Services

One of the main purposes of the Marketplace is to provide other components with the necessary data about available applications, either installed locally or on the market. These functionalities will be encapsulated in an **API Component** which will be accessible by other ALFRED components.

3.5.2.1 Downloading and installing an application

End users will be able to download an application from the Marketplace to their device. The process of download and install a new application is detailed in the diagram below (see Figure 36).

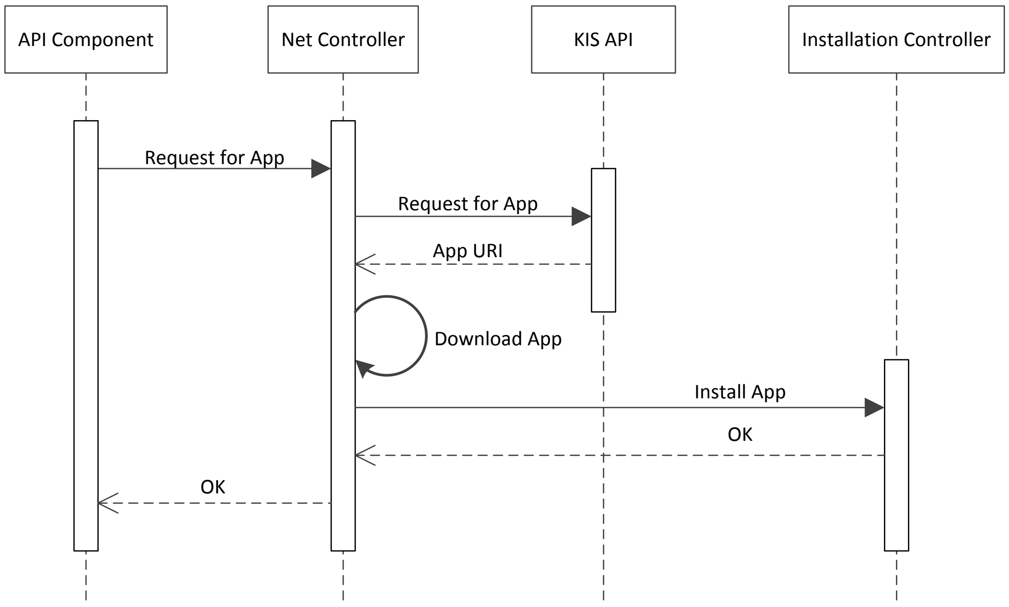

Figure 36: UML Sequence Diagram – AM: Downloading and Installing an Application

The process starts with a request for the installation of a new application or game. The Marketplace will forward the request to the KIS service, responsible for the application storage management and the download will start. Once the game or application is on the Marketplace,, the Marketplace will be responsible for the installation on the user's device and inform the involved components.

3.5.2.2 Uninstalling an application

All ALFRED applications can be uninstalled whenever the user wants.

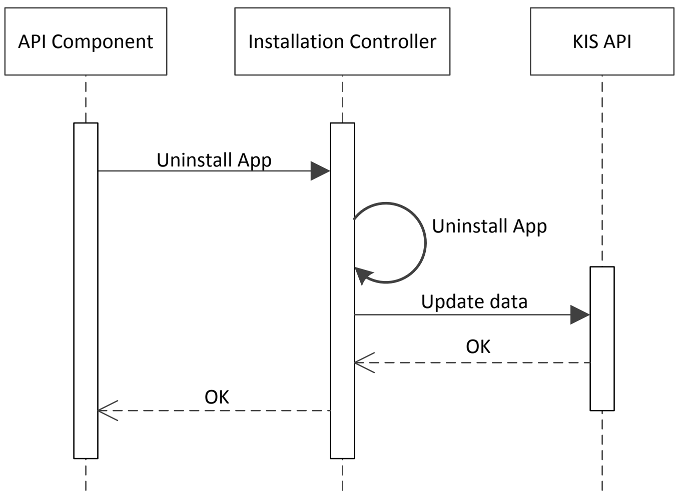

Figure 37: UML Sequence Diagram – AM: Uninstalling an Application

The process starts with the request for the uninstallation of a specified application. Then, the API Component will handle the request and the Installation Controller will uninstall the application from the user's device, deleting the record from the KIS module and informing the involved components.

3.5.2.3 Upgrading an application

Users will be able to upgrade their installed applications through the Marketplace. The process is depicted in the diagram below (see Figure 38).

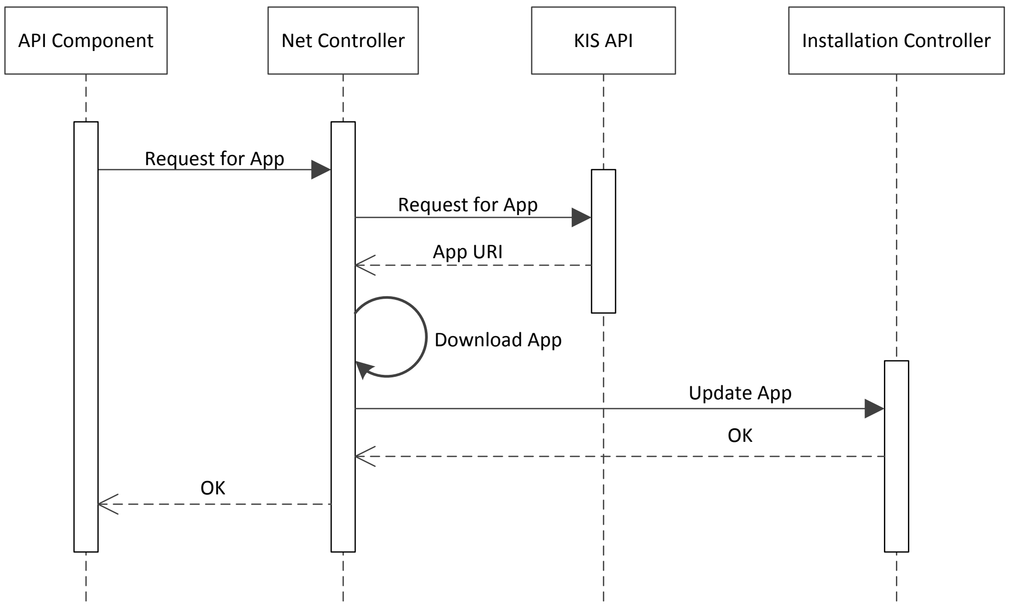

Figure 38: UML Sequence Diagram – AM: Upgrading an Application

This process can start both automatically or manually by the user. In any case, the Marketplace will be responsible for the download via Net controller by requesting the application to KIS and install the new version of the application as described in section 3.5.2.1.

3.5.2.4 Getting application information

Some components like CADE need information about an application. The Marketplace will be responsible to deliver that information to the petitioner.

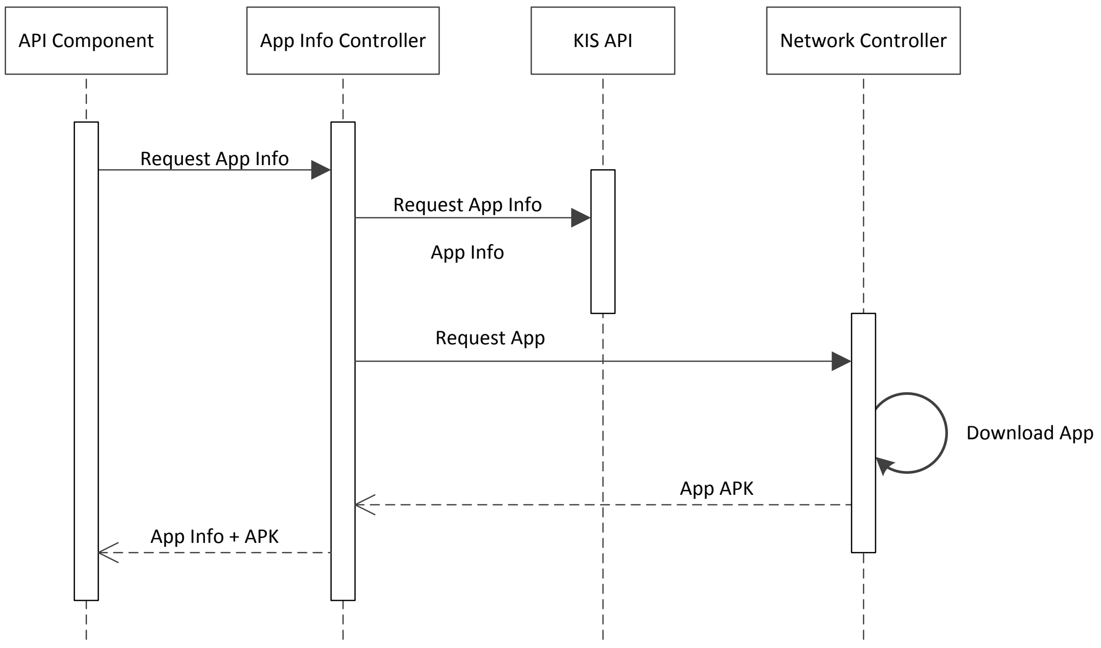

Figure 39: UML Sequence Diagram – AM: Getting Application Information

When the process is started, the Marketplace forwards the call to the KIS component which returns the information related to the application or game stored on the storage service. The Marketplace manages the information and returns it to the component that has started the process.

3.5.2.5 Searching for an application

One of the key functionalities of Marketplace is to allow other components to discover new applications by criteria. The diagram below (see Figure 40) describes the process of an application search.

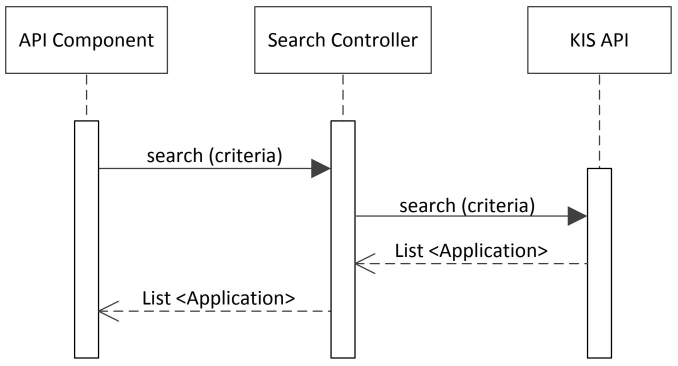

Figure 40: UML Sequence Diagram – AM: Searching for an Application

The Marketplace publishes a service via the API Component that forwards the query to the specific KIS service. Then KIS returns the list of applications and/or games that matches the criteria to the Marketplace which manages the information and returns it to the component that has started the search.

3.5.2.6 Rating an application

One of the functionalities the Marketplace offers to the ALFRED users is the possibility to rate an installed application (see Figure 41).

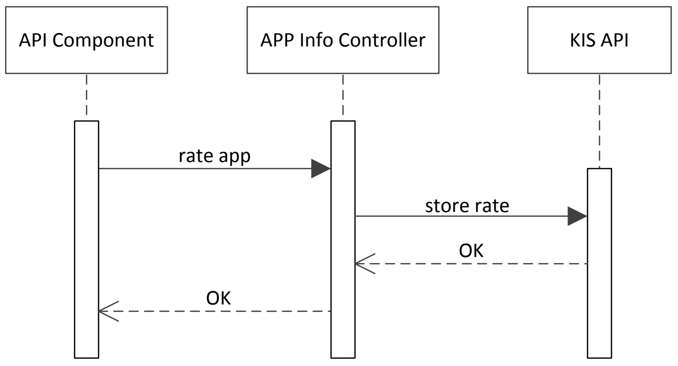

Figure 41: UML Sequence Diagram – AM: Rating an Application

The API Component is requested to rate an application, the request is forwarded to the APP Info Controller, handling the request and storing the rating via KIS.

3.5.3 Developer services

These services will only be accessible for developers and through the web based UI.

3.5.3.1 CRUD operations for developer

Developers will be able to execute all four CRUD operations (Create, Read, Update and Delete) on their authorized applications (see Figure 42).

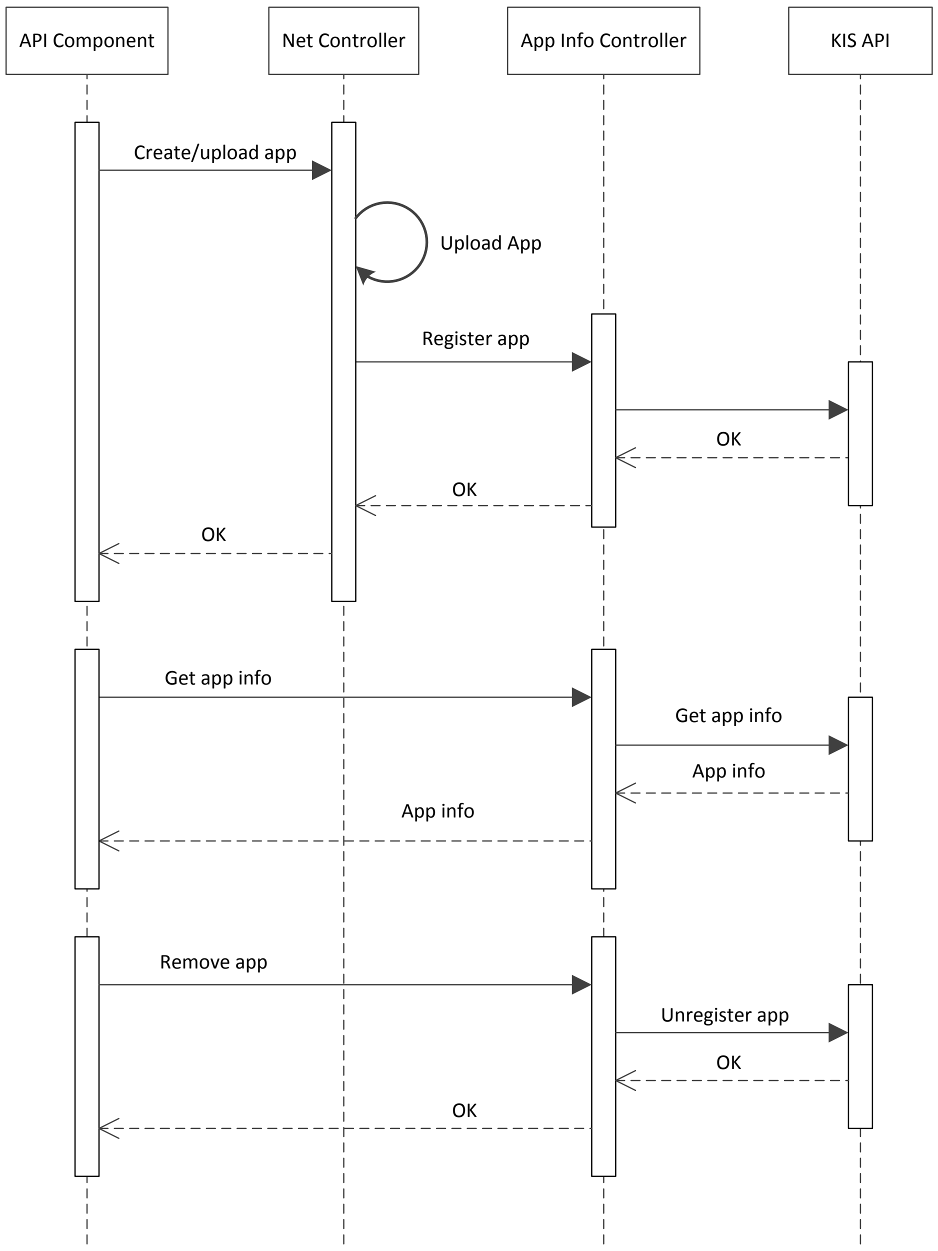

Figure 42: UML Sequence Diagram – AM: CRUD Operations for Developer

- The Create operation corresponds to the publishing process that consists of uploading the application to the Marketplace. After that, the application is open for the review process and will eventually be published on the Marketplace once it has been reviewed.
- The Update operation works in a similar way as create because the update will need to pass the review process as well. The developer will upload the application and the review process will start.
- The Read operation will allow the developer to list all his operations and see the status of the applications, e.g. under review, published or refused.

- The Delete operation will remove the application and delete it from the Marketplace.

3.5.4 Tester services

The reviewing process will be only performed by tester users and will be accessible through web or mobile based UI.

This set of operations is part of the reviewing process that will start once a developer uploads an application. Testers will be responsible for the acceptance or rejection of an application based on internal tests.

The review process of an application is depicted in the diagram below (see Figure 43).

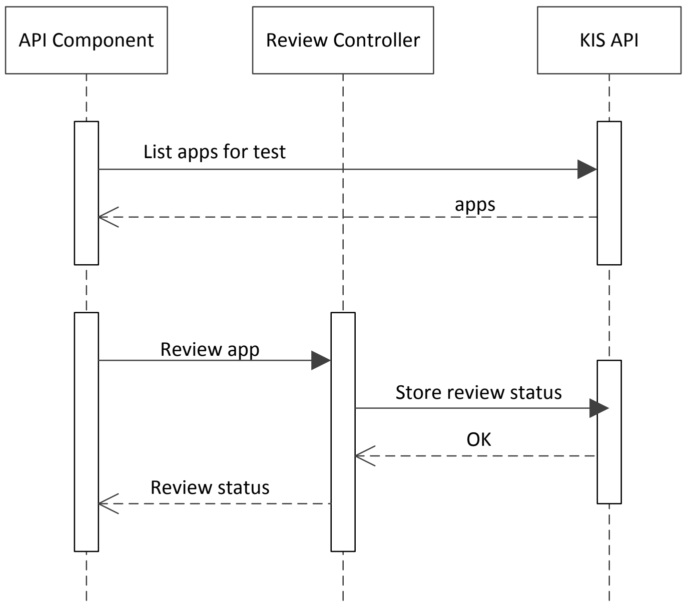

Figure 43 : UML Sequence Diagram – AM: Review Process

In order to list his/her assigned applications, a tester sends a request to the Marketplace. The API Components handles the request and queries to KIS API for the pending applications for testing.

Once a tester has an overview of the assigned applications, he/she can select an application and start reviewing it. Once an application has been reviewed, the tester can decide if the application is ready to be published or not, depending on the test cases. The tester sends the result to KIS which will store it. Finally the result is returned to the component that has started the process.

It's important to notice that an application will not be published until a tester has approved it. The application lifecycle is described on the diagram below (see Figure 44).

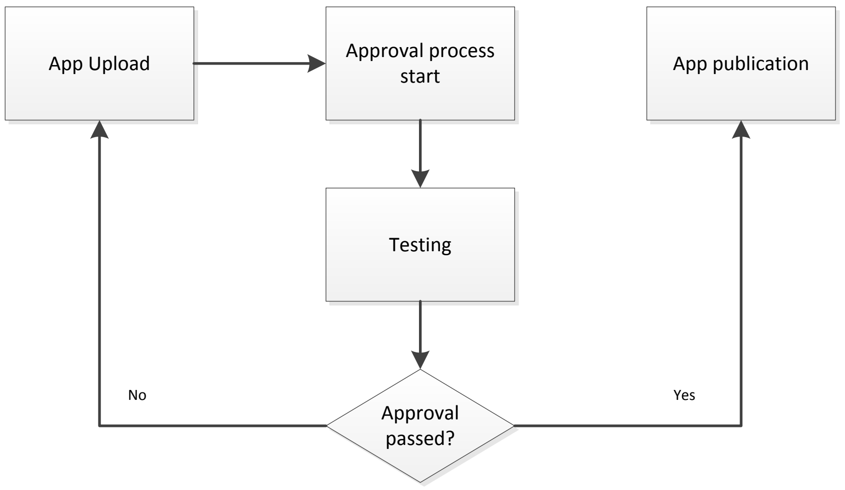

Figure 44: Activity Diagram – AM: Application Lifecycle

3.6 Personalization Manager

3.6.1 Overview

The Personalisation Manager component cooperates with most (other) components in the ALFRED system. Many different functionalities of the ALFRED system will be achieved with the help of the Personalisation Manager but the functionalities of the component itself can be categorized in three types: providing access to user (i.e. older person) profile information, providing context by reasoning on user profile information and recommending events on which the older person could be interested.

In the following figure the high level overview of the processes and interactions of components and sub-component is depicted. Sections 3.6.2 to 3.6.5 provide a more detailed description of the functionalities and interactions.

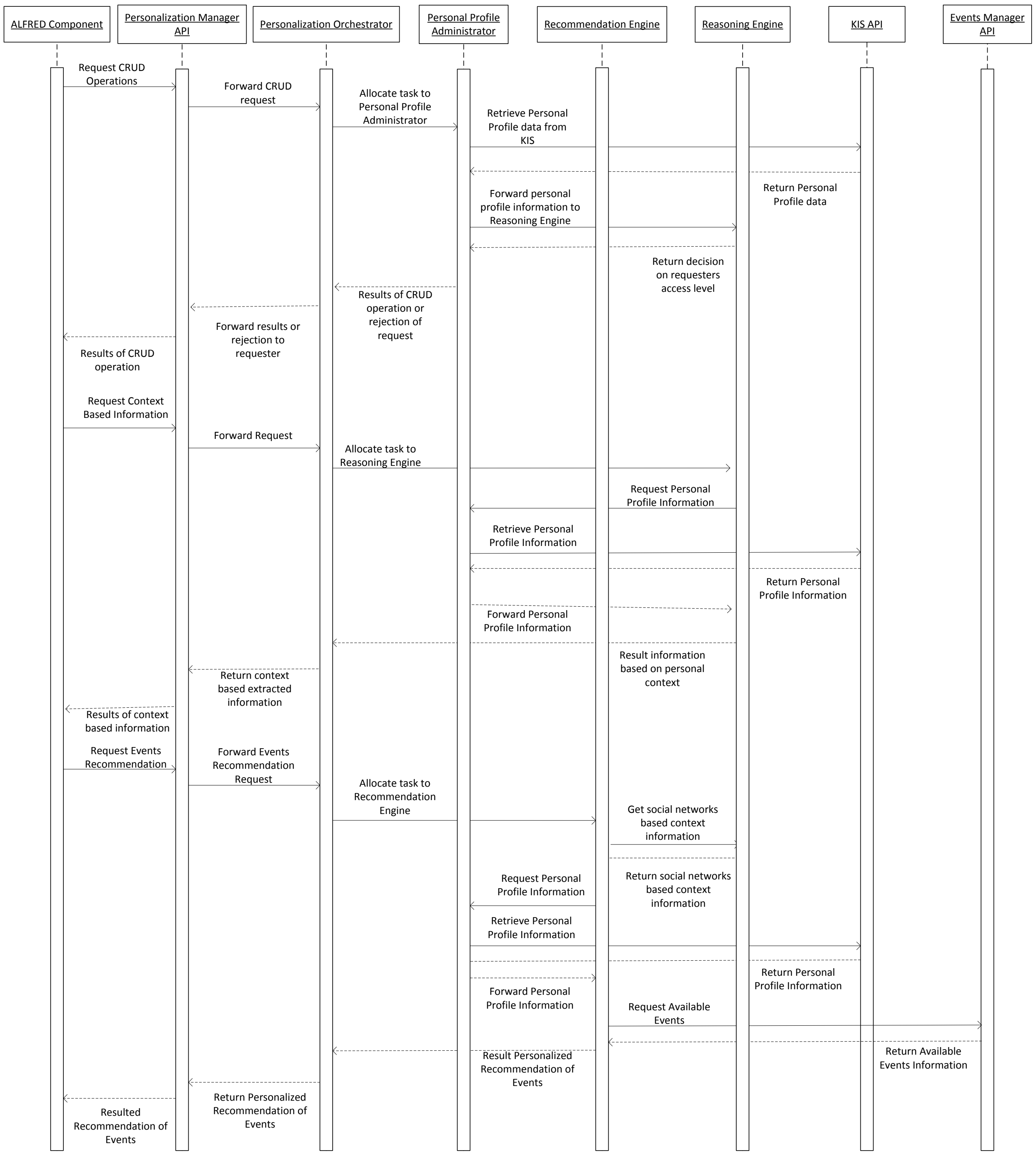
ALFRED Component
Personalization Manager API
Personalization Orchestrator
Personal Profile Administrator
Recommendation Engine
Reasoning Engine
KIS API
Events Manager API
Request CRUD Operations
Forward CRUD request
Allocate task to Personal Profile Administrator
Retrieve Personal Profile data from KIS
Return Personal Profile data
Forward personal profile information to Reasoning Engine
Return decision on requesters access level
Results of CRUD operation or rejection of request
Forward results or rejection to requester
Results of CRUD operation
Request Context Based Information
Forward Request
Allocate task to Reasoning Engine
Request Personal Profile Information
Retrieve Personal Profile Information
Return Personal Profile Information
Forward Personal Profile Information
Result information based on personal context
Return context based extracted information
Results of context based information
Request Events Recommendation
Forward Events Recommendation Request
Allocate task to Recommendation Engine
Get social networks based context information
Return social networks based context information
Request Personal Profile Information
Retrieve Personal Profile Information
Return Personal Profile Information
Forward Personal Profile Information
Request Available Events
Return Available Events Information
Result Personalized Recommendation of Events
Return Personalized Recommendation of Events
Resulted Recommendation of Events

Figure 45: UML Sequence Diagram – PM: Overview

3.6.2 Personalization Orchestrator

This sub-component receives the requests for services (functionalities) provided by the Personalization Manager. The Personalization Orchestrator will orchestrate and allocate the task for the appropriate sub-component to execute and then will provide back the results. The functionalities and communications focusing on the Personalization Orchestrator is depicted in the diagram below (see Figure 46).

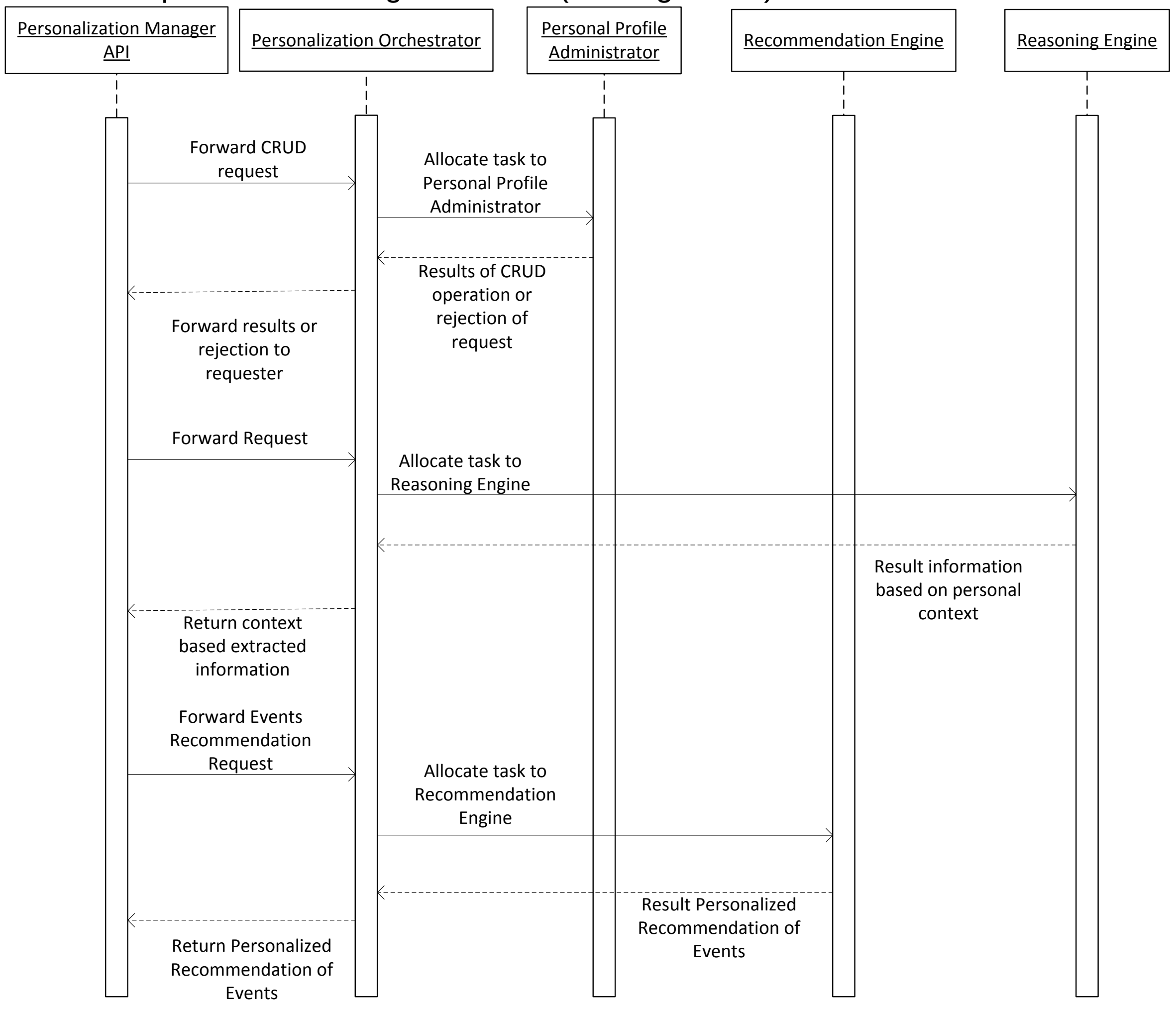
Personalization Manager API
Personalization Orchestrator
Personal Profile Administrator
Recommendation Engine
Reasoning Engine
Forward CRUD request
Allocate task to Personal Profile Administrator
Results of CRUD operation or rejection of request
Forward results or rejection to requester
Forward Request
Allocate task to Reasoning Engine
Result information based on personal context
Return context based extracted information
Forward Events Recommendation Request
Allocate task to Recommendation Engine
Result Personalized Recommendation of Events
Return Personalized Recommendation of Events

Figure 46: UML Sequence Diagram – PM: Interactions and Processes of the Personalization Orchestrator

3.6.3 Personal Profile Administrator

This sub-component is acting as a mediator to perform CRUD operations to the data that are stored in the KIS component. Moreover it will be responsible for privacy and access rights issues for the CRUD operations with the help of the Reasoning Engine. When a CRUD operation is requested, the Personal Profile Administrator will retrieve the data from the bucket where they are stored in KIS. It employs the Reasoning Engine sub-component to examine the access level of the requester and the access level of the information of the requested CRUD. If the result of the evaluation is that the requester is allowed to make the operation, and then the Personal Profile Administrator will execute it, otherwise it will not. The interactions focused on the Personal Profile Administrator are depicted in the diagram below (see Figure 47).

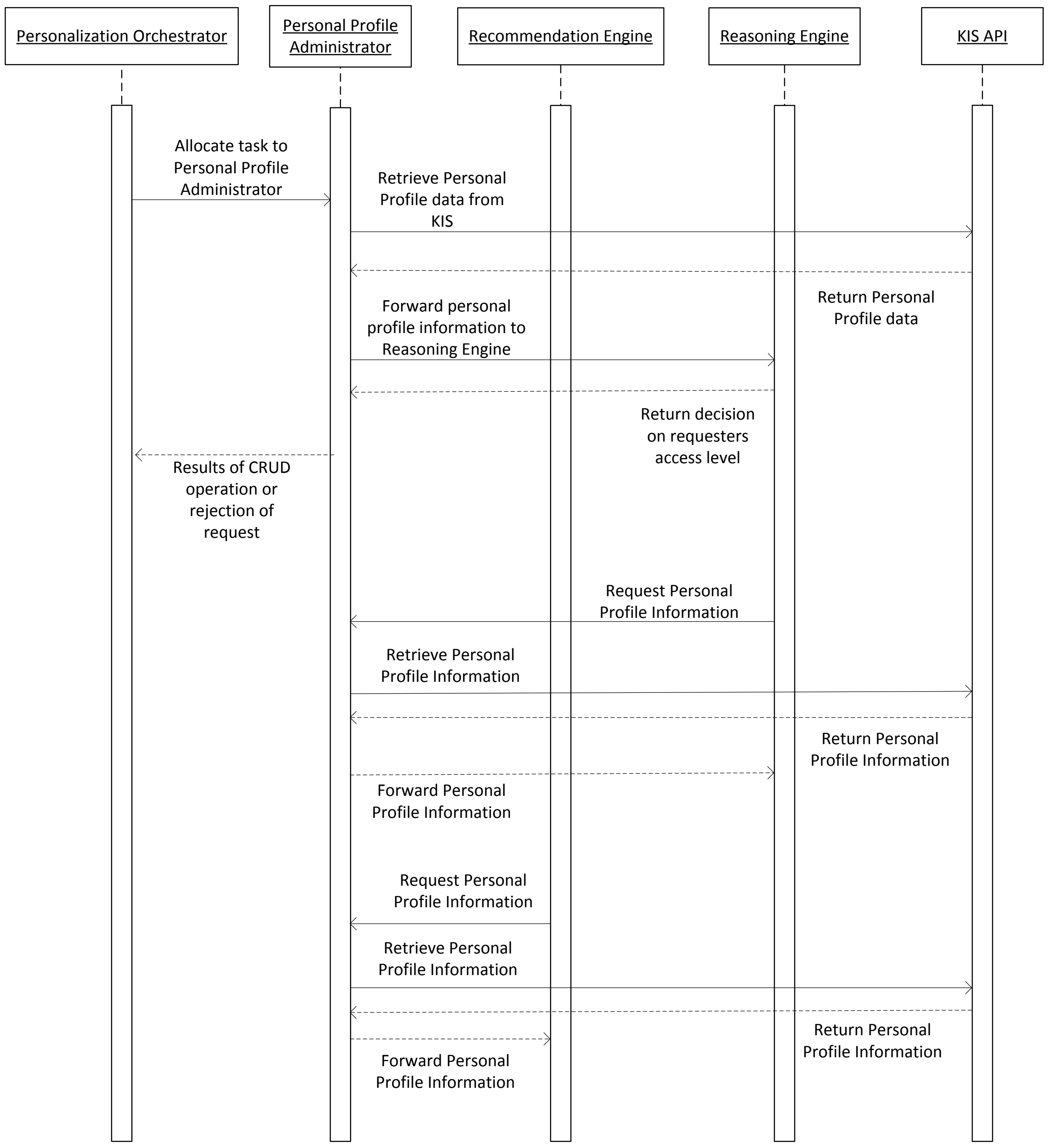
Personalization Orchestrator
Personal Profile Administrator
Recommendation Engine
Reasoning Engine
KIS API
Allocate task to Personal Profile Administrator
Retrieve Personal Profile data from KIS
Return Personal Profile data
Forward personal profile information to Reasoning Engine
Return decision on requesters access level
Results of CRUD operation or rejection of request
Request Personal Profile Information
Retrieve Personal Profile Information
Return Personal Profile Information
Forward Personal Profile Information
Request Personal Profile Information
Retrieve Personal Profile Information
Return Personal Profile Information
Forward Personal Profile Information

Figure 47: UML Sequence Diagram – PM: Interactions and Processes of the Personal Profile Administrator

3.6.4 Recommendation Engine

This sub-component is recommending events that are personalized to the likings of the older person. It uses data from various components and subcomponents such as the Events Manager and the Personal Profile Administrator. It capitalizes on these data by artificial intelligent applications (e.g., matchmaking, classification) and recommends events that could be of interest to the user. It communicates with the Reasoning Engine to obtain context from social networks of the older person, with the Event Manager to get available events, with the Personal Profile Administrator for user (older person) specific information and finally with the Health Manager to obtain (relatively) real-time information about the physical condition of the older person. The interactions and processes of the Recommendation Engine are depicted in the diagram below (see Figure 48).

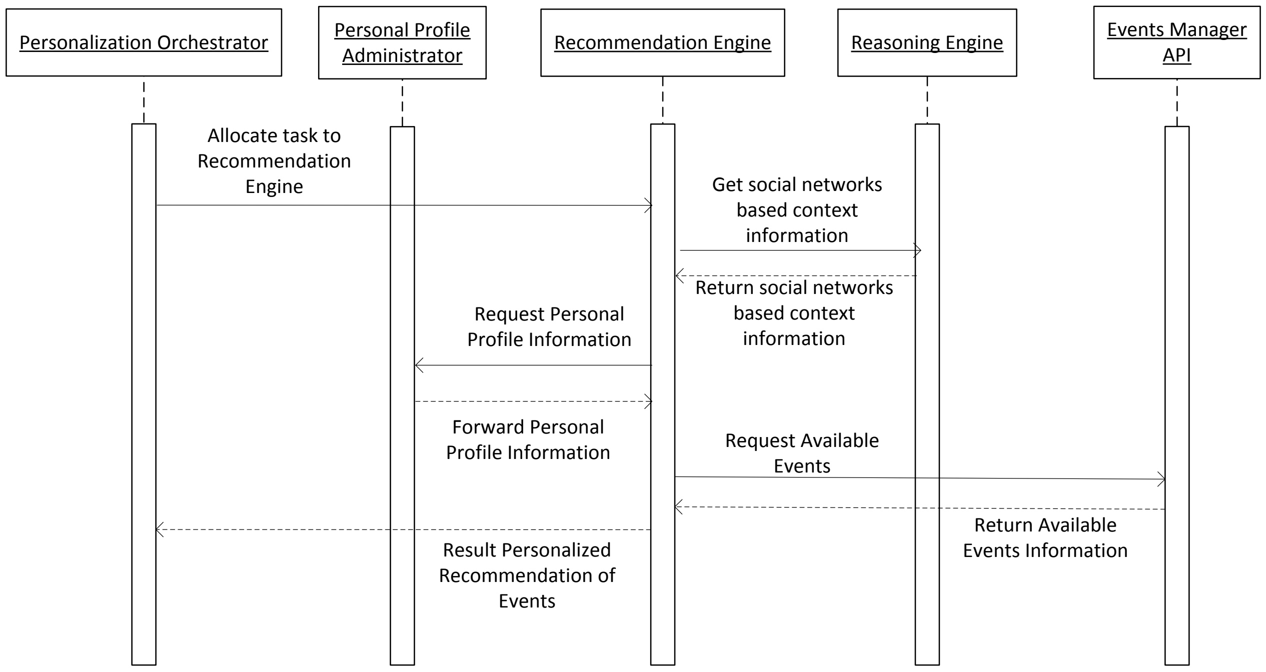
Personalization Orchestrator
Personal Profile Administrator
Recommendation Engine
Reasoning Engine
Events Manager API
Allocate task to Recommendation Engine
Get social networks based context information
Return social networks based context information
Request Personal Profile Information
Forward Personal Profile Information
Request Available Events
Return Available Events Information
Result Personalized Recommendation of Events

Figure 48: UML Sequence Diagram – PM: Interactions and Processes of the Recommendation Engine

3.6.5 Reasoning Engine

This subcomponent will perform reasoning applications for providing three main functionalities. The first functionality is a relatively simple process of identifying if a requester of CRUD operations on personal profile data has the required access level for it. This functionality involves communication with the Personal Profile Administrator only. The second functionality concerns mining social networks of the user to provide social context to the selection of recommended events. This functionality involves communications with the Recommendation Engine and Personal Profile Administrator sub-components. Finally, the Reasoning Engine will provide context information by reasoning over the personal information of the user. The older person will communicate with the ALFRED system through the CADE component (see section 3.4) and in some cases the commands/inputs of the user can be resolved only with context knowledge for the user (see example “Call my daughter” in section 2.6). Reasoning Engine can contribute making other components context aware. This functionality is achieved in cooperation with the Personal Profile Administrator sub-component.

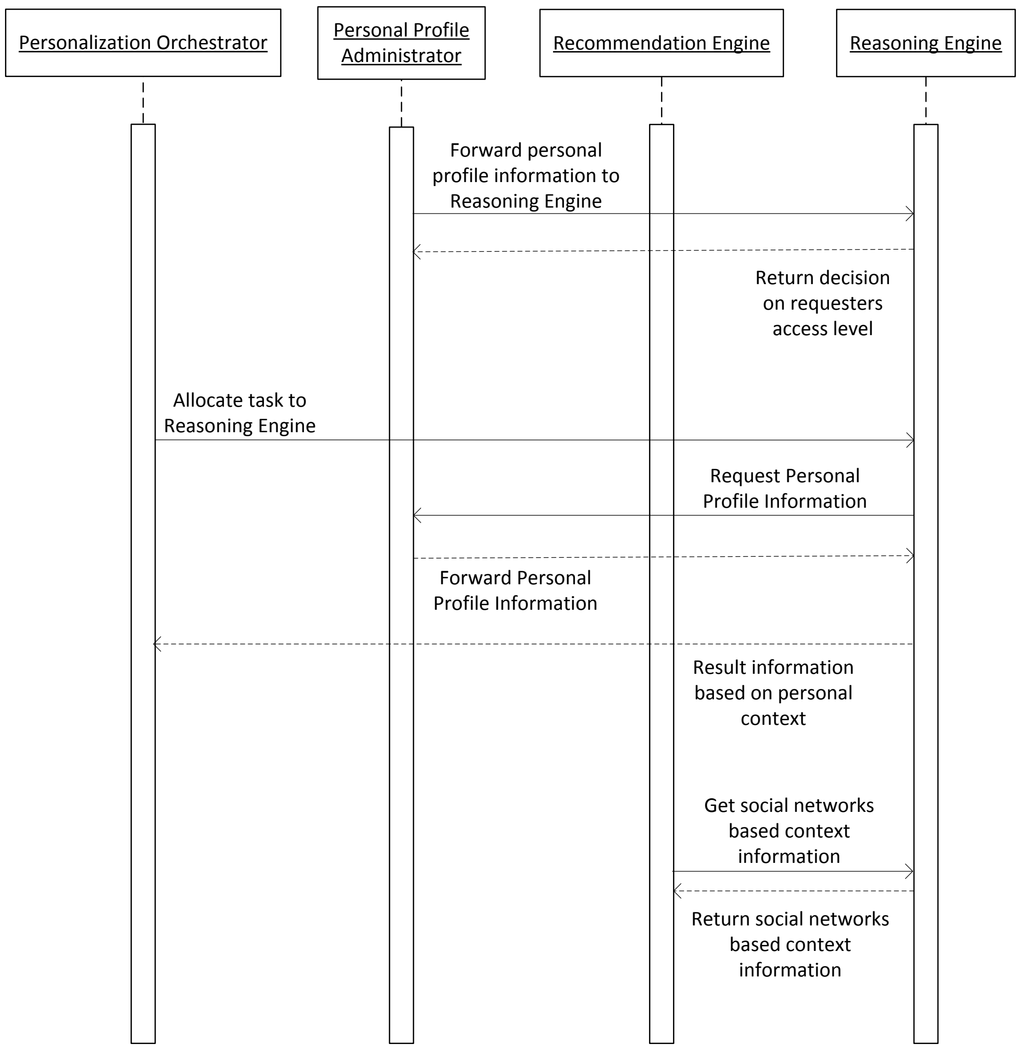
Personalization Orchestrator
Personal Profile Administrator
Recommendation Engine
Reasoning Engine
Forward personal profile information to Reasoning Engine
Return decision on requesters access level
Allocate task to Reasoning Engine
Request Personal Profile Information
Forward Personal Profile Information
Result information based on personal context
Get social networks based context information
Return social networks based context information

Figure 49: UML Sequence Diagram – PM: Interactions and Processes of the Reasoning Engine

3.7 Event Manager

3.7.1 Overview

The Event Manager's objective is to build and maintain a Knowledge Base of events which can be browsed and administered by users through a GUI; the real value of this Events Knowledge Base however, concerns the use of the persisted events in the knowledge base by the Personalisation Manager component to provide tailored recommendations of events to ALFRED users.

The Event Manager component is built on 4 main blocks (namely, Web Portal, Web Crawler, Event Miner and Events Knowledge Base Administrator) for which the functionalities they provide will be explained in the following sections. The Event Manager component's interactions are composed solely with users through the Web Portal sub-component and with the component through the Event Manager API.

The high-level functionalities of the Event Manager are:

- Entering new events to the Events Knowledge Base
- Browsing events currently persisted in the Events Knowledge Base
- Searching the Web for events and evaluating which events should be persisted in the Events Knowledge Base
- Making data from the Events Knowledge Base available through the Event Manager API

The sequence diagram for Event Manager to provide the above functionalities is depicted in the following diagram (see Figure 50).

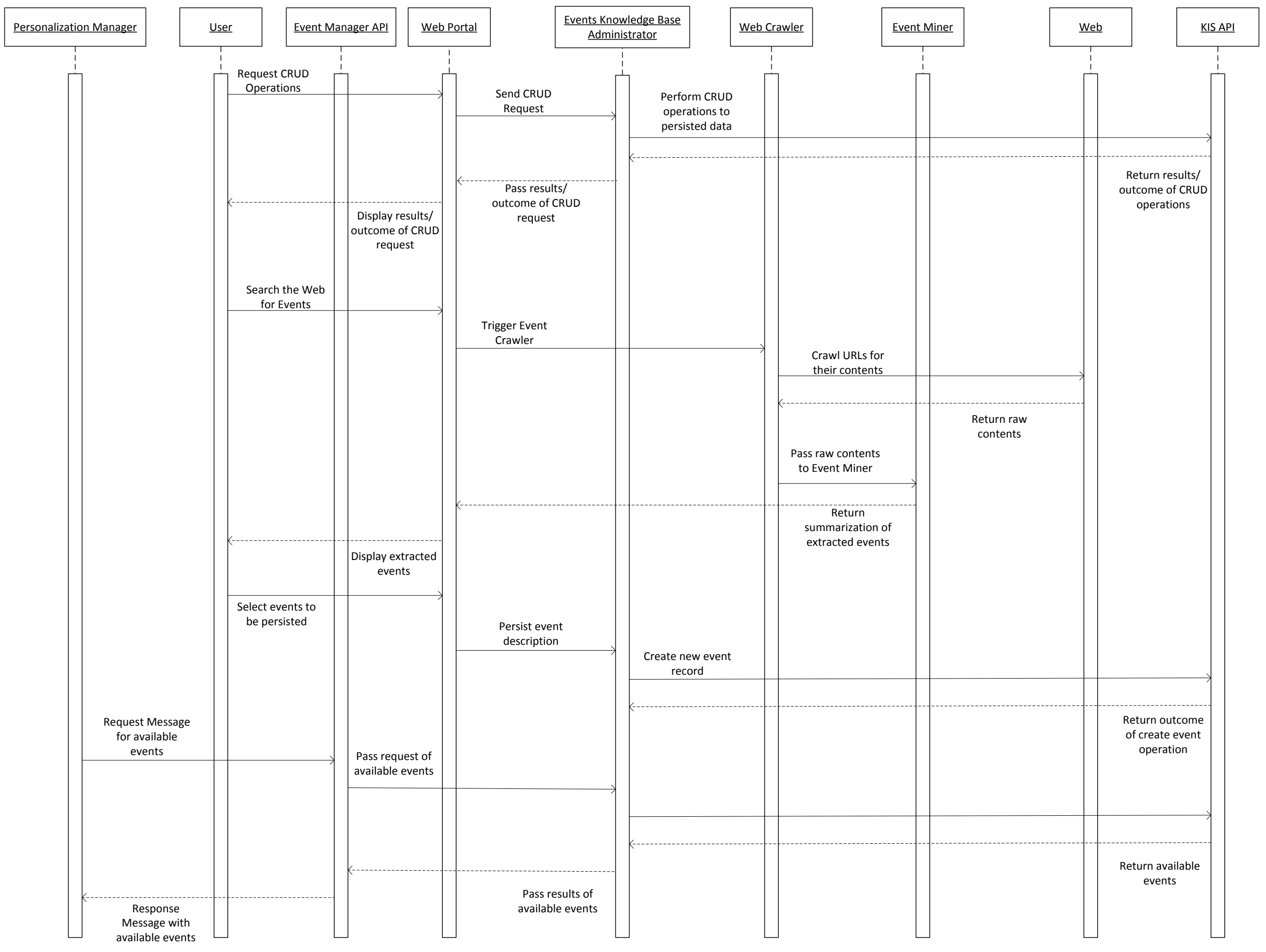
Personalization Manager
User
Event Manager API
Web Portal
Events Knowledge Base Administrator
Web Crawler
Event Miner
Web
KIS API
Request CRUD Operations
Send CRUD Request
Perform CRUD operations to persisted data
Return results/ outcome of CRUD operations
Pass results/ outcome of CRUD request
Display results/ outcome of CRUD request
Search the Web for Events
Trigger Event Crawler
Crawl URLs for their contents
Return raw contents
Pass raw contents to Event Miner
Return summarization of extracted events
Display extracted events
Select events to be persisted
Persist event description
Create new event record
Return outcome of create event operation
Request Message for available events
Pass request of available events
Return available events
Pass results of available events
Response Message with available events

Figure 50: UML Sequence Diagram – EM: Overview

3.7.2 Web Portal

The Web Portal sub-component is the interaction point of users with the Events Knowledge Base. Users through the Web portal are able to enter events to the Knowledge Base, browse events that are already persisted in this knowledge base, or search the Web for events with one or more parameters of their choosing and select which of the resulted events should be persisted in the Events Knowledge Base. In general, users will be able to perform CRUD operations through the Web Portal of the Events Knowledge Base. However every user will have “delete” rights only to events that are inserted by him/her.

One or more (admin) users will be granted with rights to delete events submitted by any user. This will be part of an “event review process” which allows special users (the reviewers) to decide whether a manually entered event will be available to the users or will be rejected (e.g. because of incorrect data). Only events which have passed this review will be available for suggestion. The sequence diagram below (see Figure 51) visualizes the interactions and processes of the Web Portal subcomponent.

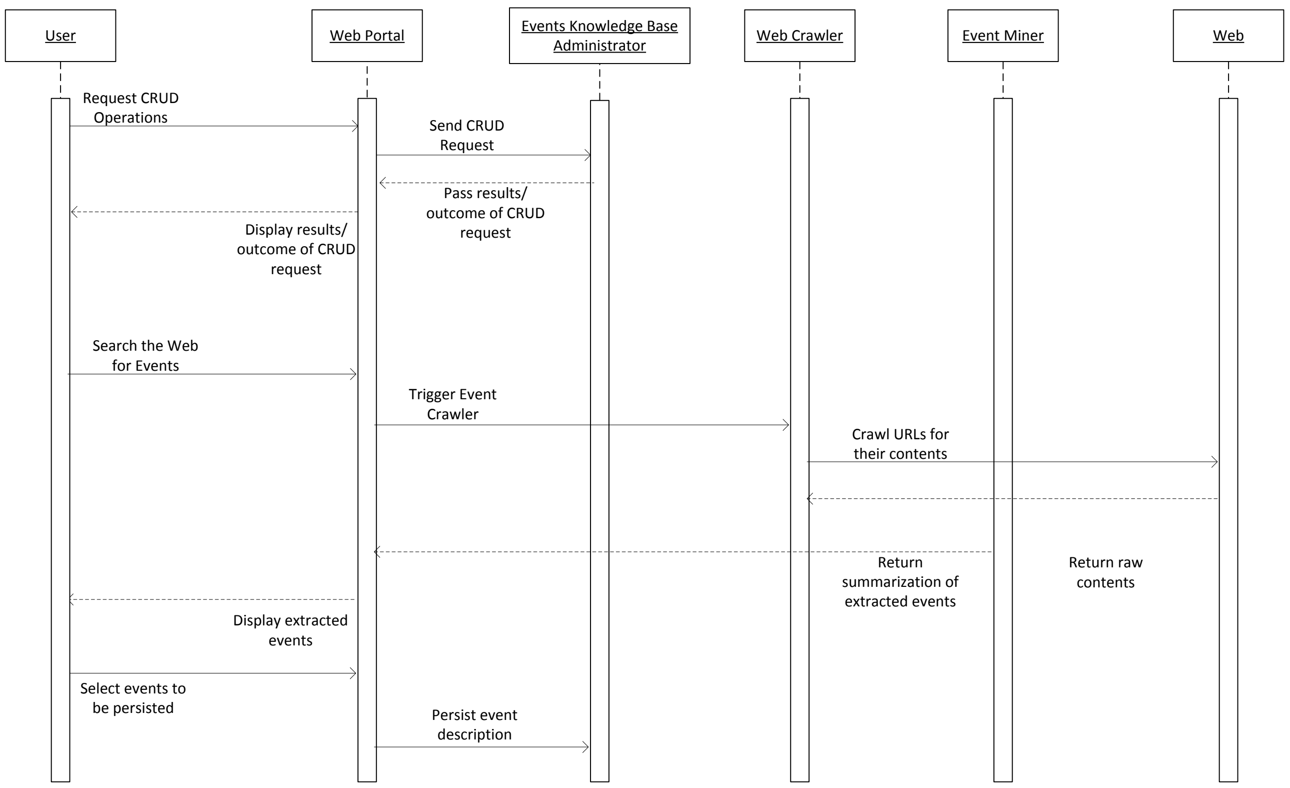
User
Web Portal
Events Knowledge Base Administrator
Web Crawler
Event Miner
Web
Request CRUD Operations
Send CRUD Request
Pass results/ outcome of CRUD request
Display results/ outcome of CRUD request
Search the Web for Events
Trigger Event Crawler
Crawl URLs for their contents
Return summarization of extracted events
Return raw contents
Display extracted events
Select events to be persisted
Persist event description

Figure 51: UML Sequence Diagram – EM: Interactions and Processes of the Web Portal sub-component

3.7.3 Web Crawler

The Web Crawler subcomponent will be triggered internally in the Event Manager either by direct request through the Web Portal, or according to certain configurations to crawl for events in defined intervals. When triggered, the crawler will check the list of URLs (called seeds) to crawl and return its raw contents, passing them to the Event Miner until the subset of remaining unvisited URLs (called frontier) are all crawled. The Web Crawler interacts with two sub-components of the Event Manager, the Web Portal and the Event Miner as well as with the Web to fetch contents. The sequence diagram (see Figure 52) below visualizes these processes.

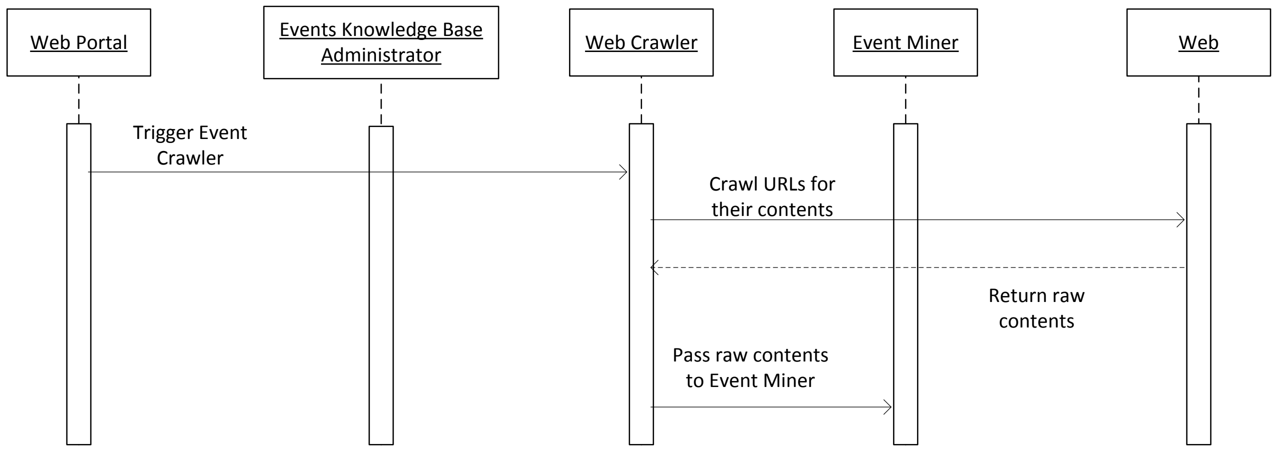
Web Portal
Events Knowledge Base Administrator
Web Crawler
Event Miner
Web
Trigger Event Crawler
Crawl URLs for their contents
Return raw contents
Pass raw contents to Event Miner

Figure 52: UML Sequence Diagram – EM: Interactions and Processes of the Web Crawler sub-component

3.7.4 Event Miner

The Event Miner sub-component is activated when it receives information from the Web Crawler of the raw contents of the various URLs. It will start processing the contents and apply techniques such as information extraction to recognize locations, dates and similar information about events, as well as text classification to recognize types of events. Furthermore it will filter classified and extracted contents to possible user inputs and will perform summarization of events. It will return extracted summarized events to the Web Portal, from where the user will be able to decide which events should be persisted in the Events Knowledge Base and which should not. The sequence of these actions is visualized in Figure 53.

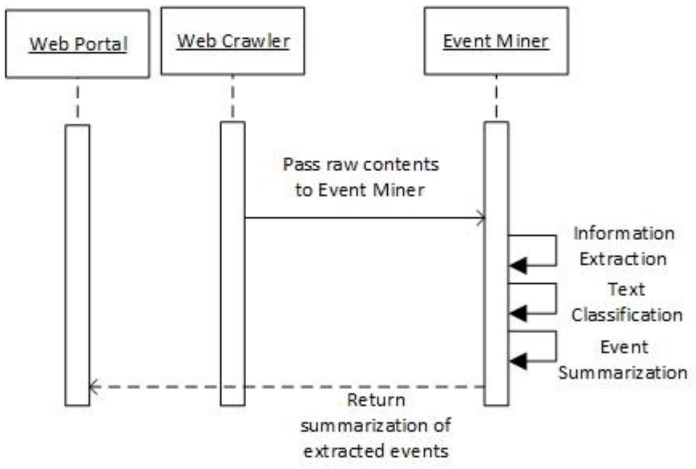

Figure 53: UML Sequence Diagram – EM: Interactions and Processes of the Event Miner sub-component

3.7.5 Events Knowledge Base Administrator

The Events Knowledge Base Administrator subcomponent is in charge of creating, reading, updating and deleting event descriptions from the Events Knowledge Base. This means that it acts as an administrator and mediator to the description of events that are persisted in KIS through its API. It receives requests from the Web Portal subcomponent such as browsing for events in the knowledge base, and in general to perform CRUD operations on it. When creating events, the user enters the description of an event or the user selects a resulted event from the mentioned semi-automatic event mining process to be persisted in the Events Knowledge Base (see Figure 54).

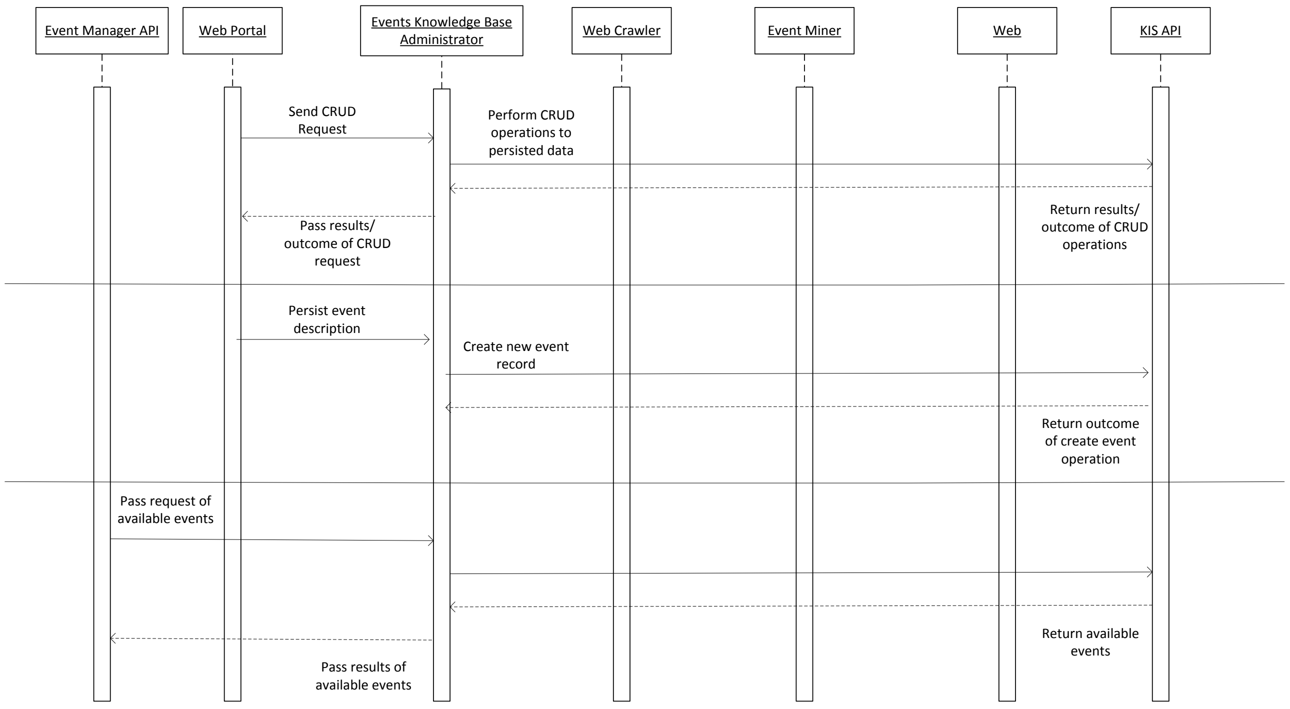
Event Manager API
Web Portal
Events Knowledge Base Administrator
Web Crawler
Event Miner
Web
KIS API
Send CRUD Request
Perform CRUD operations to persisted data
Return results/ outcome of CRUD operations
Pass results/ outcome of CRUD request
Persist event description
Create new event record
Return outcome of create event operation
Pass request of available events
Return available events
Pass results of available events

Figure 54: UML Sequence Diagram – EM: Interactions and Processes of the Events Knowledge Base Administrator sub-component

3.8 Game Manager

There are several interdependencies between devices, applications and components which have an association to the Game Manager. See Figure 55 for the numbers as referred to in the following list.

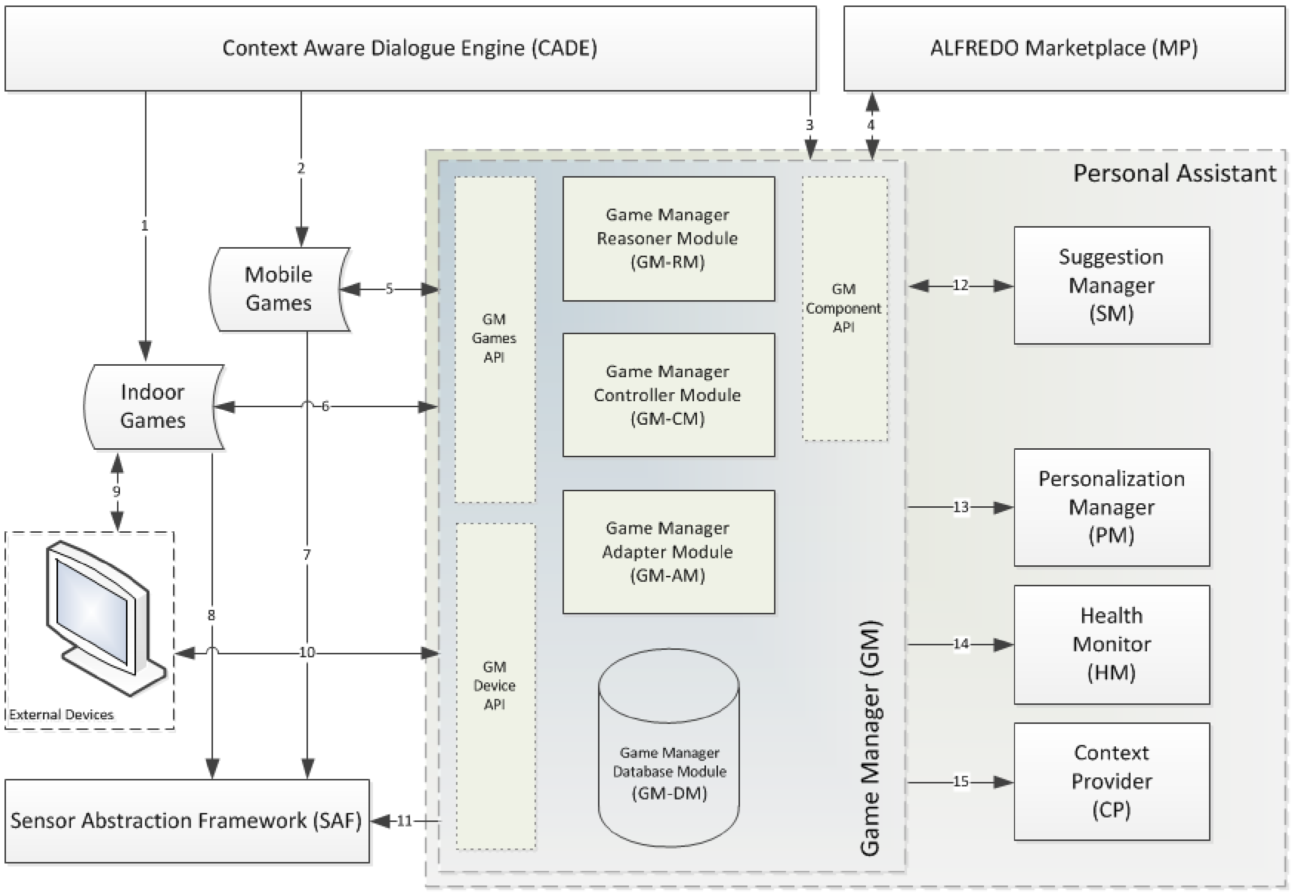
Context Aware Dialogue Engine (CADE)
ALFREDO Marketplace (MP)
Personal Assistant
Mobile Games
Indoor Games
External Devices
Sensor Abstraction Framework (SAF)
GM Games API
GM Device API
Game Manager Reasoner Module (GM-RM)
Game Manager Controller Module (GM-CM)
Game Manager Adapter Module (GM-AM)
Game Manager Database Module (GM-DM)
GM Component API
Game Manager (GM)
Suggestion Manager (SM)
Personalization Manager (PM)
Health Monitor (HM)
Context Provider (CP)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Figure 55: Game Manager and interfacing components

- **CADE to Indoor Games**: Some indoor games may be fully voice controllable, meaning that the user issues voice commands in order to influence the game mechanic (such as “up”, “down”, “left”, and “right”). As a constant pulling of such commands would be not very performant, the games need to register themselves to the CADE and then wait for a command to be issued by the user, in which case the CADE broadcasts the user command to all recipients.
- **CADE to Mobile Games**: See (1).
- **CADE to GM**: Some voice commands issued by the user may be relevant to the Game Manager, more specifically the Controller Module. Examples of such commands are “Start Game”, “Pause Game”, “Stop Game”, and so forth. For this reason, the GM registers itself to the CADE in order to receive all broadcasted voice commands.
- **GM to MP & vice versa**: When the user selects a game from the ALFREDO Marketplace for installation on her device, the MP uses the GM Component API to register the game to the Game Manager. It does this by forwarding the metadata description file associated to the game’s installation files to the GM’s Adapter Module, which then stores this data to the GM Database Module. This data is later used by the GM Reasoner Module to select an appropriate game for a specific situation. Over time, whenever the user plays that specific game, the GM Controller enriches the game’s metadata via the Game Adapter with usage information, such as for how long the user has played the game and how she has performed. This information is used to enhance the game suggestion and configuration process and

parts of it may also be sent back to the MP, if the user allows this. To this end, the GM Adapter frequently calls the Game Manager API Client of the ALFREDO MP and uploads parts of the game's usage history (for example once a week).

- **GM to Mobile Games & vice versa**: The GM Adapter Module uses various types of information (see above) to configure games that are about to be started. To this end, the games have to provide for such configuration options, e.g. different font sizes. Vice versa, all ALFRED games should report a player's performance and usage history back to the Game Manager once a game session has ended, using the GM Games API. The GM Controller receives the information and forwards it to the GM Adapter, which in turn stores it in the corresponding game profile.
- **GM to Indoor Games & vice versa**: See (5).
- **Mobile Games to SAF**: Some types of games make use of a player's vital data to adapt their game mechanic. For instance, when the game detects that the player's heart rate rises above a certain limit, it may reduce the game difficulty to allow the player to relax. To this end, such games may pull sensor data provided by the SAF. They should, however, be able to handle cases in which the SAF cannot provide for the data in question.
- **Indoor Games to SAF**: See (7).
- **Indoor Games to External Devices & vice versa**: Indoor Games require external devices to function as intended, such as bicycle ergometers or TV screens. Usually, the game clients and the devices will communicate with one another by way of the Game Manager (more specifically: they will rely on the GM Controller to exchange message between the GM Games API and the GM Devices API), but certain settings may also support a direct communication of an external device and the game client.
- **GM to External Devices & vice versa**: See (9).
- **GM to SAF**: More sophisticated implementations of the GM Reasoner Module may pull information from the SAF to take into account a user's vital data when making situation estimations.
- **GM to SM & vice versa**: If the GM Reasoner Module detects a situation which it considers appropriate for a game suggestion, it forwards an according notification to the SM. If the user agrees to play the suggested game, the SM notifies the GM's Controller Module via the GM Component API. The GM Controller then waits for the GM Adapter to configure the game and launches it afterwards.
- **GM to PM**: Both the GM Reasoner and the GM Adapter require user preference information for decision making and will occasionally fetch this data from the PM using the PM's API wrapper.
- **GM to HM**: Similar to (13), both the GM Reasoner and the GM Adapter also require information about the user's health parameters to be able to determine situations and game settings that suit the user's specific capabilities. To this end, they both will occasionally pull information from the HM using its API wrapper.

GM to CF: Similar to (13) and (14), both the GM Reasoner and the GM Adapter require elemental context information such as the user's current location. They will frequently query the CF API for this data.

3.9 Knowledge and Information Storage

3.9.1 Overview

The KIS provides the service to store, retrieve and manipulate knowledge and information. While acting as a single point of data storage it allows the benefits of different database solutions. For the whole ALFREDO ecosystem the functionalities to store and manipulate data as well as to allow the user to share the stored data is important. These two functionalities will be discussed in the first section of this chapter (see section 3.9.2). But for the ALFRED system other functionalities are important as well. This is the administration of the KIS, including the configuration of different databases as well as to manage the clients, who have access to the services of the KIS. These two functionalities will be discusses in the second section of this chapter (see section 3.9.3).

All provided functionalities of the KIS have the same base sequence. Figure 56 depicts this general sequence of actions. When a functionality of an API Wrapper is used, it will encapsulate the message and forward it to the Cloud Storage – Facade (see section 2.9.4 for all involved components). The Facade will temporally store the information of the sender and the message. The sender of the message will then be authenticated within the management segment of the KIS. If the sender is authenticated the message will be further executed. For this, the message will be analysed and interpreted. If the sender wants to execute an operation of a bucket he is not the owner of, he can act as a surrogate. For this the authorization for the specific operation for the specific bucket is checked. If the client is authenticated and authorized the operation encapsulated in the message will be executed and the results will be returned to the Facade. Since the Facade has temporally stored the information of the sender and message it can delegate the response back to the API Wrapper over a specific communication channel.

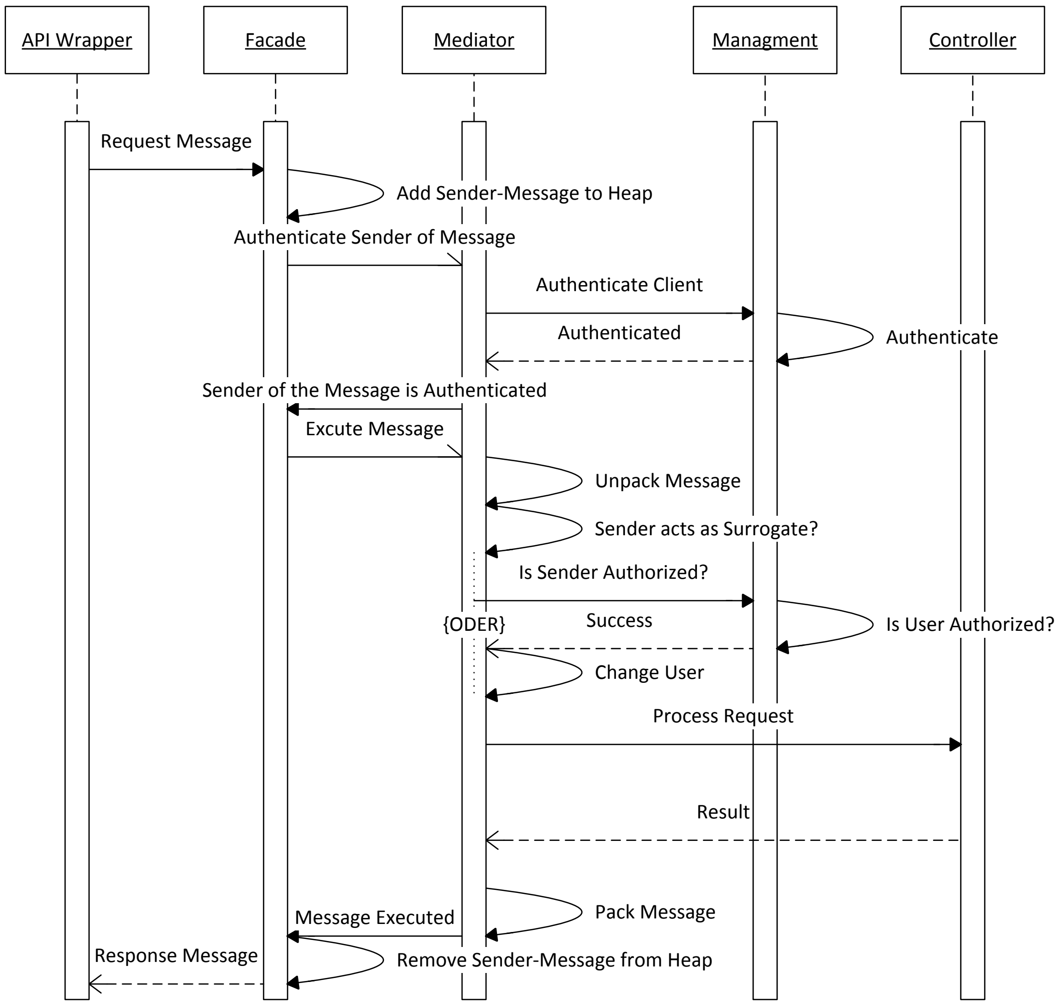

Figure 56: UML Sequence Diagram – KIS: General Message Handling

3.9.2 Public Services

The two services described in this section will be accessible by the whole ALFREDO ecosystem. This includes the storage of data as well as the service to define the access rights to the data.

3.9.2.1 Storage

The services provided encapsulated under Storage includes all services necessary for the storage and manipulation of data. The following subsections will each describe one provided service.

3.9.2.1.1 Create Bucket

In order to create a new bucket for a user, the request for the operation will be forwarded to the Storage Controller. If a bucket for the user does not already exist a new bucket will be created by the Bucket Manager. Figure 57 depicts the sequence of the involved subcomponents of the KIS for this service.

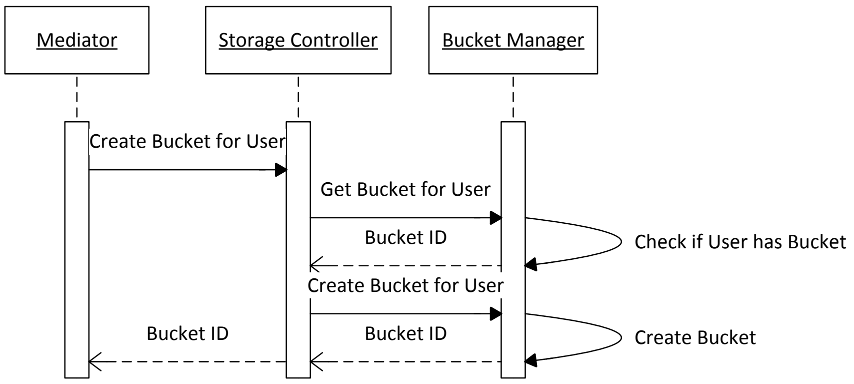

Figure 57: UML Sequence Diagram for KIS Service – Create Bucket

3.9.2.1.2 Delete Bucket

In order to create a new bucket for a user, the request for the operation will be forwarded to the Storage Controller. If a bucket for the user exists, the bucket will be deleted from the Internal Database. Afterwards the data references to this bucket will also be deleted from all External Databases, with the help of the Wrapper Manager. Figure 58 depicts the sequence of the involved subcomponents of the KIS for this service.

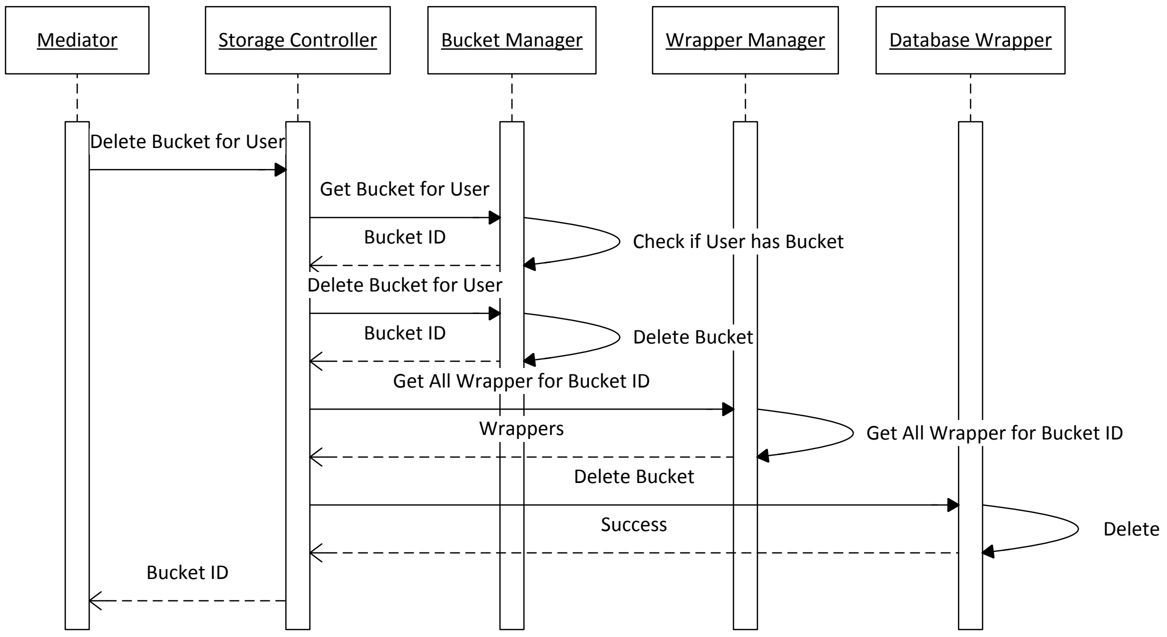

Figure 58: UML Sequence Diagram for KIS Service – Delete Bucket

3.9.2.1.3 CRUD operations for Data Object

All four CRUD operations (Create, Read, Update, and Delete) follow the same principle. After the Mediator receives the operation request, the bucket and the wrapper for the operation are identified. Finally, the operation is executed with the help of the fitting Data Type Wrapper for the Data Base Wrapper (see Figure 59).

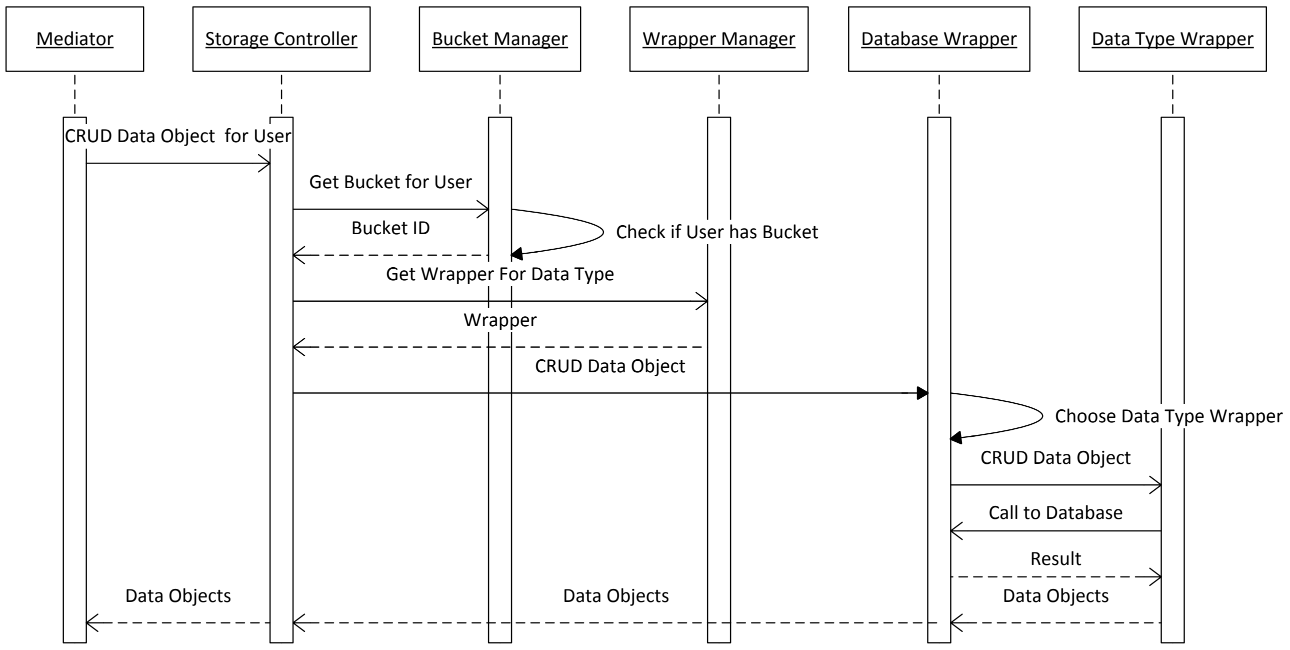

Figure 59: UML Sequence Diagram for KIS Services – CRUD Operations for Data Objects

3.9.2.2 Authorization

The owner of a bucket can manage the access to his bucket with authorization functionalities. Each bucket has its own ACL, defining the access rights of the bucket. The sequence for managing the ACL is depicted in Figure 60. After the mediator receives the request to modify the access rights for a specific bucket for a specific client, the Bucket Manager will check if the user owns the specific bucket. If so, functionalities provided by the Authorization Manager will be used in order to modify the ACL of the bucket accordingly.

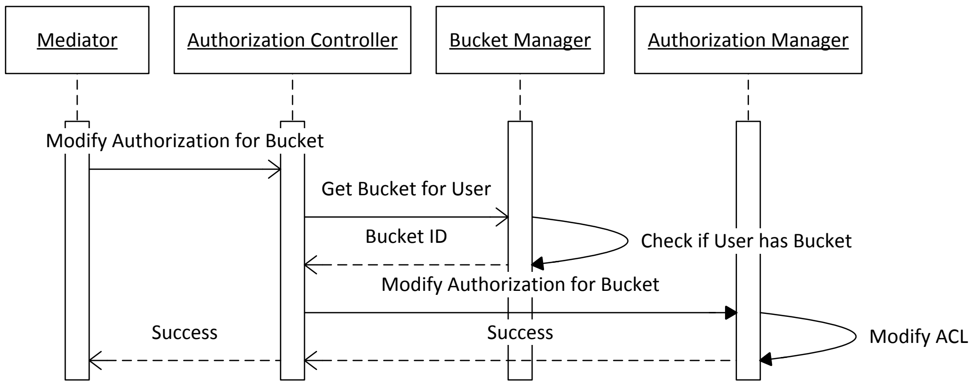

Figure 60: UML Sequence Diagram for KIS Service – Authorization Management

3.9.3 Private Services

The two services described in this section will be accessible by the administration of the ALFRED system only.

3.9.3.1 Administration

The Administration services can be used by the administration of the ALFRED system to configure the External Databases used in the KIS. For this, three different services will be provided. They allow adding, changing and removing the configuration of an External Database. Each of these services follows the same sequence depicted in Figure 61. After the Mediator receives the operation request for the External Database administration the Wrapper Manager will be used to execute the operation. Configuration alteration will be also made in the running Database Wrapper. In case of the deletion of External Databases additional steps will be taken to delete also the references to buckets for the database in question.

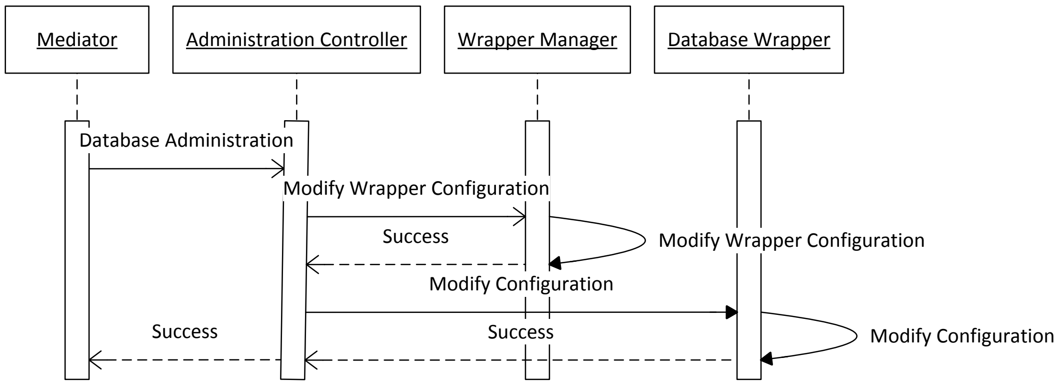

Figure 61: UML Sequence Diagram for KIS Service – Database Management

3.9.3.2 Authentication

The Authentication services can be used by the administration of the ALFRED system to manage clients allowed to access the KIS. For this, three different services will be provided. They allow adding, changing and removing of a client of the KIS. Each of these services follows the same sequence depicted in Figure 62. After the Mediator receives the operation request for the user administration, functionalities of the Authentication Manager will be trigger by the Authentication Controller. This will modify the credentials for a specific client.

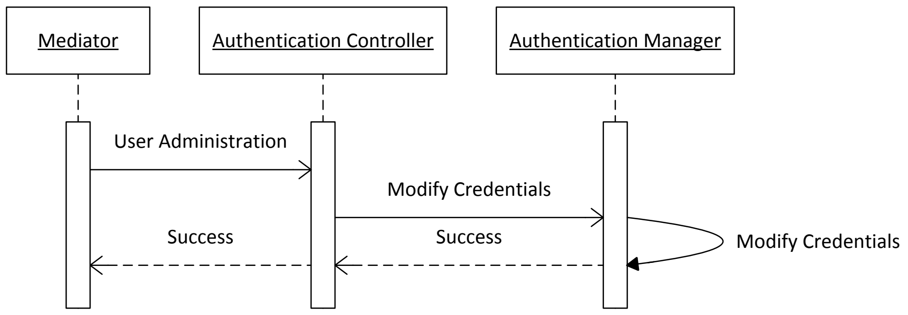

Figure 62: UML Sequence Diagram for KIS Service – Authentication Management

4 Summary and Next Steps

This deliverable introduced first the architecture of the whole ALFRED system and second all functionalities provided by the components of the ALFRED system.

In the first part of the document the global architecture of the ALFRED system was introduced. The seven main components as well as the storage solution were explained in detail. This was done by defining all subcomponents of each component and highlighting the main responsibilities of each subcomponent.

The Personal Assistant is the heart of the ALFRED system as is it will be the main interaction point for the end user. Since the focus of the ALFRED project is on the end user, special consideration was taken to ensure a high level of adaptability to generic use cases. This was done by separating all user related functionalities to ALFRED apps. This is also true for the components of ALFRED which will be represented on the mobile device of the end user with the client side of their distributed subsystem. By this “Eating Your Own Dog Food” approach[5] redundant functionalities will be limited and best practices enforced. This will ensure a high quality of the Personal Assistant already in prototypical stages.

The before mentioned distributed subsystem consists of the following components: HM, which will be responsible to gather, process and provide all health related information with a focus on data collected by wearable sensors; Context-aware Speech Recognition, which will be the interaction point for default user interaction with the end user; the ALFREDO Marketplace, which will manage the entire deployment cycle for ALFRED apps, both from end user and app provider perspective.

The other components of the ALFRED system will not be represented with a ALFRED app on the mobile device of the end user. Instead they provide their functionality solely as a web service: The Personalisation Manager will be the central point for all user profile data with reasoning capabilities to create real user information; the Game Manager will take control of the wellbeing of the end user by adapting and monitoring serious games; the Event Manager represents the central point to gather information on social events from different sources; the Knowledge and Information Storage will provide an abstracted storage location to provide all components with a fitting data schema.

This deliverable will be the guidance for the integration, over the course of the entire ALFRED project as it clearly identifies all functionalities and components needed for the ALFRED system. This allows parallel work during the implementation phases of the project while ensuring a seamless integration at a later stage. In addition, this deliverable will be the foundation for the technical specification. Based on the results of this deliverable the work on the technical specification will be done.

[5] Harrison, W., "Eating Your Own Dog Food", Software, IEEE , vol.23, no.3, pp.5,7, May-June 2006