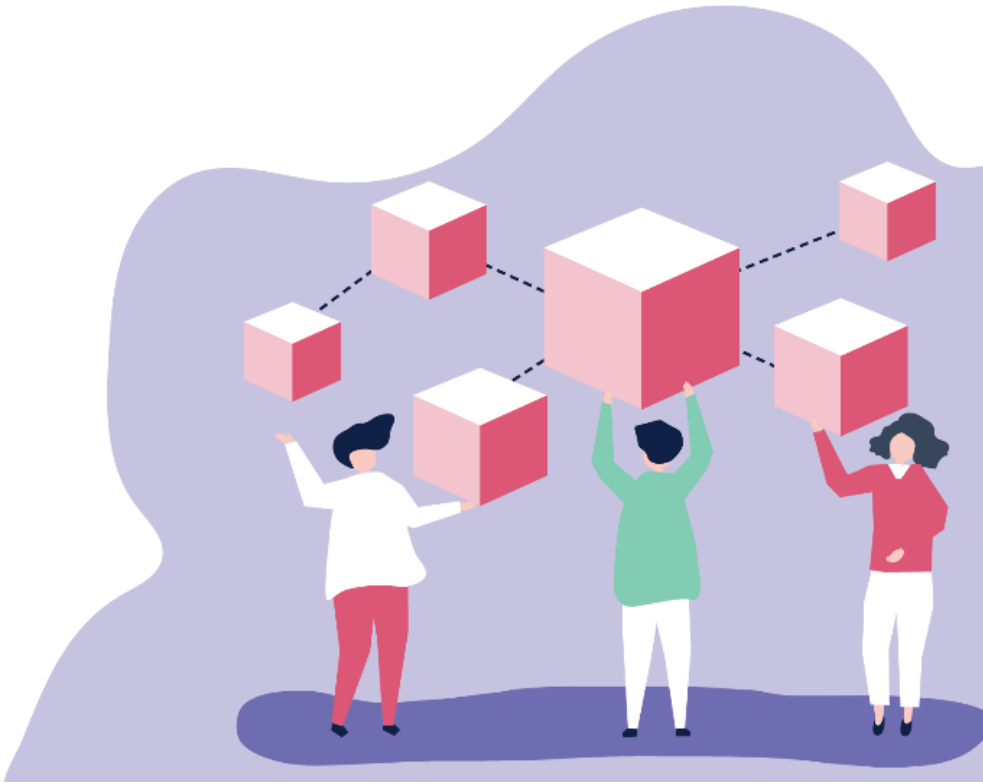
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 857223

D3.2 Overall GATEKEEPER architecture

Deliverable No.	D3.2	**Due Date**	31/07/2020
Description	Report with the overall architecture of the GATEKEEPER ecosystem and detailed description of the main components		
Type	Report	**Dissemination Level**	PU
Work Package No.	WP3	**Work Package Title**	GATEKEEPER Web of Things (WOT) Reference Architecture
Version	1	**Status**	Final



Authors

Name and surname	Partner name	e-mail
Paolo Zampognaro	ENG	paolo.zampognaro@eng.it
Valentina Di Giacomo	ENG	valentina.digiacomo@eng.it
Federica Saccà	ENG	federica.sacca@eng.it

History

Date	Version	Change
07/02/2020	0.1	Table of content and initial content
14/04/2020	0.2	Revision of ToC and contributions
13/05/2020	0.3	Integration of content up to the date
12/06/2020	0.4	Integration of 1st round of contributions
03/07/2020	0.5	Integration of 2nd set of contributions
16/07/2020	0.6	Revision of content
24/07/2020	0.7	Addressed BB internal review and other minor fixes
27/07/2020	0.8	Addressed HPE internal review and other minor fixes
29/07/2020	0.9	Version for Quality check
31/07/2020	1.0	Final version

Key data

Keywords	Reference architecture, GATEKEEPER Platform
Lead Editor	ENG
Internal Reviewer(s)	BB, HPE

Abstract

This deliverable contains a description of the GATEKEEPER Platform Architecture, the components and services that build the platform, their role and their interconnections.

It describes the underlying architectural principles leading the design and gives a detailed description of the components that are the building blocks of the GATEKEEPER platform. It also lists a first set of interactions that describe the behaviour of the platform and how its components interact. The Platform Architecture is still evolving, thus a second version of this document will be released in March 2021, describing the final version of the architecture.

Statement of originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Table of contents

List of tables

List of figures

Introduction

Starting from the technical requirements collected in T3.1 [1], informed by user requirements collected in T2.3 [3], together with the plans of the pilots to use the platform identified in T6.2 [2], and the components descriptions of WP4 and WP5, this deliverable aims to define the overall GATEKEEPER reference architecture, describing the components and services that build the platform, their role and their interconnections.

According to the phased approach adopted in the project and described in the DoA (section 1.3.2, GATEKEEPER journey), the GATEKEEPER platform architecture definition will follow an incremental approach. In this first version, the main goal is to detail the underlying principles the platform has to follow, identify and describe the base functionalities of the components building the platform, and ensure that they, together, satisfy the requirements defined so far.

This version of the architecture will inform and guide the release of the platform components and, as the project results are consolidated and initial feedback is received from the pilots, the platform architecture will be updated to reflecting the status of the developments and satisfy the emerging requirements. A second and final version of this deliverable will be released at M18 of the project, in March 2021.

The results of the activities described in this deliverable have been organized following the structure below:

Section I – Is focused on the driver principles that define the general approach in designing the GATEKEEPER system architecture and its leading architectural specification, the Web of Things approach.

Section II - Aims at defining the logical hierarchical structure of the infrastructure components, the role of each of these system components, the relationships between them and how they interact with one another, where the key components reside, where data are stored and how can they be retrieved. In order to achieve such objective, logical and deployment architectures are described, which offer (i) a clear understanding of the main data flow among components (logical architecture) and (ii) a clear representation of the software components and their relationships (deployment architecture). Such architectures will offer the general framework to supervise the design and implementation done in WP4 and WP5 to ensure the compliance of components and to smooth the component integration.

Section III - Explains the information model which captures the unified data model for all the information generated and processed within the GATEKEEPER system. The identified model will serve as the basis for realizing the requirements and goals for the rest of the architecture and design of the various components and will take care to guarantee the syntactic interoperability. An initial model for the health-related measures that will be managed by the platform (whose definition and formalization is the goal of the joint work of T3.4 and T3.5) is also given. Insights about the main adopted standards are provided as well in this part of the document.

Section IV - In this part the main functional components, implementing the GATEKEEPER services are identified and their functionalities described in terms of offered and required programmatic interfaces (i.e. APIs). Such functionalities are identified gathering information from the component providers and identifying sequence diagrams of interactions with other components associated to the project requirements (see next section).

Section V - In this section, finally, are analysed the sequence diagrams of platform behaviours, gathered from the "user stories" extracted from D6.2 and mapped to the specific functionalities of the GATEKEEPER platform, or describing core functionalities of the platform (user management, data management), with the purpose to identify which GATEKEEPER components are involved and the type of interactions among them. This work has been carried out in close collaboration with pilot owners and component providers, and in conjunction with T3.1, mapping functionalities with a first set of platform requirements.

Appendix A - Provides a table summarizing the mapping between the project's User Stories to the platform interactions of Section V. User Stories related to GK Pilot UCs have been identified by analysing D6.2 deliverable [2].

Appendix B - Provides a table mapping the platform requirements from D3.1 [1] to the platform interactions of Section V. As D3.1 is due later in the project timeline, for this work we used a draft of the document.

Appendix C - Provides an overview table of all components interactions and the components involved in each of them.

Appendix D - Shows a UML design document produced by analysing DoA description which laid the basis for the work of definition of the platform architecture and was refined along with the partners during T3.2 work.

Finally, **Appendix E -** Reports the description of GATEKEEPER spaces referenced in this deliverable as defined in the DOA.

1 Architecture definition principles

This document describes the GATEKEEPER reference architecture. The main goal is to help the platform component owners to collaborate effectively, having a coherent view of the platform as a whole, a detailed description of the main components needed for the implementation of the GATEKEEPER platform and the interaction expected between these components to satisfy the requirements coming from Technical Requirements (T3.1), as well as Pilots (WP6), and other user requirements (WP2). In this first section we describe the defining principles that drive the design of the GATEKEEPER Platform, we introduce the stakeholders of the platform, and then we give an overview of the target cloud infrastructure, as well as security and privacy concerns.

1.1 Web of Things

GATEKEEPER platform will be based on the Web of Thing (WoT) layered architecture described in [4]. The main difference between the GATEKEEPER layered architecture and the WoT layered architecture relies on the inclusion of an additional layer that is devoted to the implementation of governance rules for the platform that are applied to a "GATEKEEPER Thing" through the release of a certification (Figure 1).

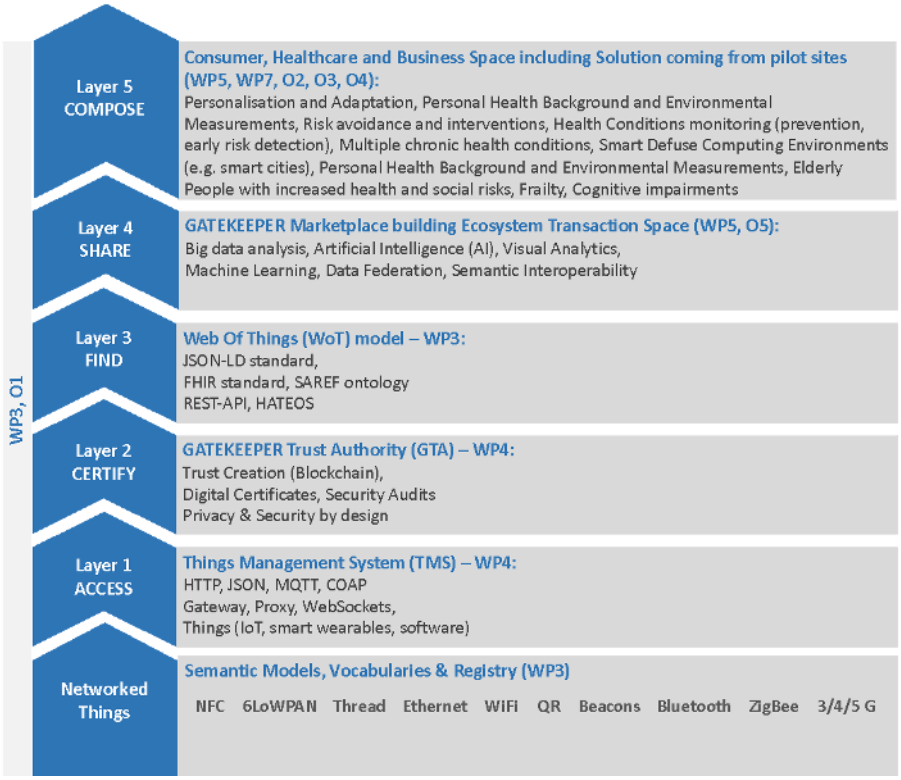


Figure 1 - GATEKEEPER layered architecture

The different sets of policies are associated with a different kind of digital certificate, and when a Thing obtains a specific certificate from the GATEKEEPER Trust Authority (GTA), it means that this Thing is compliant with the policies associated with the certificate.

Within GTA the policies will be in line to a common set of features that are related to:

- Data access compliance with current regulation (e. g. GDPR compliance);
- Alignment of data to the GATEKEEPER semantic models;

- Compliance with standards (mainly Web of Things and FHIR [14]);
- Quality of provided data and/or services.

1.1.1 Principles for GATEKEEPER data

The main objective behind the data governance inside the GATEKEEPER platform is the enhancement of data economy, in order to provide solutions for data interoperability and re-use in machine learning (ML) and artificial intelligence (AI) algorithms. This is achieved by ensuring data quality, protection, privacy and security.

In order to reach this objective, several principles will be followed for GATEKEEPER data:

1. Compliance with Findable, Accessible, Interoperable Reusable (FAIR) principles.
2. Open data as possible, and closed as necessary. GATEKEEPER will always provide access to data whenever possible;
3. Clear separation between data owner and data provider. Within GATEKEEPER, data will be treated as Things (digital twin of the data). So, interfaces in order to access data should be defined as APIs. This means that the data provider should agree with the data owner (e. g. physical database owner) on how and which subset of the data should be made publicly available and/or which kind of restricted access should be implemented.
4. Balancing the flow and wide use of data, while preserving high privacy, security, safety and ethical standards. These features should be provided by data providers and GATEKEEPER will be able to certify its accomplishment through the Certification Authority (GTA).
5. Allow the free flow of non-personal data, GATEKEEPER will treat in a high permissive way non-personal and non- sensitive data. Less or no certification will be needed in order to include these datasets as Things within the platform.
6. Provide rules for access to and use of data should be fair, practical and clear, with clear and trustworthy data governance mechanisms in place; for an open, but assertive approach to international data flows, data should flow within the EU and across sectors.
7. Allow infrastructures that should support the creation of data pools enabling Big Data analytics and machine learning, in a manner compliant with data protection legislation and competition law, allowing the emergence of data-driven ecosystems.
8. Create an Artificial Intelligence ecosystems based on the concept of GATEKEEPER data space that will contribute to the HealthCare Data Space foreseen at the European level, with the objective of providing services (WP5) for early prevention and intervention in 7 Medical Reference Use Cases (RUCs defined in WP6) in order to improve the accessibility, effectiveness and sustainability of the healthcare systems.

1.1.2 GATEKEEPER Web of Thing based architecture

1.1.2.1 Description of the layered structure

The proposed structure of GATEKEEPER Web of Things Platform architecture [5] will be framed into a layered structure composed of: Access, Certify, Find, Share, Compose layers as already shown in Figure 1.

Layer 1: Access, provide Accessibility of FAIR principle: This layer is responsible for turning any Thing into a Web Thing that can be interacted with using HTTP requests just like any other resource on the Web. A Web Thing is a REST API [15] that allows the interaction with software services, or something in the physical realm, like opening a door or reading a sensor located somewhere in the world. In GATEKEEPER this layer is provided by the Things Management System. The Things Management System (TMS) is one of the core components dedicated to the implementation of the functionality of access and find associated with the access and find layer of WoT architecture. The TMS is like a broker service that publishes the GATEKEEPER Things. Each Thing is decorated with a Web of Thing Description that is available through a web-based service. Within the TMS, the level of trustiness between the Things that are already connected to the platform is automatically managed. The interaction between different Things using Thing Descriptions is defined through a **Web of Things (WoT) interaction model**. The Thing Description enables: (i) management of multiple Things by a cloud service, (ii) simulation of devices/Things that have not yet been developed, (iii) common applications across devices from different manufacturers that share a common Thing model, and (iv) combining multiple models into a Thing. In the next sections, the Web of Things model will be presented to show the interaction model and architecture of the Web of Things platform.

Layer 2: Certify, improve the FAIR principles with Trustability: This layer is specific to the GATEKEEPER platform, with respect to the Web of Thing layered reference architecture. It is dedicated to build the concept of trustiness in the GATEKEEPER platform through certification and a way to securely share data across services. A GATEKEEPER Thing is different against a standard Thing because it has been certified by the GATEKEEPER Trust Authority (GTA). Within the GATEKEEPER architecture the certify layer is enabled through the interaction between the Things Management System (TMS) and the GATEKEEPER Trust Authority (GTA). GATEKEEPER Trust Authority will provide the Certify layer of the GATEKEEPER architecture, while the GATEKEEPER MarketService will be in charge of sharing the GATEKEEPER Things. The Trust creation will be managed using Blockchain technology [16] with the aim of having a decentralized trust system. As a decentralized system, it removes the requirement for a trusted third party by allowing participants to verify data correctness and ensure its immutability. Things can use Blockchains to register themselves and organize, store, and share streams of data effectively and reliably.

Layer 3: Find, provide Findability of FAIR principle: Giving accessibility via HTTP to Things is a good option but it does not mean applications data or services can be easily offered and/or consumed. This layer is dedicated at providing ways for easy discovery and consuming of Things. In GATEKEEPER it will be implemented through a Marketplace that will provide Things offered through the *consumer space*, the *healthcare space* and the *business space*. Each space is oriented to a different type of market user. These core features will be supported by the Networked Things architecture that will provide the reference model in home and health-oriented devices forming the GATEKEEPER Platform's Business Space. The ecosystem will be split into clear boundaries around 3 spaces, Business-to-Government (B2G), Business-to-Consumer (B2C) and Business-to-Business (B2B).

Layer 4: Share, provide Interoperability of FAIR principle: This layer will provide functionalities by which someone can really "understand" what a Thing is, what data or services it offers, and so on. Through these functionalities a Thing can not only be easily used by other HTTP clients but can also be findable and automatically usable by other WoT applications [6][7]. The approach here is to reuse Web semantic web standards to describe Things and their services. This enables searching for Things through search engines and other Web indexes as well as the automatic generation of user interfaces or tools to interact with Things. At this level, technologies such as JSON-LD (a language for semantically annotating JSON) are in use. In GATEKEEPER, all the Things will use as communication language the Web of Things standard with JSON-LD contexts, including FHIR standard and SAREF ontology [17].

Layer 5: Compose, provide Reusability of FAIR principle: This layer provides the integration of data and services from heterogeneous Things into an immense ecosystem of tools such as analytics software, mash-up platforms and developer platforms. Within GATEKEEPER the compose layer will provide all the intelligent services for early detection and intervention and a developer platform where developers can compose GATEKEEPER Things in order to provide advanced services.

All the data pushed from the Things that compounds the ecosystem to the platform will be used by the integrated Intervention Services of the GATEKEEPER Platform, which will aim to create diagnostic and prognostic algorithms, to help not only clinicians and domain experts to support their decisions but also predictive and proactive services to help elderly people at home and in their communities.

In order to build these services, techniques such as big data analysis or artificial intelligence will be of particular importance given the wide range of possibilities they provide. For instance, retrieving multiple datasets from multiple wearable devices could be used to accurately predict possible life threatening diseases such as stroke or heart attack, thus helping to provide efficient fast assistance.

These early detection, prediction and proactive services for healthcare will be validated in the pilot sites in order to populate the Consumer and Healthcare spaces within the GATEKEEPER Marketplace where these services will be available as Things to third party users in order to compose more advanced services through the open calls.

1.1.2.2 Web of Things (WoT) interaction model

Special mention must be given to the Web of Things interaction model which is intimately related to the access layer and the Things Management System (TMS). The Object model is used by the TMS and GTA, and it is composed of three layers: Binding Templates, protocol bindings, and protocol stacks. This model would be an architecture for the interconnection of the different layers of the Web of Things, integrating those Things to the Web and in particular to HTTP, WebSocket, JSON and JSON-LD, using TLS, DTLS, and/or OAuth to authenticate requests. Four main areas are considered inside the Web of Things interaction model: Protocols, Resources and Data Model and Semantic Extensions. As seen in Figure 2, the TMS model follows a structured and layered architecture where from the communication protocol, we move onto the TD, then to the contextualized TD and the semantic Web distribution.

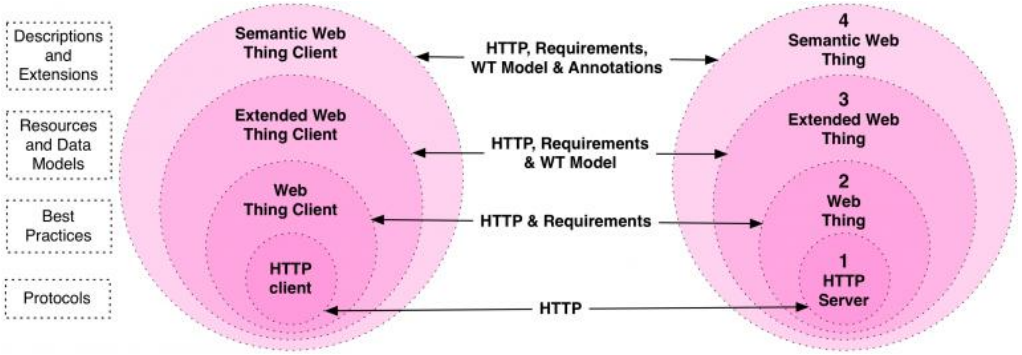


Figure 2 - Web of Things model [8]

Binding Templates are a reusable collection of templates used in communication with other platforms. These templates are mapped together with the Protocol Bindings to be used by the Protocol Stack as a guideline for implementation of the Web Services in HTTP, WebSocket, and CoAP, with JSON and JSON-LD as data-interchange format (Figure 3).

In a large-scale way such as intended with the GATEKEEPER Platform, Things pushing data to the Web can only happen if the data can be efficiently—and securely—shared across services. This layer specifies how devices and software services and their resources must be secured so that they can only be accessed by authorized users and applications. For that purpose, Things are internally configured in a way that it is divided into different layers with the implementation, definition and communication, through binding templates.

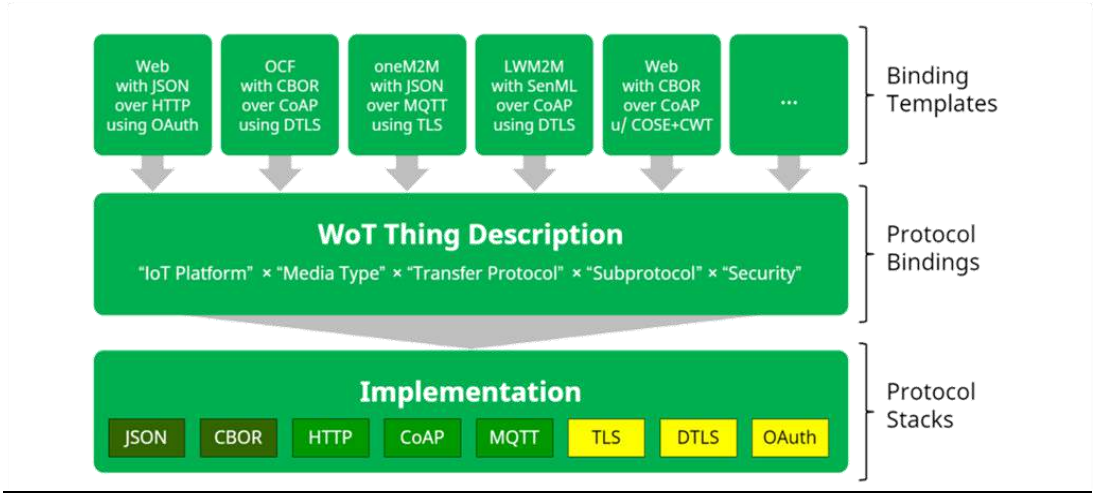


Figure 3 - From Binding Templates to Protocol Bindings [9]

1.1.3 Role of WoT Thing Description

The Thing Description (TD) is one of the key aspects of the WoT architecture and data models. It allows Things to be defined, communicate with each other and expose information. In essence, the Web of Thing Description is the entry point of a Thing, and the Thing Description of the TMS is the point of access to the GATEKEEPER ecosystem. It can be understood as the nucleus of the Thing since it provides the functionality of the interconnectivity to the Thing. A Thing Description consists of four components: (i) *textual metadata about the Thing itself*; (ii) a set of *Interaction Affordances* that indicate how the

Thing can be used; (iii) *schemas for the data exchanged* with the Thing for machine-understandability, and, (iv) *Web links to express any formal or informal relation to other Things or documents* on the Web.

An example of a WoT TD is shown in Example 1. Note that in general, the TD provides metadata for different Protocol Bindings identified by URI schemes and security mechanisms (for authentication, authorization, confidentiality, etc.)

Example 1: Temperature Event with subscription and cancellation. Extracted from [11]

```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "id": "urn:dev:ops:32473-Thing-1234",
    "title": "WebhookThing",
    "description": "Webhook-based Event with subscription and unsubscribe
form.",
    "securityDefinitions": {"nosec_sc": {"scheme": "nosec"}},
    "security": ["nosec_sc"],
    "events": {
        "temperature": {
            "description": "Provides periodic temperature value updates.",
            "subscription": {
                "type": "object",
                "properties": {
                    "callbackURL": {
                        "type": "string",
                        "format": "uri",
                        "description": "unsubscriber for Webhook.",
                        "writeOnly": true
                    },
                    "subscriptionID": {
                        "type": "string",
                        "description": "Unique subscription ID for
cancellation",
                        "readOnly": true
                    }
                }
            },
            "data": {
                "type": "number",
                "description": "Latest temperature value"
            },
            "cancellation": {
                "type": "object",
                "properties": {
                    "subscriptionID": {
                        "type": "integer",
                        "description": "Required to cancel subscription.",
                        "writeOnly": true
                    }
                }
            },
            "uriVariables": {
                "subscriptionID": { "type": "string" }
            },
            "forms": [
                {
                    "op": "subscribeevent",
```

```
                    "href":
"http://192.168.0.124:8080/events/temp/subscribe",
                    "contentType": "application/json",
                    "htv:methodName": "POST"
                },
                {
                    "op": "unsubscribeevent",
                    "href":
"http://192.168.0.124:8080/events/temp/{subID}",
                    "htv:methodName": "DELETE"
                }
            ]
        }
    }
}
```

In Example 1, a Thing Description is shown to represent a Webhook event. The context definition in this case has included HTTP protocol bindings supplements. The TD doesn't have security as defined in "securityDefinitions" and "security" fields. The TD provides an Event affordance called "temperature" that updates its latest temperature value to the consumer. It sends a POST request to a callback URI that is provided by the consumer. The "subscription" defines two properties, one is a write-only parameter called "callbackURL" that must be submitted through the *subscribeevent*. The other property, "*subscriptionID*" is read-only and returned by the subscription. In case of subscription the Thing would send periodically its state through a POST to the callback URI using "data" form defined structure. To unsubscribe, the "*unsubscribeevent*" form must be submitted, this form makes use of a URI Template to specify the subscription to cancel. The *uriVariables* member functions as a note to the consumer to include its contents. Alternatively, the member "cancellation" can be used to unsubscribe in a similar way to "subscription" and combine it with a *subscribeevent* form.

For the Thing Description the use of JSON-LD is crucial as it is a lightweight Linked Data format for linking data with vocabularies that describe the semantic of the data. Another important aspect of the JSON-LD data format is its human readability. It is based on the already existing JSON format and provides a way to help JSON data interoperate at Web-scale through the concept of context. JSON-LD is an ideal data format for programming environments, REST Web services, and unstructured databases such as CouchDB and MongoDB, although it also gives very useful functionalities to Web of Things. A simple example of a JSON-LD is shown in Example 2. The use of the contexts allows JSON-LD to map data.

Example 2: Example of a JSON-LD. Extracted from [10]

```
{
  "@context": "https://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John_Lennon",
  "name": "John Lennon",
  "born": "1940-10-09",
  "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}
```

Example 2 shows a simple JSON-LD where the context links the data structure of the JSON with the concept of Person of the ontology Friend of a Friend (FoaF[1]) described in the URI https://json-ld.org/contexts/person.jsonld. Based on such context the terms "Name", "born" and "spouse" have a clear semantic meaning and can be understood by machine and humans.

The vocabulary of the Thing Description is divided into: core, data schema, WoT security and Hypermedia Controls vocabularies. The interaction models between Things, conceptual basis of the processing of Thing Descriptions and their serialization.

The Thing Description information [11] model is built in:

- Core vocabulary, which reflects the Interaction Model with the Properties, Actions, and Events Interaction Affordances.
- Data Schema vocabulary, including (a subset of) the terms defined by JSON Schema.
- WoT Security vocabulary, identifying security mechanisms and requirements for their configuration.
- Hypermedia Controls vocabulary, encoding the main principles of RESTful communication using Web links and forms.

The vocabularies introduced before are the main parts of the TD information model, then, the elements that put together all the Things, i.e. platforms, wearables, web services. Therefore, they must be understood in order to create a framework based on this paradigm.

A Thing defined as a Thing Description includes the following properties fields: context, type, id, title, description, properties, actions, events, forms, security and security

[1] http://xmlns.com/foaf/spec/

definitions, among others. In Table 1, a compilation of all the fields that are included in the Thing Description is shown.

Table 1: Core vocabulary of Thing Description [11]

Vocabulary term	Description	Assignment	Type
@context	JSON-LD keyword to define short-hand names called terms that are used throughout a TD document.	mandatory	anyURI or Array
@type	JSON-LD keyword to label the object with semantic tags (or types).	optional	string
Id	Identifier of the Thing in form of a URI RFC3986[2] (e.g., stable URI, temporary and mutable URI, URI with local IP address, URN, etc.).	optional	anyURI
Title	Provides a human-readable title (e.g., display a text for UI representation) based on a default language.	mandatory	String
titles	Provides multi-language human-readable titles (e.g., display a text for UI representation in different languages).	optional	MultiLanguage
description	Provides additional (human-readable) information based on a default language.	optional	string
descriptions	Can be used to support (human-readable) information in different languages.	optional	MultiLanguage
version	Provides version information.	optional	VersionInfo
created	Provides information when the TD instance was created.	optional	dateTime
modified	Provides information when the TD instance was last modified.	optional	dateTime
support	Provides information about the TD maintainer as URI scheme (e.g., mailto RFC6068[3], tel RFC3966[4], https).	optional	anyURI

[2] Uniform Resource Identifier (URI): Generic Syntax. T. Berners-Lee; R. Fielding; L. Masinter. IETF. January 2005. Internet Standard. URL: https://tools.ietf.org/html/rfc3986

[3] The 'mailto' URI Scheme. M. Duerst; L. Masinter; J. Zawinski. IETF. October 2010. Proposed Standard. URL: https://tools.ietf.org/html/rfc6068

[4] The tel URI for Telephone Numbers. H. Schulzrinne. IETF. December 2004. Proposed Standard. URL: https://tools.ietf.org/html/rfc3966

Vocabulary term	Description	Assignment	Type
Base	Define the base URI that is used for all relative URI references throughout a TD document. In TD instances, all relative URIs are resolved relative to the base URI using the algorithm defined in RFC3986. Base does not affect the URIs used in @context and the IRIs used within Linked Data[5] graphs that are relevant when semantic processing is applied to TD instances.	optional	anyURI
properties	All Property-based Interaction Affordances of the Thing.	optional	Map of PropertyAffordance
actions	All Action-based Interaction Affordances of the Thing.	optional	Map of ActionAffordance
events	All Event-based Interaction Affordances of the Thing.	optional	Map of EventAffordance
Links	Provides Web links to arbitrary resources that relate to the specified Thing Description.	optional	Array of Link
forms	Set of form hypermedia controls that describe how an operation can be performed. Forms are serializations of Protocol Bindings. In this version of TD, all operations that can be described at the Thing level are concerning how to interact with the Thing's Properties collectively at once.	optional	Array of Form
security	Set of security definition names, chosen from those defined in securityDefinitions. These must all be satisfied for access to resources.	mandatory	string or Array of string
securityDefinitions	Set of named security configurations (definitions only). Not actually applied unless names are used in a security name-value pair.	mandatory	Map of SecurityScheme

The Thing Description offers the possibility to add contextual definitions in some namespace. This mechanism can be used to integrate additional semantics to the

[5] Linked Data Design Issues. Tim Berners-Lee. W3C. 27 July 2006. W3C-Internal Document. URL: https://www.w3.org/DesignIssues/LinkedData.html

content of the Thing Description instance, provided that formal knowledge, e.g., logic rules for a specific domain of application, can be found under the given namespace. The contextual information also specifies some configurations and behaviour of the underlying communication protocols declared in the forms field.

Web of Things use of Thing Description is similar to OpenAPI [18] although there are important differences.

- In terms of security, while the HTTP security schemes, Vocabulary, and syntax given in this specification share many similarities with OpenAPI, they are not compatible, making this a big challenge for harmonizing OpenAPI with Web of Thing. GATEKEEPER project is addressing this challenge in T4.2.
- While OpenAPI is an open specification standard for exposing an API with a set of rules, the Thing Description is a standard that is more general it allows to expose a Thing, being understood as a device, service, platform or whatever.
- OpenAPI does not support semantic annotation while Web of Thing Description is allowing the inclusion of contexts with JSON-LD that are used for describing the semantic of the data within the Thing Description.

In the following sections we will present how two standards widely used in GATEKEEPER domain can be used in a Thing Description.

1.1.4 Role of SAREF and relation with Thing Description

Similar to Thing Descriptions, SAREF [17] ontology would provide an object model that describes the Things within the smart appliance domain. Smart Appliances REFerence (SAREF) ontology is a shared model of consensus that facilitates the matching of existing assets in the smart appliances' domain. SAREF ontology provides building blocks that allow separation and recombination of different parts of the ontology depending on specific needs. This ontology has been used for the concept of device (e.g., a light switch). Devices are physical objects designed to accomplish a particular task, in the case of GATEKEEPER, medical devices or smart-home devices. The device may perform one or more functions and for that reason, Thing Description must summarize and show all of those functionalities in a human-machine readable way. For example, a medical device that is designed to measure the pulse and oxygen in blood of a patient (tasks) and to accomplish this task it performs the start and stop function. Then, the Thing Description should provide those two functionalities to allow the interaction between the platform and the device through Web of Things architecture. However, the Thing Description might offer a list of other functionalities that can be eventually combined in order to have more complex functions in a single device. Depending on the function(s) it accomplishes, a device can be found in some corresponding states that are also listed as building blocks. When connected to a network, a device offers a service, which is a representation of a function to a network that makes the function discoverable, registrable and remotely controllable by other devices in the network. A service can represent one or more functions. A service is offered by a device that wants (a certain set of) its function(s) to be discoverable, registrable, remotely controllable by other devices in the network. A service must specify the device that is offering the service and the function(s) to be represented. A device in the SAREF ontology is also characterized by a profile that can be used to optimize some property, such as Energy, in a home or office that are part of a building.

SAREF is domain-independent ontology; however, it provides building blocks that allow recombination and separation of parts of the ontology depending on specific needs. Therefore, SAREF is designed as a compound of simple core ontology patterns. Those ontology patterns can be instantiated for multiple engineering-related verticals. There

are many extensions standardized to facilitate domain-specific modelling and management like SAREF for Energy, for Environment and for smart cities. Each domain-specific extension inherits and reuses parts of SAREF core concepts.

SAREF ontology is mainly focussed on device ontology, including sensors and actuators, which focuses on functions and measurements given by devices [12]. Following is presented an approach for the integration of SAREF ontology into Thing Description through JSON-LD contexts.

Example 3: Example of a JSON-LD. Extracted from [13]

```
{
    "@context": [
                 "https://www.w3.org/2019/wot/td/v1",
                 {
                  "saref": "http://uri.etsi.org/m2m/saref#",
                  "OnOffState": "saref:OnOffState",
                  "Light": "saref:Light"
                  }
                ],
    "securityDefinitions":{
                  "bearer_sc": {
                      "in":"header",
                      "scheme": "bearer",
                      "format": "jwt",
                      "alg": "ES256",
                      "authorization": "/auth"
                      }
                  },
    "security": ["bearer_sc"],
    "@type": [ "Thing", "Light" ],
    "name": "Bathroom Light switch",
    "properties" : [ {
      "on" : {
        "name" : "bathroom light",
        "@type" : ["Property", "OnOffState"],
        "inputType": { "type": "boolean" },
        "outputType": { "type": "boolean" },
        "forms": [
          {
              "op": "readproperty",
              "href": "/1/properties/on",
              "htv:methodName": "GET"
          },
          {
            "op": "writeproperty",
            "href": "/1/properties/on",
            "htv:methodName": "PUT"
        }
          ]
      }
     }
     ]
  }
```

This example describes the use of a SAREF ontology device in a Thing Description. In the TD there are different fields to describe functionalities. For instance, each field is used

for defining the Thing in terms of security, properties, events, context, and more. For example, properties are used to define the attributes of the Thing like *lightStatus*, *Intensity of Light*, or others. In the specific case of the shown example in Example 3, the Thing Description corresponds to a SAREF-defined device that is a bathroom light switch which has only a boolean value that is on or off. As seen there, the light can be turned on and off while assuring security through a JWT with algorithm ES256 for authentication.

1.1.5 Role of FHIR and relation with Thing Description

FHIR [14] will be a core concept within GATEKEEPER. It is a very mature standard provided by HL7, commonly used in the healthcare domain around the world with a wide community of developers and adopters. Details on FHIR will be provided in the D3.4 and D3.5 but for understanding how it will be used within GATEKEEPER some basic notions will be provided.

FHIR is a REST-ful based approach for modelling data structures as Healthcare Resources and services as REST-APIs in order to provide a solution for health interoperability. Furthermore, it addresses the semantic health interoperability between healthcare centres providing a dynamic standardized approach for the definition of the terminology used within a healthcare centre. In a very smart way, it is solving semantic interoperability between different healthcare centres standardizing the rules that allow terminology inconsistency between them. When an adopter would use FHIR it should define a FHIR Profile Resource where is defined the health terminology (e. g. SNOMED-CT[6], ICD[7], LOINC[8], etc.) used by the adopter. In this way two different adopters will differ at semantical level only in the definition of their profiles and interoperability could be reached by mapping of the terminologies used in their Profile Resources. The definition of a FHIR implementation guide and *GATEKEEPER FHIR profiles* will be the based for the GATEKEEPER healthcare data space.

Apart of Profile resources, FHIR also provides Conformance Resource. This resource is a description of the services (signatures, profiles, data exchanged, allowed parameters, etc.) provided by each endpoint of the FHIR implementation.

In the context of GATEKEEPER, a FHIR Conformance Resource is the same of a Things Description because it is describing the whole set of services included within the FHIR implementation. So, we need to avoid an unnecessary overwriting of functionalities and find a way that Thing Description and FHIR conformance resource can coexist within the platform. In this case the solution for the integration of both approaches is to integrate a

[6] http://www.snomed.org/

[7] https://www.who.int/classifications/icd/en/

[8] https://loinc.org/

FHIR Conformance Resource within a Thing Description by linking the endpoint that is providing the Conformance Resources as showed in the following example:

Example 4: FHIR Conformance Resource in the Thing Description

```
{
  "@context": [
    "https://www.w3.org/2019/wot/td/v1",
    {
      "xsd": "http://www.w3.org/2001/XMLSchema#",
      "FHIRServer":  {"@id": "td:Thing"},
      "conformance": {"@id": "xsd:anyURI"}
    }
  ],
  "@type":"FHIRServer",
  "title": "GATEKEEPER pilot x FHIR server",
  "description" : "A FHIR server implementation",
  "securityDefinitions": {
    "no_sec": {
      "scheme": "nosec"
    }
  },
  "security": [
      "no_sec"
  ],
  "conformance": "http://hapi.fhir.org/conformance?serverId=home_r4"
}
```

1.2 GATEKEEPER Stakeholders

This section aims at combining the analysis of D2.3 and the domain knowledge expressed in D6.2, with a specific focus on GK stakeholders, to identify them and the role they cover in the usage of the platform and of the solutions developed through the platform itself. As this deliverable focuses on the GATEKEEPER Platform, we will focus only in those stakeholders that have an active role in interacting with the platform, and the Things registered to it. The actors identified and described here below are classified based on the categories of: Business Space, Healthcare Space, and Consumer Space. These categories, in fact, were already used in the DoA to point out the main affected stakeholders and have been further analysed and specified in WP9. In this respect such categories are associated to the actors' survey here reported following the specification produced in WP9 and available at the moment this document was written (which excerpt is reported in Appendix E). A broader study of GATEKEEPER stakeholders will be performed in WP2, where deliverable D2.1 will detail the GATEKEEPER economical ecosystem sustaining the business model of the project. In that context, the list of stakeholders will be broadened, classified and reordered framing them based on the involvement they have in the platform strategy

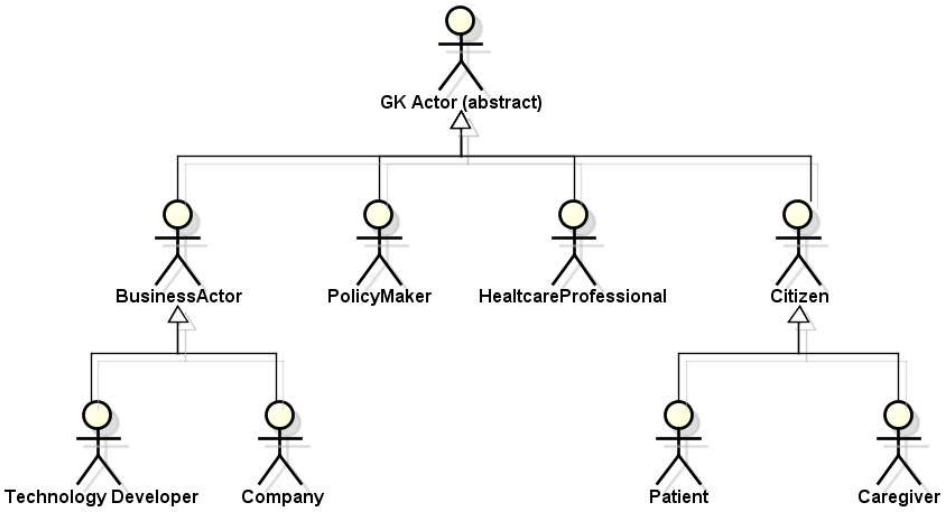


Figure 4 - GATEKEEPER actors

BUSINESS ACTOR

Extends: GK Actor (Abstract)

Description: Generic actor of the **Business Space**. He/she is the provider of marketable solutions that integrate with the GK platform

Concrete Implementers: Medtech Companies, Developers, IoT or HC Device providers

TECHNOLOGY DEVELOPER

Extends: Business Actor

Description: Develops solutions that exploit the existing GATEKEEPER services

Concrete Implementers: Solution developer

COMPANY

Extends: Business Actor

Description: Produces and markets health and wellbeing KETs

Concrete Implementers: Medtech Company

POLICY MAKER

Extends: GK Actor (Abstract)

Description: Administrator of the GK Platform. Manages the governance policies regulating the platform in the **Transaction Space**

Concrete Implementers: Governments, HC Systems

HEALTHCARE PROFESSIONAL

Extends: GK Actor (Abstract)

Description: A Professional Caregiver is a person in the **Healthcare and Consumer Space** who provide care to those who need supervision or assistance in illness or disability. They use GATEKEEPER technology and solutions to assist person or citizen

Concrete Implementers: General Practitioner, Nurse, Pharmacist

CITIZEN

Extends: GK Actor (Abstract)

Description: Citizen represents people in the **Healthcare and Consumer Space** who might be interested in the results of GATEKEEPER Interventions, directly (in the case of patients) or indirectly (for Caregivers). They consume health services.

Concrete Implementers: Patients, Informal caregivers

PATIENT

Extends: Citizen

Description: A Patient is a person receiving or registered to receive medical treatment. He/she is the owner of personal health and wellbeing data

Concrete Implementers: Elderly Citizen, Patients with co-morbidities.

CAREGIVER

Extends: Citizen

Description: Provides formal or informal care to one or more Elderly Citizens

Concrete Implementers: Assistant, Social Worker, Family Member

1.3 Security and Privacy considerations

The conceptual approach of the Security and Privacy module of GK follows the principles of the Reference Model of International Data Space Association [19]. The trustworthy Architecture focuses on exploiting and sharing Things from various sources in any type of scenarios, including cross-border cases. The Security and Privacy framework leverages existing standards, technologies and established governance models, to facilitate secure and standardized data exchange and data linkage in trusted ecosystems.

In detail, security and privacy considerations will be ensured by five main architectural elements: a) the User management, b) the Certification Authority, c) the Dynamic Attribute Provisioning Service, d) the Dynamic Trust Monitoring (DTM) and e) the Clearing House (exposed as Thing).

The User management that manages authentication and authorization within the platform based on rules.

The Certification Authority (CA) is responsible for issuing, validating and revoking digital certificates. A digital certificate is provided for a user and a Thing based on the validation mechanism. The Validation is implemented based on standardisation methods that will be delivered by T8.1.

The Dynamic Attribute Provisioning Service (DAPS) includes master data and information on security profiles. Since the CA provides the details on the digital certificate, the participant registers at the DAPS. Then the User Management mechanism identifies the validated users and authorizes the trusted users to access the GK platform. Furthermore, the validated Things are delivered to the TMS system. DAPS is also responsible for the management of dynamic consents and FAIRification Principles of Things.

Dynamic Trust Monitoring (DTM) is necessary for classification of the trustworthiness of all participants in the ecosystem. DTM implements a monitoring function for everything and shares information with the DAPS to notify on the trustworthiness of the transactions. Furthermore, trails of all transactions related to Things, maintaining a detailed history of the whole Thing lifecycle.

The Clearing House logs all activities performed in the course of a data exchange. After a data exchange, or parts of it, has been completed, both the data provider and the data consumer confirm the data transfer by logging the details of the transaction at the Clearing House. The logging information can also be used to resolve conflicts. The Clearing House also provides reports on the performed (logged) transactions for billing, conflict resolution, etc. This task is responsible for the adoption of FAIRification principles after a Thing is consumed. Thus the Clearing House will be also exposed as a Thing and delivered to the end-users of the GK platform.

In the following two diagrams the interactions of the GTA with the TMS component appear for two use cases, (i) registering a new Thing (by a registered User), and (ii) consuming a certified Thing (by a registered User). In these diagrams appear the processes of User authentication (and authorisation), Thing certification, and (Immutable) Logging of actions covering the security features of privacy, confidentiality, accountability, and non-repudiation.

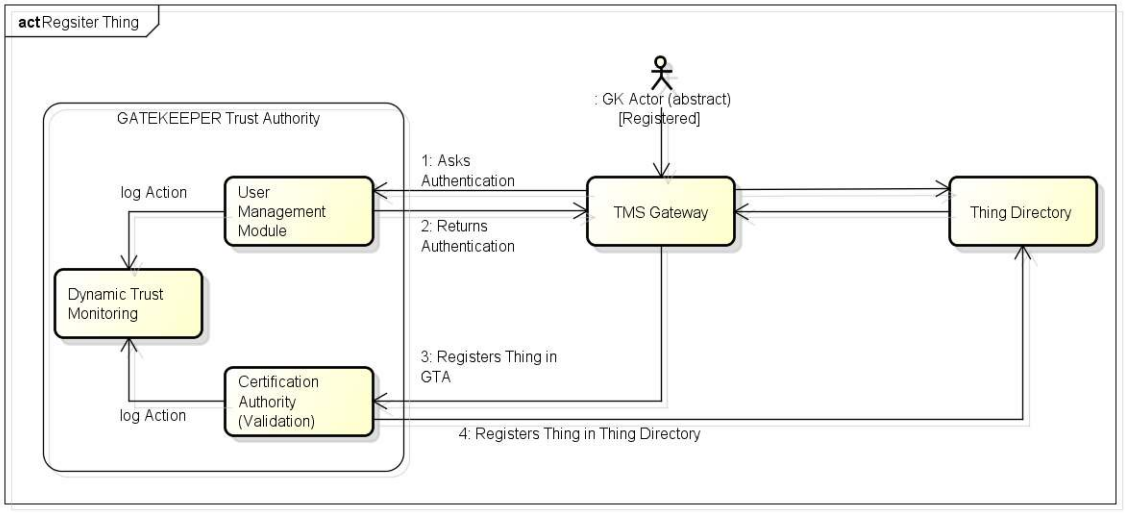


Figure 5 - Thing Registration in the GTA

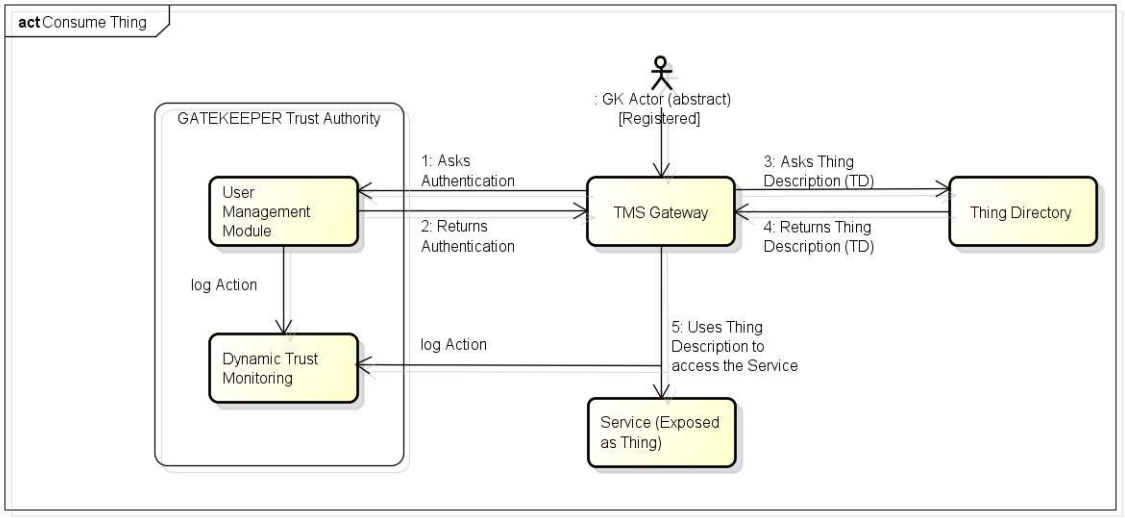


Figure 6 - Consuming a Thing

1.3.1 Infrastructure security

In order to address security and privacy issues, the GATEKEEPER Data Centre infrastructure, managed by HPE, adopt a number of technologies, products and services here briefly reported (further details will be provided as output of T4.3 activity):

- VPN access, by means of OpenVPN open-source software. Two kinds of VPN profiles are available:
  - *Road warrior,* for GATEKEEPER partners users, supporting on-demand connections from PC clients
  - *Site-to-site,* for GATEKEEPER pilots, supporting always-on connections from Pilot sites
- Support for different VPN access authentication types:
  - *Single Factor (user and password)*
  - *Two Factor Authentication (2FA)*
  - *Multi Factor Authentication (Client Certificate + Password + OTP)*
- Firewall devices and policies. They are used to determine whether a given user/pilot can access a network or a GATEKEEPER service
- Security services. They are managed by HPE and include:
  - Identity Management: user identities to access services (e.g. VPN, servers, VMs) are centrally managed by HPE
  - Public Key Infrastructure (PKI): HPE manages an internal private Certification Authority that releases digital certificates (e.g. for VPN user access or internal web sites/services) and manages their lifecycle (e.g. revocation)
  - Intrusion Detection System (IDS): a service to block malicious attacks based on security rulesets
  - Proxy Server: access to the Internet from HPE Data Centre is controlled and filtered via an HTTP Proxy Server
  - Log Management: all devices (e.g. operating system, backup, switches, firewalls, etc.) are traced, and logs are kept in a Log Management system for security purposes

2 GATEKEEPER Architecture

2.1 Logical Architecture

This section highlights the context and role of GATEKEEPER platform by giving an overview of the platform as a whole and the roles of the single components.

The following figures highlight the context of the GATEKEEPER platform by means of functional views. Component colours highlight their role.

- Pink components represent **Core Platform Things** (from WP4): they are responsible for providing the core services to the platform Things such as access to Things and data, user management and data integration,
- Blue components are Integrated **Dynamic Intervention Things** (WP5): these are the Things that expose the early detection, prediction and proactive services for healthcare
- Yellow components represent **External Things**: that can interact with the platform and respond to specific needs of Pilots or, in general, respond to specific application requirements.

The solid arrows demonstrate the main flows of the significant data managed by the Platform. The dashed arrows in the figures demonstrate the significant interactions of actors that trigger the main data flows. For clarity, we split such flows to highlight the ones that concern the Business and Transactions spaces (Figure 7) (see Appendix E) and the ones that emerge from the use in the Consumer and Healthcare spaces (Figure 8). A detailed description of the actors involved can be found in Section 1.2. The role of the Platform components is described in more details below, but the flows can be summarized in the following coarse-grained sequences:

1. The Policy Maker manages the platform by moderating the MarketService content and managing the security and privacy policies in the Trust Authority (Transaction space). Such policies regulate and define the access modalities to the Thing ecosystem (see also sec. 1.3 for further details)
2. The Technology Developers integrate the services provided by GATEKEEPER Things in customer solutions or develop new Things to be integrated in the platform. Business actors (Developers, but also Companies) publish new offerings of Things in the Marketplace, (Business space)
3. Sensor data produced along the execution of activities/exercises by the patient are fed (collected in connectors or directly) to the platform together with data from EHRs. Data are federated in the platform and processed by Dynamic Intervention services. Data and results are visualized by Healthcare professionals and Patients using the registered applications (Consumer and Healthcare spaces).

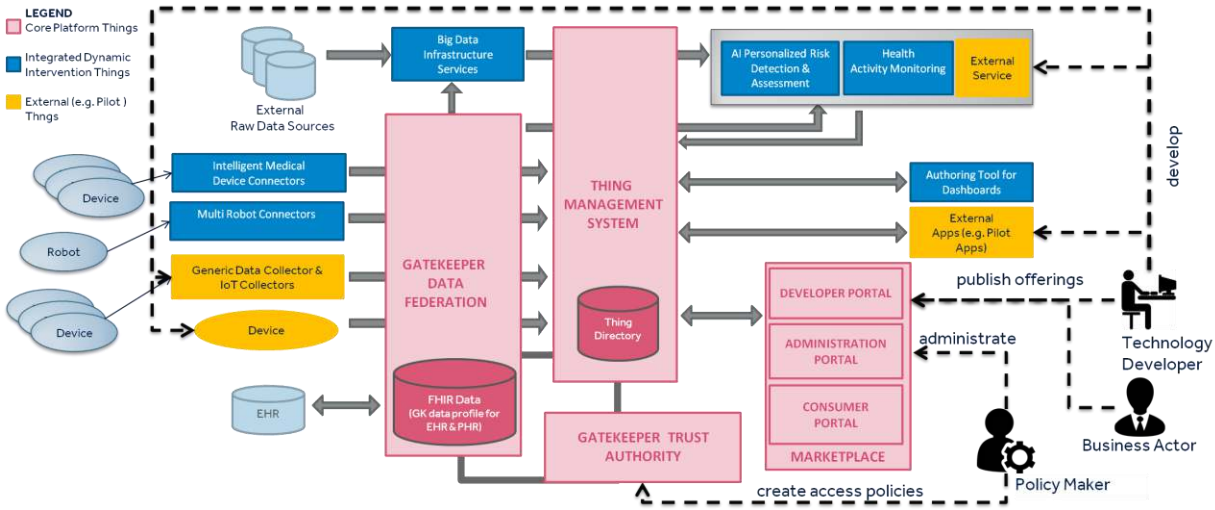


Figure 7 –GATEKEEPER Architecture view - Business and Transaction spaces

In all contexts of usage the **Things Management System** functions as the entry point of the GATEKEEPER platform. It manages Things (devices, services, platform or other assets to be operated as an individual elements) represented as Thing Descriptions, following the Web of Things approach (Section 1.1). It keeps a registry of all Things registered in the Platform in the Thing Directory and acts as an API Gateway mediating any interaction between Things and their consumers using the access policies set by the Trust Authority.

Such policies and usage rights are managed by the *Policy Maker*, who administers the Platform ensuring laws and regulations are enforced by such policies and supervises registered users and Things.

The **Trust Authority** is the component that is responsible to enforce all policies regulating the access to the platform (that are enforced by the TMS) and act as a Certification Authority for Things. It applies certification tests to the Things ensuring that a Thing respects the rules of the different GATEKEEPER Thing profiles (medical device certification, interoperability with standards, GDPR compliance, etc). The Trust Authority also checks authorization rights for the access to services and data throughout the platform. A detailed representation of the main interaction flow covered by this component has been reported in Section 1.3.

The **GATEKEEPER Marketplace** is the single-entry point for all users to explore, conceptualize, test and consume the added value services they are interested in. It will allow a uniform access to the Things ecosystems and will achieve interoperability by enabling service/application exchange between deployment sites, third parties, etc. For developers in particular, it will provide a **Developer Portal** allowing to find development and deployment material in order to publish applications and services. It will also deploy applications/services to the cloud or on premise at ease.

Developers will be also supported by the **Authoring Tool** to build and integrate UI dashboard targeting pilots' needs.

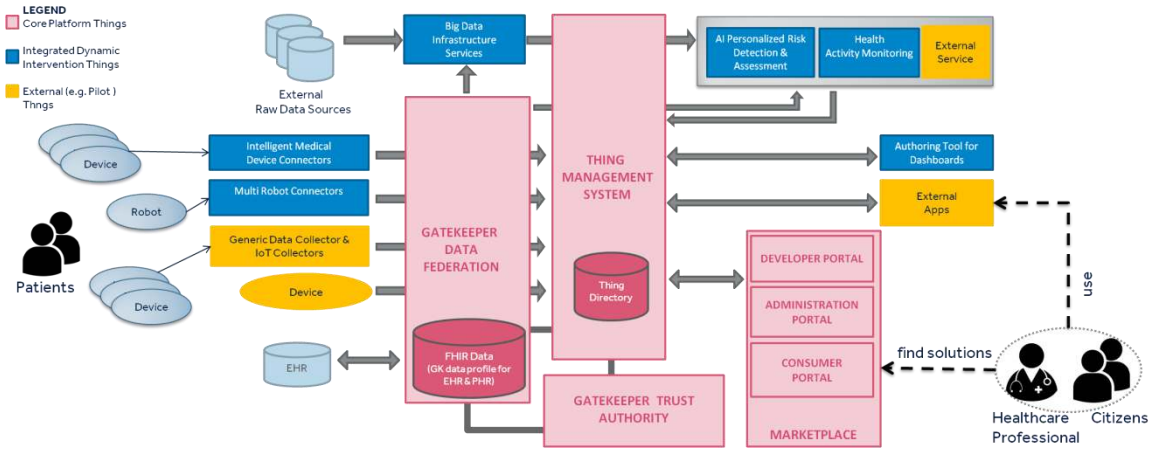


Figure 8 –GATEKEEPER Architecture view – Consumer and Healthcare spaces

Moving to the interactions affecting Consumer and Healthcare spaces (Figure 8), the main data flow and components involvement will be here briefly summarized.

A Consumer or Healthcare space actor (e.g. Patient or Healthcare Professional) finds a solution (a "Thing" in GK) by exploring the Marketplace core platform Thing. The Marketplace retrieves the description of the requested Thing (TD from now on) interacting with the Things Management System core platform Thing. In turn, the Things Management System will interact with the GATEKEEPER Trust Authority core platform Thing to get the detailed information about the requested Thing quality and certification degree. Once accepted the usage conditions the actor starts using the Thing.

Health and environmental data that are processed by platform can come from data connectors or devices provided by pilots or companies and are registered and certified in the platform as Things, or even directly accessed from EHRs. GATEKEEPER Platform already provides two types of connectors:

The **Intelligent Medical Device Connector**, that allows accessing device measurement data regardless of their differences in interfaces or connection protocols, and homogenizing their data format; the **Multi robot connector**, the connector that allows to interact and get information from robots.

GATEKEEPER Data Federation is responsible to integrate and federate data coming from the different sources. Different data acquisition modalities are supported: data can be sent explicitly by the external data sources (i.e. connectors, devices, proprietary EHR etc.) exploiting the REST interface provided by this Thing (southbound API). As alternative, if properly configured, data can be periodically fetched directly from the external data sources. Using semantic models, data are transformed in a unique format, the GATEKEEPER FHIR Data Profile, and made available to the rest of the GATEKEEPER Platform and all Things authorized to access them. It is worth to remember that being Data Federation a Thing any interaction with it is mediated (as for all the GK Things) by the TMS.

Data integrated in the data federation are also pushed to the **Big Data Infrastructure**, where they can be further processed and merged with further external data sources.

The infrastructure will provide services to perform Big Data analysis and generic models that can be exploited from the other services registered in the platform as Things.

In the platform will be also available two processing services: the **AI Personalized Risk Detection & Assessment**, that will provide diagnostic and prognostic algorithms that can

help both professionals to support their decisions and elderly with no technical knowledge to improve their independency and ability over the time; **Home and Health Activity Monitoring** that can combine Personal Health Background and Environmental Measurements, mapping of daily activities and environmental threats at home, to identify and notify abnormal conditions.

2.2 Deployment Architecture

The deployment strategy of the platform will have to fulfil constraints and requirements from Pilots. For this reason, a number of different deployment alternatives have been identified. In the following paragraphs it is described at high level the Infrastructure made available in the HPE Data Centre to host the GATEKEEPER reference architecture (a work carried out and detailed in the context of T4.1), then we will describe the reference deployment and finally alternative deployment strategies that can meet different pilot constraints are discussed.

2.2.1 GATEKEEPER Cloud Infrastructure

The following figure represents the software layers of the platform provided by HPE:

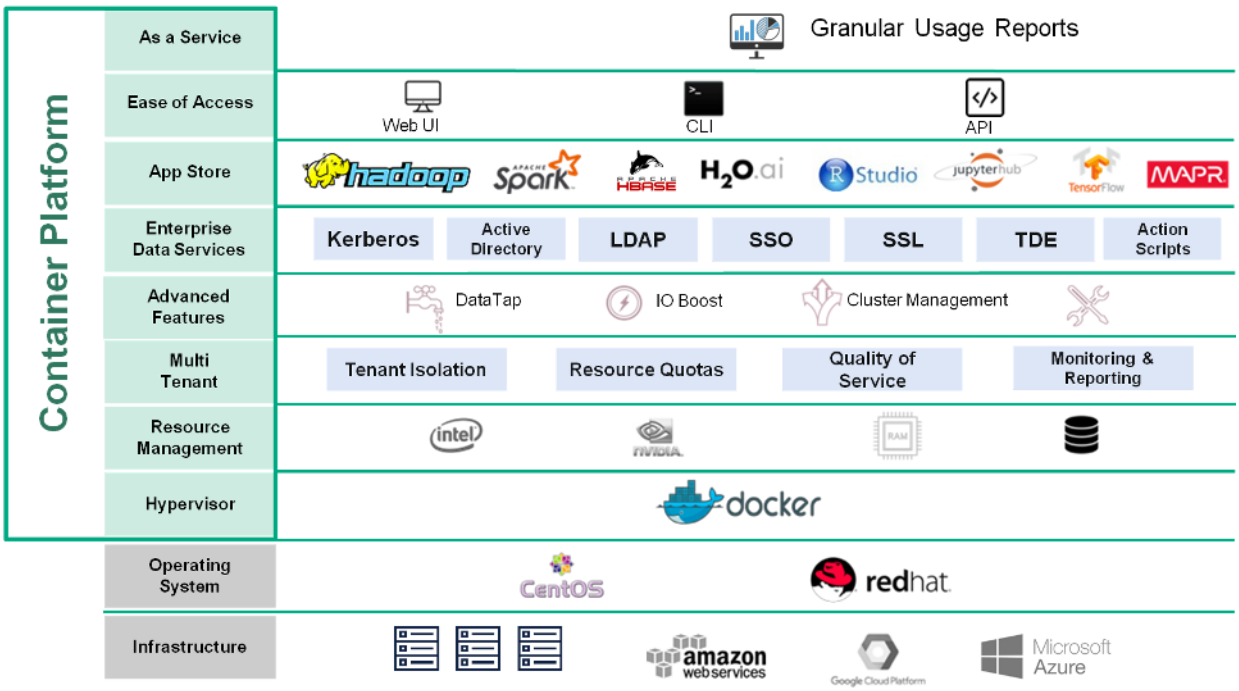


Figure 9 - Container Platform Full Stack

The bottom layer represents the infrastructure where the platform will be installed: it will be made of virtual nodes running on KVM (in principle the infrastructure nodes can include bare metal machines, or cloud instances, or both, thus allowing for hybrid deployments). The operating system will be RHEL or CentOS (CentOS will be preferred since it is open source). Internally, the HPE platform uses Docker as core containerization technology and Kubernetes for related orchestration. On top of the containerization layer, there are resource management tools, responsible for administering physical resources of nodes (CPU, RAM, storage) and obtain a complete abstraction of the underlying infrastructure, which is presented as a homogenous resources pool to the overlying layers.

The multitenancy level is responsible for segregating the resources pool into tenants. Tenants can be defined at whatever level: in GATEKEEPER, tenants can be created for

each partner, or each use case, or whatever other entity. Resource quotas can limit the resource usage for each tenant; Quality of Service is meant to tune the resource allocation between tenants.

On top of this, the platform offers other high level functionalities: the capability to connect to existing data lakes without moving data, and cluster management functions allowing users to create new clusters, stop them, expand/contract them depending on their needs, or create auto-sizing rules.

The Enterprise Data Services layer includes security services applied at container level: authentication, authorization, encryption, TDE, TLS, and SSL.

The AppStore presents a number of reference applications available to users: they can be used as they are, or they can be customized for specific requirements, e.g. using Action Scripts technology.

Finally, users can access the HPE Container Platform in multiple ways: through a web UI, or a command line interface, or REST APIs.

Further details on this Infrastructure will be reported in the context of Task 4.1.

The schema to deploy the GATEKEEPER functional blocks described above is shown in the UML diagram of Figure 10. Such deployment diagram provides a high-level view of the interactions between all components. The diagram reports the different deployment locations involved, the deployed components and their provided and required interfaces, drawing an overall picture of their relationships. The detailed description of each component, and its related interfaces, are reported in Section 4, while the possible interactions to implement the target functionalities are reported in Section 5.

The nodes can be of different types: <<platform>> node, which represents vertical platform. If the platform is a cloud platform, it is noted as such, if it can be either local or cloud, it is left unspecified. In the <<platform>> nodes, the assumption is that it will provide containerization services (Docker, Kubernetes) relying on HPE cloud infrastructure capabilities (see Section 2.2.1). All components will be deployed in such a container, so, to improve readability, in the diagram this is not reflected. <<execution environment>> nodes represent a software process providing the runtime for the components, like Java Virtual Machine or Web Server. Each container can host, naturally, one or more <<execution environments>>.

2.2.2 Reference Deployment model

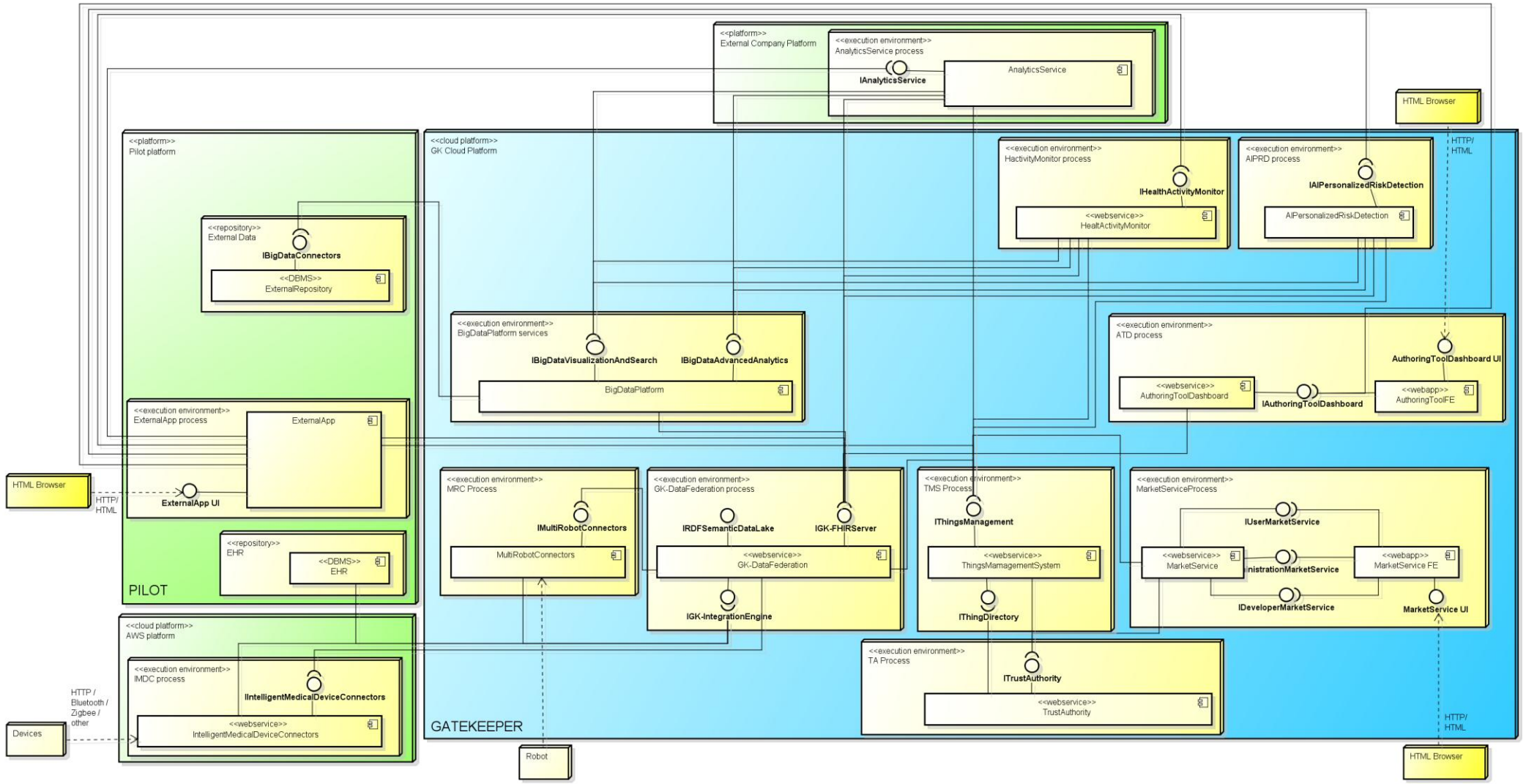


Figure 10 – GATEKEEPER platform reference deployment model

2.2.3 Deployment alternatives

2.2.3.1 GK Platform is deployed on GK Cloud

This is the reference choice of GATEKEEPER also reflected in the diagram of Figure 10.

All main components of the platform are hosted in GK cloud at HPE data centre in a single reference tenant and connects to pilot sites to fetch data and provide results. Security, updates and maintenance can be managed centrally ensuring the highest level of service. In this case GATEKEEPER Platform will be responsible to ensure separation of data and multitenancy.

2.2.3.2 Pilots own a private space on GK Cloud and share some data with GK Platform

In case Pilots require to keep part of their data isolated from the other pilots, GK Cloud can provide private storage spaces in dedicated private "*pilot cloud tenants*", while the GK Platform remain centralized. Pilot systems running in the separate spaces interact with the Platform to exploit its services from within GK Cloud.

2.2.3.3 Pilots manage a private copy of the GK platform within GK Cloud Infrastructure

To ensure a greater isolation, an alternative deployment implies the creation of separated "*pilot cloud tenants*" within the HPE data centre, where replicas of the GATEKEEPER platform are deployed separately (not only storage as in the solution above). In this solution, maintenance of the GK Cloud becomes more complex.

2.2.3.4 Pilots install an instance of GK Platform on its own premise

In this solution the Pilot decides to install GK Platform locally in its own premises. This implies that the Pilot will have to take care of the maintenance of the infrastructure and performing the updates. GK Cloud will host testing and Integration environments for the Platform. No data will be shared across pilots.

3 Information Model

The Information Model shown in the diagrams below describes the main types of data exchanged between the GATEKEEPER components.

The Information Model described focuses on two aspects: entities, and their relationships, directly used as input and output parameters of the operations provided by components (listed in section 4); an initial entity diagram that represents the Health related measurements used by pilots, as gathered by the analysis of D6.2, that will be the basis of the work of tasks 3.4 and 3.5 for the creation and formalization of a unified GATEKEEPER semantic model.

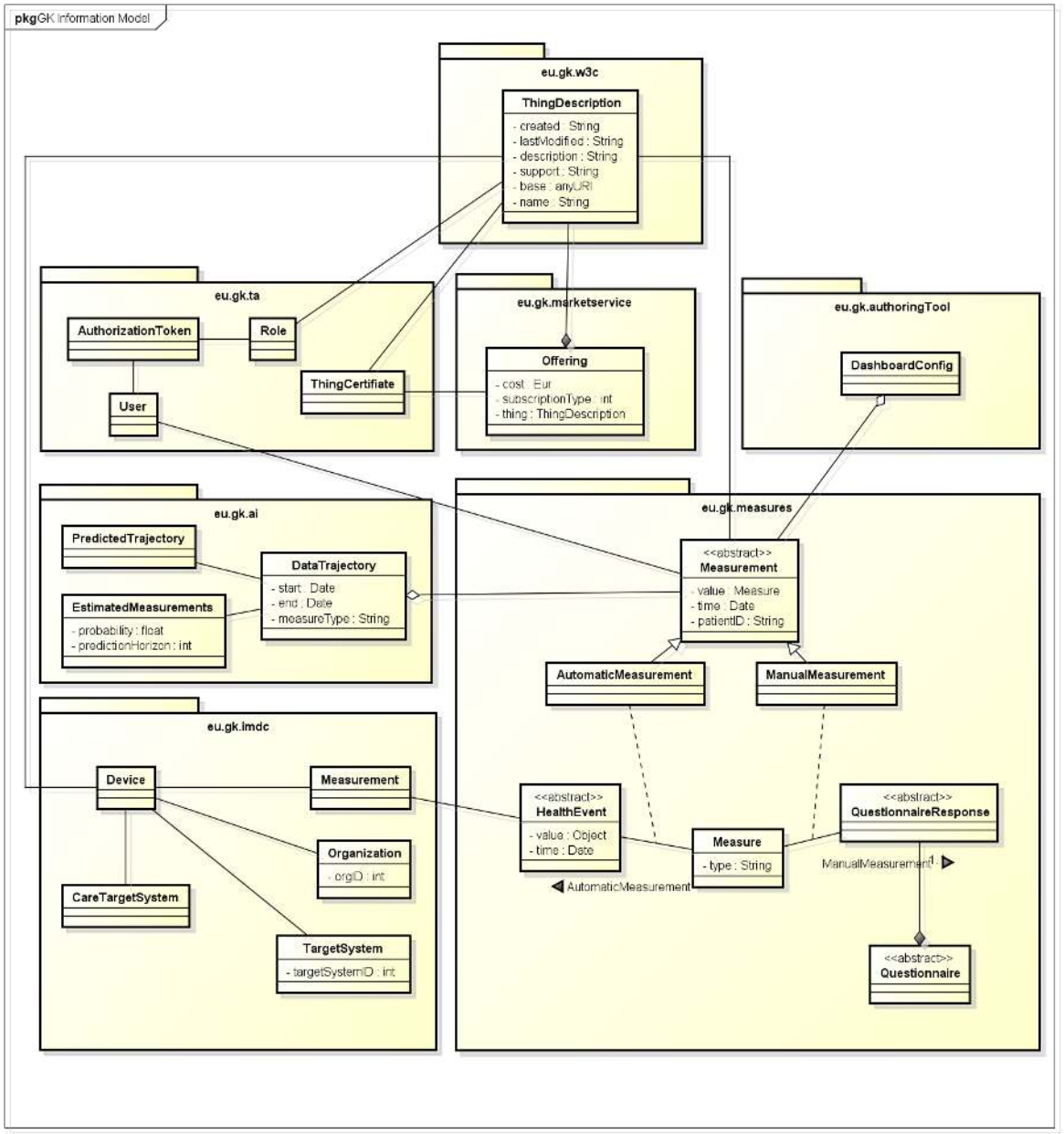


Figure 11 - GATEKEEPER Information Model

The reported Information Model (Figure 11) focuses on data involved in the component interactions defined so far in the project. The model will be continuously enriched with new entities when the interfaces of the components will be further defined.

The main data type for this platform is the *ThingDescription* described in detail in section 1.1.3. It contains descriptions of all services and Things (sensors, *Devices*) that by invoking their actions can produce or elaborate health data (*Measurements*) in the platform.

To regulate the access to the ThingDescription, and perform actions, the Trust Authority links to the TD a set or authorized *Roles.* Roles are assigned to registered *Users. Users* obtain *AuthorizationTokens* to prove their identities.

A Thing is referred in the MarketService by means of the *Offering* entity. This entity is the representation of all added value services exposed by the GATEKEEPER Platform. When an Offering describes a software service or device that can be connected to the GK platform, it links to its *ThingDescription*.

The *IntelligentMedicalDeviceConnectors* Thing can manage *Devices*, representing sensors that generate *Measurements* of patients' status. Devices belong to *Organizations*. Devices and their measurements can be accessed by *Users* with different *Roles.*

Measurements represent the Health data managed by the platform. They refer to *Measure* types, of a variety of health-related aspects. An initial set of Measures obtained by the analysis of Pilots is detailed in the next section.

All Measurements the platform manages are collected from connectors in a variety of formats (the task dealing with the identification of such formats is T3.4) and being translated in a homogeneous format to federate them and allow a homogeneous access. The target format will be formalized in the GATEKEEPER FHIR profile, output of T3.5.

Federated data can be visualized using the services of the AuthoringTool that uses *DashboardConfigs* to customize views on the *Measurements*.

Data are also processed by Dynamic Intervention services that take as input *DataTrajectories* and by the use of AI algorithms can produce Risk assessments or *Predicted Trajectories*. Details on the input and output requirements for such services is detailed in section 3.2.

3.1 Health Measures

Although details on the work of mapping input measures and their formats from pilots and defining a unique GATEKEEPER FHIR profile that is able to represent them is a joint work of T3.4 (for the concepts identification and mapping) and T3.5 (for the definition of the FHIR profile), here we give an initial overview of health measure types that the platform will manage.

Figure 12 show the result of the analysis of the list of measures required by pilots as reported in D6.2. *Measures* that the GATEKEEPER platform will have to manage comprehend *Vital Signs* (such as *Body Temperature*, *ECG*, *Respiratory Rate*), as well as data on the patient *Activity* or other parameters as *Glucose* or Sweat level.

A specific value in time of a Measure is captured by the *Measurement* entity, which describes a Measure, its value, the patient identifier and the time it was captured.

Measurements are first categorized based on the way they are captured. They can be *AuomaticMeasurements*, produced by *HealthEvents*, or *ManualMeasurements* coming from the *QuestionnaireResponses* of *Questionnaires* of self-assessment or interviews with professionals.

HealthEvents can be generated from Devices or be the result of the data processing of Dynamic Intervention Services. In the latter case they are referred as *Risk Events*.

Questionnaires can cover a variety of topics, from *Healthy Habits*, to *Cognitive Impairment*, *Dependencies*, *Medications* or *Emotional Situation*.

3.1.1 GATEKEEPER FHIR profile

A FHIR profile is a set of rules which allows a FHIR resource to be constrained or include extensions so it can add additional attributes. T3.5 will take as input all the information on relevant Resources to be included in the profile (output of T3.4), and formalize a GATEKEEPER FHIR profile to ensure data will be semantically interoperable. The profile will be based on v4 of FHIR [14].

The translation from the original format to the GK FHIR profile will be performed by the GATEKEEPER federation component, which will also provide a FHIRv4 compliant database to store the translated data and make them available for the rest of the platform.

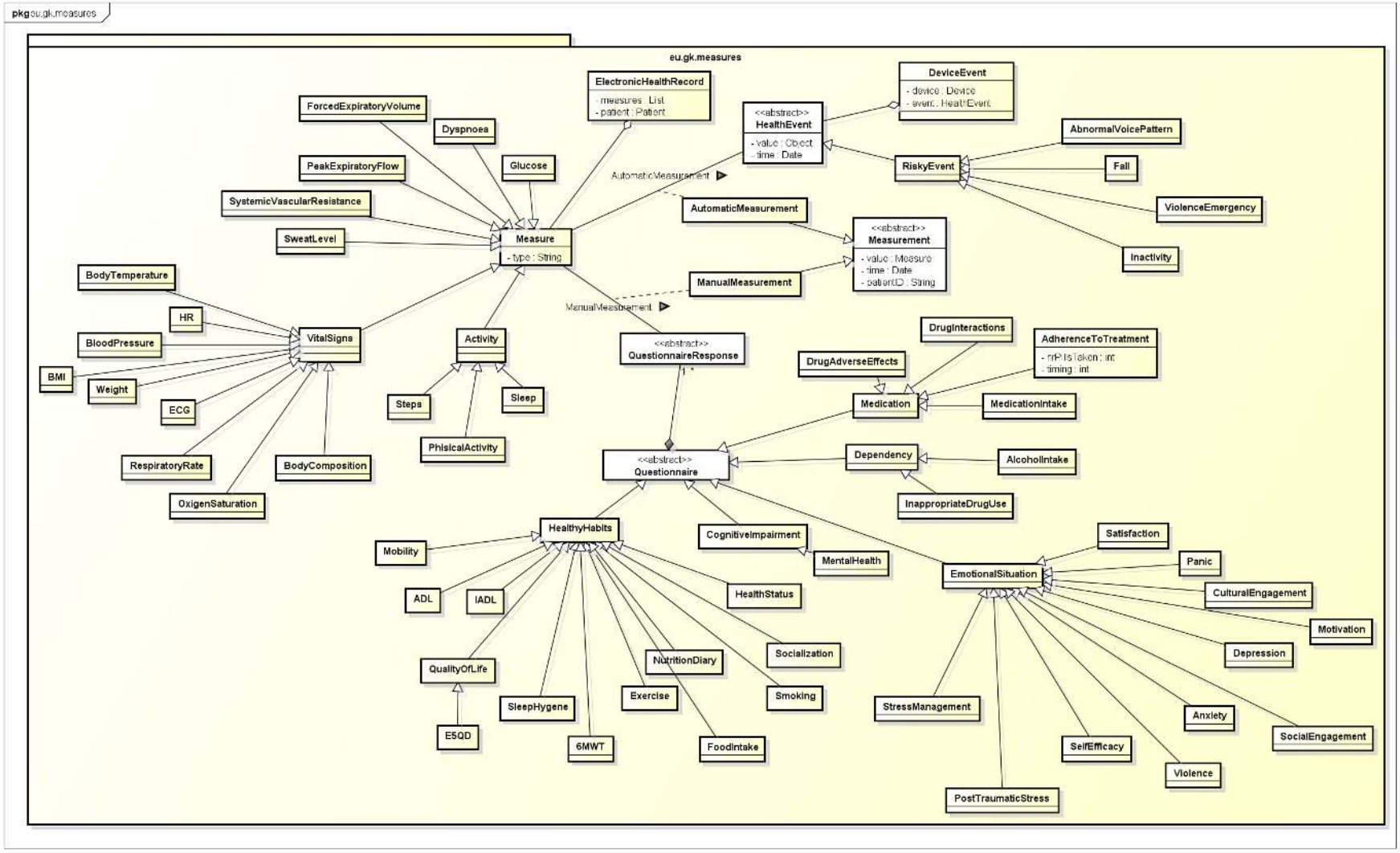


Figure 12 - GK Health Information Model

3.2 Data Structures arising from the Input-Output Requirements of the AI/ML Models

AI/ML-based risk stratification and prevention models in GATEKEEPER, as they will be specified in D6.3.1 and outlined in the description of the AIPersonalizedRiskDetection&Assessment component herein (Section 0), feature a rigid representation of their input (feature) and output spaces. The input space of each AI/ML model encompasses the set of features upon which the AI/ML based function or knowledge base is defined and learnt with respect to a research hypothesis, whereas the output space, in the case of supervised ML problems, defines the set of output variables.

An intermediate layer (southbound interface) between the GATEKEEPER Data Federation Component and the AI Personalised Risk Detection and Assessment Component shall provide, in both phases of training/validation and real-world performance monitoring of the defined and specified models within the GATEKEEPER AI Reasoning Framework, the needed data trajectories upon which each AI/ML model's feature / output space will be built. An additional intermediate layer (northbound interface) shall be able to provide the representation of each built (trained) AI/ML model, its internally validated performance, as well as each AI/ML model's output (e.g. a vector describing the predicted trajectory of an output variable over a specific prediction interval $[t, t+PH]$, for a specific prediction horizon PH, or the probability p of an event, or the probability p of each of the identified classes over a specific prediction horizon PH).

Herein, we specify the basic data structures required in the formation of the required input/output data trajectories over a specified time interval $[t_{start},\ t_{end}]$ (i.e. monitoring period) given a specific individual, provided the sampling interval Δt and $start \leq end$, such that the associated training/validation datasets (in the training/validation phase) or the defined feature vectors (in the real-world performance monitoring phase) can be constructed. The undermentioned information will be refined as the definition and specification of the AI/ML models within WP5 and WP6/T6.3 advances. More specifically, both in the case of supervised learning – risk stratification models and unsupervised learning tasks, the training and validation phases of model building require: (i) the formation of synchronised multivariate input data trajectories for a specific number of subjects, and, (ii) the formation of the time course of output variables expressing disease-related symptoms or annotated events. Box 1 presents the process of the construction of the input data trajectories as a function of a pilot study, a reference use case (RUC) and an investigated research hypothesis; the aforementioned parameters drive the selection of the sample of subjects and the related measures (e.g. EHR data, vital signs, questionnaires). Hence, the basic AI/ML-related data class pertains to the definition of the time course of measure x_m over the time interval $[t_{start},\ t_{end}]$ for one individual p: $X_{p,m} = \left[x_{m,start}^{(p)}, \dots, x_{m,start+k\Delta t}^{(p)}, \dots, x_{m,end}^{(p)}\right]$. Similarly, as it is shown in Box 2, the time course of each output variable y is encapsulated within the vector $\left[y_{start}^{(p)}, \dots, y_{start+k\Delta t}^{(p)}, \dots, y_{end}^{(p)}\right]$.

Box 1 - Construction of the input data trajectories in the training/validation phase

Retrieval of the Input Data (Pilot study, RUC, Research Hypothesis)

1. For each individual $p_i,\ i = 1, \dots, P$
   a. For each measure $m_j,\ j = 1, \dots, M$
      i. Specify the data trajectory over a specified time interval $[t_{start}, t_{end}]$ (i.e. monitoring period) provided the sampling interval Δt and $start \leq end$.

$$X_{p_i,m_j} = \left[x_{j,start}^{(i)}, \dots, x_{j,start+k\Delta t}^{(i)}, \dots, x_{j,end}^{(i)}\right]$$

   b. Specify the full data trajectory over the monitoring period $[t_{start},\ t_{end}]$.

$$X_{p_i} = \left[X_{p_i,m_1}, \dots, X_{p_i,m_j}, \dots, X_{p_i,m_M}\right] = \left[X_{p_i,m_j}\right]_{j=1}^{M}$$

2. Construct the input dataset:

$$X = \left[X_{p_1}, \dots, X_{p_i}, \dots, X_{p_P}\right]^T = \left[X_{p_i}\right]_{i=1}^{P}$$

Box 2 - Construction of the output data trajectories in the training/validation phase

Retrieval of the Output Data (Pilot study, RUC, Research Hypothesis)

1. For each individual $p_i,\ i = 1, \dots, P$
   a. Specify the data trajectory of the output variable y over a specified time interval $[t_{start},\ t_{end}]$ (i.e. monitoring period) provided the sampling interval Δt and $start \leq end$.

$$Y_{p_i} = \left[y_{start}^{(i)}, \dots, y_{start+k\Delta t}^{(i)}, \dots, y_{end}^{(i)}\right]$$

2. Construct the training output set:

$$Y = \left[Y_{p_1}, \dots, Y_{p_i}, \dots, Y_{p_P}\right]^T = \left[Y_{p_i}\right]_{i=1}^{P}$$

During the real-world performance monitoring phase, the same input data structure (i.e. : $X_{p,m} = \left[x_{m,start}^{(p)}, \dots, x_{m,start+k\Delta t}^{(p)}, \dots, x_{m,end}^{(p)}\right]$) shall capture the time course of the measures required to construct/instantiate the feature vector of each of the identified AI/ML models subject to the history window pertaining to each of the measures (Box 3). This formulation describes well the cases of supervised AI/ML models or unsupervised ones defining a mapping from the input space to a feature space e.g. (clustering). On the other hand, each AI/ML model's output formulation in the real-world performance monitoring phase depends on the specified research hypothesis and the output of the model *per se*. Box 4 provides some indicative examples of model's output formulation.

Box 3 - Construction of the input data vector in the real-world performance monitoring phase

Construction of the Input Data Vector (Pilot, RUC, Research Hypothesis, Individual)

1. For each measure $m_j,\ j = 1, \dots, M$
   a. Specify the data trajectory over a specified time interval $[t,\ t - History\ Window j]$ provided the sampling interval Δt:

$$X_{m_j} = \left[x_{j,t-History\ Window_j}, \dots, x_{j,t-k\Delta t}, \dots, x_{j,t}\right] =$$

2. Specify the full data trajectory over a specified time interval $[t,\ t - History\ Window]$.

$$X = \left[X_{m_1}, \dots, X_{m_j}, \dots, X_{m_M}\right] = \left[X_{m_j}\right]_{j=1}^{M}$$

Box 4 Formulation of a model's output in the real-world performance monitoring phase.

Formulation of AI/ML Model's Output (Pilot, RUC, Research Hypothesis, Individual)

1. Output:
   a. Vital Signals: Vector describing the predicted trajectory of the output variable y over a specific prediction interval $[t, t + PH]$, for a specific prediction horizon PH.
   b. Events: The probability p of an event, or the probability p of each of the identified classes over a specific prediction horizon PH
   c. Disease or Condition: The probability p of a condition, or the probability p of each of the identified classes over a specific prediction horizon PH

3.3 Standards

The definition and implementation of GK platform architecture will follow the project approach of standardization by design, leveraging on existing and emerging open standards.

As stated in the Section 1 of this deliverable, the main guiding standard for the architecture is the Web of Things, with JSON-LD contexts and including FHIR standard and SAREF. References to the intended use of these in GK platform have already been provided in section 1.1.

In addition to these, based on the analysis made in D8.1 that provides an exhaustive list of standards applicable in the different domains that GATEKEEPER addresses, here below we identify those specially relevant for the specification of the platform architecture:

- **EN ISO 12967-1:2011 Health informatics - Service architecture**[9]

  GK Platform is intended to be deployed in real healthcare settings, integrating, and making available existing information assets, and most likely also making possible the integration and interoperability of existing applications. This objective can be realised with the help of this standard, which purpose is to:

  - Identify a methodology to describe healthcare information systems through a language, notation and paradigms suitable to facilitate the planning, design and comparison of systems;
  - Identify the fundamental architectural aspects enabling the openness, integration and interoperability of healthcare information systems.

  The architecture is therefore intended as a basis both for working with existing systems and for the planning and construction of new systems.

  Relevant to the work of GK platform architecture are the following characteristics of this standard:

  - The architecture is described according to the methodology of ISO/IEC 10746 to provide a formal, comprehensive and non-ambiguous specification suitable to serve as a reference in the planning, design and implementation of healthcare information systems. It is structured in three parts that reflects the Enterprise viewpoint, Information viewpoint and Computational viewpoint,
  - The scope of the architecture comprises the support to the activities of the healthcare organization as a whole, from the clinical, organizational and managerial point of view. It therefore does not detail specificities of different sub domains but provides overarching comprehensive information and services framework to accommodate requirements.
  - The architecture is intrinsically compatible, complementary and synergistic with other models and standards, such as HL7 CDA, FHIR, ISO 13940[10] (Contsys) and the Electronic Health Record Architecture ISO 13606[11]. Specific information objects and services are explicitly foreseen in

[9] https://www.iso.org/obp/ui#iso:std:iso:12967:en

[10] https://www.iso.org/obp/ui#iso:std:iso:13940:en

[11] https://www.iso.org/obp/ui#iso:std:iso:13606:en

the architecture to facilitate the implementation of views and communication mechanisms based on such standards.

- **ISO TR 12300:2014 Health informatics — Principles of mapping between terminological systems**:

  Due to the need to accommodate existing infrastructures as well as to enable interoperability across different data models, GK platform will include components to facilitate mapping and connections between different terminologies. This Technical Report provides guidance for organizations charged with creating or applying maps to meet their business needs. It explains the risks inherent in the mapping process and discusses the issues that need to be considered in the development, maintenance, and use of maps in health care. Importantly, this Technical Report establishes and harmonizes the basic principles for developing, maintaining, and using maps and gives guidelines for good practice that underpin the mapping process. Terminological resources include terminologies, classifications, and code systems used in the regulatory environment as it relates to healthcare and reporting requirements in healthcare.

- **ISO/IEC 20547-3:2020 Information technology — Big data reference architecture — Part 3: Reference architecture**

  This document specifies the big data reference architecture (BDRA) and includes concepts and architectural views intended to:

  - provide a common language for the various actors;
  - encourage adherence to common standards, specifications, and patterns;
  - provide consistency of implementation of technology to solve similar problem sets;
  - facilitate the understanding of the operational intricacies in big data;
  - illustrate and understand the various big data components, processes, and systems, in the context of an overall big data conceptual model;
  - provide a technical reference for government departments, agencies and other consumers to understand, discuss, categorize and compare big data solutions; and
  - facilitate the analysis of candidate standards for interoperability, portability, reusability, and extendibility.

- **ISO/IEC TR 23186:2018 Information technology — Cloud computing — Framework of trust for processing of multi-sourced data**

  This document describes a framework of trust for the processing of multi-sourced data that includes data use obligations and controls, data provenance, chain of custody, security and immutable proof of compliance as elements of the framework.

- **ISO/IEC TR 10032:2003 Information technology — Reference Model of Data Management**

  This Technical Report defines common terminology and concepts pertinent to all data held within information systems. Such concepts are used to define more specifically the services provided by particular data management components, such as database management systems or data dictionary systems.

- **TLS Transport Layer Security**

  Transport Layer Security (TLS) are cryptographic protocols designed to provide communications security over a computer network. The TLS protocol aims primarily to provide privacy and data integrity between two or more communicating computer applications.

- **OAuth**

  OAuth is an open standard for access delegation, it provides to clients a "secure delegated access" to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. Designed specifically to work with Hypertext Transfer Protocol (HTTP), OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner. The third party then uses the access token to access the protected resources hosted by the resource server.

4 GATEKEEPER Components

In this document the components reported below are considered as black-boxes and, as such, no information is reported about their internal architecture which is documented in other deliverables. The following table provides a guide of the context where these components are provided and thus documented.

Table 2: Components list overview

Component Name	Responsible	Task
ThingsManagementSystem	UPM	4.2
ThingsDirectory	UPM	4.2
BigDataInfrastructureService	HPE	4.3
GK-IntegrationEngine	ENG	4.4
GK-FHIRServer	ENG	4.4
GK-SemanticDataLake	ENG	4.4
TrustAuthority	CERTH	4.5
MarketService	CERTH	4.6
HealthActivityMonitoring	SAMSUNG	5.2
AIPersonalizedRiskDetection&Assessment	MYS	5.3
IntelligentMedicalDeviceConnectors	MEDISANTE	5.4
AuthoringToolForDashboards	TECNALIA	5.5
MultiRobotConnectors	OU	5.6

4.1 ThingsManagementSystem

The Things Management System (TMS) is the entry point of the GATEKEEPER platform. In analogy to classical micro-services architectures, it is like an API gateway component.

The TMS will not manage directly REST-API like a common API Gateway but it will manage Things represented as Thing Description. A representation of this functionality is shown in Figure 13, and it is based on the intermediary architecture described in the Web of Things architecture specifications (https://www.w3.org/TR/wot-architecture/).

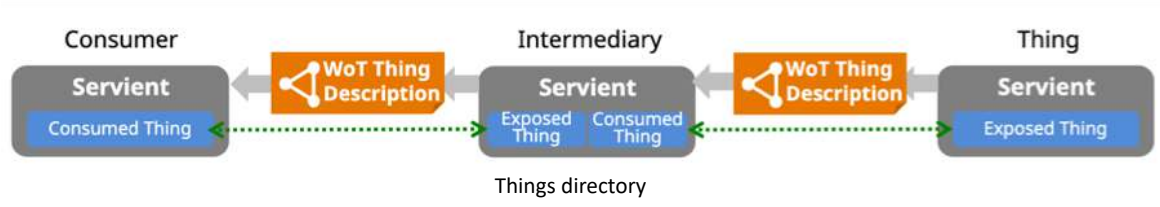


Figure 13 - Conceptual Diagram of the GATEKEEPER Things Management System

The Things Management System is the intermediate in any interaction between Things and consumers. We define Thing as any device, service or platform that is standardized with a Thing Description and with a model of data to be operated as an individual element derived from a set of predefined templates like smart light-bulbs, smart-watches, AI service or marketplace analytics platform. In Figure 14 it can be seen the initial inner architecture of Things Management System and its components. This architecture will be updated and detailed in the next deliverable D4.2.

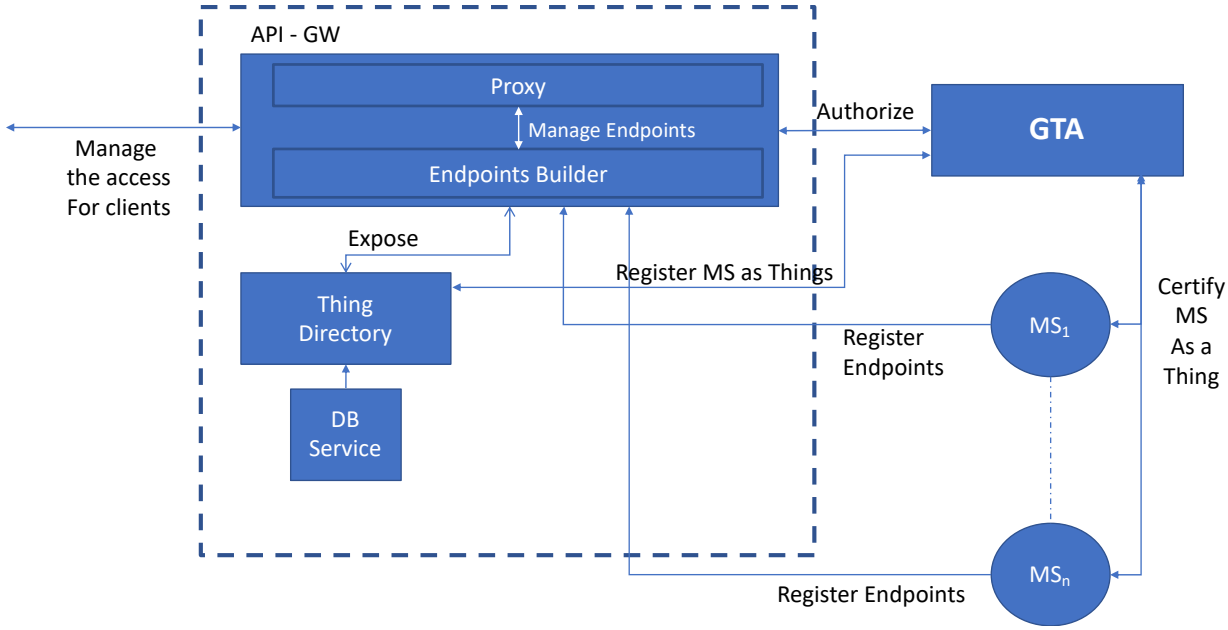


Figure 14 - GATEKEEPER Things Management System inner architecture.

In the architecture of the GATEKEEPER Things Management System, it can be identified the following components:

- GTA: GATEKEEPER Trust Authority (T4.5), it manages authentication and authorization of users in order to consume Things
- Thing Description (TD) – descriptor of the Thing compliant with WoT object model and GK semantics (T3.3, T3.4).
- API - GW – Gateway for RESTful interfaces (or other protocols) of GK services
    - Proxy: Redirect requests to different micro-services (only for REST interfaces)
    - Builder: Interact with GTA for building and register new endpoints
- Thing Directory TMS – TD – Directory that collect all GK Thing Descriptions
    - It is like a broker service that publish a directory of Thing Description of the components available within the platform
- MS_x: Microservice *X* providing service X as REST API associated to a logical Thing *X*

Two use cases have been described for the analysis of the components and functionalities that must be considered for the definition of the interfaces: (i) registration of a new Thing and (ii) normal use of the Thing.

The first use case, which is shown in GKPLAT_02, represents the interaction between the user and the platform for the registration and the first time usage of Thing. This use case foresees a deep integration with the Trust Authority that will validate the Thing before its first usage. The second use case is shown in GKPLAT_03, it describes how to use a Thing that is already registered into the platform and the interaction with the Trust Authority in order to be authenticated and authorized for its usage.

Also, the TMS is providing a Thing Description of its functionalities so in the interaction the first step is always to ask the TMS for its Thing Description that will describe the functionalities it provides.

Following are described the initial interfaces that the component should provide. Further details and updates of such interfaces will be provided in D4.2 that will describe the first version of the Things Management System.

ThingsManagementSystem Provided Interface		
access() : anyURI		
Ask to the TMS for the access to the Thing.		
Input(s)	--	
Output	thing:anyURI	*Thing Description of the TMS with security definition (e. g. Bearer authentication)*
registerForUser(String): anyURI		
Register a Things for the user		
Input(s)	thing:String	*The serialized Thing (see section 1.1.3) to register*
Output	thing:anyURI	*Thing Description of the TMS with security definition (e. g. Bearer authentication)*
getTMSDescription(String): anyURI		
Ask to the TMS for the Thing Description		
Input(s)	thingDescription: String	*The serialized Thing (see section 1.1.3) to grant access to*
Output	thing:anyURI	*Thing Description of the TMS with security definition (e. g. Bearer authentication)*
verifyUserCredentials(String, String) : String		
Users send credentials for authentication		
Input(s)	username:String	*Username*
	password:String	*Password*
Output	jwt:String	*A JSON Web Token to allow the access to the platform*

discoverThing(String): anyURI		
Request the list of Things to TMS		
Input(s)	thingDescription: String	*The serialized Thing (see section 1.1.3)*
Output	thing:anyURI	*Thing Description*
consumeTMS (thingID): anyURI		
Request one Thing to TMS		
Input(s)	Thing ID	*The ID of the requested Thing*
Output	thing:anyURI	*Thing Description*

ThingsManagementSystem Requested Interface				
Provider	**Method**	**Description**	**Input**	**Output**
TrustAuthority	*RegisterInGTA()*	*Register a Thing in the GTA*	*String* *Thing Description*	*AnyURI: Thing Description*
	verifyCredentialsInGTA ()	*User sends credentials for authetication*	*String: Username, String: Password*	*String: Json Web Token*
Thing Directory	*discoverThingInTD()*	*Request the list Things in Thing Directory*	*String* *Thing Description*	*AnyURI: Thing Description*
	RegisterInThingDir()	*Register a Thing in the Thing Directory*	*String* *Thing Description*	*AnyURI: Thing Description*
	consumeTD()	*Request one Thing to Thing Directory*	*Thing/<id>*	*AnyURI: Thing Description*
	Consume()	*Ask the new created Thing to the Thing directory*	*String* *Thing Description*	*AnyURI: Thing Description*
	update()	*upload the endpoint of the Thing Description to be accessed by the gateway*	*String* *Thing Description*	*AnyURI: Thing Description*
MSx	*ConsumeActionInMS()*	*Consume an action on a Thing in its Micro Service.*	*Thing/<id>/action*	*Object*

4.2 ThingsDirectory

This component is the repository of Thing Descriptions. It is managed by the Things management System. It acts like a broker service that publishes a directory of Thing Descriptions of the components available within the GATEKEEPER platform.

ThingsDirectory Provided Interface		
discoverThingInTD(String): anyUri		
Request the list Things in Thing Directory		
Input(s)	*String*	*Filter*
Output	*[anyUri]*	*List of url of the selected Things*
RegisterInThingDir()		
Register a Thing in the Thing Directory		
Input(s)	string	*Thing Description*
Output	bool	State of success
consumeTD()		
Request one Thing to Thing Directory		
Input(s)	AnyURI	ID of the Thing the consume
Output	String	The associated Thing Description
Insert()		
Ask the new create Thing to the Thing directory		
Input(s)	Thing Description	The Thing Description of the new Thing to insert
Output	bool	State of success
Update()		
upload the endpoint of the Thing Description to be accessed by the gateway		
Input	AnyURI	*Thing ID of the Thing to modify*
Input(s)	string	*New Thing Description*
Output	bool	State of success
Delete()		
Delete a Thing from the Thing Directory		
Input(s)	AnyURI	Thing ID of the Thing to modify
Output	bool	State of success

4.3 BigDataInfrastructureService

The Big Data Platform built by HPE offers the Big Data infrastructure services used by GATEKEEPER.

The following diagram shows the main logic components included in the Big Data Platform:

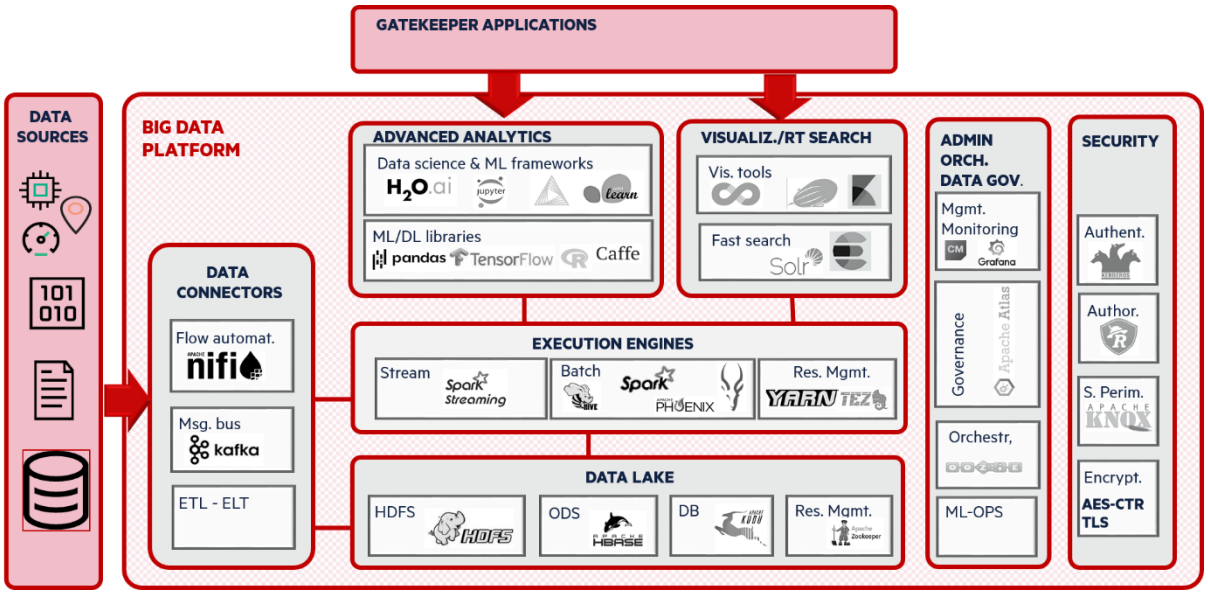


Figure 15 - Big Data Platform Architecture

In particular:

- Data Connectors are responsible for ingesting data from multiple sources: GATEKEEPER components, external DBs, IoT devices, etc. Open-source platforms like Apache NiFi allow automating ingestion flows, and support the most common data types and protocols. In case of real time flows, Apache Kafka allows to stream input data reliably, by means of publish/subscribe mechanisms. Data preparation and transformation may happen before data is loaded (ETL), or afterwards (ELT).
- The Data Lake embeds the storage systems where multiple data sets are persisted, in different forms (structured, unstructured, and semi-structured). The Data Platform takes advantage of components like HDFS and Operational data stores (e.g. HBase) to persist and organize different data formats. Resource management tools like Zookeeper are used to guarantee high availability and fault tolerance of distributed storage systems.
- The Execution Engines allow to access, search and analyse in parallel big volumes of data. Many execution engines allow SQL-like queries directly on HDFS (like Apache Hive). The workloads generated by the execution engines can in turn leverage resource management frameworks like YARN. At a higher level, the execution engines include tools for processing RT/NRT data in streaming mode (such as Flink or Spark streaming module) and for processing historical data (Hive, Spark, Phoenix, etc.). In particular, Spark includes SQL-like query modules (SparQL) and Machine Learning libraries (MLlib).
- Advanced Analytics functionalities are provided by data science frameworks and ML/DL libraries and tools, like H2O, Knime, Scikit learn, R studio, etc.

- Visualization and RT search block is meant to quickly access/show data or insights. Sometimes it is useful to offer real time search capabilities to collect RT information on new data (especially for monitoring purposes). Tools like Elasticsearch or Solr are used for fast queries, and also index data in form of documents which can be searched via full text search. Visualization capabilities are important both for data exploration / descriptive analytics and for presenting data insights: they are also provided by open source tools (e.g. Apache Superset or Apache Zeppelin).
- Administration, Orchestration, and Data Governance. They are horizontal functionalities required for the overall Data Platform: Cluster administration tools allow managing instances/parameters of big data platform components, by performing cluster maintenance, and monitoring all software and hardware resources. In terms of data governance, Apache Atlas allows to centralize and manage different types of metadata, share them with users, classify data in a dynamic way, and manage and monitor data lineage. In terms of orchestration, tools like Oozie are used to orchestrate big data workloads.
- Data Security. This layer is meant to address all the security aspects across the Data Platform:
    - Authentication. Data segregation through authentication is achieved using Apache Kerberos. Authentication mechanisms can integrate with external directory services such as LDAP or AD.
    - Authorization:
        - Permissions on file systems are enforced through ACLs. For instance, in HDFS ACLs have a definition similar to POSIX for regular Linux file systems.
        - For authorizing services (e.g. the execution of a query), Apache Ranger allows to define access policies for HDFS, Hive, HBase, Kafka, Knox, YARN. A definition of fine-grained permission policies is allowed: it is possible to define table-based or even column-based policies for execution engines like Hive and HBase.
    - Perimetral security. It is realized mainly through Apache Knox, which provides a gateway to communicate in a secure way with REST API and Hadoop user interfaces. Knox integrates with Kerberos and AD and supports WebHDFS, Oozie, JDBC, Ranger UI and other protocols.
    - Data encryption. To encrypt data at rest, on HDFS the AES-CTR (Advanced Encryption Standard-Counter) algorithm is employed. To encrypt data in motion, TLS will be always applied.
- Auditing. Apache Ranger allows to perform auditing in terms of access (e.g. a Hive query launched by a user), Administration (e.g. update of a Hadoop parameter), Login sessions, Plugin status.

BigDataInfrastructureService Provided Interface

As the infrastructure will be built on existing tools, the following interface refers to the list of candidate tools that will provide the actual interfaces. The final list of such tools will be decided in T4.3 and reported in the next version of the architecture.

advancedAnalytics
ML/DL libraries and tools, like ▪ H2O (https://www.h2o.ai/) ▪ Knime (https://www.knime.com/) ▪ Scikit learn (https://scikit-learn.org/) ▪ R studio (https://rstudio.com/)
visualization and RT search
Queries tools like: ▪ Elasticsearch (https://www.elastic.co/) ▪ Solr (https://lucene.apache.org/solr/) Visualization Tools: ▪ Apache Superset (https://superset.apache.org/) ▪ Apache Zeppelin (https://zeppelin.apache.org/)

4.4 GK-IntegrationEngine

The GK-Integration Engine is the component able to convert raw data, coming from different data sources EHR, sensors, IoT devices, wearables etc.), to HL7/FHIR v4.0.1 and RDF representation. Data can be sent to this component invoking the REST APIs that it exposes. For IOT, it accepts as input data in the formats XML, JSON and CSV and provides as output their representation in RDF. The rules for the transformation are written with the language RML using the terminologies provided by the task T3.4. Transformed data is sent to the component GK-SemanticDataLake.

For Electronic Health Records, it converts custom EHRs into FHIR v4.0.1 representation according to the GK FHIR profiles defined in the task T3.5. Data can be sent to this component invoking the REST APIs that it exposes. GK-IntegrationEngine accepts as input data in the formats XML and JSON and provides as output their representation in FHIR standard (JSON/FHIR). The terminology to be used for the conversion is provided by the task 4.4. Finally transformed data is sent to the component GK-FHIRServer.

IGK-IntegrationEngine Provided Interface
create(pilot: String, sensorId: String, data: File): responseBody: String
Interface accepting data in XML/JSON/CSV format coming from IOT devices (or connector services). If a FHIR processor has been preliminary registered for that device/service, data will be converted and persisted in a FHIR R4 repository. The data will be also converted in RDF and made available in to GK-SemanticDataLake component. If the registered converter produces data compliant to other ontologies (e.g. SAREF) then they will be loaded only in GK-SemanticDataLake repository. In order to select the appropriate rules to be applied for the transformation, this method accepts as input the name of the pilot, the id of the sensor and a file contacting data. **[POST method]**

Input(s)	pilot: String	*The name of the pilot. Knowing the name of the pilot this method can apply the right transformation for each pilot. Note that each pilot uses a different data schema*
	sensorId: String	*The id of the sensor. The main goal of this parameter is to select which converter rules should be applied to the data. The pair pilot+sensorId allows to select the specific transformation rules for the data*
	data: String	*Actual raw data that must be transformed in RDF and sent to GK-SemanticDataLake. The format of the data can be JSON, XML and CSV. In order to write the rules for the conversion in RDF, it is necessary to know the schema of JSON,XML/CSV.*
Output	String	*data in the new format (XML or JSON)*

create(pilot: String, data: String): responseBody: String

Transforms data in FHIR representation and sends it to GK-FHIRServer component. It returns the output of operation returned by the GK-FHIRServer together with the HTTP codes describing the execution outcome.

This component defines and implements specific conversion rules for each type of data of each use case. In order to select the appropriate rules to be applied for the transformation, this method must know the name of the pilot to which data belong to. **[POST method]**

Input(s)	pilot: String	*The name of the pilot. Knowing the name of the pilot this method can apply the right transformation for each pilot. Note that each pilot uses a different data schema*
	data: String {json/xml}	*Actual raw data that must be transformed and persisted in the GK-FHIRServer. In order to perform the rules for the transformation in FHIR standard, the structures of the data should be known.* *The format of the data can be JSON or XML. In order to write the rules for the conversion in FHIR standard, it is necessary to know the schema of JSON/XML.*

Output	responseBody: String	*Operation outcome returned by the FHIRServer in JSON/XML format together with the HTTP code that provides feedback about execution outcome.*

GK-IntegrationEngine Requested Interface				
Provider	**Method**	**Description**	**Input**	**Output**
GK-SemanticDataLake	*create(rdfData: String): HTTPResponse*	*Persist transformed data (RDF) into GK-SemanticDataLake. POST*	*rdfData: String*	*HTTPResponse*
GK-FHIRServer	*create (resource: Bundle): Bundle*	*Send to the GK-FHIRServer raw data (belonging to the pilot) transformed in FHIR standard. It is required that the GK-FHIRServer implement all the operation defined in the FHIR standard. https://hl7.org/FHIR/http.html#operations. POST*	*resource: Bundle*	*Bundle*

4.5 GK-FHIRServer

The GK-FHIR-Server is a component implementing the HL7/FHIR v4.0.1 specification. It provides all RESTful operations described by the standard. Refer to the specification for more details: https://www.hl7.org/fhir/http.html.

This component has been developed relying on HAPI FHIR Library (https://hl7.org/FHIR/index.html) that is an open-source implementation of the FHIR specification in Java which defines model classes for every resource type and data type defined by the standard.

Persisted data are translated in RDF format and sent to the component GK-SemanticDataLake thought its REST APIs.

IGK-FHIRServer Provided Interface		
create(resourceType: String, resource: Resource): HTTPResponse		
Create a new resource in a server-assigned location. *POST method*		
Input(s)	resourceType: String	*resource type of the resource to create*
	resource: Resource	*FHIR Resource to create. The resource does not need to have an id element (this is one of the few cases where a resource exists without an id element). If an id is provided, the server SHALL ignore it.*
Output	HTTPResponse	*The server returns a 201 Created HTTP status code, and SHALL also return a Location header which contains the new Logical Id and Version Id of the created resource version*
read(resourceType: String, id: String): resultBody: Resource		
Read the current state of the resource. *GET method*		
Input(s)	resourceType: String	*resource type of the resource to read*
	id: String	*id of Resource*
Output	resource: Resource {json/xml}	*Resource returned by the GK-FHIRServer with the content specified for the resource type in JSON/XML format together with the HTTP code that provides feedback about execution outcome*
vread(resourceType: String, id: String, vid: String): resultBody: Resource		
Read an individual resource instance given a version ID to retrieve a specific version of that instance to vread that instance). *GET method*		
Input(s)	resourceType: String	*resource type of the resource to read*
	id: String	*id of resource*
	vid: String	*version ID to retrieve a specific version of that instance (optional)*

Output	resultBody: Resource {json/xml}	*Resource returned by the GK-FHIRServer with the content specified for the resource type in JSON/XML format together with the HTTP code that provides feedback about execution outcome*
update(resourceType: String, id: String, resource: Resource): resultBody: Resource		
Update an existing resource by its id (or create it if it is new) *PUT method*		
Input(s)	resourceType: String	*resource type of the resource to update*
	id: String	*id of resource*
	resource: Resource	*FHIR Resource to update*
Output	resultBody: Resource {json/xml}	*Resource returned by the GK-FHIRServer with the content specified for the resource type in JSON/XML format together with the HTTP code that provides feedback about execution outcome*
delete(resourceType: String id: String): HTTPResponse		
Delete an individual instance of the resource. DELETE method		
Input(s)	resourceType: String	*resource type of the resource to delete*
	id: String	*id of Resource to delete*
Output	HTTPResponse	*Operation outcome returned by the FHIRServer in JSON/XML format together with the HTTP code that provides feedback about execution outcome*
history(resourceType: String, [id: String]): responseBody: Bundle		
Retrieve the update history for a particular resource type, or against a specific instance of that resource type if an ID is specified. GET method		
Input(s)	resourceType: String	*resource type of the resource to read*
	id: String (optional)	*id of Resource to read*

Output	responseBody: Bundle {json/xml}	*The return content is a Bundle with type set to history containing the specified version history, sorted with oldest versions last, and including deleted resources*
search(resourceType: String, parameters: String[]): responseBody: Bundle		
Search all resources of a particular type using the criteria represented in the parameters. GET method		
Input(s)	resourceType: String	*resource type of the resource to perform the search*
	parameters: String[]	*parameter of the search request*
Output	responseBody: Bundle {json/xml}	*The return content is a Bundle the set of the resources fitting the input parameters*

IGK-FHIRServer Requested Interface				
Provider	**Method**	**Description**	**Input**	**Output**
GK-SemanticDataLake	*create(rdfData: String): HTTPResponse*	*Persist RDF data into GK-SemanticDataLake. POST*	*rdfData: String*	*HTTPResponse*

4.6 GK-SemanticDataLake

This component is an open source modular Java framework for working with RDF data. This includes parsing, storing, inferencing and querying of/over such data. It offers an easy-to-use API that can be connected to all leading RDF storage solutions. It allows you to connect with SPARQL endpoints and create applications that leverage the power of Linked Data and Semantic Web.

This server should be configured to be compliant to GATEKEEPER. For now, the only REST operation that it is used is described in the following that allows to store RDF file.

GK-SemanticDataLake provided interface		
create(rdfData: String): HTTPResponse		
REST operation that allows to store RDF file – *POST method*		
Input(s)	rdfData: String	*RDF data to be persisted*
Output	HTTPResponse	*Http response of the requests*

4.7 TrustAuthority

The "TrustAuthority" is the component that will be responsible for validating and certifying the Things of the GATEKEEPER platform. It will apply validation tests to the "Things" based on a predefined set of specifications that will ensure that a Thing respects the rules of the different GATEKEEPER Thing profiles (medical device certification, interoperability with standards, GDPR compliance) and levels of trustiness will be calculated as a score. Besides, it will act as a Certification Authority (CA) able to issue digital certificates, which will certify a Thing by giving it the appropriate attributes, and by describing the ownership of a public key by the named subject of the certificate. Furthermore, it will use a distributed ledger so as to keep an audit trail of all transactions related to Things, thus maintaining a detailed history of the whole Thing lifecycle. Furthermore, the ledger will track of operations performed on the available data, such as creation, access, deletion and sharing among parties, without access to the actual personal data due to security and regulatory compliance. This component will interact with the "ThingsManagementSystem" to secure all transaction related to Thing lifecycle when an external system (e.g. a User) is authenticated and allowed to perform actions to a "Thing".

TrustAuthority Provided Interface		
authenticateUser(domain: string): string		
This method will take as an input the domain used by the User in which the User has valid credentials; this domain will be used for the credential validation during the Single sign on mechanism to be used by the User Management module. This method will interact with any GK component that will need to authenticate Users using the User Management module of the GTA component (e.g. the Marketplace). *POST: /authenticateUser*		
Input(s)	domain:string	*this is a string representing the domain to which the user will be redirected by the User Management module, in order to validate their credentials using the OAuth2.0 mechanism*
Output	authorisation_token:string	*this is a token provided by the User Management that will contain encoded authorisation information about the User based on their certificate issued by the GTA*
registerThing(authorisation_token:string, thing:ThingDescription) file		
This method will take as an input an authorisation token string that will contain encoded authorisation information about the User, and a Thing Description (TD) object, and after applying proper Validation of the Thing based on a set of predefined standards, it will produce a validation score. This validation score will be linked with levels of certification and corresponding permissions/roles. A certificate will be issued for the Thing by the Certificate Authority having as an attribute this Validation Score. This method will interact with the TMS. *POST: /registerThing*		

Input(s)	authorisation_token:string	*this is an authorisation token string that will contain encoded authorisation information about the User*
	thing:ThingDescription	*this is the object representing the device, application, service etc. See Thing Description in the Information Model*
Output	Thing Certificate:file	*this is the certificate file (probably in X.509 format) of the Thing as provided by the GTA. It will contain the public key for the Thing as well as the Validation score as attribute of the Certificate*

logAction(string:UserID, string:ThingID, string:ActionType, timestamp:Timestamp)

This method will take as an input a User ID, a Thing ID, and the Type of Action the User wants to perform on the Thing (e.g. register, consume, etc.) and will log this triplet on the ledger along with the timestamp of the action. This method will be called by the TMS API for logging actions on Things and by the User Management Module for logging actions of Users.

POST: /logAction

Input(s)	userID	*this is the ID of the User as contained in the Thing Description (TD) of the User*
	thingID	*this is the ID of the Thing as contained in the Thing Description (TD) of the Thing*
	actionType	*this is the description of the Action the User wants to do on a Thing (e.g. register, consume, etc.)*
	timeStamp	*this is the timestamp when the action was performed by the User in the User Interface, e.g. the timestamp when the User clicked the button to register a new service with the GK Marketplace.)*

4.8 MarketService

MarketService will provide a single-entry point for all users to explore, conceptualize, test and consume the added value services they are interested in. This high-level component will include several services and User Interaction (UI) interfaces in order to make accessible to the interested users the Things belonging to the respected space.

The key goals of MarketService are:

- Unify platform/service ecosystems whether they have a marketplace or not;
- Achieving interoperability by enabling service/application exchange between deployment sites, third parties, etc.;
- Find development and deployment material in order to publish apps/services;
- Deploy applications/services to the cloud or on premise at ease.

Its interface consists of 3 separate panels that offer multiple level of functionality and include all of the operations of the MarketService.

- **END USER PORTAL OR MARKET:**

  End users will be able to log in, search, discover, download-register and review offerings already available in MarketService.

- **DEVELOPERS PORTAL:**

  Registered developers will be able to upload new offerings, edit and update already listed ones, check reviews of end users to their offerings, find useful development and deployment material.

- **ADMINISTRATION PORTAL:**

  Administration portal is the central management console of the MarketService. There administrators can locate information about transactions, reviews, offering reports (abusive content etc...), inspect security checks for new and updated offerings, grant access to new administrators.

UserMarketService provided Interface		
downloadOffering(offeringID): string		
Downloads or deploys a given offering		
Input(s)	*Offering_id*	The id of the offering to download
Output	*url or outcome*	*The url to download or the outcome of the deploy*
login(user_creds): Auth_outcome		
Performs the login of the user to the marketplace		
Input(s)	*User_creds*	User credentials object
Output	*Auth_outcome*	*Authentication outcome object*
register(user_data): Auth_outcome		
Registers a new user to the marketplace		
Input(s)	*User_data*	Object of the user data
Output	*Auth_outcome*	*Registration outcome object*
elevateUser(user_id): Elevation_result		
Elevates an end-user to developer		
Input(s)	*User_id*	Identification number for the user to be elevated
Output	*Elevation_result*	*Result of the elevation process*

AdministrationMarketService Interface		
approveUser(user_id): Appr_outcome		
Elevates user to marketplace administration		
Input(s)	*User_id*	The id of the elevating user
Output	*Appr_outcome*	*The outcome of the approval process*

DeveloperMarketService Interface		
createOffering(Offering): Validation		
Create a new offering to be listed to the marketplace		
Input(s)	*Offering_data*	An object with all the metadata of the offering
Output	*Validation*	*The result of the validation process of the offering*

4.9 HealthActivityMonitoring

HealthActivityMonitoring as a single software component for elderly, family and friends and professionals for supporting independent living, remote monitoring and improving quality of life. *HealthActivityMonitoring* has been developed as an open Internet of Things (IoT) platform with semantics & ontologies for richer data interpretation and ML facilitation, where homes have been equipped with non-intrusive sensors in order to capture regular passive sensor data and infer regular home activities based on the data. On this basis, we detect Directly Detected Home Activities (DDHA), such as, TV Watching and Bathroom Usage and Indirectly Detected Home Activities (IDHA), such as, Meal Preparation and Taking Shower, using an Abduction-Inference paradigm. On the top, a model of anomaly behaviour detection for the users is created. At this stage, as our first attempt in the context of assisted living, we define anomaly behaviour when performing any of the above activities as either doing it too much or doing it too less with respect to user's normal activity account, over a period (time interval). *HealthActivityMonitoring* and its models are built and evaluated with real users in a real-life setting across several pilot in this project.

From a technological point of view, it is based on a combination of IoT technologies, wearable and smart phone. More importantly, when possible and appropriate, it has been based on the combination of the above mentioned data with the semantically formalized health data from the WP4, T4.4 (GK Semantic Data Lake) which, in turns, relies on FHIR based Data Model developed in WP3, T3.4.

The key goals of *HealthActivityMonitoring* are:

- monitoring daily living activities (ADL), their classification, and recognition of routine daily patterns and habits of elderly, including Bathroom-Usage, TV-Viewing, Sleep-Patter, Mean-Preparation (Breakfast, Lunch and Dinner);
- monitoring daily physical activities (ADL) and recognition of routine daily physical patterns and habits of elderly, including exercise, distance, number of steps, average speed, and so on;

- detecting anomalies to supporting the remote monitoring from the caregiver prospective;
- building on the top of the semantic data lake, WP4, T4.4, to enable the personalized prospective of the above services.

HealthActivityMonitoring Interface		
A set of APIs that can we used to extend any fronted application with the ADL AI based monitoring and anomaly detection engine as described below:		
getTVViewingBehaviour(userID:int, startDate:Data, endDate:Data): Result<Object>		
Given a specific userID, a start date and an end date, it gives the average hours of TV watched during that period, plus a Boolean indicating whether the amount is an anomaly or not. *GET:* orgs /{ userID }{ startDate }{ endDate } /tv-viewing-behaviour		
Input(s)	UserID: int startDate:Data endDate:Data	*Id of the user participant* *startDate* *endDate*
Output	Result<Object>	*the average hours of TV watched by a specific user, with a Boolean indicating whether the amount is an anomaly or not*
getBathroomUsageBehaviour (userID:int, startDate:Data, endDate:Data): Result<Object>		
Given a specific userID, a start date and an end date, it gives the Average number of times bathroom used over a specific period of time, plus a Boolean indicating whether the amount is an anomaly or not. GET: /orgs /{ **userID** }{ **startDate** }{ **endDate** } /bathroomusage-behaviour		
Input(s)	UserID: int startDate:Data endDate:Data	*Id of the user participant* *startDate* *endDate*
Output	Result<Object>	*It gives the Average number of times bathroom used over a specific period of time, plus a Boolean indicating whether the amount is an anomaly or not.*
getActiveTimeBehaviour(userID:int, startDate:Data, endDate:Data): Result<Object>		
Given a specific userID, a start date and an end date, it gives the Average hours spent active – moving from one room to another - at home over a specific period of time, plus a Boolean indicating whether the amount is an anomaly or not. GET: /orgs /{ **userID** }{ **startDate** }{ **endDate** } /**activetime**-behaviour		

Input(s)	UserID: int startDate:Data endDate:Data	*Id of the user participant* *startDate* *endDate*
Output	Result<Object>	*it returns the Average hours spent active – moving from one room to another - at home over a specific period of time, plus a Boolean indicating whether the amount is an anomaly or not.*

getKitchenUsageBehaviour(userID:int, startDate:Data, endDate:Data): Result<Object>

Given a specific userID, a start date and an end date, it gives the Average hours spent in kitchen room over a specific period of time, plus a Boolean indicating whether the amount is an anomaly or not.

GET: /orgs /{ **userID** }{ **startDate** }{ **endDate** } / **kitchen-usage-behaviour**

Input(s)	UserID: int startDate:Data endDate:Data	*Id of the user participant* *startDate* *endDate*
Output	Result<Object>	it returns the Average hours spent in kitchen room over a specific period of time, plus a Boolean indicating whether the amount is an anomaly or not.

getStepsTakenBehaviour (userID:int, startDate:Data, endDate:Data): Result<Object>

Given a specific userID, a start date and an end date, it gives the number of steps carried by the user over a specific period of time (steps taken are recorded by the users' smartwatches and smartphones), plus a Boolean indicating whether the amount is an anomaly or not.

GET: /orgs /{ **userID** }{ **startDate** }{ **endDate** } / **steps-taken-behaviour**

Input(s)	UserID: int startDate:Data endDate:Data	*Id of the user participant* *startDate* *endDate*
Output	Result<Object>	it returns the number of steps carried by the user over a specific period of time (steps taken are recorded by the users' smartwatches and smartphones), plus a Boolean indicating whether the amount is an anomaly or not.

getBreakfastPreparationBehaviour (userID:int, startDate:Data, endDate:Data): Result<Object>		
GET: /orgs /{ **userID** }{ **startDate** }{ **endDate** } / **breakfast-preparation-behaviour**		
Input(s)	UserID: int startDate:Data endDate:Data	*Id of the user participant* *startDate* *endDate*
Output	Result<Object>	it returns the number of times the user prepared breakfast at home over a specific period of time, plus a Boolean indicating whether the amount is an anomaly or not.
getLunchPreparationBehaviour (userID:int, startDate:Data, endDate:Data): Result<Object>		
GET: /orgs /{ **userID** }{ **startDate** }{ **endDate** } / **lunch-preparation-behaviour**		
Input(s)	UserID: int startDate:Data endDate:Data	*Id of the user participant* *startDate* *endDate*
Output	Result<Object>	it returns the number of times the user prepared lunch at home over a specific period of time, plus a Boolean indicating whether the amount is an anomaly or not.
getDinnerPreparationBehaviour (userID:int, startDate:Data, endDate:Data): Result<Object>		
GET: /orgs /{ **userID** }{ **startDate** }{ **endDate** } / **dinner-preparation-behaviour**		
Input(s)	UserID: int startDate:Data endDate:Data	*Id of the user participant* *startDate* *endDate*
Output	Result<Object>	it returns the number of times the user prepared dinner at home over a specific period of time, plus a Boolean indicating whether the amount is an anomaly or not.

getTakingShowerBehaviour (userID:int, startDate:Data, endDate:Data): Result<Object>		
GET: /orgs /{ **userID** }{ **startDate** }{ **endDate** } / **taking-shower-behaviour**		
Input(s)	UserID: int startDate:Data endDate:Data	*Id of the user participant* *startDate* *endDate*
Output	Result<Object>	it returns the number of time the user took the shower over a specific period of time, plus a Boolean indicating whether the amount is an anomaly or not.

4.10 AIPersonalizedRiskDetection&Assessment

GATEKEEPER AIPersonalizedRiskDetection&Assessment Component will encompass a set of AI/ML-based solutions to the research hypotheses identified and specified within Tasks 6.1 and 6.2, respectively. The analysis of (i) the medical reference use cases (RUCs 1-7), as they have been described within D6.1.1, along with (ii) the precise mapping of interventions to measures and respective KETs in D6.2.1, yielded a first description of the early risk detection and prevention research problems need to be addressed through AI/ML strategies within GATEKEEPER (Table 3). The definition of the AI/ML research problems and the analysis of their respective modelling requirements (i.e. the specification of the input-output spaces and the understanding of the dynamics of each examined system), considering the description of each Reference Use Case (RUC) population, will drive the AI/ML model design and development process. In particular, the design of the AI/ML models development pipeline (data curation, model selection, hyper-parameter optimization and model evaluation) and the specification of its individual components (classes of ML algorithms, centralized or distributed learning schemes) will be affected by the intrinsic characteristics and complexity of each problem as well as the physical characteristics of GATEKEEPER architecture (e.g. topology of GATEKEEPER datasets). AI/ML clinical risk prediction models are typically represented as a classification model or as a time-to-event prediction model; nonetheless, the integration of longitudinal continuous biological/physiological monitoring data and EHR data, transfuses dynamic/temporal features to the algorithmic approaches which will be adopted in GATEKEEPER. To this, searching for novel data patterns which may lead or be associated with a crucial event will be also examined.

Table 3 - AI/ML Research Hypotheses within GATEKEEPER

Reference Use case	Research Problems
RUC1: Lifestyle-related early detection and interventions	Predictive Modelling (Inductive Reasoning) • Stratification (classification) of the patients according to the associated risk of frailty, or equivalently, prediction of the risk of frailty. Pattern Mining and Clustering (Inductive Reasoning) – Behavioural Modelling • Searching for novel patterns - Recognition of data patterns leading to/associated with frailty. Identification of high-risk groups.
RUC2: COPD exacerbations management	Predictive Modelling (Inductive Reasoning) • Stratification (classification) of the patients according to the risk of adverse outcomes such as exacerbations in COPD, or equivalently, prediction of the probability of a COPD exacerbation over a predefined time interval treated via classification algorithms (e.g. Class 0: No Event, Class I: Event, or Class 0: Low Risk, Class I: Intermediate Risk, Class II: High Risk). • Time to event prediction modelling. Pattern Mining and Clustering (Inductive Reasoning) – Behavioural Modelling • Searching for novel patterns - Recognition of data patterns leading to/associated with an exacerbation. Identification of high-risk groups.
RUC3: Diabetes, predictive modelling of glycaemic status	Predictive Modelling (Inductive Reasoning) • Blood glucose prediction refers to the short-term prediction of the time series of subcutaneous glucose concentration. Prediction horizon may range between 5 and 60 min. • Dynamic regression problem addressed through adaptive linear time series or non-linear ML approaches. Pattern Mining and Clustering (Inductive Reasoning) – Behavioural Modelling • Searching for novel patterns - Recognition of data patterns consistently leading to hypoglycaemic events or related to hyperglycaemic excursions.
RUC4: Parkinson's disease treatment decision support system	Predictive Modelling (Inductive Reasoning) • Stratification (classification) of the patients according to the risk of adverse PD progression over a predefined time interval treated via classification algorithms (e.g. Class 0: Low Risk, Class I: Intermediate Risk, Class II: High Risk). Pattern Mining and Clustering (Inductive Reasoning) – Behavioural Modelling • Searching for novel patterns - Recognition of data patterns leading to/associated with adverse PD progression. • Identification of high-risk groups.

Reference Use case	Research Problems
RUC5: Predicting readmissions and decompensations in Heart Failure	Predictive Modelling (Inductive Reasoning) • Stratification (classification) of the patients according to the risk of adverse outcomes such as a HF decompensation event, or equivalently, prediction of the probability of a HF decompensation over a predefined time interval treated via classification algorithms (e.g. Class 0: No Event, Class I: Event, or Class 0: Low Risk, Class I: Intermediate Risk, Class II: High Risk). • Time to event prediction modelling. Pattern Mining and Clustering (Inductive Reasoning) – Behavioural Modelling • Searching for novel patterns - Recognition of data patterns leading to/associated with a HF event. • Identification of high-risk groups.
RUC6: Primary and secondary stroke prevention	Predictive Modelling (Inductive Reasoning) • Stratification (classification) of the patients according to the risk of adverse outcomes such as an incipient stroke event, or equivalently, prediction of the probability of an incipient stroke over a predefined time interval treated via classification algorithms (e.g. Class 0: No Event, Class I: Event, or Class 0: Low Risk, Class I: Intermediate Risk, Class II: High Risk). • Time to event prediction modelling. Pattern Mining and Clustering (Inductive Reasoning) – Behavioural Modelling • Searching for novel patterns - Recognition of data patterns leading to/associated with a stroke event. • Identification of high-risk groups.
RUC7: Multi-chronic elderly patient management including polymedication	Predictive Modelling (Inductive Reasoning) • Stratification (classification) of the patients according to the risk for drug-induced or related health problems, or equivalently, prediction of the probability of a event over a predefined time interval due to non-adherence or drug interactions treated via classification algorithms (e.g. Class 0: No Event, Class I: Event, or Class 0: Low Risk, Class I: Intermediate Risk, Class II: High Risk). • Polypharmacy complications prediction modelling. Pattern Mining and Clustering (Inductive Reasoning) – Behavioural Modelling • Searching for novel patterns - Recognition of data patterns leading to/associated with polypharmacy event. Identification of high-risk groups.

In this direction, the GATEKEEPER AI Reasoning Framework has been developed and implemented as joint effort between WP5 and WP6/T6.3. Starting from the specific appliances, sensors and wearable devices, solutions and applications that the technological consortium partners have made available through others previous templates, the purpose of the GATEKEEPER AI Reasoning Framework is to clarify the AI-

Reasoning-Relationship between Technological partners in WP5 and WP6/T6.3 (let's call them Producer) and LSP needs (let's call them Consumer), encouraging AI-model reusability and cross validation, exchange practices in order to support the GK innovation statement. Each LSP can be schematized in 4 layers: (i) Consumer Layer; (ii) AI Reasoning Layer; (iii) Feature Layer, and (iv) Device Layer. The GATEKEEPER AI/ML research hypotheses for each of the RUCs along with the associated AI/ML modelling needs will be refined according to the continuing analysis of the pilot's requirements. On the basis of a well-defined research hypothesis space, the architecture of the AI/ML models and their individual components (data selection and management, model training and tuning, model performance evaluation and clinical evaluation) will be defined in line with good practices for constantly developing AI/ML-based software in healthcare. The abovementioned analysis will be reported in the first deliverable of Task 6.3 (i.e. D6.3.1, M12).

Both phases of training/validation and real-world performance monitoring of the defined and specified models within the GATEKEEPER AI Reasoning Framework require that an intermediate layer (interface) between the GATEKEEPER Data Federation Component and the AI Personalised Risk Detection and Assessment Component provides/formulates the needed data trajectories. For instance, in the case of a supervised learning-based risk stratification model such an interface shall provide the required data trajectories for each individual and input/output measure over a specified time interval $[t_{start},\ t_{end}]$ (i.e. monitoring period), provided the sampling interval Δt and $start \leq end$, such that the associated training/validation datasets (in the training/validation phase) or the specified input data vectors (in the real-world performance monitoring phase). An additional intermediate layer (interface) shall be able to provide the representation of each built (trained) AI Reasoning model, their internally validated performance, as well as each AI Reasoning model's output (e.g. a vector describing the predicted trajectory of an output variable over a specific prediction interval $[t, t + PH]$, for a specific prediction horizon PH, or the probability p of an event, or the probability p of each of the identified classes over a specific prediction horizon PH).

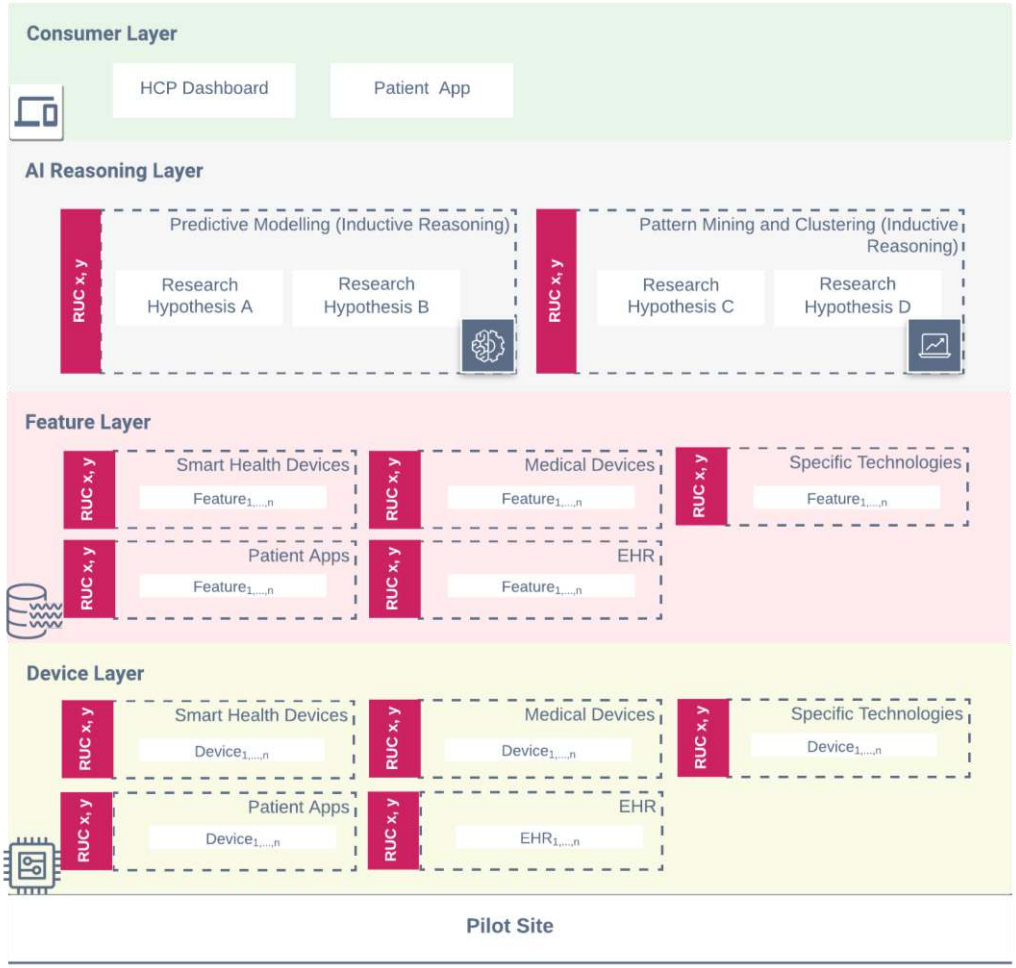


Figure 16 - The GATEKEEPER AI Reasoning Framework.

AIPersonalizedRiskDetection&Assessment provided interface
ConsumeAIService
Interface provided toward any external UI service to allow requests for usage of AI developed tools over specified datasets

AIPersonalizedRiskDetection&Assessment Requested Interface				
Provider	**Method**	**Description**	**Input**	**Output**
BigDataInfrastructureService		*Retrieve specified datasets from the semantic data lake component to execute AI reasoning*		

4.11 IntelligentMedicalDeviceConnectors

IntelligentMedicalDeviceConnectors offers services allowing to connect devices to the platform. This component has been designed to reformat device data containing measurements. This, assuming appropriate configuration (including security protocols) should be expected to be received by a patient system. There should be no need for the customer to use our login or UI in a general sense. Although a UI can be used for device management out of the box, our API endpoints allow CRUD operations with just a bearer token. So, in this respect the IntelligentMedicalDeviceConnectors should be described as a medical data integration engine, connecting devices to clinical systems via a cloud based integration (as described in the Deployment Model), all without needing Bluetooth pairing or an app to download (for sending measurements).

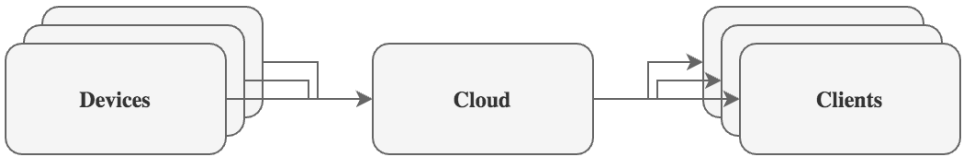


Figure 17 - Conceptual component overview

IntelligentMedicalDeviceConnectors Provided Interface

A rich API that powers the frontend. A pull-based, public API is available for:

- device management (configuration, retrieve details),
- target system management (destinations for measurements),
- user management (role-based access)
- organisation management.

A push-based API is also available for the purpose of sending measurements in different formats.

Server

https://api.medisante.net/v1

getDevicesByOrgID(orgID: int):devices: List<Device>		
List all devices owned by an org *GET:* orgs /{orgID} /devices		
Input(s)	orgID: int	*Id of the organisation*
Output	List<Device>	*List all devices belonging to the specific organisation*

postDevicesByOrgID(orgID: int, device: Device): int
Create a new device for an org POST: /orgs /{orgID} /devices

Input(s)	orgID: int device: Device	*Id of the organisation* *json with the features of the device to be created*
Output	int	
getJobStatus(jobID: int):status: String		
Get the status of an async job *GET:* /jobs /{jobID}		
Input(s)	jobID: int	*Id of the async job*
Output	status: String	*Return the status of an async job*
postDevicesChangeWeightUnit(orgID:int, params: Object): Job		
Create an async job to change the weight unit of devices *POST:* /orgs /{orgID} /jobs /change-devices-weight-unit		
Input(s)	orgID: int params: Object	*Id of the organisation* *Object containing device in order to change weight unit*
Output	Job	*Return Job created*
postDevicesChangeMinWeight(orgID:int, params: Object): Job		
Create an async job to change the minimum weight of devices *POST:* /orgs /{orgID} /jobs /change-devices-minimum-weight		
Input(s)	orgID: int params: Object	*Id of the organisation* *Object containing device in order to change min weight*
Output	Job	*Return async Job*
postDevicesChangeTimeZone(orgID: int, params:Object):Job		
Create an async job to change the time zone of devices *POST:* /orgs /{orgID} /jobs /change-devices-time-zone		
Input(s)	orgID: int params: Object	*Id of the organisation* *Object containing device in order to change time zone*
Output	Job	*Return async Job*
postDevicesChangeHBConfig(orgID: Job, params: Object): Job		
Create an async job to change the heartbeat config of devices POST: /orgs /{orgID} /jobs /change-devices-heartbeat-config		

Input(s)	orgID: int params: Object	*Id of the organisation* *Object containing device in order to change heartbeat configuration*
Output	Job	*Return async Job*
postResetDeviceToken(orgID: int, params: Object):Job		
Create an async job to reset devices tokens *POST:* /orgs /{orgID} /jobs /reset-devices-token		
Input(s)	orgID: int params: Object	*Id of the organisation* *Object containing device in order to reset device tokens*
Output	Job	*Return async Job*
postDeviceTargetSystem(orgID: int, params: Object):Job		
Create an async job to change the target systems of devices *POST:* /orgs /{orgID} /jobs /change-devices-target-systems		
Input(s)	orgID: int params: Object	*Id of the organisation* *Object containing device in order to change device target systems*
Output	Job	*Return async Job*
postTransferDeviceOwnership(orgID: int, params: Object):Job		
Create an async job to transfer the ownership of devices *POST:* /orgs /{orgID} /jobs /transfer-devices-ownership		
Input(s)	orgID: int params: Object	*Id of the organisation* *Object containing device in order to change device target systems*
Output	Job	*Return async Job*
getDevice(deviceID: int): Device		
Retrieve the details of a device *GET:* /devices /{deviceID}		
Input(s)	deviceID: int	*Id of the device to return*
Output	Device	*Detail of the requested device*
getDeviceMetrics(deviceID: int): Metrics		
Retrieve the metrics for a device *GET:* /devices /{deviceID} /metrics		

Input(s)	deviceID: int	*Id of the device to return*
Output	Metrics	*Metrics of the specific device*
getAllDevices(): List<Device>		
List all devices *GET:* /devices		
Input(s)	-	-
Output	List<Device>	*List all devices*
getAllTargetSystem(orgID: int): List<TargetSystem>		
List all target systems owned by an org *GET:* /orgs /{orgID} / target-systems		
Input(s)	orgID: int	*Id of the organisation which target system belong to*
Output	List<TargetSystem>	*Return the list of target system within specific organisation*
postTargetSystem(orgID: int, ts: TargetSystem): int		
Create a new target system for an org *POST:* /orgs /{orgID} / target systems		
Input(s)	orgID: int ts: TargetSystem	*Id of the organisation* *Target system for the specific organisation*
Output	int	
postCareTargetSystem(orgID: int, ts: TargetSystem): int		
Create a new Eliot Care target system for an org *POST:* /orgs /{orgID}/care-target-systems		
Input(s)	orgID: int ts: TargetSystem	*Id of the organisation* *Care target system for the specific organisation*
Output	int	
getTargetSystem(targetSystemID: int): TargetSystem		
Retrieve the details of a target system *GET:* /target-systems /{targetSystemID}		
Input(s)	targetSystemID: int	*Id of the target system*

Output	TargetSystem	*Details of the specific target system*
patchTargetSystem(targetSystemID: int, ts: TargetSystem): void		
Modify a target system *PATCH:* /target-systems /{targetSystemID}		
Input(s)	targetSystemID: int ts: TargetSystem	*Id of the target system to modify* *Target system to modify*
Output	void	
deleteTargetSystem(targetSystemID: int): void		
Delete a target system *DELETE:* /target-systems /{targetSystemID}		
Input(s)	targetSystemID: int	*Id of the target system to delete*
Output	void	
postSendTestMeasurement(targetSystemID: int, test: Measurement): void		
Sends test measurements to a target system *POST:* /target-systems /{targetSystemID} /send-test-measurements		
Input(s)	targetSystemID: int test: Measurement	*Id of the target system* *Test measurement to send to target system*
Output	void	
getAllUsers(): List<User>		
Retrieve all users the user has access to *GET:* /users		
Input(s)	none	
Output	users: List	*List of all users*
postCreateUser(user: User): int		
Create a new user *POST:* /users		
Input(s)	user: User	*Information to create a new user*
Output	int	

postAddRoles(userID: int, roles: List<Role>): void		
Add global roles for a user *POST:* /users /add-global-roles		
Input(s)	userID: int roles: List<Role>	*Id of the user* *List of the roles provide for the specific user*
Output	Void	
postRemoveGlobalRoles(userID: int): void		
Remove global roles for a user *POST:* /users /remove-global-roles		
Input(s)	userID: int	*Id of the user*
Output	none	
getUser(userID: int): User		
Retrieve the details about a user *GET:* /users/{userID}		
Input(s)	userID: int	*Id of the user to retrieve information*
Output	User	*The user*
postResendPWD(userID: int): void		
Resends a password activation email to the user *POST:* /users /{userID}/resend-user-password		
Input(s)	userID: int	*Id of the user to which resend password activation email*
Output	none	
getRoles(): List<Role>		
Retrieve all roles the user has right to manage *GET:* /roles		
Input(s)	none	
Output	role: List	*Return the list of all roles the user has right to manage*
getOrgs(): List<Organization		
Retrieve all orgs the user has access to *GET:* /orgs		

Input(s)	none	
Output	List<Organization>	*Return the list of organizations the user has access*
postOrganization(orgs: Organization) : int		
Create a new org *POST:* /orgs		
Input(s)	orgs: Organization	*Information to create a new organization*
Output	int	*The id of the organization*
getOrg(orgID: int): Organization		
Retrieve the details about an org *GET:* /orgs /{orgID}		
Input(s)	orgID: int	*Id of an organization*
Output	Organization	*Detail about the specific organization*
patchOrg(orgID: int): org: Organization		
Update an org *PATCH:* /orgs /{orgID}		
Input(s)	orgID: int	*Id of an organization*
Output	Organization	*Detail about the specific organization*
postAddRole(orgID: int, roles: List<Role>): void		
Add org roles for a user POST: /orgs /{orgID}/add-roles		
Input(s)	orgID: int roles: List<Role>	*Id of organization* *List of roles to add for a user*
Output	void	
postRemoveRole(orgID: int): void		
Remove org roles for an organization *POST:* /orgs /{orgID}/remove-roles		
Input(s)	orgID: int	*Id of organization*
Output	void	

IntelligentMedicalDeviceConnectors Requested Interface

To exploit all IntelligentMedicalDeviceConnectors service functionalities, it will need from the GK Platform:

- An endpoint to send measurement data to.
- Basic authentication (API keys, TLS, OAuth 2.0)

4.12 AuthoringToolForDashboards

This component offers a visual service allowing pilot sites to configure specific dashboards targeting healthcare professionals in order to show a graphical representation of the patient data and observations collected. For doing so, this component will use the Health semantic model based on FHIR as common basis for the data visualization.

This component will be web-based, and it will provide a user-friendly and intuitive Graphical User Interface. The component enables the visualization not only of patient's current status but also historical trends and comparison analysis between patient groups.

AuthoringToolForDashboards Provided Interface

A visual interface where the following initial features will be provided:

- Patient management
- GK data categorization and visualization
- Dashboard configuration and visualization

4.13 MultiRobotConnectors

The MultiRobotConnectors implement a fully autonomous and modular robotic platform combined with an intervention for in-house assistance. The application will focus on three or four main functionalities depending on the hardware configuration.

- **Exploration.** A patrolling robot that navigates autonomously around the house by exploiting a pre-existing map of the building. While patrolling it scans the environment using cameras and depth sensors and use them to identify potentially hazardous situations, the location of some key objects, habits of the user to help maintain a routine or react to an unexpected event.
- **Reminders.** The robot can act as an embodied reminder system to rely to the user notifications coming from different applications. With its ability to detect activities and habits, the robot can perform intelligent reminders depending on the behaviour and actions of the user.
- **Telepresence.** Multiple forms of teleoperation can be used to remotely pilot the robot and interact with the user or explore the environment. Goal-based asynchronous teleoperation (e.g. "go to the kitchen" or go to a specific location). Semi-assisted free navigation. Direct teleoperation using a dedicated connection.

- **Manipulation.** In the case of platforms equipped with a robotic arm the robot can directly interact with the environment to resolve hazardous situation or to help the user.

The core functionalities of the robot are developed using ROS (Robot Operating System). ROS is a modular and open source framework based on asynchronous message exchange to develop a component-based architecture; it is the de facto standard for robot software development. Additionally, the robot will interface with a web-based data hub that will collect most of the raw measurements and all the processed information, such as user location, user activities, object location, hazards detected, etc.

MultiRobotConnectors Provided Interface

The data hub acts as a web-based interface to let external entities access the sensor readings and processed information generated by the robot. Such data streams include:

- 2D map of the environment
- Position of the robot
- Position of the user
- Position of objects and environmental hazards
- Approximation of activities of the user (e.g., working, sleeping, cooking)
- Raw data stream from the robot sensors

Example API:

/{robotID}/robot-2D-location: GET

Return the most recent robot position given the ID of specific robot

GET

Input(s)	robotID: string	*Id of the robot*

Output	<pre>{
	"@id": "string",
	"@type": "RobotLocation",
	"timestamp": "2020-06-29T16:08:34.256Z",
	"x": 0,
	"y": 0
	}</pre>
		Timestamped 2D location of the robot with respect to a predefined origin.

The data hub also acts as an interface to asynchronously teleoperate the robot by collecting precise goals and destinations.

Example API:

/{robotID}/robot-2D-goal: POST

Return the most recent robot position given the ID of specific robot

GET

Input	robotID: string	*Id of the robot*

Input	{
	"@id": "string",
	"@type": "RobotGoal",
	"timestamp": "2020-06-29T16:08:34.256Z",
	"x": 0,
	"y": 0
	}
		Timestamped 2D goal to be reached by the robot with respect to a predefined origin.

MultiRobotConnectors Requested Interface

One of the key features of the robotic platform is the ability to create a bidirectional communication between the robot and the smart environment. The GATEKEEPER platform needs to provide interfaces to access data streams generated by environmental sensors and personal devices. This can be used by the robot to enhance its predictions, for example by correcting the estimate of the user activities by using data provided by a smart watch.

5 Components Interactions

In this section, the main components interactions are listed as sequence diagrams.

Each flow was formalized analysing user stories from different sources:

- D6.2 deliverable [2], that gives an overview of the overall functionalities requested by pilots;
- A draft[12] of D3.1 deliverable [1] on platform and user requirements.
- Each user story has been analysed to extract expected behaviours of the platform and, interacting with the pilot owners, alternative solutions have been identified to reflect any difference in the integration strategy of each pilot.
- The interfaces detailed in the section above have been used as the "building blocks" for the realization of the sequence diagrams.
- For each flow, together with the textual description and the diagram describing it, there is the list of the user stories that are related. There is also pointer to alternative flows, if there are any, and an explanation of the need for an alternative.
- A summarization of the components involved in the data flows can be found in Appendix C.

Interactions that relate more directly to user stories are named GKPILOT_xx while base interactions supporting requirements at different levels are named GK_PLAT_xx.

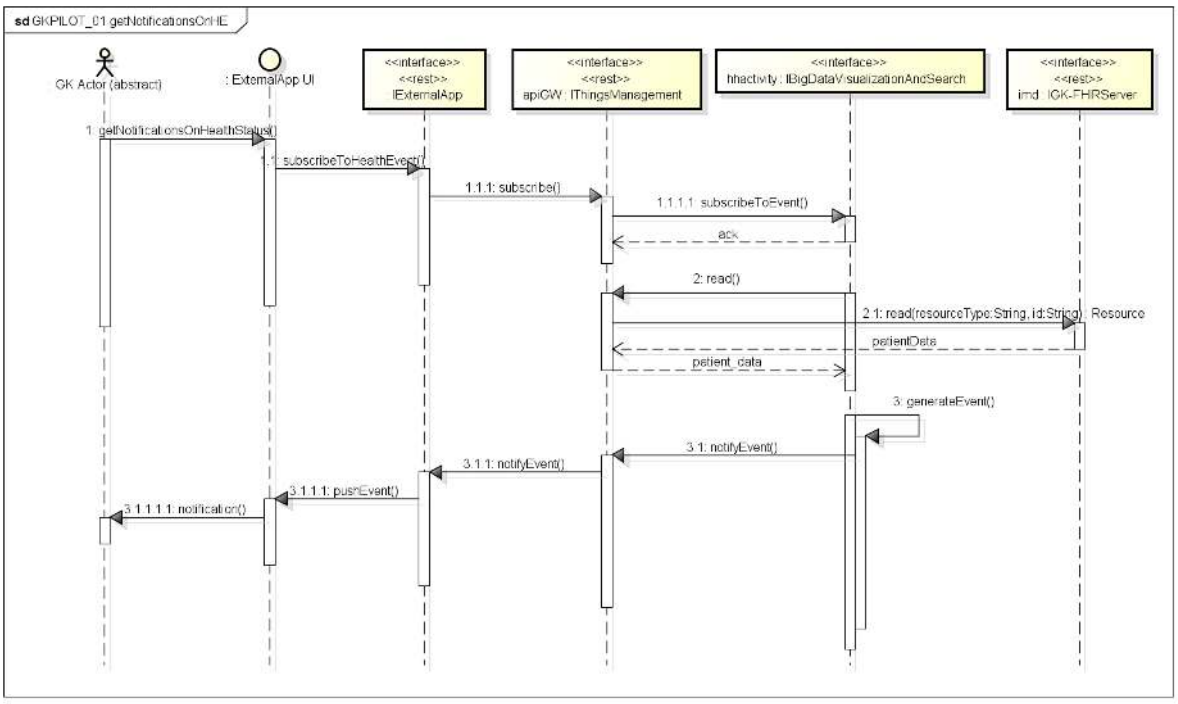
[12] the final version will be available at M10

5.1 GKPILOT_01

An AuthorizedActor subscribes to health events of a patient and receives notifications

This flow describes interaction between an Authorized user (e.g. GP, Nurse) and the Platform in order to monitor in remote mode the Patient.

User uses the External App UI (a thing registered to the platform) to get notifications on the Patient health status. ThingsManagementSystem forwards the requested query to the BigDataInfrastructure. The BigDataInfrastructure continuously receives events from the GK-FederationService and in case the requested conditions are met notifies the event back to the requesting Thing.
The notification is then pushed to the requesting actor as notifications/alarms.



ASSOCIATED USER STORIES

d62.13.03 – GP receives notifications [Puglia, Basque Country, Lodz]

d62.13.04 – Nurse receives notifications [Puglia, Basque Country]

d62.13.05 – Contact Centre receives notifications [Aragòn, Basque Country]

ASSOCIATED PLATFORM REQUIREMENTS

Req_UI_03- System connects and provides information to Social Services and primary healthcare for better intervention

Req_UI_04 - System connects and provides information to healthcare professionals

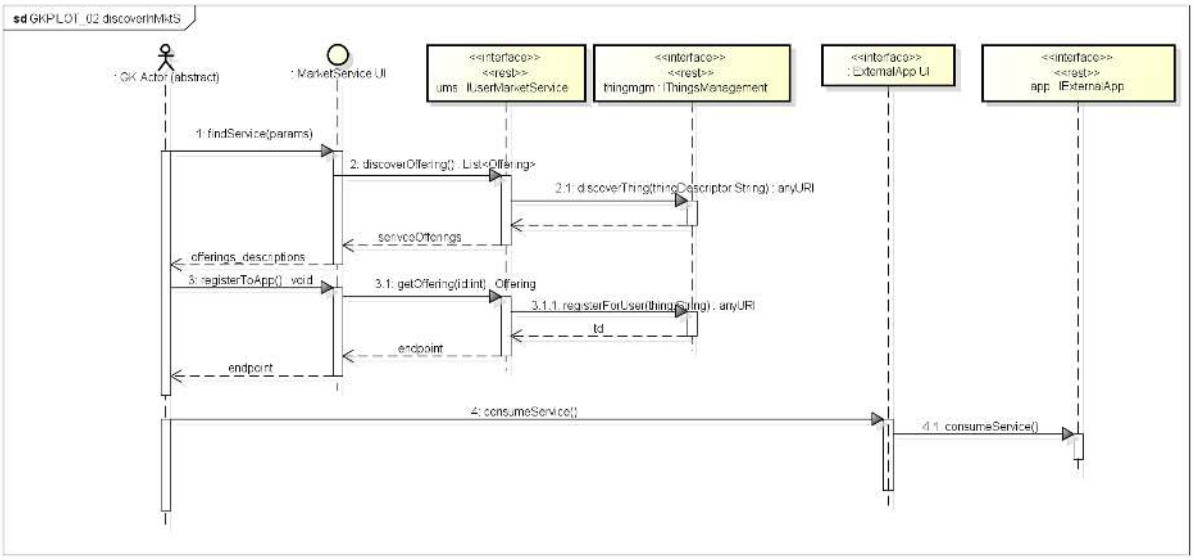
Req_AI_15- System provides periodic reports and historical data about assisted elderly activity

5.2 GKPILOT_02

A GK actor browses the MarketService catalogue to find a suitable solution and obtains the endpoint of the service

This flow describes interaction between a GK actor and the Platform to discover a solution for his needs (e.g. to monitor a Patient).

The User searches for a solution through MarketService UI that forwards to ThingsManagementSystem in order to find it. An endpoint of the service (linked to the suitable solution) is returned to user. Then he can register to use this service (using ExternalApp UI)



ASSOCIATED USER STORIES

d62.14.01 – GP accesses MarketPlace [Aragòn, Puglia]

d62.14.10 – MedTech Company promotes closed platform [Attica and Central Greece]

ASSOCIATER PLATFORM REQUIREMENTS

Req_UI_19 Users must be able to choose the services, service components and applications they will use.

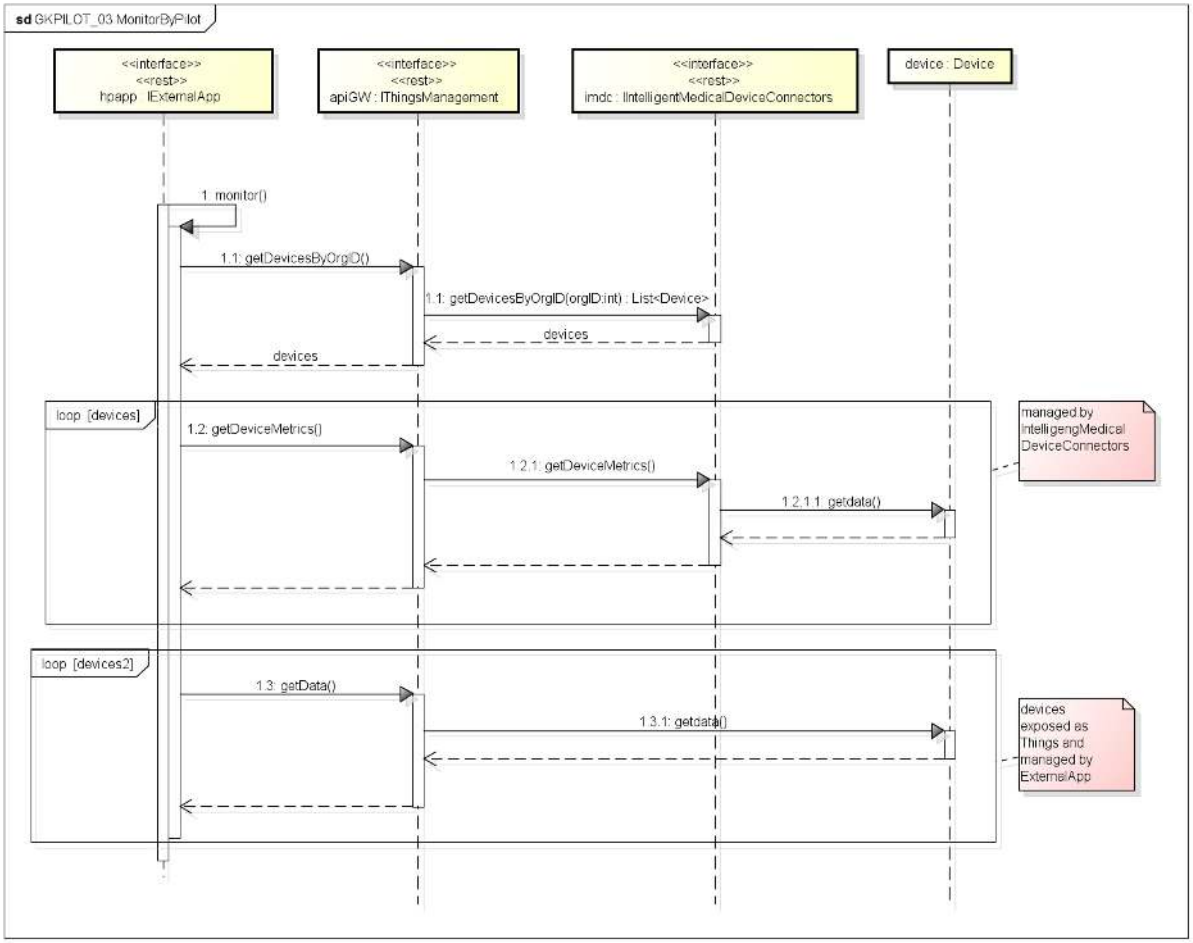
5.3 GKPILOT_03

Pilot app monitors data directly getting metrics from devices

This flow describes interaction between app belonging to Pilot and smart devices within the Platform in order to directly monitor the data.

In a first loop ThingsManagementSystem processes and forwards the request to the IntelligentMedicalDeviceConnectors component. In this case, metrics of device are collected and then sent back to the requesting service.

The second loop refers to devices that are registered directly as Things and the collection of data is managed by the External service.



ASSOCIATED USER STORIES

d62.13.06 - GP consults Dashboard [Attica and Central Greece, Basque Country, Milton Keynes, Lodz]

d62.13.09 - Caregiver consults Dashboard [Attica and Central Greece, Saxony]

ASSOCIATED PLATFORM REQUIREMENTS

Req_UI_09 Allow the configuration of activity monitoring by caregivers (formal and informal)

Req_UI_23 Allow the configuration of activity monitoring by caregivers (formal and informal)

ALTERNATIVE FLOWS

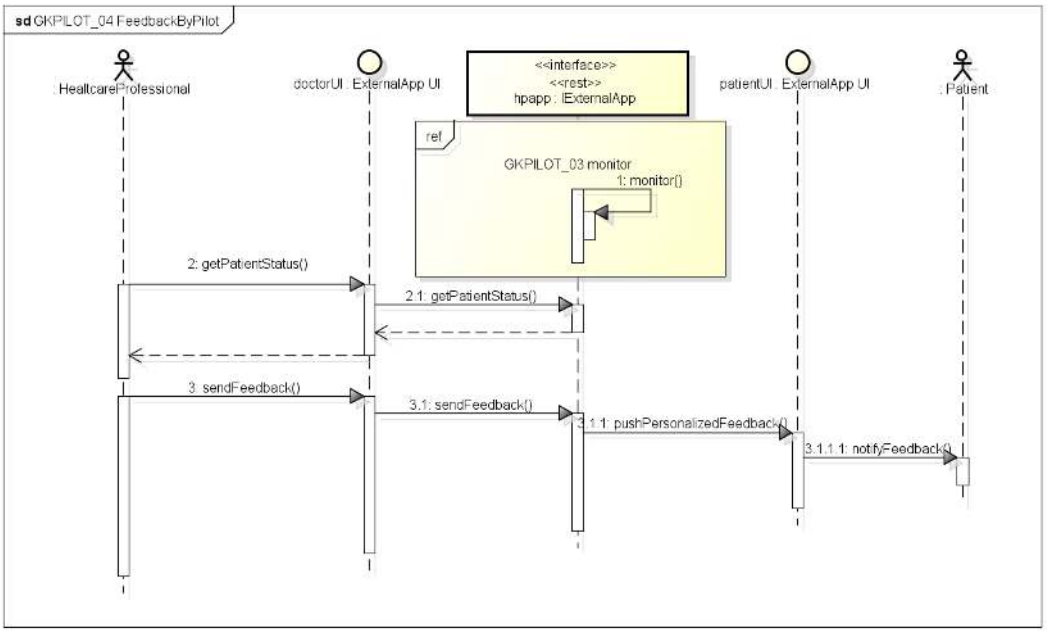
This flow represents a low integration to the GK platform, where most of the computation is kept in the external applications. A use case that has a similar integration approach is Attica and Central Greece. An alternative flow that involves a deeper integration with the platform is described in GKPLAT_04.

5.4 GKPILOT_04

HealtcareProfessional uses pilot app to send feedback to patient

This flow describes interaction between Reference HealthCare Professional and ExternalApp (i.e. Pilot app) in order to notify feedback to Patient.

The HC Professional checks for patient status through the UI that gets this information by the user story described above (GKPILOT_03). Once information has been received, the RHP sends feedback to the Patient that is conveyed in the patient UI.



ASSOCIATED USER STORIES

d62.13.02 GP prescribes Digital Coach

ASSOCIATED PLATFORM REQUIREMENTS

Req_UI_06 Help Desk services will be implemented to support / inform final users and informal caregivers.

Req_AP_08 Notification system on the watch must be configurable to deliver medication reminders, and able to be programmed remotely

ALTERNATIVE FLOWS

The data that allow the HC Professionals to formulate their feedback can be the result of integration with the platform as in GKPLAT_04. Also, digital coaching can be driven by AI Algorithms of integrated Dynamic Interventions services (see GKPILOT_09)

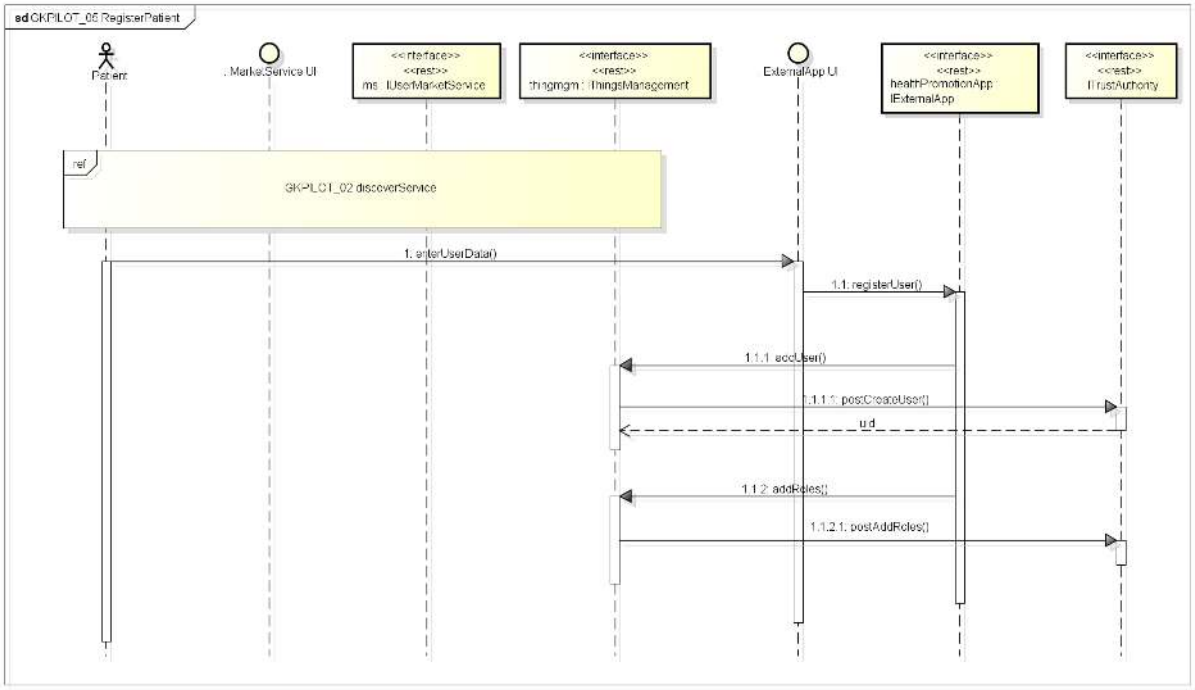
5.5 GKPILOT_05

A Citizen registers his or her identity to the Thing s/he bought through the MarketService

This flow describes interaction between a Citizen and the GK Platform in order to allow access to register to the health promotion app.

The User through the MarketService UI searches for the specific Service in which is interested. ThingsManagementSystem provides to discover the corresponding Thing and through the MarketService returns the endpoint of the service.

Then the logged user (login in the ExternalApp UI) registers his/her credentials using the ExternalApp related to the health promotion. Then ThingsManagementSystem provides to finalize this operation and to assign a role to patient (in terms of permissions) to access the application.



ASSOCIATED USER STORIES

d62.14.10 – MedTech Company promotes closed platform [Attica and Central Greece]

ASSOCIATED PLATFORM REQUIREMENTS

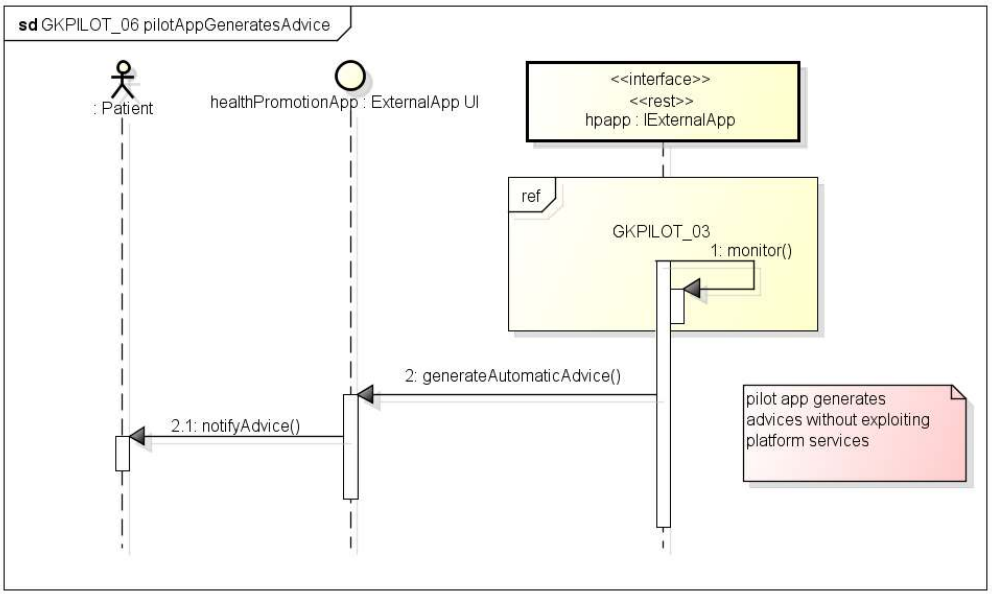
Req_PS_02 The use of information will be closely linked to identification mode - enabled users will only have access to the information they have been enabled for.

5.6 GKPILOT_06

Pilot app generates advices without exploiting platform services

This flow describes a direct interaction between a Patient and the pilot app.

Notifications are generated and sent to the patient without the pilot app interacting with the services offered by the platform (s. description of second loop in GKPILOT_03).



ASSOCIATED USER STORIES

d62.13.02 – GP prescribes Digital Coach [Aragòn, Puglia, Attica and Central Greece, Cyprus, Basque Country, Lodz]

ASSOCIATED PLATFORM REQUIREMENTS

Req_AI_08 System sends alerts to remind the elderly daily habits / routines (decided by elderly and also by caregivers)

ALTERNATIVE FLOWS

The recommendation can be formulated by data processing of Integrated Dynamic intervention services. (See GKPILOT_09)

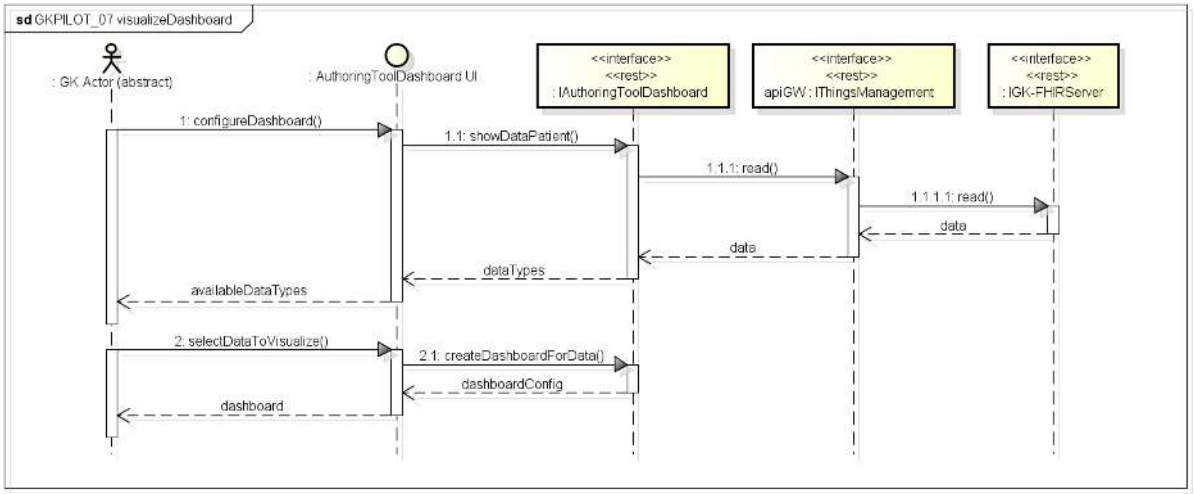
5.7 GKPILOT_07

A Pilot actor configures the dashboard for patient's data visualization.

This flow describes interaction between a Pilot actor (e.g. reference healthcare professional or Citizen) in order to visualize patient's data within a dashboard.

Through the authorization tool the request of the patient's data is forwarded to the GK-FHIR Server which provides the required data.

Then the user selects the data he wants to visualize and sends the request to the AuthoringTool for the configuration of the dashboard. The Dashboard is displayed.



ASSOCIATED USER STORIES

d62.13.06 - GP consults Dashboard [Aragon, Puglia, Attica and Central Greece, Basque Country, Milton Keynes, Lodz]

d62.13.07 - Nurse consults Dashboard [Aragon, Puglia, Basque Country, Milton Keynes, Lodz]

d62.13.08 - Contact Centre consults Dashboard [Aragon]

d62.13.09 - Caregiver consults Dashboard [Puglia, Attica and Central Greece]

d62.13.10 - Pharmacist Consults Dashboard [Puglia]

ASSOCIATED PLATFORM REQUIREMENTS

Req_AP_03 The solution shall provide a dashboard displaying the user's personal data

Req_AP_06 Participants should be able to have an overview of the data gathered and trends, changes etc. through a web portal

Req_AI_01 Generation of reports on routines to detect trends on the assisted person behaviour

Req_AI_11 Trend information may require graphic representations.

ALTERNATIVE FLOWS

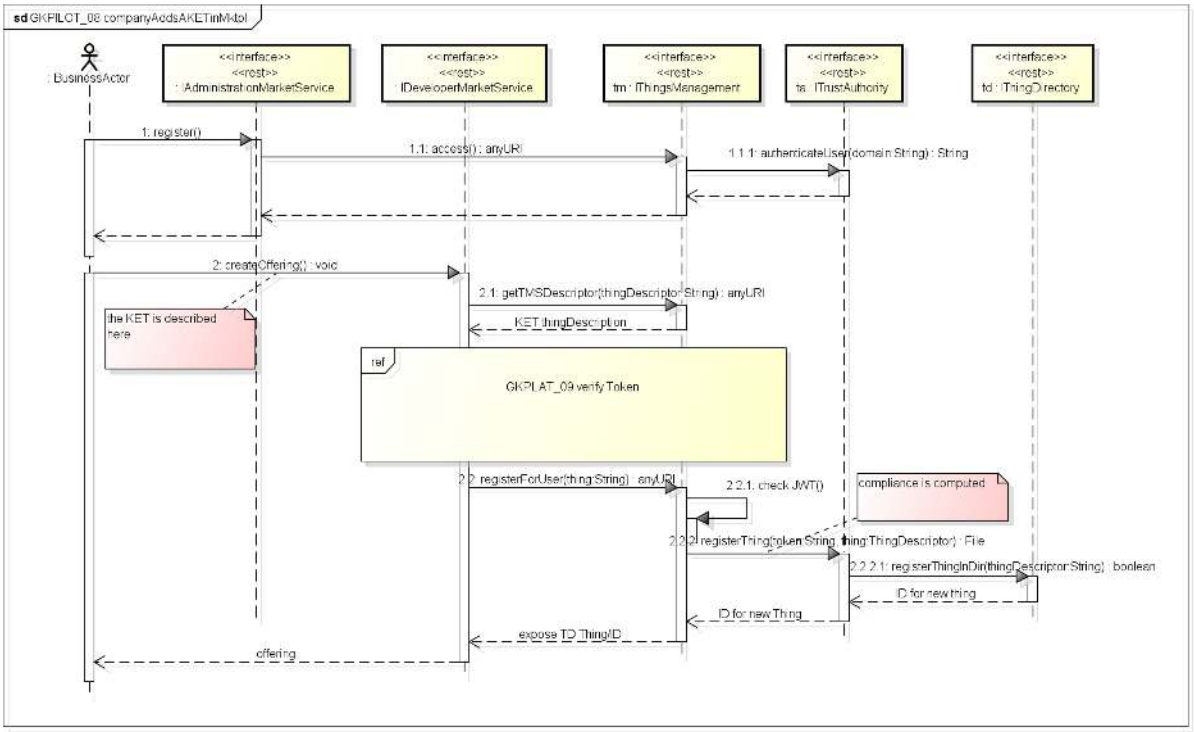
Dashboards are managed directly by the External Application (GKPLAT_06)

5.8 GKPILOT_08

A BusinessActor adds a new KET in the MarketService

This flow describes how a Business actor can create an Offering in the market service to promote a closed platform or KET.

The flow highlights also the certification process that is triggered by the registration of the Thing in the Platform.



ASSOCIATED USER STORIES

d62.14.09 - Medtech Company promotes modular KET [Aragòn, Puglia, Saxony]

d62.14.11 - Medtech Company integrates KET in existing platform [Puglia]

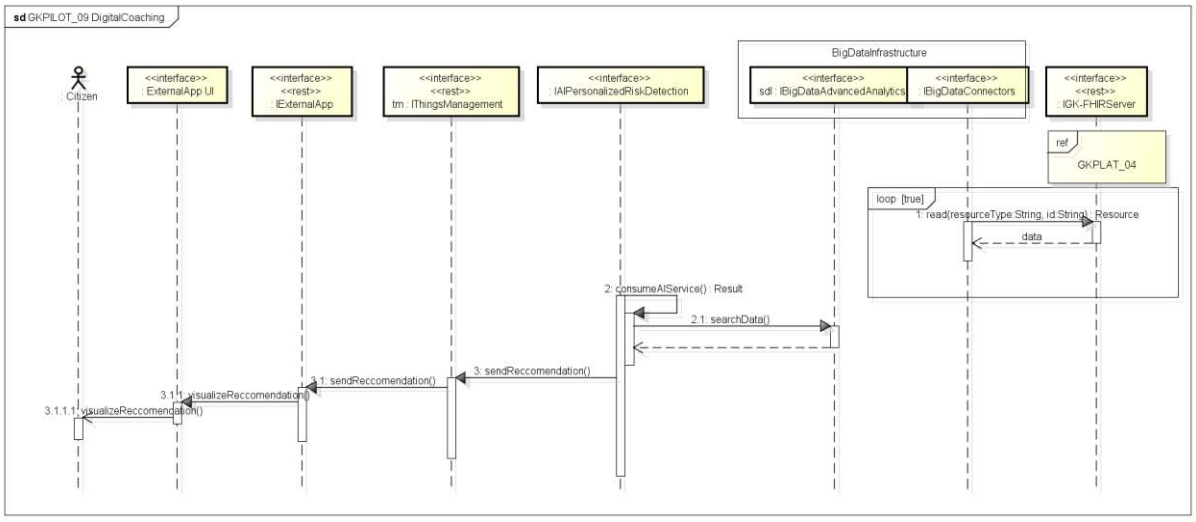
ASSOCIATED PLATFORM REQUIREMENTS

Req_DSP_02 The IoT platform should allow its own extension with other components (new sensors, new connectors etc) without a need for adaptation

5.9 GKPILOT_09

A patient gets automatic recommendations

This flow describes how a Citizen (Patient or caregiver) can get automatic recommendations generated by computations of the platform AI Services.



ASSOCIATED USER STORIES

d62.13.02 - GP prescribes Digital Coach [Aragòn, Puglia, Attica and Central Greece, Cyprus, Basque Country, Lodz]

ASSOCIATED PLATFORM REQUIREMENTS

Req_UI_06 Help Desk services will be implemented to support / inform final users and informal caregivers.

Req_AP_08 Notification system on the watch must be configurable to deliver medication reminders, and able to be programmed remotely

ALTERNATIVE FLOWS

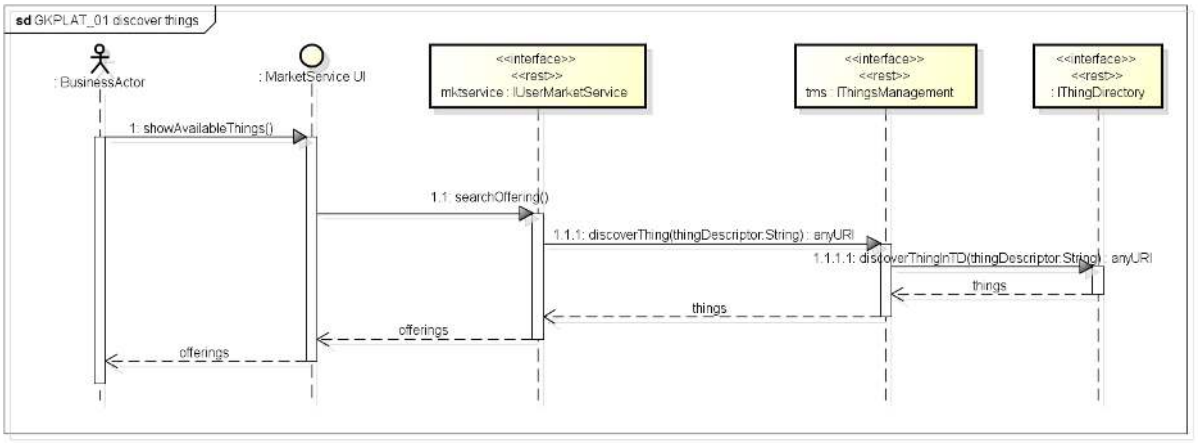
GKPILOT_04 (recommendations are generated by the External App (Pilot App) accessing data through the platform.

5.10 GKPLAT_01

A Business actor browses the MarketPlace catalogue using specific constraints

This flow describes interaction between a Business actor and the MarketPlace.

A Business actor asks through MarketPlace User Interface for available Things within the MarketPlace Catalogue. The ThingsManagementSystem provides to get requested Things.



ASSOCIATED USER STORIES

d62.13.01 – GP prescribes Digital Patient Monitoring [Aragòn, Puglia, Attica and Central Greece, Cyprus, Saxony, Basque Country, Lodz]

d62.13.02 GP prescribes Digital Coach [Aragòn, Puglia, Attica and Central Greece, Cyprus, Basque Country, Lodz]

d62.14.01 – GP accesses MarketPlace [Aragòn, Puglia]

d62.14.12 – Hospital Clinician recommends Big Data Analytics Algorithm [Puglia]

d62.14.14 – Caregiver recommends platform [Milton Keynes]

ASSOCIATED PLATFORM REQUIREMENTS

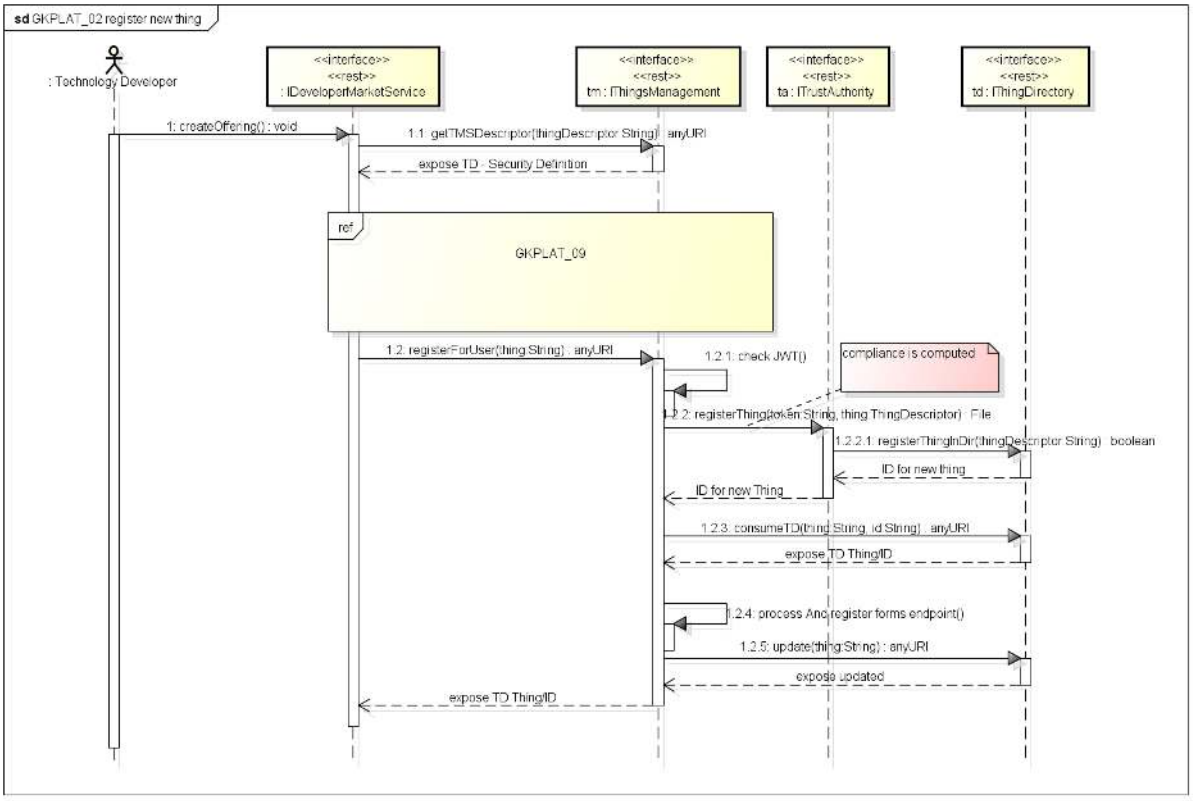
None

5.11 GKPLAT_02

An Authorized user / service registers a new GK Thing in the TMS through the Market Service.

This flow describes the interaction between the user and the platform for the first time that a Thing is used.

Since in the first use case, it is needed to ask for an authorization and certify the component before registeding it in the ThingDirectory and then using it.



ASSOCIATED USER STORIES

d62.14.02 GP assesses regulatory compliance [Aragon, Puglia]

d62.14.08 Healthcare Provider assesses sustainability [Aragon]

ASSOCIATED PLATFORM REQUIREMENTS

None

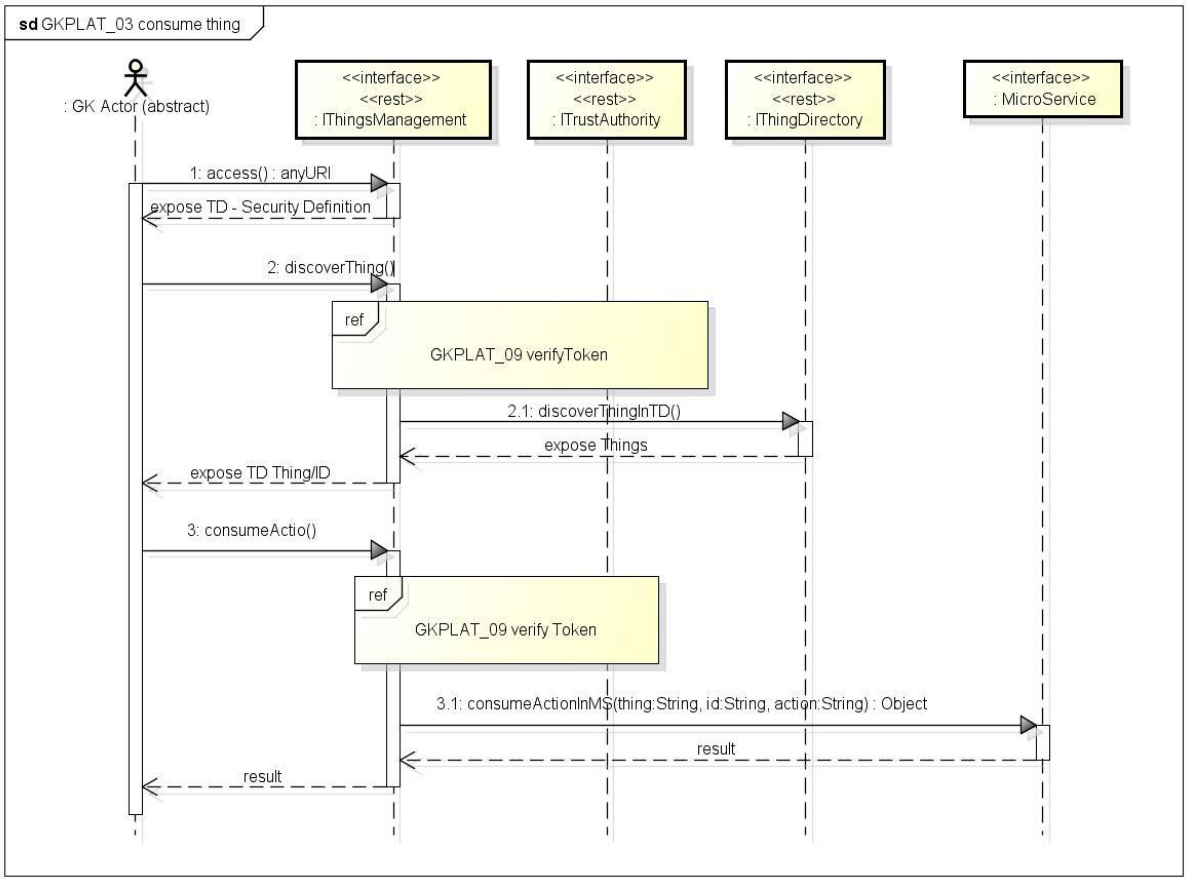
5.12 GKPLAT_03

An authorized user / service wants to find a Thing in the TMS to use it

This flow describes the interaction between an authorized user and the ThingsManagementSystem in order to search for a Thing and use it.

The ThingsManagementSystem discover the Thing in the ThingDirectory and then expose it. The user is able to consume the Thing sending a request to the TMS or MicroService.

The TrustAuthority is responsible for Security.



ASSOCIATED USER STORIES

d62.14.04 Healthcare Provider integrates Digital Patient Monitoring data in EHR [Aragon, Puglia]

d62.14.05 Healthcare Provider combines existing KETs available in GATEKEEPER Spaces [Aragon, Puglia, Basque Country]

d62.14.10 - Medtech Company promotes closed platform [Attica and Central Greece]

d62.14.11 - Medtech Company integrates KET in existing platform [Puglia]

d62.14.12 Hospital Clinician recommends Big Data Analytics Algorithm [Puglia]

ASSOCIATED PLATFORM REQUIREMENTS

Req_DSP_02 The IoT platform should allow its own extension with other components (new sensors, new connectors etc) without a need for adaptation

5.13 GKPLAT_04

A KET sends data to the platform

This flow describes interaction between KETs producing patient data and the GATEKEEPER Platform.

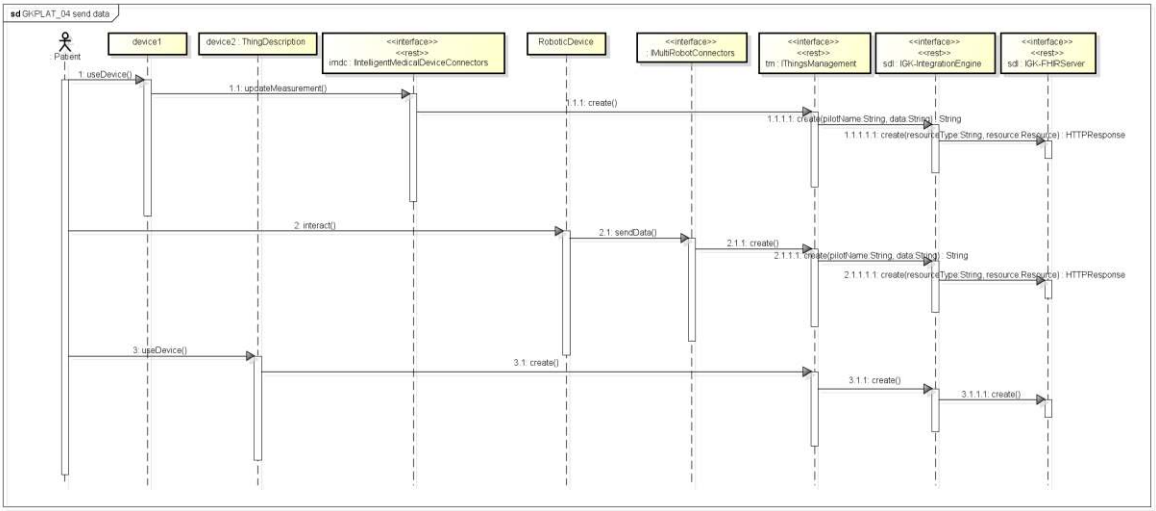
A user can use a variety of KETs (e.g. medical devices) without worrying on how they will send data to the platform.

KETs registered as Things can interact with the platform directly sending updated measurements.

KETs with no HTTP support or not registered as Things in the platform can send data (e.g. measurement) to a data collector (e.g. IntelligentMedicalDeviceConnectors) that will take care to collect all the measurements and forward them to the platform in a suitable format.

Other KETs (supported robots in particular) can send their data mediated by the MultiRobotConnectors.

ThingsManagementSystem will mediate the interaction between Things producing data and the GK-IntegrationEngine that provide a mapping to the GATEKEEPER FHIR profile and store incoming data in the GK FHIRServer.



ASSOCIATED USER STORIES

d62.13.01 – GP prescribes Digital Patient Monitoring [Aragòn, Puglia, Attica and Central Greece, Saxony, Basque Country, Lodz]

d62.14.05 – Healthcare Provider combines existing KETs available in GATEKEEPER Spaces [Aragòn, Puglia, Basque Country]

ASSOCIATED PLATFORM REQUIREMENTS

Req_DS_01 The data produced by IoT sensors must be stored in local premises or in the HPE Cloud

Req_DSP_03 The system must be able to acquire body weight measurements collected by smart devices

Req_DSP_04 System collects information useful for the early detection of pathologies (insomnia, physical inactivity through wearables…)

Req_DSP_05 Connection with medical devices to measure vital signs (blood pressure, glucose…)

Req_DSP_07 User monitoring must be performed using IoT sensors able to provide their measurements via an Internet or a Bluetooth connection.

Req_DSP_08 Data collection must be as "transparent" to the user as possible (i.e. no need for user intervention). The IoT sensors should not require the user to change their daily living habits.

Req_AI_06 The data collected by IoT sensors must be processed to generate information about the well-being and health status of the user

ALTERNATIVE FLOWS

Data are sent and managed by registered External Applications (GKPLAT_03)

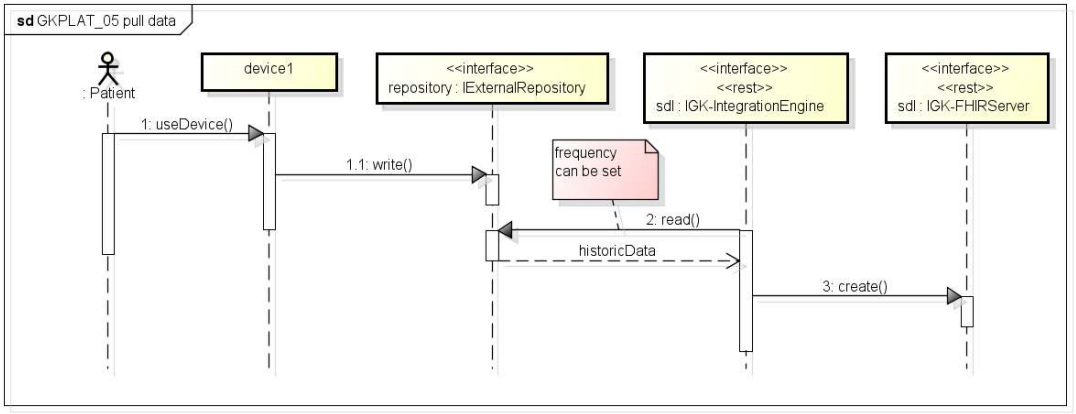
5.14 GKPLAT_05

The GK Integration Engine pulls data from configured external repos to integrate them in the platform

This flow describes interaction between the GK-IntegrationEngine and the GATEKEEPER Platform.

A user uses a KET (e.g. medical device) and it sends data (e.g. measurement) to an external repository.

GK-IntegrationEngine reads (at regular intervals) data stored and sends it to GK-FHIRServer



ASSOCIATED USER STORIES

d62.14.03 – Healthcare Provider make EHR data accessible within GATEKEEPER Spaces [Aragòn, Puglia, Basque Country]

d62.14.04 - Healthcare Provider integrates Digital Patient Monitoring data in EHR [Aragòn, Puglia]

d62.14.06 – Healthcare Provider avoids overlaps with legal KETs [Aragòn, Puglia, Basque Country]

ASSOCIATED PLATFORM REQUIREMENTS

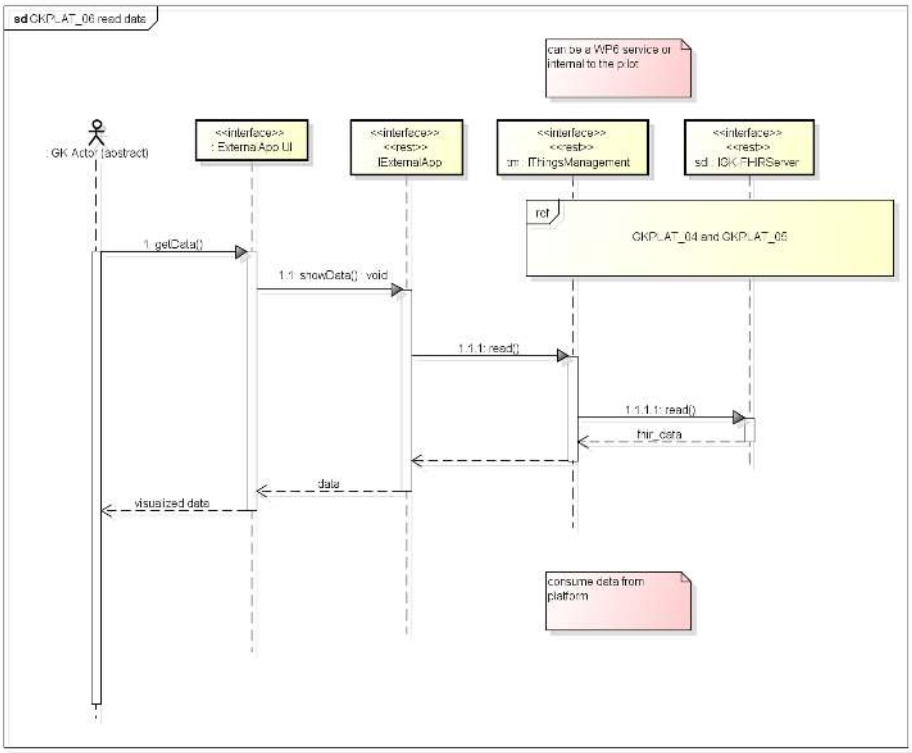
Req_DA_02 Data should link to GP record so can view activity, plus gather info on hospital admissions etc. Will need to fit with services provided by EMIS and TPP to achieve

5.15 GKPLAT_06

An Authorized user reads federated data from the Platform

This flow describes interaction between an Authorized user and the Platform to read a set of data stored in the Platform.

Authorized user makes a request through an UI for specific data to visualize. The request is mediated by the ThingsManagementSystem that checks authorization and allows the access to the data by reading the GK-FHIRServer.



ASSOCIATED USER STORIES

d62.13.11 Specialist Clinician evaluates Digital Patient Monitoring reports [Puglia]

d62.13.12 Hospital Clinician evaluates Digital Patient Monitoring reports [Puglia]

ASSOCIATED PLATFORM REQUIREMENTS

Req_UI_03 System connects and provides information to Social Services and primary healthcare for better intervention

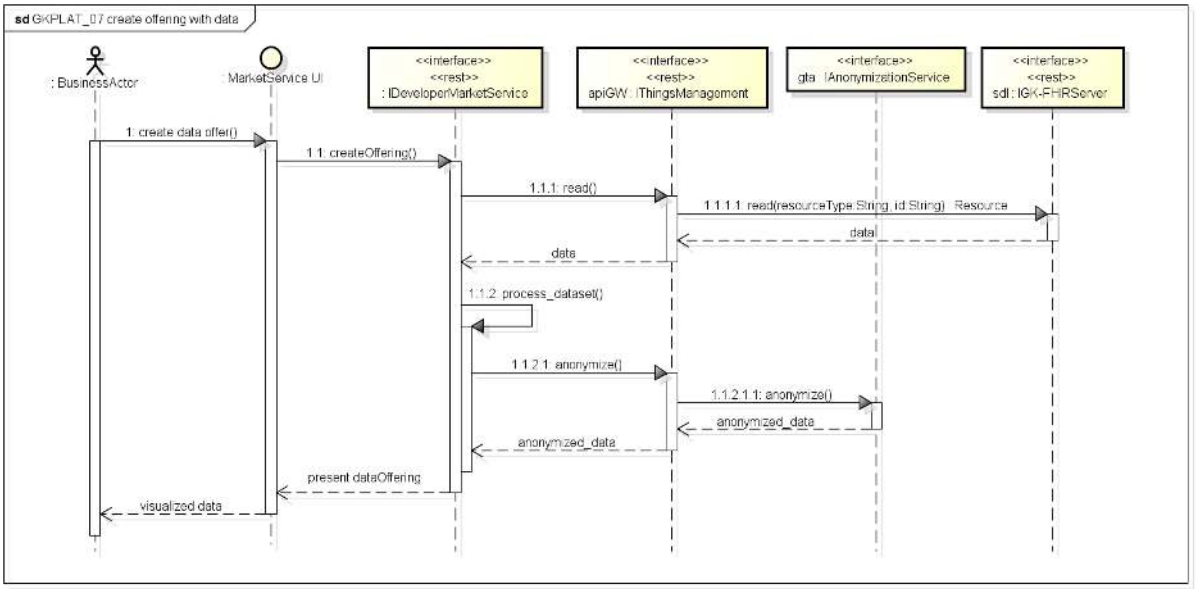
Req_UI_04 System connects and provides information to healthcare professionals

Req_AI_06 The data collected by IoT sensors must be processed to generate information about the well-being and health status of the user

5.16 GKPLAT_07

Data series can be aggregated / anonymised in order to be included in an offering in the MarketService

This flow describes interaction the flow of creation of purely informative offerings. In this case the offering is on anonymised data series. The Business Actor (Developer or Company) will have the possibility to choose the dataset he/she wants to include, then the MarketService will gather the data and anonymise them before returning them to the authorized developer to detail the commercial offering.



ASSOCIATED USER STORIES

None

ASSOCIATED PLATFORM REQUIREMENTS

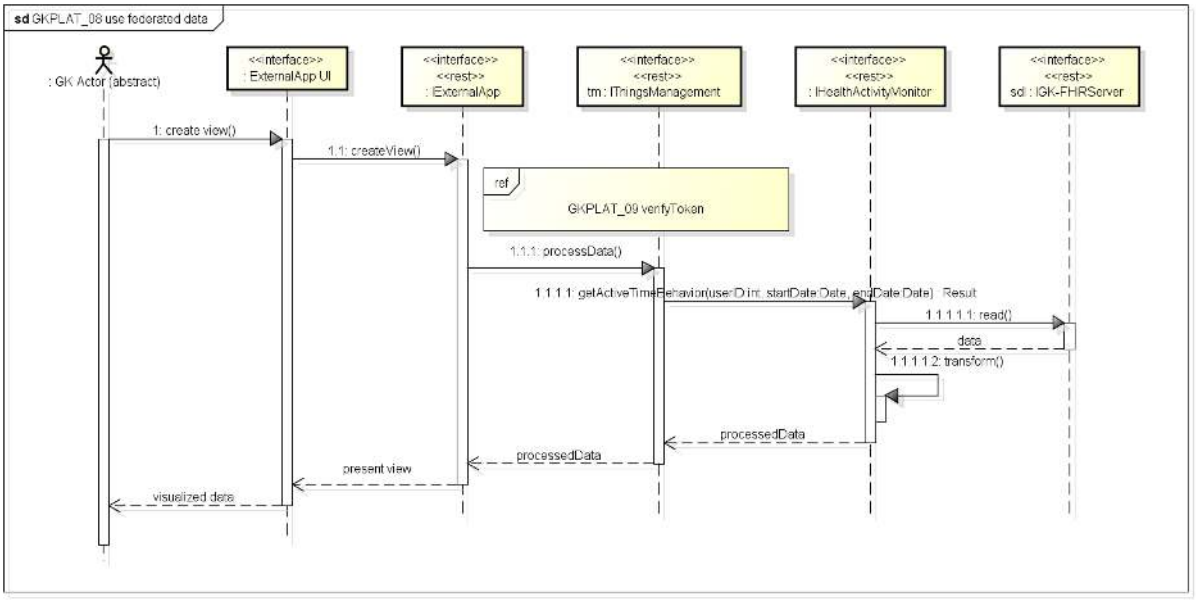
Req_AI_18 System products (data, analysis, messages, highlights) could be purely informative.

5.17 GKPLAT_08

Health data are processed by Processing Services to create views and trends to offer to Authorized users

This flow describes interaction between an authorized GK-Actor (HealthCare Professional or Citizen) and the platform in order to retrieve processed data for a patient or a set of patients.

The process starts with a request through an ExternalApp (e.g. Pilot App). The ThingsManagementSystem resolves the request and verifies the authorization using services of the GTA (see GKPLAT_09). In this case the action is then forwarded to the Home and Health Activity Monitor to compute the active time of a patient. The service takes care to retrieve the data from the GK FHIR server and transform them in the required format and then send them back to the user to be visualized.



ASSOCIATED USER STORIES

d62.13.06 – GP consults Dashboard [Aragòn, Puglia, Basque Country, Milton Keynes, Lodz]

d62.13.07 - Nurse consults Dashboard [Aragòn, Puglia, Basque Country, Lodz]

d62.13.08 – Contact Centre consults Dashboard [Aragòn]

d62.13.09 - Caregiver consults Dashboard [Puglia, Saxony]

d62.13.10 – Pharmacist consults Dashboard [Puglia]

d62.13.11 - Specialist Clinician evaluates Digital Patient Monitoring reports [Saxony, Basque Country]

d62.13.12 - Hospital Clinician evaluates Digital Patient Monitoring reports [Puglia]

ASSOCIATED PLATFORM REQUIREMENTS

Req_AI_01 Generation of reports on routines to detect trends on the assisted person behaviour

Req_AI_06 The data collected by IoT sensors must be processed to generate information about the well-being and health status of the user

Req_AI_09 System suggests to the elderly to do physical activity or other activities

Req_AI_14 The solution shall detect if the assisted is in bed or chair and estimate the time spent in bed or chair

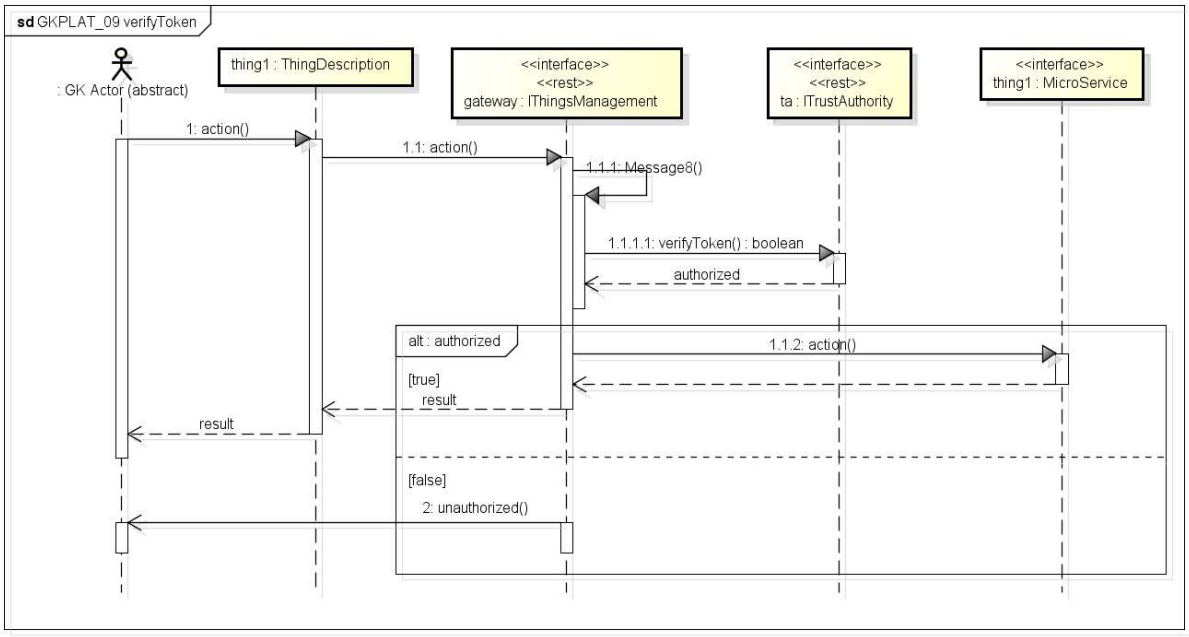
ALTERNATIVE FLOWS

GKPLAT_12 (use AI Services to process data)

5.18 GKPLAT_09

A GK Actor sends a request to the platform and its authorization is verified

This is the detailed flow that shows how authorization is checked in the GATEKEEPER platform, when a registered actor requests to perform an action on a registered Thing.



ASSOCIATED USER STORIES

None

ASSOCIATED PLATFORM REQUIREMENTS

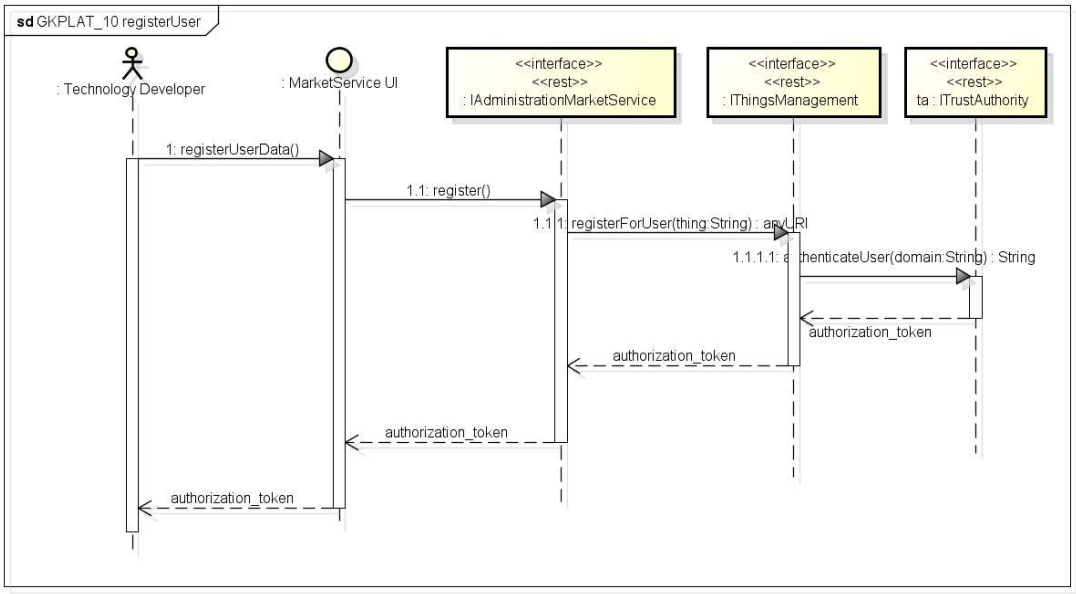
Req_PS_02 The use of information will be closely linked to identification mode - enabled users will only have access to the information they have been enabled for.

Req_DSP_10 The solution shall communicate only with authenticated devices

5.19 GKPLAT_10

A user registers to platform

This flow represents the case when a Developer registers to the platform and is given an authorization token to be used in the services he/she will develop



ASSOCIATED USER STORIES

None

ASSOCIATED PLATFORM REQUIREMENTS

Req_PS_01 Security of data and information generated (prevention from unauthorised access)

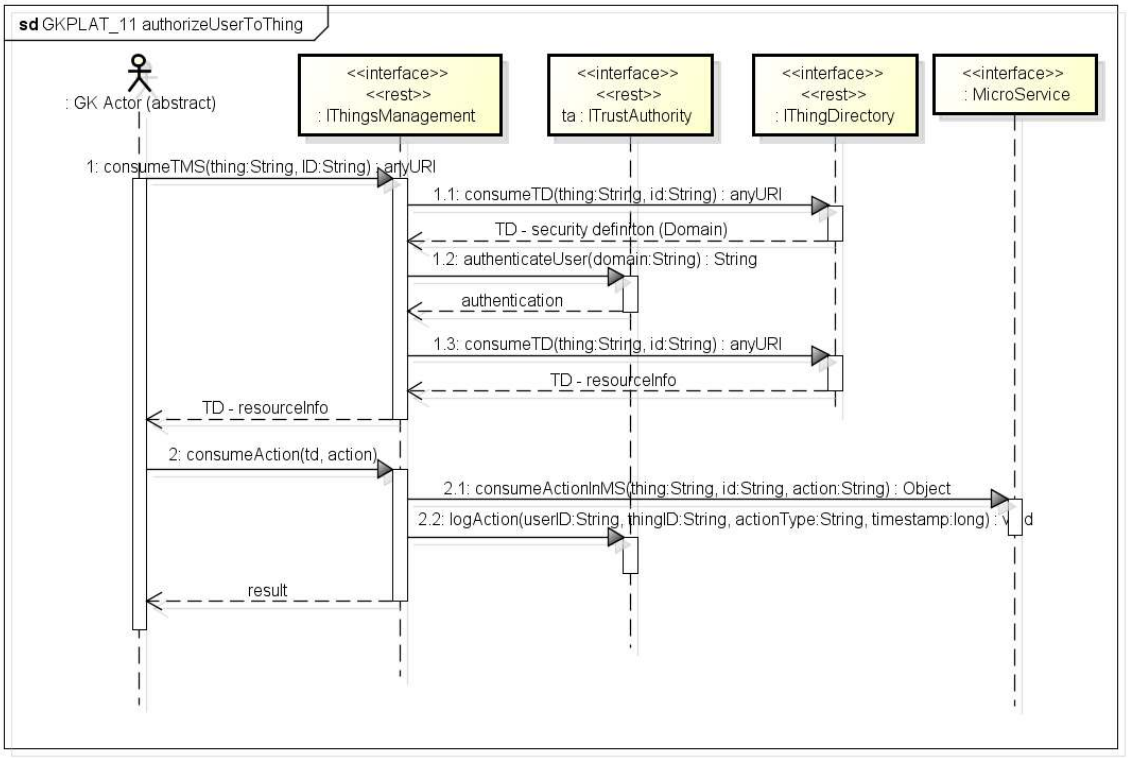
Req_PS_07 The system should be managed by a User Management module to control access in UIs and services

5.20GKPLAT_11

A User is authorized to accesses a Thing in the platform

This flow shows the process of authorizing a used to access a specific Thing. The flow highlights the mechanism of associating an authorization token to a Thing Description in the Trust Authority

First the User requests the ThingDescription to the TMS, using its authorization Token. The TMS delegates the authentication of the user to the GTA, and requests the ThingDescription to the ThingsDirectory, that is returned to the User. The user is now authorized to consume actions on the Thing, and every access to it is logged in the GTA.



ASSOCIATED USER STORIES

None

ASSOCIATED PLATFORM REQUIREMENTS

Req_PS_02 The use of information will be closely linked to identification mode - enabled users will only have access to the information they have been enabled for.

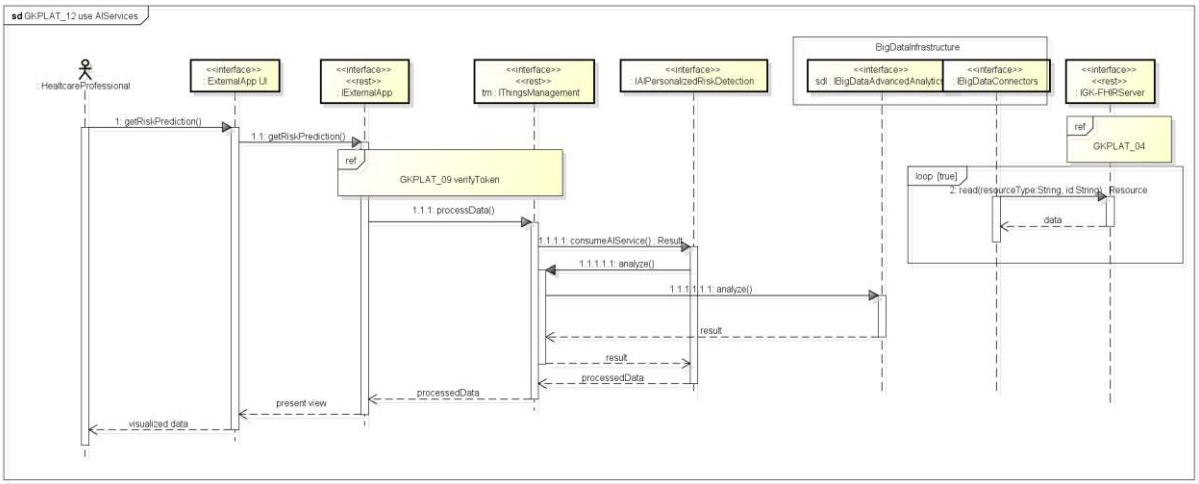
Req_PS_07 The system should be managed by a User Management module to control access in UIs and services

Req_DSP_10 The solution shall communicate only with authenticated devices

5.21 GKPLAT_12

A Healthcare professional requests through an External App (e.g. Pilot app) the platform to compute a risk prediction for a patient exploiting the AI services

The flow shows the interaction between External Apps and the Personalized Risk detection service. To provide its services this component exploits the functionalities and tools provided by the BigDataInfrastructure, that gets data from the GK Platform FHIR server.



ASSOCIATED USER STORIES

d62.14.12 Hospital Clinician recommends Big Data Analytics Algorithm [Puglia]

ASSOCIATED PLATFORM REQUIREMENTS

Req_AI_05 Alarm to alert informal caregivers or emergence services of a risky situation stemming from a period of inactivity

Req_AI_13 Algorithm on falls identification decides where to send alerts based on risk and information given on sign up -informal carers, services etc.

6 Conclusions

The first design of the GATEKEEPER architecture lays the foundation for the GATEKEEPER platform components integration. Basing its principles in the Web of Things reference architecture, it addresses pilot requirements in deliverable D6.2, and a first set of functional requirements defined in an early version of deliverable D3.1 (also informed by user requirements by D2.3).

The role of each of the primary components provided by WP4 and WP5 has been identified, as long as the component interactions among them and with external services. The architecture design provides to the reader an overall view of the GATEKEEPER components and their interfaces, together with an early definition of the hardware requirements and the infrastructure that is being set up to allow the correct operation of the system.

This architecture will serve as an instrument for the supervision of the design and implementation done in WP4 and WP5 to ensure the compliance of components with the reference architecture and to smooth the component integration.

This architecture will be improved and better defined in the next months. A second version of this deliverable is planned at M18 (8 months from now) to report the new resulting architecture.

7 References

[1] GATEKEEPER Consortium, Deliverable D3.1 - Functional and technical requirements of GATEKEEPER platform [due September, 2020]
[2] GATEKEEPER Consortium, D6.2 - Early detection and interventions operational planning [March, 2020]
[3] GATEKEEPER Consortium, D2.3 - User Requirements and Taxonomy [June, 2020]
[4] Guinard, Dominique; Vlad, Trifa (2015). Building the Web of Things. Manning. ISBN 9781617292682.
[5] https://webofthings.org/2017/04/08/what-is-the-web-of-things/
[6] https://www.computer.org/csdl/magazine/cd/2018/04/mcd2018040012/13rRUy0ZzUF
[7] https://publications.csiro.au/rpr/pub?pid=csiro:EP189892
[8] https://www.w3.org/blog/wotig/2017/01/13/web-thing-model-member-submission/
[9] https://www.w3.org/TR/wot-architecture/#sec-building-blocks
[10] https://json-ld.org/
[11] https://w3c.github.io/wot-thing-description/
[12] Li, W., Tropea, G., Abid, A., Detti, A., & Le Gall, F. (2019, June). Review of Standard Ontologies for the Web of Things. In 2019 Global IoT Summit (GIoTS) (pp. 1-6). IEEE
[13] https://json-ld.org/
[14] http://hl7.org/fhir/summary.html
[15] Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine [2000]
[16] Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, [2017], pp. 557-564
[17] https://saref.etsi.org/
[18] https://www.openapis.org/
[19] https://www.internationaldataspaces.org/

Appendix A User Stories

The pilot use case specifications were collected from pilots (WP6) by extending (with further details) the "#5 reference use cases" included in DoA (section 1.3.4.3) and using as basis the user stories listed in D6.2 [2] in tables 13 and 14.

Such user stories have been analysed and high-level "common" requirements for the platform have been extracted (as agile user stories) and described in section 5 in the form of components interactions. An overview of the mapping is summarized in the table below.

User story ref	User story Title	Interaction IDs	Applicable Pilots
AS IS			
d62.13.01	GP prescribes Digital Patient Monitoring	GKPLAT_01, GKPLAT_04	Aragón, Basque Country, Cyprus, Attica and Central Greece, Puglia, Lodz, Saxony
d62.13.02	GP prescribes Digital Coach	GKPILOT_04, GKPILOT_06	Attica and Central Greece
		GKPLAT_01, GKPILOT_09	Aragón, Basque Country, Cyprus, Puglia, Lodz, Saxony
d62.13.03	GP receives notifications	GKPILOT_01	Basque Country, Puglia, Lodz
d62.13.04	Nurse receives notifications	GKPILOT_01	Basque Country, Puglia
d62.13.05	Contact Centre receives notifications	GKPILOT_01	Aragón, Basque Country
d62.13.06	GP consults Dashboard	GKPILOT_07, GKPLAT_08	Aragón, Basque Country, Puglia, Lodz
		GKPILOT_03, GKPILOT_07	Attica and Central Greece
d62.13.07	Nurse consults Dashboard	GKPILOT_07, GKPLAT_08	Aragón, Basque Country, Puglia, Lodz
d62.13.08	Contact Centre consults Dashboard	GKPILOT_07, GKPLAT_08	Aragón
d62.13.09	Caregiver consults Dashboard	GKPILOT_07, GKPLAT_08	Puglia, Saxony
		GKPILOT_03, GKPILOT_07	Attica and Central Greece
d62.13.10	Pharmacist Consults Dashboard	GKPILOT_07, GKPLAT_08	Puglia
d62.13.11	Specialist Clinician evaluates Digital Patient Monitoring reports	GKPLAT_06, GKPLAT_08	Basque Country, Saxony

User story ref	User story Title	Interaction IDs	Applicable Pilots
d62.13.12	Hospital Clinician evaluates Digital Patient Monitoring reports	GKPLAT_06, GKPLAT_08	Puglia
d62.13.13	Informal Caregiver trained to recognize risks	-	Basque Country
d62.13.14	Elderly Citizen trained to recognize risks	-	Basque Country
d62.13.15	Caregiver supports Elderly Citizen with KET	-	Aragón, Basque Country
TOBE			
d62.14.01	GP accesses Marketplace	GKPILOT_02, GKPLAT_01	Aragón, Puglia
d62.14.02	GP assesses regulatory compliance	GKPLAT_02	Aragón, Puglia
d62.14.03	Healthcare Provider make EHR data accessible within GATEKEEPER Spaces	GKPLAT_05	Aragón, Basque Country, Puglia
d62.14.04	Healthcare Provider integrates Digital Patient Monitoring data in EHR	GKPLAT_03, GKPLAT_05	Aragón, Basque Country, Puglia
d62.14.05	Healthcare Provider combines existing KETs available in GATEKEEPER Spaces	GKPLAT_03, GKPLAT_04	Aragón, Basque Country, Puglia
d62.14.06	Healthcare Provider avoids overlaps with legacy KETs	GKPLAT_05	Aragón, Basque Country, Puglia
d62.14.07	Healthcare Provider compiles DMP		Aragón, Basque Country
d62.14.08	Healthcare Provider assesses sustainability	GKPLAT_02	Aragón, Basque Country
d62.14.09	Medtech Company promotes modular KET	GKPILOT_08	Aragón, Basque Country, Puglia
d62.14.10	Medtech Company promotes closed platform	GKPILOT_02, GKPILOT_05, GKPLAT_03	Attica and Central Greece
d62.14.11	Medtech Company integrates KET in existing platform	GKPILOT_08, GKPLAT_03	Puglia
d62.14.12	Hospital Clinician recommends Big Data Analytics Algorithm	GKPLAT_01, GKPLAT_03, GKPLAT_12	Puglia
d62.14.13	Heath Policy Body conducts analysis		Aragón, Basque Country, Puglia
d62.14.14	Caregiver recommends platform	GKPLAT_01	Milton Keynes

Appendix B Platform Requirements

The list of platform requirements as per deliverable D3.1 draft (June 2020)

Type	ID	Description	Platform Interaction
USER	Req_UI_01	Solution and information visually accessible and friendly for elderly people users (big displays, big buttons, voice feedback…)	
USER	Req_UI_02	Easy interaction for elderly people (to write, to read, hand tremor handling…)	
USER	Req_UI_03	System connects and provides information to Social Services and primary healthcare for better intervention	GKPILOT_01, GKPLAT_06
USER	Req_UI_04	System connects and provides information to healthcare professionals	GKPILOT_01, GKPLAT_06
USER	Req_UI_05	The system must provide the stakeholders with information (data, elaboration, warning, state, …) in an appropriate manner (by type of content and by style of associated with that particular subject (User, Caregiver, HCP, ….)representation) with the skills	
USER	Req_UI_06	Help Desk services will be implemented to support / inform final users and informal caregivers.	GKPILOT_04, GKPILOT_09
USER	Req_UI_07	The whole system must not be dangerous for the user	
USER	Req_UI_08	Allow the configuration of restitution information related to users / Designation of a third party to get access to the data	
USER	Req_UI_09	Allow the configuration of activity monitoring by caregivers (formal and informal)	GKPILOT_03
USER	Req_UI_10	The solution has to be intuitive and easy to use	
USER	Req_UI_11	Provide feedback to users concerning the equipment installed in the assisted home	
USER	Req_UI_12	User input can determine opt-outs and data correction	
USER	Req_UI_13	Participants to be given full training on all hardware and software on delivery	
USER	Req_UI_14	Users and referring services offered training on digital literacy & privacy	
USER	Req_UI_15	Have easily available support for participants, email, phone, peer support and forums	
USER	Req_UI_16	Participants to complete comprehensive information sheet at sign up to determine which aspects of the project suitable and what will participate in	
USER	Req_UI_17	Potential participants are identified and informed about the project through a variety of methods - self identifying, NHS or council services, or third sector services	
USER	Req_UI_18	The services shall provide ways of promoting physical exercising and mental training.	

Type	ID	Description	Platform Interaction
USER	Req_UI_19	Users must be able to choose the services, service components and applications they will use.	GKPILOT_02
USER	Req_UI_20	The system shall be easy to learn and remember for older adults	
USER	Req_UI_21	The system shall have an attractive user interface for older adults	
USER	Req_UI_22	Enabled users must be able to delete a measurement	
USER	Req_UI_23	Allow the configuration of activity monitoring by caregivers (formal and informal)	GKPILOT_03
APP	Req_AP_01	System alerts on medical appointments to elderly and caregivers	
APP	Req_AP_02	System sends alerts to assisted elderly and caregiver to take medication. Reminder could be configurable remotely	
APP	Req_AP_03	The solution shall provide a dashboard displaying the user's personal data	GKPILOT_07
APP	Req_AP_04	The solution shall provide tools to question the users about any issues/trouble he/she could have	
APP	Req_AP_05	When emergency trigger alert raised will need to alert either named contact or statutory service	
APP	Req_AP_06	Participants should be able to have an overview of the data gathered and trends, changes etc. through a web portal	GKPILOT_07
APP	Req_AP_07	Alerts should be interactive so as to ask questions alongside to gather more information	
APP	Req_AP_08	Notification system on the watch must be configurable to deliver medication reminders, and able to be programmed remotely	GKPILOT_04, GKPILOT_08
APP	Req_AP_09	The system shall provide useful help for users on how to do the tasks (virtual assistant)	
APP	Req_AP_10	Audience information: Information about the users of the marketplace, their profile and services downloaded to estimate potential for exploitation and attract more service providers	
SEC	Req_PS_01	Security of data and information generated (prevention from unauthorised access)	GKPLAT_10
SEC	Req_PS_02	The use of information will be closely linked to identification mode - enabled users will only have access to the information they have been enabled for.	GKPILOT_05, GKPLAT_09, GKPLAT_11
SEC	Req_PS_03	The system should respect the privacy and data protection of the users	
SEC	Req_PS_04	The system must ensure adequate security of the acquired data in terms of protection against accidental damage and deletion	

Type	ID	Description	Platform Interaction
SEC	Req_PS_05	Ensure the discretion of the solution	
SEC	Req_PS_06	The solution shall preserve anonymous data and respect data privacy	
SEC	Req_PS_07	The system should be managed by a User Management module to control access in UIs and services	GKPLAT_10, GKPLAT_11
SEC	Req_PS_08	The system should respect the privacy and data protection of the users	
D_ABS	Req_DA_01	Data gathered must be able to be fed into the Leeds Care Record (integrated health & social care).	
D_ABS	Req_DA_02	Data should link to GP record so can view activity, plus gather info on hospital admissions etc. Will need to fit with services provided by EMIS and TPP to achieve	GKPLAT_05
D_STOR	Req_DS_01	The data produced by IoT sensors must be stored in local premises or in the HPE Cloud	GKPLAT_04
NT	Req_NT_01	Participants will need mobile data/WiFi access on devices to link to watch, and also externally to log into portal	
NT	Req_NT_02	The supporting system allows messages between carers and includes options for notes from GPs, services, carers etc.	
NT	Req_NT_03	The IoT sensors must communicate with the system using known "standard" protocols	
DEVICE	Req_DSP_01	System detects location of assisted elderly person outdoors and informs caregivers of it	
DEVICE	Req_DSP_02	The IoT platform should allow its own extension with other components (new sensors, new connectors etc) without a need for adaptation	GKPLAT_03, GKPILOT_08
DEVICE	Req_DSP_03	The system must be able to acquire body weight measurements collected by smart devices	GKPLAT_04
DEVICE	Req_DSP_04	System collects information useful for the early detection of pathologies (insomnia, physical inactivity through wearables…)	GKPLAT_04
DEVICE	Req_DSP_05	Connection with medical devices to measure vital signs (blood pressure, glucose…)	GKPLAT_04
DEVICE	Req_DSP_06	System receives information about the physical activity of the elderly via wearable: measuring activity, steps, distance walked…	
DEVICE	Req_DSP_07	User monitoring must be performed using IoT sensors able to provide their measurements via an Internet or a Bluetooth connection.	GKPLAT_04

Type	ID	Description	Platform Interaction
DEVICE	Req_DSP_08	Data collection must be as "transparent" to the user as possible (i.e. no need for user intervention). The IoT sensors should not require the user to change their daily living habits.	GKPLAT_04
DEVICE	Req_DSP_09	Instructions to the measurement devices used must be in the native language of the user	
DEVICE	Req_DSP_10	The solution shall communicate only with authenticated devices	GKPLAT_09, GKPLAT_11
AI	Req_AI_01	Generation of reports on routines to detect trends on the assisted person behaviour	GKPILOT_07, GKPLAT_08
AI	Req_AI_02	Alerts/alarms inform of changes in parameters of routine	
AI	Req_AI_03	The services offered should be selected and adapted in relation to the current situation of the user	
AI	Req_AI_04	Personalized agenda for assisted elderly person and its relatives/ informal caregivers. Elderly can add events to this agenda	
AI	Req_AI_05	Alarm to alert informal caregivers or emergence services of a risky situation stemming from a period of inactivity	GKPLAT_12
AI	Req_AI_06	The data collected by IoT sensors must be processed to generate information about the well-being and health status of the user	GKPLAT_04, GKPLAT_06, GKPLAT_08
AI	Req_AI_07	Solution provides information on activities that assisted elderly people like or are keen on	
AI	Req_AI_08	System sends alerts to remind the elderly daily habits / routines (decided by elderly and also by caregivers)	GKPILOT_06
AI	Req_AI_09	System suggests to the elderly to do physical activity or other activities	GKPLAT_08
AI	Req_AI_10	To detect sleep patterns (influence of polimedication)	
AI	Req_AI_11	Trend information may require graphic representations.	GKPILOT_07
AI	Req_AI_12	Contingency abnormal alert information must be formatted in a not alarming manner - even because they only represent suspicions of anomalies.	
AI	Req_AI_13	Algorithm on falls identification decides where to send alerts based on risk and information given on sign up -informal carers, services etc.	GKPLAT_12
AI	Req_AI_14	The solution shall detect if the assisted is in bed or chair and estimate the time spent in bed or chair.	GKPLAT_08
AI	Req_AI_15	System provides periodic reports and historical data about assisted elderly activity	GKPILOT_01

Type	ID	Description	Platform Interaction
AI	Req_AI_16	System generates information for KPI to be included dedicated Platform for monitoring of the service and evaluation of service benefits (number of detected alerts, people assisted, life expectancy, number of informal caregivers participating)	
AI	Req_AI_17	Produce location-based reminders of events/activities the user is interested in to promote social inclusion	
AI	Req_AI_18	System products (data, analysis, messages, highlights) could be purely informative.	GKPLAT_07

Appendix C Platform components interactions overview

Requirement ID	Narrative	Things Management System	BigData Infrastructure Service	GK Data Integration	Trust Authority	Market Service	Health Activity Monitoring	AI Personalized Risk Detection & Assessment	Intelligent Medical Device Connectors	Multi Robot Connectors	Authoring Tool
GKPILOT_01	an AuthorizedActor (Patient, GP, Caregiver) subscribes to health events receives notifications	X	X	X	X						
GKPILOT_02	a GK actor browses the MarketService catalogue to find a suitable solution and obtains the endpoint of the service	X			X	X					
GKPILOT_03	External Pilot service gets updated metrics from collectors or devices	X			X				X		
GKPILOT_04	GP uses pilot app to send feedback to patient	X			X						
GKPILOT_05	a User registers his identity to the Thing he bought through the MarketService	X			X	X					
GKPILOT_06	External Pilot Service generates advices without exploiting platform services	X			X						
GKPILOT_07	HC Actor (Patient/GP/Caregiver) configures and visualizes a personalized dashboard	X		X	X						X
GKPILOT_08	A Company Actor adds a new KET in the MarketService	X			X	X					
GKPILOT_09	A patient gets automatic recommendations	X	X	X	X			X			
GKPLAT_01	a Business Actor browses the MarketPlace catalogue using specific constraints	X			X	X					
GKPLAT_02	An Authorized user / service registers a new GK Thing in the TMS through the MarketService	X			X	X					
GKPLAT_03	An authorized user / service wants to find a Thing in the TMS to use it	X			X						

Requirement ID	Narrative	Things Management System	BigData Infrastructure Service	GK Data Integration	Trust Authority	Market Service	Health Activity Monitoring	AI Personalized Risk Detection & Assessment	Intelligent Medical Device Connectors	Multi Robot Connectors	Authoring Tool
GKPLAT_04	A KET sends data to the platform and it is federated and stored in the platform	X		X	X				X	X	
GKPLAT_05	the Data Federation Component pulls data from configured external repositories	X		X	X						
GKPLAT_06	An HC User (Patient/GP/Caregiver) reads federated data from the Platform	X		X	X						
GKPLAT_07	Data series can be aggregated / anonymised in order to be included in an offering in the MarketService	X		X	X	X					
GKPLAT_08	Data are used by processing services	X		X	X		X				
GKPLAT_09	The verification of a JWT token	X			X						
GKPLAT_10	the GK-Actor (person or service) verifies its credentials and gets a JWT token associated to his role	X			X	X					
GKPLAT_11	A User is authorized to accesses a Thing in the platform	X			X						
GKPLAT_12	A HealthProfessional requests a risk prediction for a patient	X	X	X	X			X			

Appendix D Logical Architecture diagram

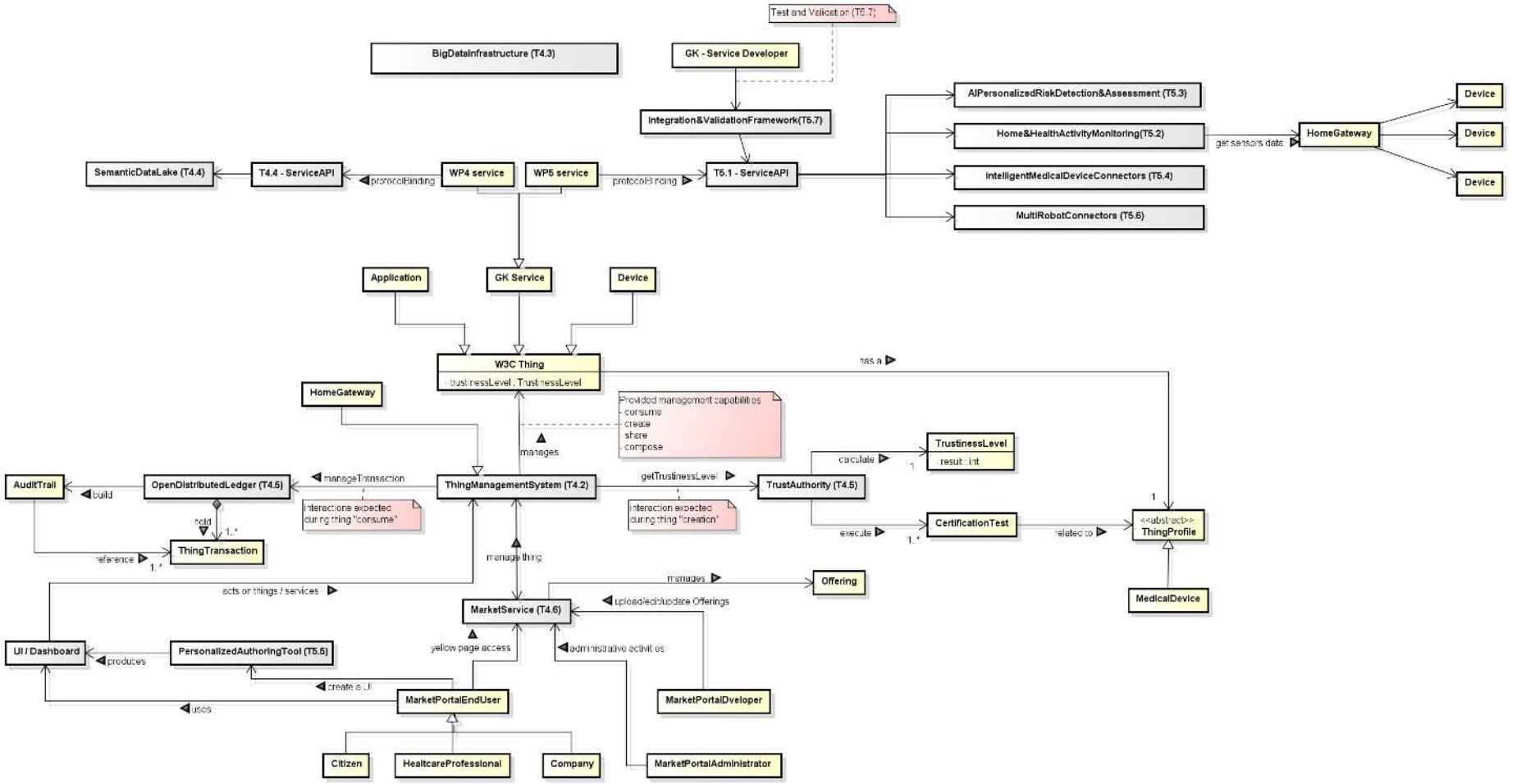


Figure 18 - High level UML Domain Model

Appendix E GATEKEEPER Spaces

This Appendix reports the definition of GATEKEEPER Spaces referenced in this deliverable as defined in the DOA:

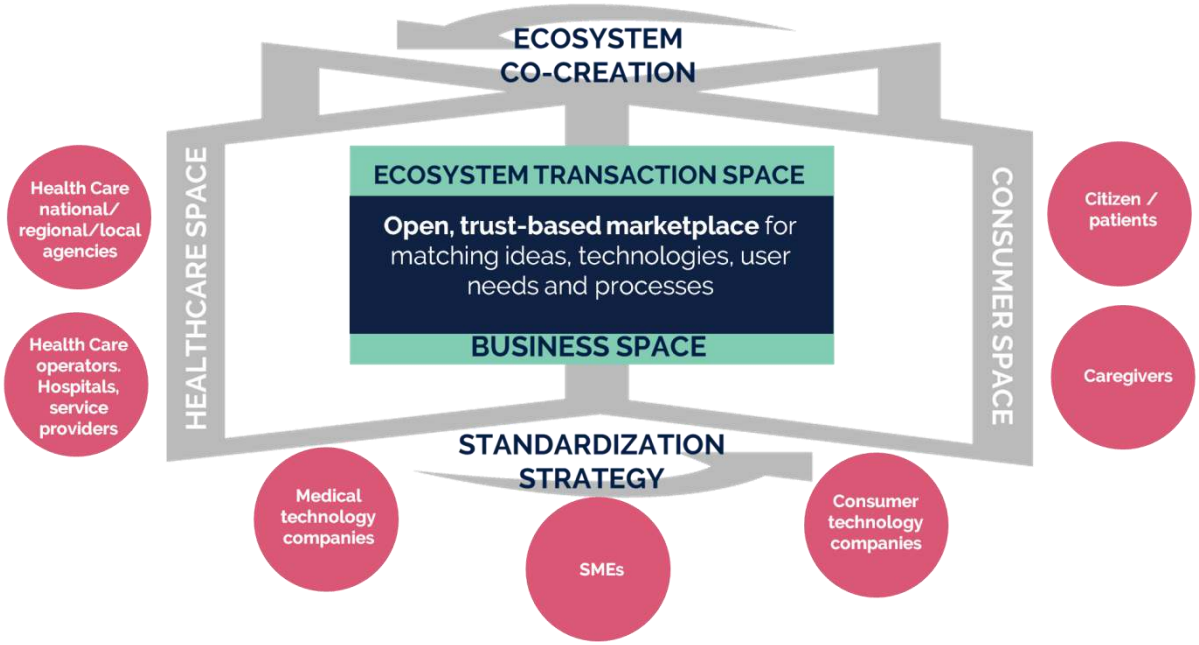


Figure 19 – GATEKEEPER Spaces and Stakeholders

The Healthcare Space provides a set of services, tools, data and components for healthcare, complying with the health protocols and regulations. It also connects with health information systems and records. It enables to build Business-to-Business (B2B) solutions and services from companies to healthcare providers.

The Consumer Space provides a set of services, tools and support components that allow the integration and interoperability of consumer-oriented solutions, appliances, robots, applications, data, sensors and platforms. It allows to build Business to Consumer (B2C) solutions and services to be used by end users for health or life-style monitoring, as well as integrated with solutions from the Healthcare Space to combine services and provide a holistic health view and monitoring in return.

The Business Space provides the adequate ecosystem for small, medium and large companies to develop solutions, services and devices alone or in partnership with other companies following a set of standards in order to reach end-users (Consumer Space) or health providers (Healthcare Space).

The Ecosystem Transaction Space provides a large selection of applications and devices leveraging AI, Big Data, machine learning and IoT technologies; coupled with a variety of smart objects (e.g. wearables, sensors, robots) currently available in the market to support Data Sharing and Value-based healthcare.

Appendix F Glossary

Term	Description
ACL	Access Control List
AD	Active Directory
ADL	Activities of Daily Living
AI	Artificial Intelligence
B2B	Business to Business
B2G	Business to Government
CA	Certification Authority
CoAP	Constrained Application Protocol
CVS	Comma Separated Value
DAPS	Dynamic Attribute Provisioning Service
DoA	Description of the Action
ECG	Electrocardiogram
EHR	Electronic Health Record
FHIR	Fast Healthcare Interoperability Resources
GDPR	General Data Protection Regulation
GTA	GATEKEEPER Trust Authority
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
JSON-LD	JSON for Linked Data
JWT	JSON Web Token
KET	Key Enabling Technology
LDAP	Lightweight Directory Access Protocol
ML	Machine Learning
OTP	One Time Password
PKI	Public Key Infrastructure
RDF	Resource Description Framework
REST	Representational State Transfer

Term	Description
RUC	Reference Use Case
SAREF	Smart Appliance REFerence
TD	Thing Description
TMS	Things Management System
UI	User Interface
URI	Uniform Resource Identifier
VPN	Virtual Private Network
WoT	Web of Things
XML	eXtensible Markup Language