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

Platform Overview

Version 1.0
31st October 2020

Table of contents

List of tables

List of figures

Abstract

Gatekeeper platform is a digital platform that provides AI and data-oriented services for the development of health and care solutions.

Gatekeeper is based on the concept of digital twin where every platform asset, such as devices, services, data or even if other platforms, has a digital replica that is described with a Thing Description in agreement with Web of Thing standard specification.

Within Gatekeeper the Things are virtual entities that are, decoupled but linked with their physical and/or technological implementation. Based on that at data level Gatekeeper allows a high degree of separation between data owner (the physical owner of a database for instance) from the data provider (the service that wraps the data into a digital twin).

The Gatekeeper core data are the data related to the Gatekeeper users. Gatekeeper users are developers and customers, nor patients neither healthcare professionals are expected to be Gatekeeper user.

Anyway, when a developer builds an application by using Gatekeeper services he can associate to several Gatekeeper services sensible data such as personal patients' and/or healthcare professionals' data. This data are private data owned by the developers that Gatekeeper is hosting and for which is granting security and privacy implemented into the platform and the deployment infrastructure. Data stored into the Gatekeeper platform associated with a user are isolated from any other user.

Data belonging to a developer are only accessible and visible by that developer. For the applications developed on the top of the Gatekeeper services, the developer is responsible to implements the adequate privacy and security mechanisms to avoid data breaches to his applications by using appropriate countermeasures.

Anyway, developers can rely on some core security and privacy services provided by Gatekeeper that can help them to do this job, such as standard user management services for custom authentication and authorization, secure connections and communications provided by Gatekeeper infrastructure services, intrusion detection systems (IDS) for network traffic generated by developers' applications, etc.

Furthermore, Gatekeeper can provide to pilot developers some additional feature in terms of a federation of their physical resources (not only data) into the Gatekeeper platform.

The Gatekeeper platform by default is deployed into the data centre provided by HPE in Rome. Such data centre provides both physical security (security personnel, access control to the facilities, locks of the physical infrastructure) either IT security such as above-mentioned services IDS, etc. Based on that, the storage and data extraction operations are physically done into the HPE data centre.

1 Architecture definition principles

This document describes 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 as well as Pilots (WP6), and other user requirements (WP2). In this first section we will describe the defining principles that drive the design of the Gatekeeper Platform, then we introduce the stakeholder of the platform and 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 the rules of governance of the platform that are applied to a "Gatekeeper thing" through the release of a certification (Figure 1).

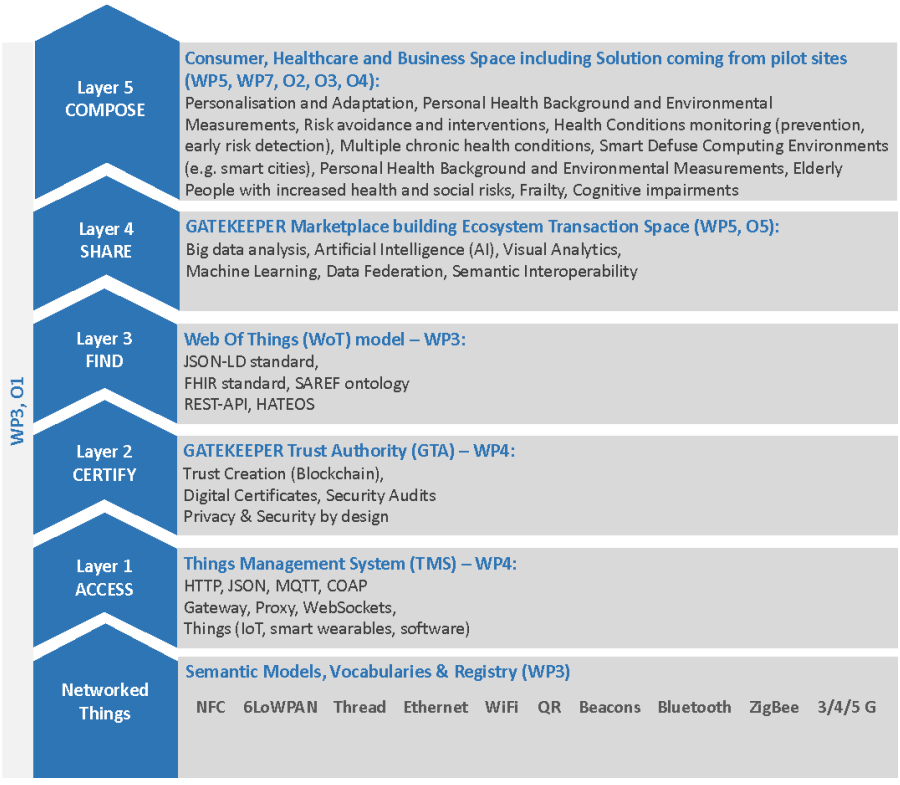

Figure 1 - Gatekeeper layered architecture

The different sets of policies are associated with a different kind of 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 with:

- 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);
- 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 providing solutions for data interoperability and re-use in machine learning (ML) and artificial intelligence (AI) algorithms 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 a Thing (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- sensible 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 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 that allows to interact with something in the physical world, like opening a door or reading a sensor located somewhere in the world. In Gatekeeper this layer is provided by the Thing Management System. The Thing 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 an **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, (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 against 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 Marketplace will be in charge of sharing the Gatekeeper things. The Trust creation will be managed using Blockchain 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 to provide 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 the 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 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.

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 associated with Gatekeeper services, 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 a 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 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 Thing Management System (TMS). The Object model 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.

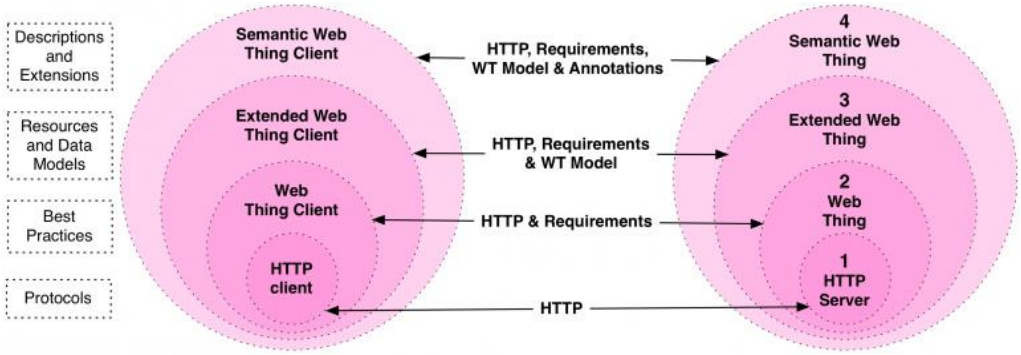

Figure 2 - Web of Things model [8]

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.

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 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.

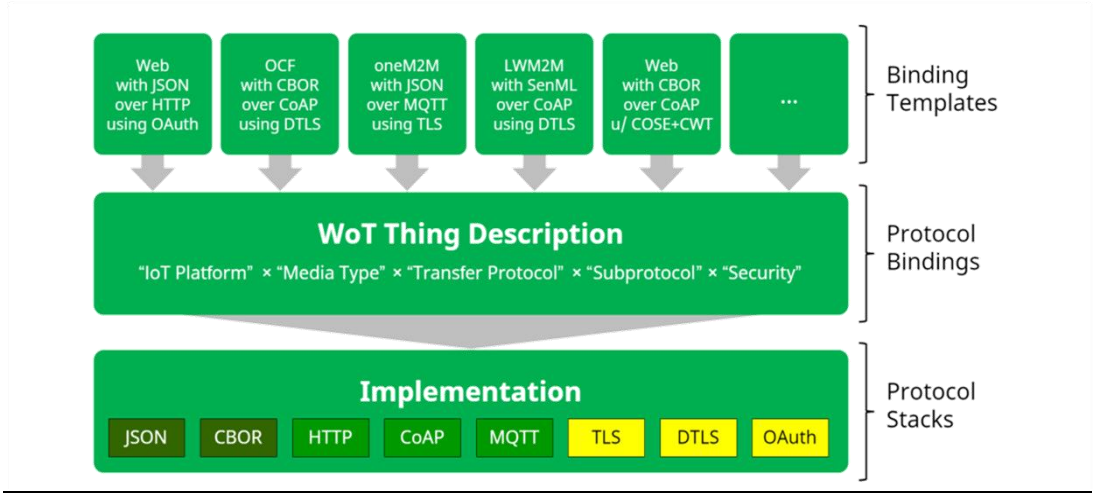

Figure 3 - From Binding Templates to Protocol Binding [9]

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) 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 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[1] (e.g., stable URI, temporary and	optional	anyURI

[1] 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

Vocabulary term	Description	Assignment	Type
	mutable URI, URI with local IP address, URN, etc.).		
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[2], tel RFC3966[3], https).	optional	anyURI
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[4] 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

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

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

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

Vocabulary term	Description	Assignment	Type
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 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 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.
- 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.

1.1.4 Role of FHIR and relation with Thing Description

FHIR 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 centre 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, ICD, LOINC, etc.) used by the adopter. In this way 2 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 FHIR Conformance Resource within a Thing Description by linking the endpoint that is providing the Conformance Resources as showed in the following example:

Example 3: FHIR Conformance Resource in the Thing Descriptor

```
{
  "@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 Platform Stakeholders

Stakeholders that are interested in the results of the Gatekeeper project can be differentiated of two types: the platform stakeholders, who interact and use the software, and project stakeholders, who are the ones who do not interact directly with the solution but are somehow affected by it.

This deliverable tries to combine the analysis of D2.3 and the domain knowledge expressed in D6.2, with a specific focus on platform stakeholders to identify them and the role they cover in the usage of the platform.

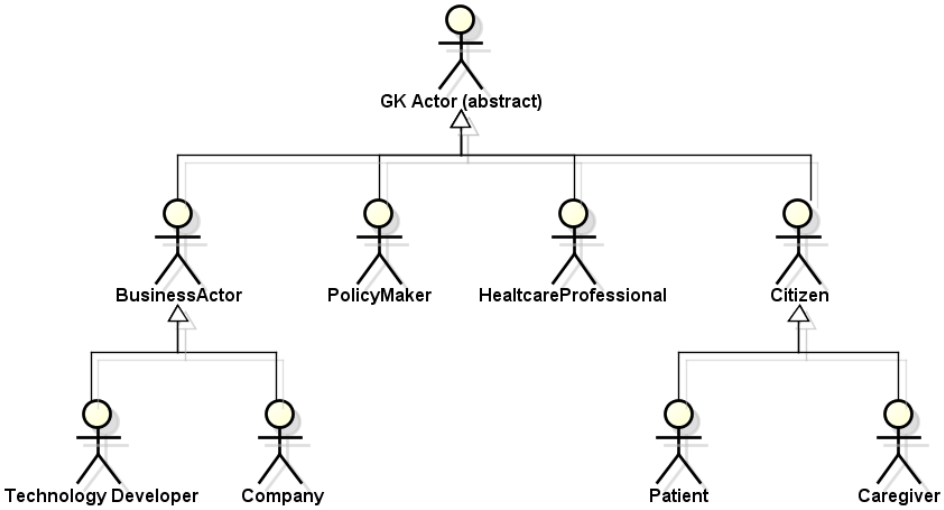

Figure 4 - Gatekeeper platform stakeholders

BUSINESS ACTOR

Extends: GK Actor (Abstract)

Description: Generic stakeholder 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:

POLICY MAKER

Extends: GK Actor (Abstract)

Description: Administrator of the GK Platform. Manages the governance policies of the regulating the platform

Concrete Implementers: Governments, HC Systems

HEALTHCARE PROFESSIONAL

Extends: GK Actor (Abstract)

Description: A Professional Caregiver is a person 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 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 Memeber

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. 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, d) the Thing Action Tracking / Audit and e) the Clearing House (exposed as Thing).

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 gives permissions for the trusted 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.

1.3.1 Infrastructure security

In order to address security and privacy issues, the Gatekeeper Data Centre infrastructure managed by HPE uses a number of technologies, products and services:

- 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 functionals views. Components colors highlight their role. Pink components represent Core Pltform Things (from WP4), blue components are Integrated Dynamic intervention Things (WP5), while yellow components represent External things that can interact with the platform and respond to sepcific 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 figues demonstrate the significant interactions of stakeholders that trigger the main data flows. For clarity, we split such flows to hilight the ones that concern the Business and Transactions spaces and the ones that emerge from the use in the Consumer and Healthcare spaces. 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 moterating the MarketService content and managing the security and privacy policies in the Trust Authority (Transaction space)
2. Developers integrate the Platform serivices 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).

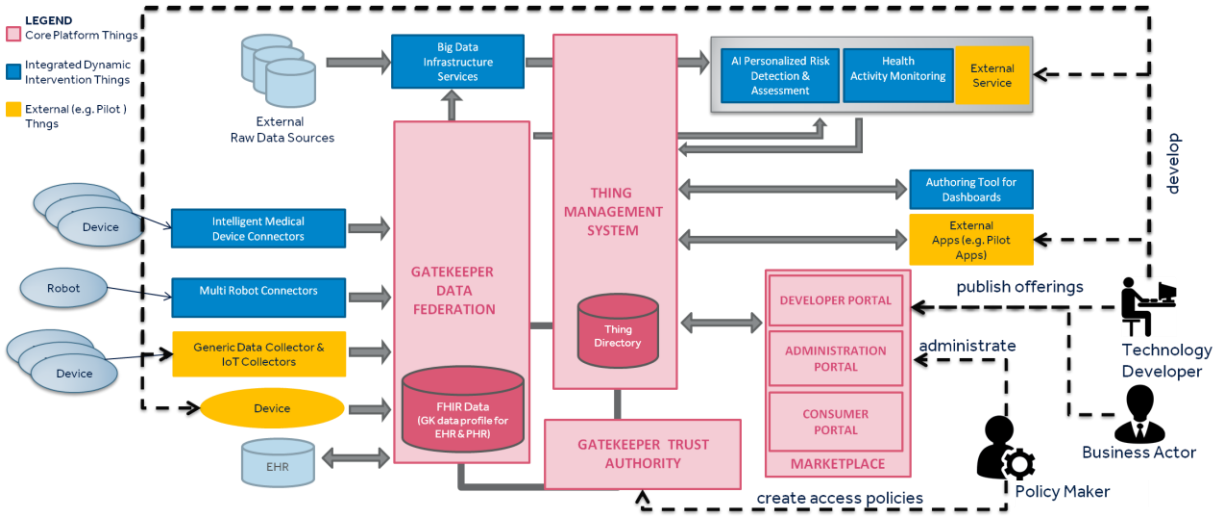

Figure 5 –Gatekeeper Architecture view - Business and Transaction spaces

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 Descriptors, following the Web Of Things approach (Section 1.1). It keeps a registry of such Things in the Thing Directory and also acts as an API Gateway mediating any interaction between Things and their consumers using the 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 such policies 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.
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 acheive 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 easily in their solutions.

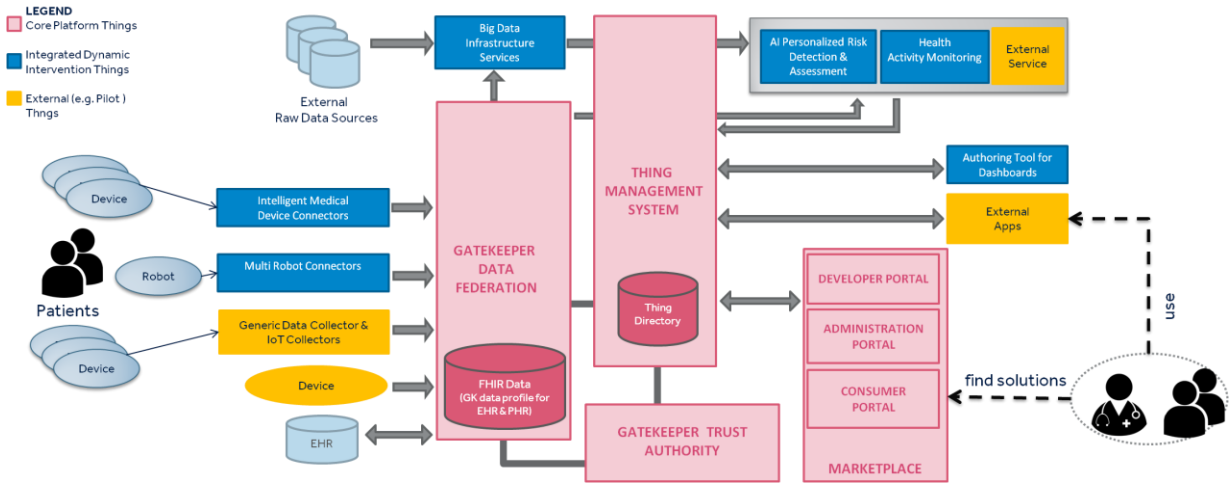

Figure 6 –Gatekeeper Architecture view – Consumer and Healthcare spaces

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 to access device measurement data regardless of their differences in inferfaces or connection protocols, and homogenize their data format; the **Multi robot connector**, the connector that allows to interact and get information from robots.
GateKeeper Data Federation service is responsible to integrate and federate data coming from the different sources. It provides a set of southbound interfaces to connect to the different data providers. Data can be sent explicitly by the connectors and devices

exploiting the provided rest interface or be configured to periodically read data from EHR or other 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.
Data integrated in the data federation are also pushed to the **Big Data Infrastructure**, where they can further processed and merged with external data sources.
The infrastructure will provide services to preform Big Data analysis and generic models that can be exploited from the other services registered in the platform as Things.
In the plaform 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.
Following figure 16 shows the UML domain model associated to the Gatekeeper platform.

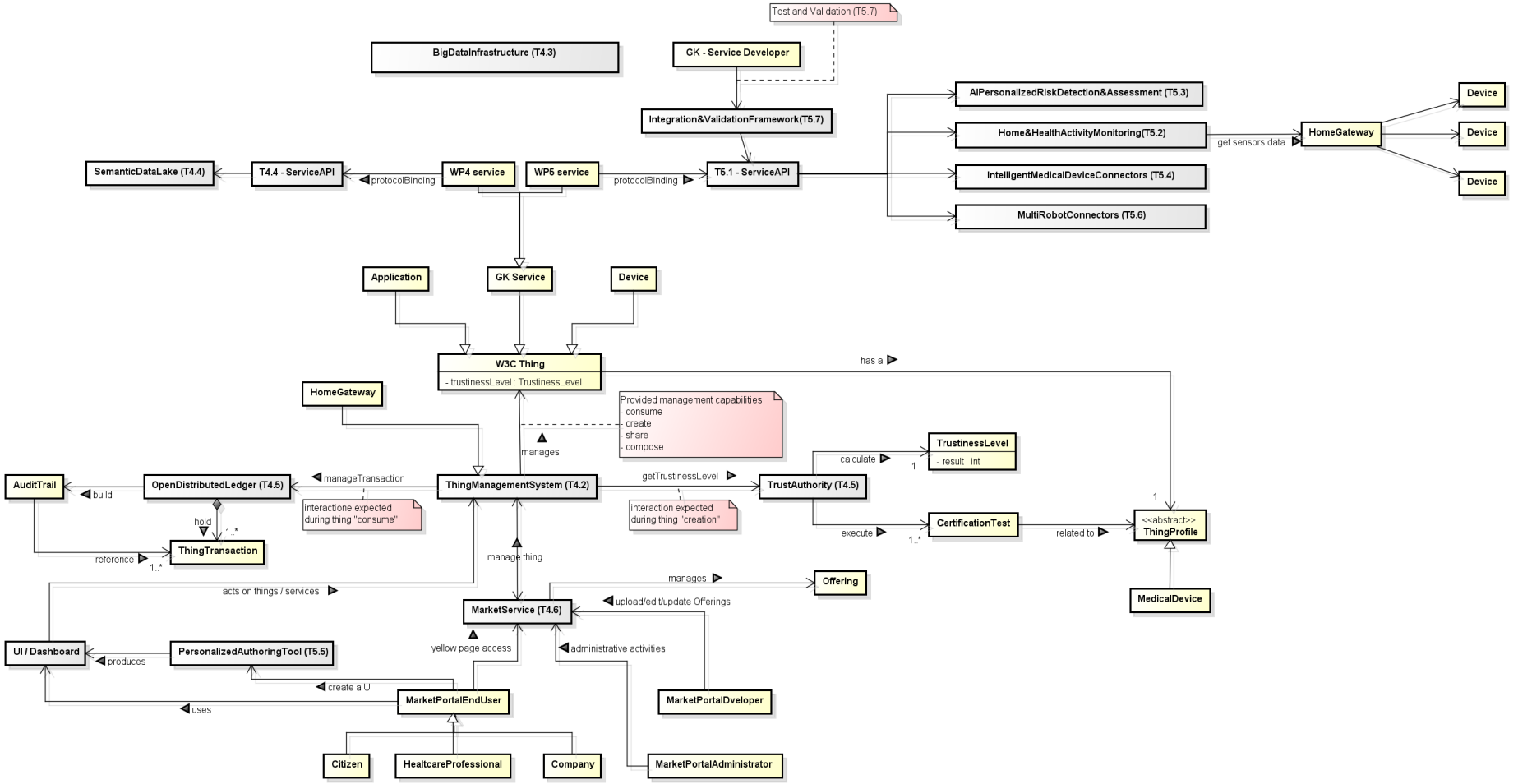

Figure 7 - High level UML Domain Model

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.

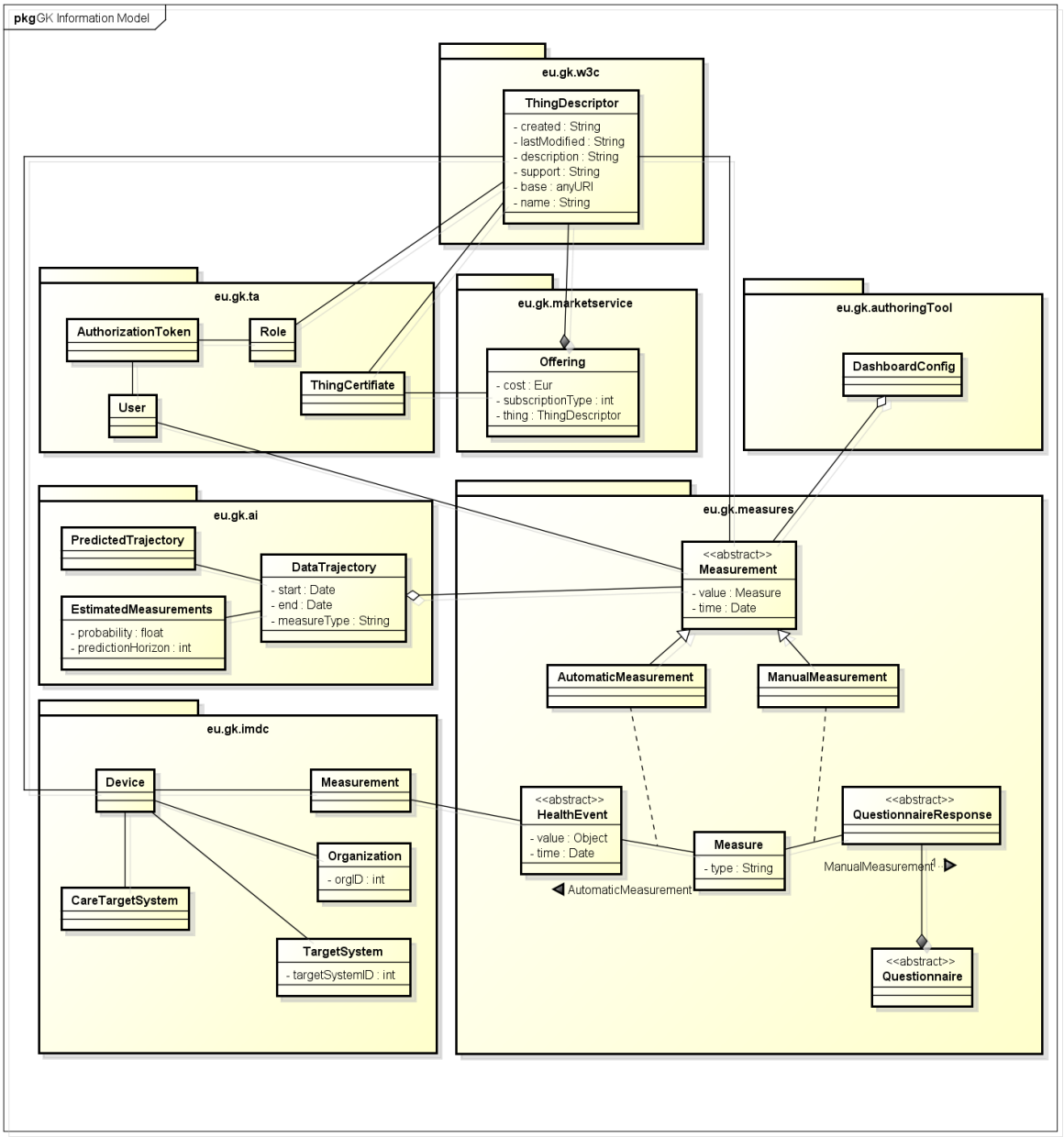

Figure 8 - Gatekeeper Information Model

The reported Information Model (Figure 8) 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 *ThingDescriptor* 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 ThingDescriptor, and perform actions, the Trust Authority links to the TD a set or authorized *Roles.* Roles are assigned to registered *Users* and they 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 *ThingDescriptor*.

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 **Error! Reference source not found.**.

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 9 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, that 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 *AuomaticMearurements*, 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 *Helthy 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, that will also provide a FHIRv4 compliant database to store the translated data and make them available for the rest of the platform.

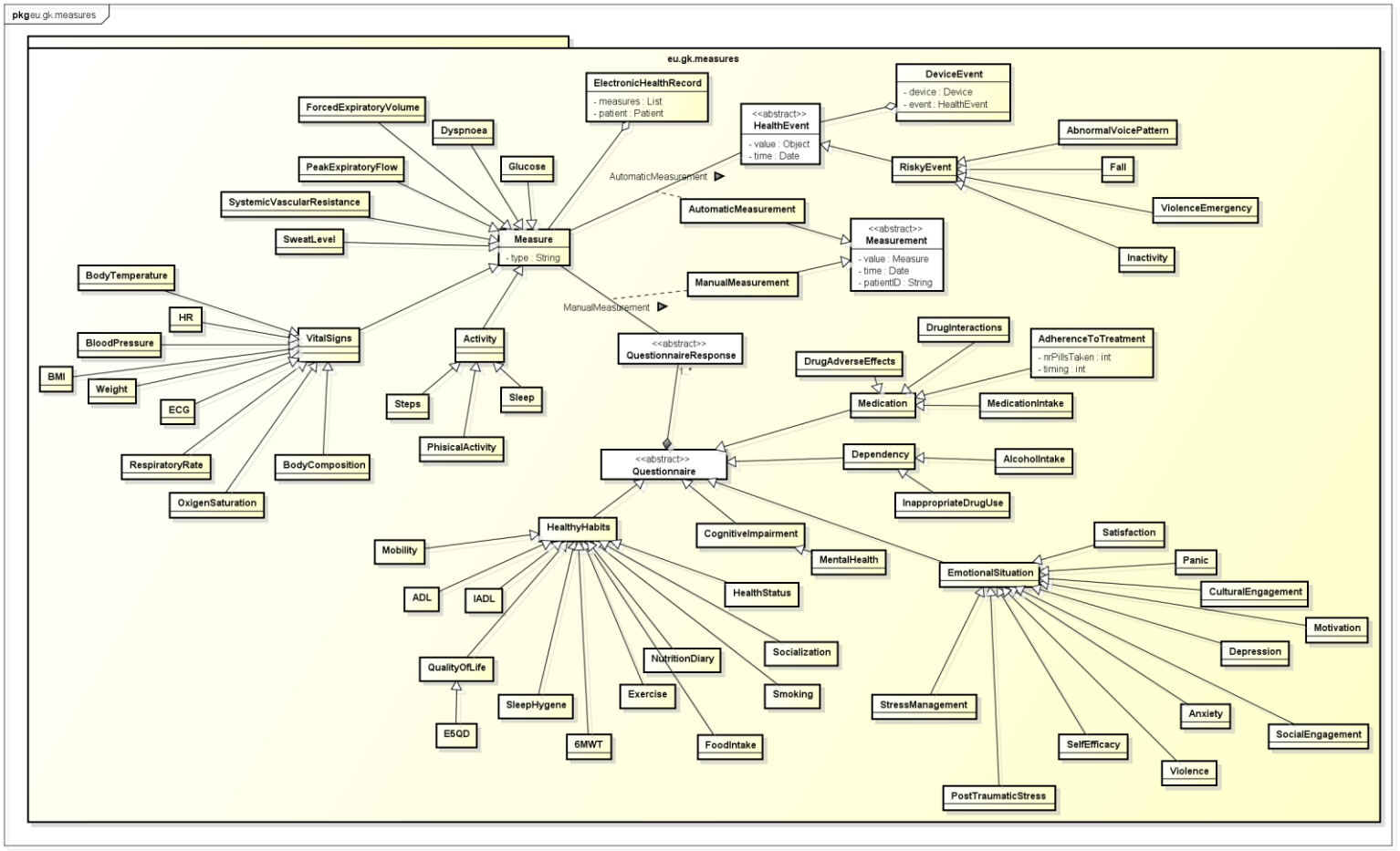

Figure 9 - GK Health Information Model

4 GateKeeper Components

In this document the components reported below are considered as black-box 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
Things Management System	UPM	4.2
Things Directory	UPM	4.2
Error! Reference source not found.	HPE	4.3
GK-IntegrationEngine	ENG	4.4
GK-FHIRServer	ENG	4.4
RDF Semantic Data Lake	ENG	4.4
Trust Authority	CERTH	4.5
Error! Reference source not found.	CERTH	4.6
Error! Reference source not found.	SAMSUNG	5.2
Error! Reference source not found.	MYS	5.3
Error! Reference source not found.	MEDISANTE	5.4
Error! Reference source not found.	TECNALIA	5.5
Error! Reference source not found.	OU	5.6

For the first release of the platform only components highlighted in gray in table 2 will be provided.

4.1 Things Management System

The Things Management System (TMS) is the entry point of the GateKeeper platform. In analogy to a classical microservices architecture 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 Descriptor. A representation of this functionality is shown in Figure 10, and it is based on the intermediary architecture described in the Web of Things architecture specifications (https://www.w3.org/TR/wot-architecture/).

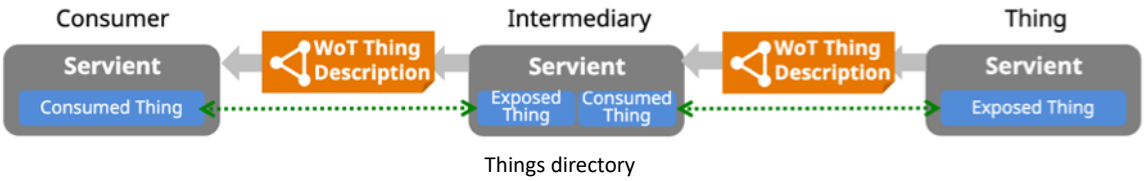

Figure 10 - Conceptual Diagram of the GateKeeper Thing Management System (TMS).

The Thing Management System is the intermediate in any interaction between things and consumers. We define thing as any device, service or platform that is standardize with a model of data to be operated as an individual element derived from a set of predefined templates like smart light-bulbs, smartwatches, AI service or marketplace analytics platform. In Figure 11 it can be seen the inner architecture of Thing Management System and its components.

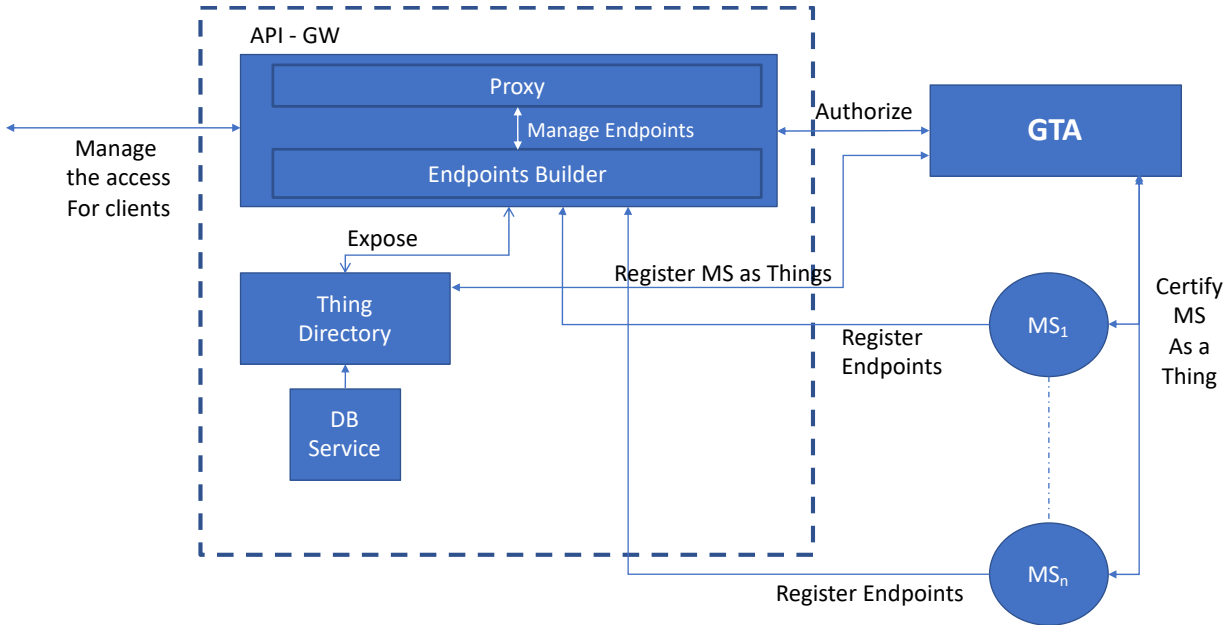

Figure 11 - GateKeeper Thing Management System (TMS) inner architecture.

In the architecture of the GateKeeper Thing 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 Descriptor (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 microservices (only for REST interfaces)
  - Builder: Interact with GTA for building and register new endpoints
- Thing Directory TMS – TD – Directory that collect all GK Things
  - It's the WoT Runtime (e. g. ArenaWebHub) It should provide access to standardized URL for properties, actions, events like Mozilla does
- MSx: Microservice x providing service x as REST API

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) first use of a thing and (ii) normal use of the thing.

The first use case, which is shown in **Error! Reference source not found.**, represents the interaction between the user and the platform for the first time that the thing is used. Since in the first use case it is needed to ask for an authorization and certify the component. In the second use case (**Error! Reference source not found.**), the component is already recognizable by the GateKeeper Trust Authority and used directly thought the right permission.

ThingManagementService Provided Interface		
access() : anyURI		
Ask to the TMS for the access to the thing.		
Input(s)	--	
Output	thing:anyURI	*Thing Descriptor 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 Descriptor of the TMS with security definition (e. g. Bearer authentication)*
getTMSDescriptor(String): anyURI		
Ask to the TMS for the Thing Descriptor		
Input(s)	thingDescriptor: String	*The serialized Thing (see section 1.1.3) to grant access to*
Output	thing:anyURI	*Thing Descriptor 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)	thingDescriptor:	*The serialized Thing (see section 1.1.3)*

	String	
Output	thing:anyURI	*Thing Descriptor*
consumeTMS (thingID): anyURI		
Request one thing to TMS		
Input(s)	thing ID	*The ID of the requested Thing*
Output	thing:anyURI	*Thing Descriptor*

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

4.2 Things Directory

Description of the component

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 Things 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 Descriptor 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()		
Consume an action on a thing in its Micro Service		
Input(s)	AnyURI	Thing ID of the thing to modify
Output	bool	State of success

4.3 GK-IntegrationEngine

The GK-IntegrationEngineis the component able to convert raw data, coming from different data sources EHR, sensors, iot devices, wearables etc.), in 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 CVS 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 RDFSemanticDataLake.

For electronic health record, it converts custom EHR into FHIR v4.0.1 representation according 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 terminologies to be used for the conversion is provided by the task 4.4. Finally transformed data is sent to the component GK-FHIRService.

IGK-IntegrationEngine Provided Interface		
create(pilot: String, sensorId: String, data: File): responseBody: String		
Interface accepting data in XML/JSON/CVS 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 RDFSemanticDataLake component. If the registered converter produces data complaint to other ontologies (e.g. SAREF) then they will be loaded only in RDFSemanticDataLake 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 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 RDFSemanticDataLake. 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 od 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 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.*

IIOTSemanticMappingService Requested Interface

Provider	Method	Description	Input	Output
RDFSemanticDataLake	*create(rdfData: String): HTTPResponse*	*Persist transformed data (RDF) into RDFSemanticDataLake. 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*

4.4 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 using the 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 datatype defined by the standard.

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

IFhirServer 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[]	*paremeter of the seach request*
Output	responseBody: Bundle {json/xml}	*The return content is a Bundle the set of the resources fitting the input parameters*

FHIR Server Provided Interface

IFHIRServer Requested Interface				
Provider	**Method**	**Description**	**Input**	**Output**
RDFSemanticDataLake	*create(rdfData: String): HTTPResponse*	*Persist rdf data into RDFSemanticDataLake. POST*	*rdfData: String*	*HTTPResponse*

4.5 RDF Semantic Data Lake

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.

RDFSemanticDataLake 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*

4.6 Trust Authority

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 a 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.)*

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

Appendix A Glossary

Term	Description
TD	Thing Description
TMS	Things Management System
KET	Key Enabling Technology
DoA	Description of the Action
WoT	Web of Things
GK	Gatekeeper
GTA	Gatekeeper Trust Authority