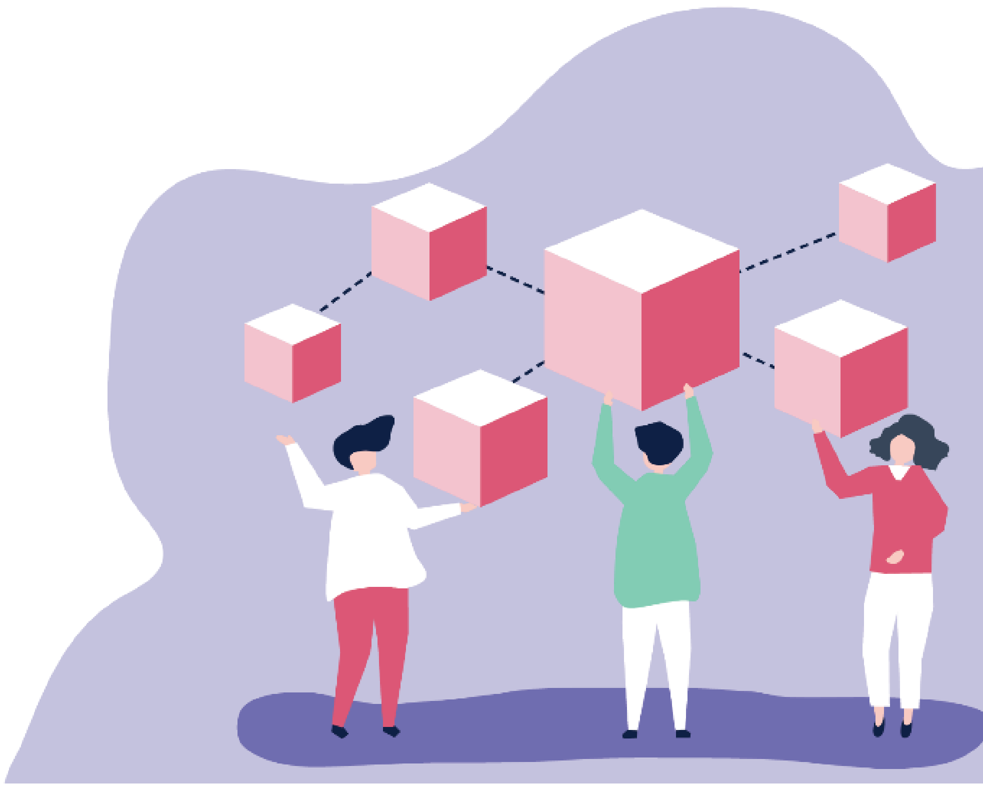
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 857223

GATE KEEPER

D3.3.2 Interoperability within GATEKEEPER, second edition

Deliverable No.	D3.3.2	**Due Date**	31/12/2020
Description	Updated report on interoperability within GATEKEEPER		
Type	Report	**Dissemination Level**	PU
Work Package No.	WP3	**Work Package Title**	GATEKEEPER Web of Things (WOT) Reference Architecture
Version	1.0	**Status**	Ready for quality check



Authors

Name and surname	Partner name	e-mail
Dave Raggett	W3C	dsr@w3.org
François Daoust	W3C	fd@w3.org
Gabriel Galeote	UPM	ggaleote@lst.tdo.upm.es
Eugenio Gaeta	UPM	eugenio.gaeta@lst.tdo.upm.es
Valentina Di Giacomo	ENG	valentina.digiacomo@eng.it
Domenico Martino	ENG	domenico.martino@eng.it
Thanos Stavropoulos	CERTH	athstavr@iti.gr
Ioannis Kompatsiaris	CERTH	ikom@iti.gr
Eleftheria Polychronidou	CERTH	epolyc@iti.gr
Konstantinos Votis	CERTH	kvotis@iti.gr
Giorgio Cangioli	HL7	giorgio.cangioli@gmail.com

History

Date	Version	Change
16/10/2020	0.0	Initial draft outline
09/11/2020	0.03	Revised structure to better align with tasks
17/11/2020	0.04	Section on data federation
24/11/2020	0.05	Sections from previous version adjusted Section on Web of Things
02/12/2020	0.06	Section on FHIR added, Changelog
03/12/2020	0.07	Section on WoT Service Marketplace completed Dupicated WoT and Semantic description sections

		merged Editorial pass (styles, references)
04/12/2020	0.08	Clean separation between general standard descriptions in section 6 and specific recommendations in section 7
08/12/2020	0.09	Further improvements and integration of ENG contribution
12/12/2020	0.10	Ready for internal project review
22/12/2020	0.11	Integrate review comments
14/01/2021	0.12	Quality review: drop outdated analysis of reference use cases, merge sections on interoperability layers, adjust recommendations and conclusions
27/01/2021	1.0	Final version for submission

Key data

Keywords	interoperability, web of things, graph data, abstraction layers
Lead Editor	Dave Raggett (W3C)
Internal Reviewer(s)	Giuseppe Fico (UPM), Armand Castillejo (STM) and Daniel Rodriguez (S4C)

Abstract

This deliverable explores the interoperability challenges facing the GATEKEEPER pilots and the GATEKEEPER platform in respect to integrating heterogeneous devices, information sources, protocols, data formats and data models. The aim is to maximise interoperability through the use of existing and emerging standards, and best practices, across the various services and devices used by each pilot. What are the minimum interoperability mechanisms (MIMs), considering the solutions available from the GATEKEEPER technology partners? What are the interoperability implications for different choices of architecture (edge, centralised and federated)? This report updates D3.3 which was developed earlier in the project (M6), and addresses emerging challenges such as the coexistence of FHIR APIs and the Web of Things, the integration of different vocabularies, the role of validation, and future opportunities involving the combination of symbolic graphs and sub-symbolic statistics.

Statement of originality

Table of contents

List of figures

1 Introduction

European citizens are living longer, and elderly and frail people will be living with various chronic ailments, physical disabilities and mental incapacities that require long term medical attention and care. In many cases, people would prefer to remain in the familiar environment of their own homes if this is practical. This motivates work on smart home healthcare technologies that support well-being at home.

New medical devices and the discovery of bio-markers are enabling improved therapies, based upon the means to collect and analyse wider sources of information to empower patients and caregivers, and provide accurate and objective information to healthcare professionals. GATEKEEPER explores such possibilities via a series of pilots and the development of an open-source platform.

The GATEKEEPER platform integrates a graph database, statistics, rule engine and graph algorithms. It integrates heterogeneous data from sensors and external databases, and exposes devices, services, applications that can leverage the data. To improve consistency across concepts, the platform builds on the Web of Things and represents all items (sensors, actuators, services, applications) as "Web Things".

This second report on interoperability within GATEKEEPER reviews enabling technologies for the pilots that can improve the overall interoperability of the GATEKEEPER platform and makes specific recommendations on key components of the GATEKEEPER architecture to further address interoperability issues.

Interoperability can be defined as the ability of computer systems and software to effectively exchange and make use of information. Interoperability is a key concern for the GATEKEEPER project given the complex requirements for each of the pilots in terms of consumer and medical devices, different kinds of networking technologies, and different kinds of software needed, e.g. in smartphones, home hubs and cloud-based systems.

Interoperability can be considered at multiple layers of abstraction[1]. These are listed in order, such that each layer depends on the next lower layer:

- **Organisational interoperability**: e.g. terms and conditions for using a service
- **Semantic interoperability**: vocabularies[2] for shared meaning, including units of measure
- **Syntactic interoperability**: data formats such as XML and JSON, and APIs
- **Technical interoperability**: protocols such as Bluetooth, HTTP and Web Sockets

There are additional lower layers that are not shown here as they will not be considered in this report, since they are subsumed by the choice of communication technologies.

1.1 Structure of the report

This report follows the following structure:

- Section 2 describes the relationship of this report to other GATEKEEPER reports.
- Section 3 frames the interoperability challenges that the project needs to address.

[1] See: Winters, Leslie & Gorman, Michael & Tolk, Andreas. (2006). Next Generation Data Interoperability: It's all About the Metadata.

[2] *Vocabularies* are used to define concepts and relationships, see: https://www.w3.org/standards/semanticweb/ontology

- Section 4 presents interoperability layers, discusses relevant core technologies for each of the them, and introduces cross-layer technologies such as the Web of Things and FHIR.
- Section 5 reviews specific interoperability issues in GATEKEEPER and makes recommendations for addressing them.
- Section 6 discusses additional considerations.
- Section 7 lists the conclusions.

1.2 Changelog since first edition

The second edition builds on top of the first one, and various parts have remained intact as a result. Main changes are:

- Section 3 was re-written to better align the story with the evolution of the GATEKEEPER platform since publication of the first edition.
- The section that analysed reference use cases (section 4 in the first edition) was removed. A detailed analysis of pilot needs and architecture can now be found in D3.1 – Functional and technical requirements of GATEKEEPER platform.
- Previous section on GATEKEEPER Architectural Patterns was dropped as well. The architecture of the GATEKEEPER platform has been refined in the meantime and is the topic of D3.2 – Overall GATEKEEPER architecture.
- Section 4 merges previous sections on technical, syntactic, semantic, and organisational interoperability. The content of this section is similar to the previous edition, with some notable changes:
  - Section 4.1.1 was added to describe the CSV format.
  - Sections 4.2.2 and 4.2.3 were added to present topic-based pub-sub interfaces and asynchronous message exchanges.
  - Section 4.3.4 on ontologies was updated to refer to D3.4 – Semantic Models, Vocabularies & Registry for details.
  - The WoT, FHIR and OpenAPI standards are now introduced in section 4.5.
- Section 5 is new and looks at concrete recommendations to address interoperability challenges in GATEKEEPER.
- A couple of additional considerations to compare relativist and reductionist approaches, and discuss challenges for managing semantics at scale were added to section 6.

2 Relationship to other GATEKEEPER deliverables

By definition, interoperability deals with interfaces between components, commonalities in envisioned scenarios, semantics and standards. This deliverable reviews interoperability requirements derived from reference use cases defined in D6.1 Medical use cases specification [1] and implementation guide and D6.2 Early detection and interventions operational planning [2].

From a component perspective, this deliverable builds on top of D3.2 Overall GATEKEEPER architecture [3] coupled with the analysis of the architecture of the project's pilot in D3.1 Functional and technical requirements of GATEKEEPER platform [4], and leverage the project's research on key components such as D4.5 GATEKEEPER Trust Authority [5] and the upcoming D4.2 Thing Management System [6]. It presents generic interoperability recommendations for the platform, and specific recommendations for key components.

This deliverable complements D8.1 Overview of relevant standards in smart living environments and gap analysis [7], and also builds on top of D6.3 GATEKEEPER Big Data and Data Analytics Strategies [8] to make interoperability recommendations in that area.

Semantic models and vocabularies are being investigated in D3.4 Semantic Models, Vocabularies & Registry [9], which this deliverable refers to. Key interoperability challenges between FHIR APIs, which the platform needs to support, and the Web of Things approach that the platform is based on, are also being discussed. D3.5 GATEKEEPER HL7 FHIR optimization for IoT and Smart and healthy living environments [10] will complement the picture in that space.

3 Monitoring elderly and frail patients

This section expands the ideas from the introduction and explains the main overall and interoperability challenges that GATEKEEPER needs to overcome to create a uniform platform to monitor elderly and frail patients.

3.1 Framing overall challenges

The aim of GATEKEEPER is to improve the care of elderly and frail patients through better monitoring and support for the patients themselves, their caregivers and healthcare professionals. This is relevant whether the patients are living in their own homes, living in care homes with 24x7 nursing staff, or in hospital wards.

GATEKEEPER further seeks to support different approaches to healthcare, e.g. state provided healthcare or free-market solutions such as in the USA which involve a complex ecosystem and payments for individual services. Countries like the UK use a mix of the two approaches with General Practice health centres and hospitals funded by the state, whilst care homes and home help must be paid for by the patients themselves out of their own savings.

It is also increasingly common for people approaching retirement themselves to have to look after their elderly parents. This can be very demanding for patients with dementia, or for those who fall out of bed in the early hours of the morning, requiring the immediate aid of their caregivers. Can improvements in monitoring help to arrest or slow the decline of elderly and frail patients, with consequent improvements in well-being for themselves and their caregivers?

The GATEKEEPER Pilots plan to use a wide variety of devices including wearables such as wristbands, static devices such as pressure pads, weighing scales, and sensors that detect when doors are opened and closed. These devices use a variety of communication technologies, e.g. Bluetooth, WiFi and cellular modems. In addition, whilst some devices use open standards, others involve proprietary approaches.

This heterogeneity introduces complexity and increases the costs and risks for developing monitoring solutions that combine multiple sources of information and provide integrated dashboards for use by healthcare professionals. The challenge is to mitigate this complexity through an architecture that splits responsibilities so that monitoring services can be developed easily, without having to be concerned about the range of protocols, data formats and other technological details for the different devices and their vendors. Those details are delegated to specialist developers who can create and support the "connectors" that feed information into a uniform framework for use by monitoring services.

GATEKEEPER further seeks to support a marketplace for services as a means to enable an ecosystem with providers and consumers. The project proposal envisions market "spaces" for consumers, healthcare and businesses, and presumes that this can be implemented in terms of providers and consumers of "Things". Further work is needed to turn this from an abstract concept into practical examples with sellers, customers, and services with clear value propositions.

Well defined business models are also needed to support multiple stakeholders: the GATEKEEPER platform operator, the monitoring devices, their provision and installation, and the services provided by caregivers and healthcare professionals.

3.2 Framing interoperability challenges

To achieve its goals, the GATEKEEPER project develops a platform, detailed in D3.2 Overall GATEKEEPER architecture [3]. This platform addresses functional and technical requirements, detailed in D3.1 [4]. D3.1 also describes the architecture of the project's pilots with regards to the GATEKEEPER platform[3].

The main interoperability challenges evidenced by D3.1 stem from:

- The heterogeneity of data sources, be them wearables and other medial devices, medical records, answers to user questionnaires, or other sensors.
- The heterogeneity of application services being considered and of targeted audiences. For instance, all pilots develop applications that need to run across mobile devices. Similarly, all pilots develop applications for professional healthcare teams as well as applications for regular users that partially leverage the same privacy-sensitive data. In other words, different apps need different privileges.

To address such heterogeneity, the GATEKEEPER project resolved to develop a platform based on a Web of Things (WoT) layered architecture. This involves the use of virtual objects that act as digital twins for sensors. Moreover, the identifiers for these digital twins are used as part of a knowledge graph that describes the kinds of things, their properties and interrelationships. These things form part of a uniform framework for data and metadata. On top of things description, GATEKEEPER is working with medical experts to adopt and/or create vocabularies of terms for a knowledge graph that can be used by all of the reference use cases described in D6.1 [1] and D6.2 [2].

The main challenge that needs to be addressed is specifying how the Web of Things can be applied to representing concepts that go beyond physical sensors and actuators. This includes the use of Things to model the knowledge graph. Let's consider some examples:

The patient could be asked to measure his or her body weight at a given time of day. The weighing machine[4] integrates a 3G modem and sends the measurement via the mobile network to a cloud gateway. The measurement is in kilograms and is associated with a timestamp and a unique device identifier that can be used to relate the measurement to the given patient.

From a GATEKEEPER perspective, a Thing gets created with a URL that identifies the particular weighing machine. An HTTP GET request on the URL returns the Thing Description as a JSON-LD resource. The Thing has a numeric property for the weight. The unit of measure is indicated as metadata for the property. The communications metadata in the Thing Description indicates that a GET request on a specified path can be used to request the most recent measurement.

There might be a way for clients to subscribe to Thing events that signal new measurements, for instance, the client platform could register a callback URL. The server exposing the Thing can then deliver events via HTTP POST requests to that URL. This is sometimes referred to as a Web postback API.

In this scenario, the client is the GATEKEEPER cloud platform that would save the measurements to a graph database for subsequent use by services that monitor the

[3] The previous edition of this deliverable included an analysis of enabling technologies for reference use cases, which is no longer included. These reference use cases have given birth to concrete project pilots, which the D3.1 deliverable analyses with regards to technologies and the GATEKEEPER platform.

[4] e.g. the Medisanté Body composition scale

patient. The use of a Thing Description is hidden from services which interface with the graph database. Should services still access the data in the graph knowledge through Thing representations?

In another example, a battery-operated device takes regular measurements and buffers them for efficient bulk transfer. This allows the device to run in a very low power mode that enables the battery to last for many months and possibly even longer, perhaps even the entire working lifetime of the device. A Thing Description for this might involve a Thing action to retrieve the buffered measurements as a JSON array. However, it may also be simpler for the device gateway to push the buffered measurements to the GATEKEEPER cloud directly.

Another example is where the patient or caregiver enters information into a form on a mobile app or desktop web page. The form contents are submitted to an HTTP server and saved to the GATEKEEPER cloud graph database. How to represent this content as a Thing, and is the use of the Web of Things warranted in this situation?

Now consider a sensor that streams data, e.g. an ECG, where the instantaneous value is not of interest as clinical staff are instead interested in the waveform formed by the sequence of values, and what that can tell them about the patient's heart condition, e.g. the presence of arrhythmia and diseased valves. An application could present a scrolling view of the waveform, and might also apply pattern recognition to classify the waveform, and to raise alarms when something that needs urgent attention is detected. Clinical staff may also want to look at past data, e.g., from a previous episode.

This points to the need for a client API for live and historical data, including the means to query for particular patterns and alarms. This is also the case for sensors whose data is batched and uploaded to the cloud every now and then.

In principle, GATEKEEPER could define such an API in terms of an "action" that returns the specified data, e.g. an array of objects whose properties are the sensor reading and the time it was taken. For replaying old data, we could also define a streaming API where values are passed to a call-back function. GATEKEEPER partner, W3C/ERCIM, has a web-based ECG demo, where the call-back is used to update an array of values which is then rendered to an HTML canvas element as a multi-channel scrolling display.

The ECG machine[5] streams data to a cloud server, which then can be accessed from a mobile or desktop app. Ideally, the server would be the GATEKEEPER cloud so that clinician can save a snapshot for later use. WebSockets would be a natural choice for the streaming protocol along with JSON for the message format. A Thing Description may be used here as well, but the current Thing Description specification lacks support for buffered updates.

A final example is where a monitoring service wants to make use of external electronic health records that are exposed via the HL7 FHIR standard. This defines a deeply hierarchical data model expressed as XML, JSON or RDF/Turtle. Having used HTTPS to retrieve the data, applications can then use the XML DOM or XPath to traverse it in the case of an XML resource, or the object path in the case of a JSON resource.

This points to the need to model an HL7 FHIR resource as a Thing Description, or to bypass the Web of Things layer and implement a driver that pulls the XML or JSON resource from

[5] e.g. MediLynx PocketECG

the FHIR endpoint, transforms it into a graph representation, and then saves it into the GATEKEEPER cloud graph database.

There is value in being able to map FHIR to RDF, and HL7 has already paved the way. For example, if you have "29463-7" as the LOINC[6] identifier for body weight, this is mapped to "http://loinc.org/rdf#29463-7". This is also the case for HL7's own terminology, e.g. to combine a system value with a code value to yield a URI such as "http://terminology.hl7.org/CodeSystem/v2-0203/rdf#MR", whilst the subject of an observation is associated with the RDF URI "http://hl7.org/fhir/Observation.subject". HL7 is now working on a standard for representing the FHIR data model in terms of JSON-LD.

Having mapped an HL7 FHIR resource to an RDF graph, the GATEKEEPER platform could then expose this to application code via an API to traverse graphs, as well as an API to make SPARQL queries, and to apply RDF shape constraints (e.g. expressed in SHACL).

In short, to follow a Web of Things approach consistently throughout the GATEKEEPER platform, the project needs to model novel types of Things interoperably, be it because the resources being considered do not directly fit in the sensors/actuators category, or because they follow data formats, such as FHIR, for which the mapping to Things needs to be defined. The project believes that the benefits of following such an approach outweighs the costs of having to perform this modeling effort. These benefits include the ability to create a marketplace with a common model for data, and the ability to create a single trust authority that can validate, certify, and authorize access to Things, both of which are key enablers to implement suitable platform governance logic.

[6] https://loinc.org/

4 Core Interoperability Technologies for GATEKEEPER

We consider interoperability at different layers of abstraction, where each layer depends on the layer below: Organisational, Semantic, Syntactic and Technical. The following figure shows a high-level architecture schema for the GATEKEEPER Platform, highlighting the interoperability challenge that each subsystem must fulfil.

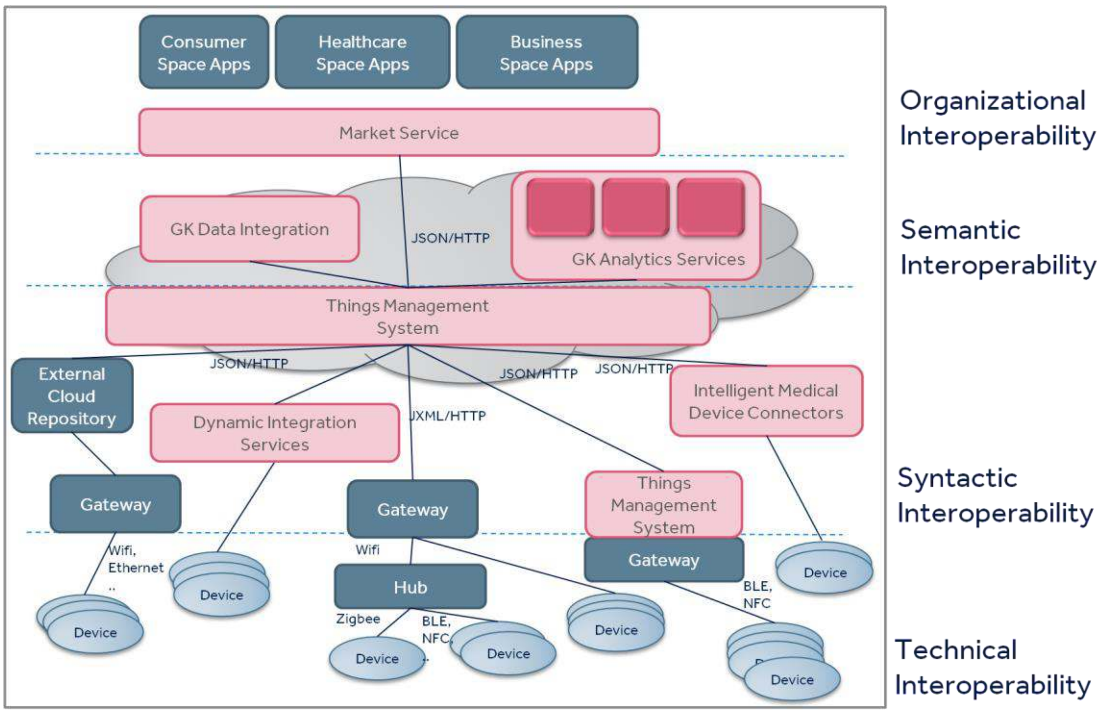


Figure 1 - Interoperability and the GATEKEEPER Platform architecture

- **Technical Interoperability**

  The guarantee of Interoperability at this level of abstraction is not a concern of the platform itself, but it is taken care of by the technologies provided by pilots. One exception is the work of Task T3.5 which will propose an HL7 FHIR binary optimization for IoT.

  Another perspective is that this is taken care of by gateway software, which could, for example, run as a mobile app on a smartphone, or run as a cloud service. The gateway functions as a "connector" that uses whatever IoT technologies are appropriate to collect data, and then forwards it to the GATEKEEPER cloud platform using either HTTPS or, for streaming data, WebSockets Secure.

- **Syntactic Interoperability**

  The main component ensuring interoperability at this level is the Things Management System. All data coming from the sensors will be collected by this component. The Web of Things paradigm and its protocol bindings will ensure a homogeneous representation of data at syntactic level. In GATEKEEPER, devices, hubs, gateways, data sources and actual data are represented as digital twins with REST interface, described through Thing Descriptions.

The Things management system can access data directly provided from Gateways or External Cloud repositories, or mediated by other GATEKEEPER services provided by WP5 (e.g. Intelligent medical device Connectors, Dynamic Integration Services).

Another perspective is that "connectors" (see above) are responsible for transforming data gathered from IoT devices into a uniform format such as JSON or Chunks for ingestion into the GATEKEEPER database, and subject to validation against an agreed ontology. This assumes a strict access control mechanism that safeguards privacy. This involves a management system that deals with authentication and access control, along with the means to register and unregister connectors.

- **Semantic Interoperability**

  Semantic interoperability refers to the agreement between the supplier and consumer of a service as to the meaning of data. For example, that a given value is a number denoting the patient's body weight in kilograms as measured at a specified date and time, and on a particular weighing machine. Traditionally, this kind of information was included as part of the system documentation, and as such was implicit in the design of the system.

  More recently, the trend is to make this information explicit in the form of machine-interpretable metadata using an agreed vocabulary of terms and following a well-defined ontology. Explicit metadata facilitates search and transformation when mapping data between different ontologies, something that is important when you want to exploit information from heterogeneous sources. GATEKEEPER will host metadata on a graph database that is subject to appropriate access control in order to safeguard privacy, e.g. segregating data by patients.

  In GATEKEEPER, JSON-LD is typically used to make the semantic of digital twins explicit.

- **Organizational Interoperability**

  Organizational Interoperability relates to the business agreements between suppliers and consumers of services. This covers privacy, security and other terms and conditions. GATEKEEPER pilots will involve hardware and software components from multiple entities, e.g. certified medical grade devices, mobile applications, cloud platforms and database vendors. The project notably intends to develop a novel ontology in order to describe organization, governance and business processes of the platform.

  The GATEKEEPER marketplace is intended to provide a forum for suppliers and consumers of services relating to home healthcare, segmented into consumer and business dataspaces. Further details are given in the chapter on organizational interoperability.

4.1 Technical Interoperability

Technical interoperability covers the interoperable use of protocols such as Bluetooth and HTTP. Whilst the protocols may have well-defined standards, there is often latitude for using them in different ways that can then result in a lack of interoperability.

The guarantee of interoperability at this level of abstraction is not a concern of the platform itself, but it is taken care of by the technologies provided by pilots. One exception is the work of Task T3.5 which proposes an HL7 FHIR binary optimization for IoT. This work will be detailed in D3.5 [10].

Another perspective is that this is taken care of by gateway software, which could, for example, run as a mobile app on a smartphone, or run as a cloud service. The gateway functions as a “connector” that uses whatever IoT technologies are appropriate to collect data, and then forwards it to the GATEKEEPER cloud platform using either HTTPS or, for streaming data, secure WebSockets.

Thing Descriptions include communications metadata that describe how a client platform interacts with a server platform when the latter exposes things using a REST API. In principle, GATEKEEPER could ingest data from external sources where those sources expose Things. In addition, GATEKEEPER itself could expose things for use by client applications.

An alternative framework would be for GATEKEEPER to expose a single HTTPS based API for uploading data to the GATEKEEPER platform using a standard data format (JSON or Chunks) and agreed data models. The Pilots would be given a means to install services on the GATEKEEPER platform, where services can use scripting APIs exposed by the platform to access and manipulate data for each patient. In addition, and if appropriate, GATEKEEPER could expose network APIs for remote clients.

Technical interoperability is related to how to convert complex objects to sequences of bits, i.e. the means to support data serialization across different systems and technologies.

Based on the standards that are foreseen to be part of GATEKEEPER, such as FHIR and Web of Things, the most important serialization formats are: CSV, XML, JSON and JSON-LD. An additional format is “Chunks”, an amalgam of RDF and Property Graphs under exploration by the project.

The FHIR specification allows data to be serialized as XML or JSON and soon JSON-LD, which is also used for describing digital twins for the Web of Things.

All these formats are also agnostic of the underlying technologies used for marshalling and unmarshalling serialization of data, in contrast to other formats such as POJO or JavaBeans that are only for Java.

More details on the standards for HL7 FHIR, XML, JSON and JSON-LD are given in deliverable D8.1 [7]. The following describes the serialization formats themselves.

Non-machine-readable and/or proprietary formats such as paper forms, spreadsheets, Word or PDF documents are still often used to serialize data. They are, by definition, harder to process and their usage is discouraged. These formats are not covered in this report.

4.1.1 Comma Separated Values (CSV)

CSV is one of the most popular formats for publishing data. It is concise, easy to understand by both humans and computers, and aligns nicely to the tabular nature of most data.

- See W3C *“CSV on the Web: A Primer”* [11] for more details and links to related standards.

But CSV also creates various interoperability issues. Despite its apparent simplicity, there is no formal specification in existence that defines the CSV format, which allows for a wide variety of interpretations of CSV files.

For instance, it is common to use separators other than the comma in CSV files to separate columns, but there is no mechanism within CSV to indicate the separator being used. The first line in a CSV file may be a header row… or not. Lines may also be delimited by a CRLF line break, or a single LF line feed character. Additionally, there is also no mechanism within CSV to indicate the type of data in a particular column, or whether values in a particular column must be unique. It is not uncommon for numbers within CSV files to be formatted for human consumption, which may involve using commas for decimal points, grouping digits in the number using commas, or adding percent signs to the number.

The format is therefore hard to validate and process on an automated basis, and prone to errors such as missing values, differing data types within a column, or misinterpreted numeric values.

To improve interoperability when ingesting/exporting CSV files from/to external platforms, a companion metadata file needs to be provided to describe the CSV files. The W3C standardized a JSON-LD dialect to create such annotations, known as Metadata Vocabulary for Tabular Data [12].

4.1.2 The Extensible Markup Language (XML)

The standard of Extensible Markup Language (XML) is a markup language. It was created as a both, human-readable and machine-readable format for document encoding. The first specification was given by The World Wide Web Consortium's (WWW) XML 1.0 Specification in 1998 [13] and was updated in 2008 [14].

As a markup language, it is a system for annotating a document in a way that is syntactically distinguishable from the text. That means that this standard allows the communication in the way that the information can be extracted from it while it is used in another format. The XML is just a shell that wraps the data. Therefore, the purpose is to be used for store and transport data between applications and users of a communication process across the internet. It focuses on simplicity, generality, and usability across the Internet, but it does not do anything to the data that is carried.

To understand XML, you need to understand its structure:

- Character: An XML document is a string of characters.
- Processor and application: The processor analyses the markup and passes structured information to an application.
- Markup and content: The characters making up an XML document are divided into markup and content, which may be distinguished by the application of simple syntactic rules.
- Tag: A tag is a markup construct that begins with < and ends with >.
- Element: An element is a logical document component that either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag.
- Attribute: An attribute is a markup construct consisting of a name–value pair that exists within a start-tag or empty-element tag.
- XML Declaration: XML documents may begin with an XML declaration that describes some information about themselves.

It is important to highlight that the XML above does not do anything, it is just information wrapped in tags. XML was developed as a simplification of the preceding SGML standard with the goal of facilitating interoperable processing of structured documents. XML

Schema is defined by a pair of W3C Recommendations for describing families of structured documents. [15] [16]

```xml
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

4.1.3 JavaScript Object Notation (JSON)

The JavaScript Object Notation (JSON) [17] is an open standard file format, and data interchange format, with the purpose of using a human-readable text to store and transmit data objects. The objects consist of attribute–value pairs and array data types. It is a very used data format that almost replaced XML due to high versatility and simplicity.

The JSON format is agnostic about the semantics of numbers, that means that it does not distinguish between types of numeric data (fixed or floating, binary or decimal). That can make interchange between different programming languages difficult. JSON instead offers only the representation of numbers that humans use: a sequence of digits. All programming languages know how to make sense of digit sequences even if they disagree on internal representations.

An example of a JSON file is shown here:

```json
{
    "id": 1,
    "name": "Foo",
    "price": 123,
    "tags": [
         "Bar",
         "Eek"
    ],
    "stock":
       "warehouse": 300,
       "retail": 20
    }
}
```

4.1.4 JSON Linked Data (JSON-LD)

This format is a modification of the JSON file format for linked data (JavaScript Object Notation for Linked Data). It is a method of encoding Linked Data using JSON. It allows data to be serialized similarly as JSON. The latest version is JSON-LD 1.1 [18], which was published as a W3C Recommendation in July 2020. JSON-LD allows existing JSON to be interpreted as Linked Data with minimal changes. It is primarily intended to be a way to use Linked Data in Web-based programming environments, to build interoperable Web services, and to store Linked Data in JSON-based storage engines. A JSON-LD example is provided below:

```json
{
     "@id": "http://store.example.com/products/links-swift-chain",
     "@type": "Product",
     "name": "Links Swift Chain",
     "description": "A fine chain with many links.",
     "category": [
         "http://store.example.com/categories/parts",
        http://store.example.com/categories/chains
    ],
    "price": "10.00",
    "stock": 10
}
```

Since JSON-LD is 100% compatible with JSON, there is a huge community that supports this standard. Main features of JSON-LD are:

- Universal identifier mechanism for JSON objects via the use of IRIs.
- A tool for disambiguate keys shared among different JSON documents by mapping them to IRIs via a context.
- Mechanism in which a value in a JSON object may refer to a JSON object on a different site on the Web.
- Ability to annotate strings with their language.
- A way to associate datatypes with values such as dates and times.
- Facility to express one or more directed graphs, such as a social network, in a single document.

4.1.5 Chunks

The W3C Cognitive AI Community Group's[7] Chunks format[8] is simpler than JSON and JSON-LD, and makes it very easy to represent binary and n-ary relationships. A chunk is a typed collection of properties whose values are literals, chunk identifiers or comma separated lists thereof. Here is an example of an employment record with identifiers for the employee and employer, along with the start date:

```
worksFor {employee smith123; employer acme2645; from 2017-04-10}
```

Chunks are based on work in Cognitive Science. Further details are given in section 6.3.3.

4.2 Syntactic Interoperability

Syntactic interoperability covers the APIs and associated data formats and encodings, e.g. the representation of numbers, and the character set and encoding for strings. GATEKEEPER needs to expose different kinds of APIs:

- APIs for ingesting data into the GATEKEEPER platform
  - For example, a REST API for uploading data
- APIs for use by applications hosted by GATEKEEPER
  - Scripting APIs for local (server-side) applications
  - REST APIs for remote (client-side) applications
- APIs for management purposes, e.g. privacy, trust and security

To ensure robust operation, GATEKEEPER should validate all data passed through these APIs. In addition, to support good security practices, APIs usage should be subject to

[7] https://www.w3.org/community/cogai/

[8] https://w3c.github.io/cogai/

access control and logging. We will need to integrate security agents that monitor the system and are capable of spotting suspicious patterns of behaviour (including denial of service attacks), alerting security staff, and taking remedial actions.

A further goal would be to include security "honeypots". These are mechanisms that are designed to lure attackers as a means to detect, analyse and block attacks. The mechanisms can include known security vulnerabilities that attackers will seek out as a means to compromise systems. We may want to provide a trap database that we can redirect attackers to in place of the operational databases. Honeypots may be designed to fool attackers into thinking that they have succeeded, meanwhile allowing system administrators to trace attackers, and work with ISPs to cancel the attackers Internet accounts.

Some data security standards:

- ISO 27001
- ISO 27799
- HIPAA
- GDPR

Deliverable D8.1 [7] provides a detailed survey of standards and standardisation gaps in that area.

4.2.1 RESTful Interfaces

REpresentational State Transfer (REST) is an architectural style, derived from Roy Fielding's seminal work on *Representational State Transfer* (REST)[9] and defined as an abstraction on top of the Hypertext transfer protocol (HTTP). There are many principles and constraints behind the REST style, they are really helpful when we face integration challenges in a microservices world, and when we're looking for an alternative style to RPC for service interfaces.

In REST, most important is the concept of resources. A resource could be seen as a thing that the service itself knows about, like a Device. The server creates different representations of this Device on request. How a resource is shown externally is completely decoupled from how it is stored internally. A client might ask for a JSON representation of a Device, for example, even if it is stored in a completely different format. Once a client has a representation of this Device, it can then make requests to change it, and the server may or may not comply with them.

REST itself does not really talk about underlying protocols, although it is used over HTTP. Some of the features that HTTP gives part of the specification, such as verbs, make implementing REST over HTTP easier, whereas with other protocols it needs to handle these features from scratch. HTTP itself defines some useful capabilities that play very well with the REST style. For example, the HTTP verbs (e.g., GET, POST, and PUT) already have well understood meanings in the HTTP specification as to how they should work with resources. The REST architectural style actually tells that methods should behave the same way on all resources, and the HTTP specification happens to define a bunch of methods that can be used. GET retrieves a resource in an idempotent way, and POST creates a new resource. This means that it can be avoided lots of different create Device or edit Device methods. Instead, we can simply POST a device representation to request that the server create a new resource, and initiate a GET request to retrieve a

[9] Roy Field's PhD dissertation: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

representation of a resource. Conceptually, there is one endpoint in the form of a Device resource in these cases, and the operations can carry out upon it are baked into the HTTP protocol.

The use of standard textual formats gives clients a lot of flexibility as to how they consume resources, and REST over HTTP lets us use a variety of formats. The XML and JSON formats are the much more popular content types for services that work over HTTP.

The fact that JSON is a much simpler format means that consumption is also easier. Some proponents also cite its relative compactness when compared to XML as another winning factor, although this is not often a real-world issue.

JSON does have some downsides, though. XML defines the link control we used earlier as a hypermedia control. The JSON standard does not define anything similar, so in-house styles are frequently used to shoehorn this concept in. The Hypertext Application Language (HAL) attempts to fix this by defining some common standards for hyperlinking for JSON (and XML too, although arguably XML needs less help). If follows the HAL standard, it's possible to use tools like the web-based HAL browser for exploring hypermedia formats and controls, which can make the task of creating a client much easier.

In a hypermedia format, hypermedia controls represent protocol information. A hypermedia control includes the address of a linked resource, together with some semantic markup. In the context of the current resource representation, the semantic markup indicates the meaning of the linked resource.

The phrase hypermedia as the engine of application state, sometimes abbreviated to HATEOAS, was coined to describe a core tenet of the REST architectural style [19]. HATEOAS means that hypermedia systems transform application state. An application as being computerized behaviour that achieves a goal, it can be described by an application protocol as the set of legal interactions necessary to realize that behaviour. An application state is a snapshot of an execution of such an application protocol. the protocol lays out the interaction rules; application state is a snapshot of the entire system at a particular instant.

A hybrid RESTful is a class of web services that fit somewhere in between the RESTful web services and the purely RPC-style services. These services are often created by programmers who know a lot about real-world web applications, but not much about the theory of REST. Anywhere there is a clear matching on the protocol messages in objects it is well classified as RESTful service. An example of a hybrid RESTful is the Flickr web service[10]. Despite the "rest" in the URI, this was clearly designed as an RPC-style service, one that uses HTTP as its envelope format. It has got the scoping information ("photos tagged 'penguin'") in the URI, just like a RESTful resource-oriented service. But the method information ("search for photos") also goes in the URI. In a RESTful service, the method information would go into the HTTP method (GET), and whatever was leftover would become scoping information. As it is, this service is simply using HTTP as an envelope format, sticking the method and scoping information wherever it pleases.

This optical illusion happens when an RPC-style service uses plain old HTTP as its envelope format, and when both the method and the scoping information happen to live in the URI portion of the HTTP request. If the HTTP method is GET, and the point of the web service request is to "get" information, it is hard to tell whether the method information is in the HTTP method or in the URI. Look at the HTTP requests that go across

[10] http://www.flickr.com/services/rest?api_key=xxx&method=flickr.photos.search&tags=penguin

the wire and you see the requests you would see for a RESTful web service. They may contain elements like "`method=flickr.photos.search`" but that could be interpreted as scoping information, the way "`photos/`" and "`search/`" are scoping information. These RPC-style services have elements of RESTful web services. Many read-only web services qualify as entirely RESTful and resource-oriented, even though they were designed in the RPC style. But if the service allows clients to write to the data set, there will be times when the client uses an HTTP method that does not match up with the true method information. This keeps the service from being as RESTful as it could be. Services like these are the ones I consider to be REST-RPC hybrids.

Here is one example. The Flickr web API asks clients to use HTTP GET even when they want to modify the data set. To delete a photo you make a GET request to a URI that includes "`method=flickr.photos.delete`". That is just not what GET is for. The Flickr web API is a REST-RPC hybrid: RESTful when the client is retrieving data through GET, RPC-style when the client is modifying the data set.

4.2.1.1 Stateless and security

Both RESTful and Hybrid RESTful services rely on HTTP. In order to maintain the confidentiality and integrity of resource representations, it is quite recommended to use TLS and make resources accessible over a server configured to serve requests only using HTTPS.

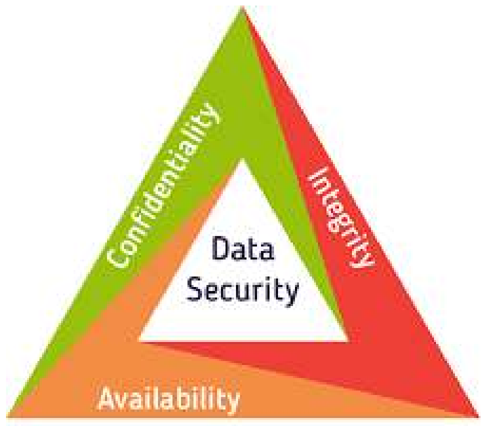


Figure 2 - HTTPS capabilities

HTTP is a layered protocol. It relies on a transport protocol such as TCP/IP to provide the reliability of message transport. By layering HTTP over the TLS (RFC 8446 [20]) protocol (HTTPS), which is a successor of SSL, you can maintain the confidentiality and integrity of request and response messages without dealing with encryption and digital signatures in client and server code (Figure 2).

TLS can also be used for mutual authentication where both the server and the client can be assured of the other party's identity. For instance, you can use basic authentication to authenticate users but rely on TLS to authenticate the client and the server.

When you use TLS for confidentiality and integrity, you can avoid building protocols for such security measures directly into request and response messages. Moreover, TLS is message agnostic. It can be used for any media type or request.

With HTTPS (HTTP over TLS) we are solving confidentiality and integrity but we are still missing another fundamental aspect of security that is authentication.

When users or services interact with an application they will often perform a series of interactions that form a session. A stateless application[11] is an application that needs no knowledge of previous interactions and stores no session information, it is usually based on an architecture that does not need user data. Such an example could be an application that, given the same input, provides the same response to any end-user. A stateless application can scale horizontally since any request can be serviced by any of the available compute resources (e.g., EC2 instances, Google cloud functions, AWS Lambda functions).

With no session data to be shared, you can simply add more compute resources as needed. When that capacity is no longer required, any individual resource can be safely terminated (after running tasks have been drained). Those resources do not need to be aware of the presence of their peers – all that is required is a way to distribute the workload to them.

A stateless service implies a stateless authentication, where the state of a user is not maintained at server-side and where the server is a priori completely unaware of who sends the request. Stateless authentication can be achieved by using a token-based approach. Token-based approaches solve the problem of more traditional approaches in which the server has to store session ids and relevant data for each individual. One of the token based approaches is the JSON Web Token (JWT) standard (RFC 7519 [21]), described in Figure 3.

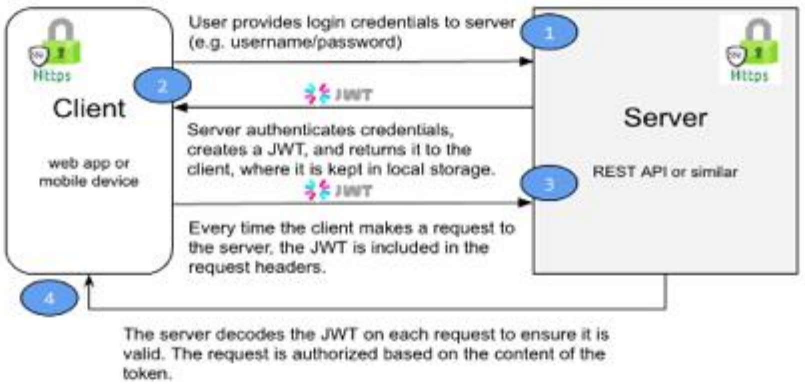

Https
Client
web app or mobile device
1
User provides login credentials to server (e.g. username/password)
2
JWT
Server authenticates credentials, creates a JWT, and returns it to the client, where it is kept in local storage.
3
JWT
Every time the client makes a request to the server, the JWT is included in the request headers.
4
The server decodes the JWT on each request to ensure it is valid. The request is authorized based on the content of the token.
Https
Server
REST API or similar

Figure 3 - JSON Web Token authentication for a stateless service

4.2.1.2 FHIR APIs

The Fast Healthcare Interoperability Resources (FHIR) API[12], standardized by HL7, is an important example of a RESTful protocol for medical data.

The API describes the FHIR resources as a set of operations (known as "interactions") on resources where individual resource instances are managed in collections by their type. Servers can choose which of these interactions are made available and which resource types they support. All interactions with FHIR resources take the form of HTTP operations that follow the form:

```
VERB [base]/[type]/[id] {?_format=[mime-type]}
```

[11] Also see definition of "Stateless Applications" in "Architecting for the Cloud AWS Best Practices - February 2016": https://aset.az.gov/sites/default/files/media/AWS_Cloud_Best_Practices.pdf

[12] http://hl7.org/implement/standards/fhir/http.html

- The first word is the HTTP verb used for the interaction
- Content surrounded by `[]` is mandatory and will be replaced by the string literal identified. Possible insertion values:
  - `base`: The Service Base URL
  - `mime-type`: The Mime Type
  - `type`: The name of a resource type (e.g. "Patient")
  - `id`: The Logical Id of a resource
  - `vid`: The Version Id of a resource
  - `compartment`: The name of a compartment
  - `parameters`: URL parameters as defined for the particular interaction
- Content surrounded by `{}` is optional

4.2.2 Topic-based pub-sub Interfaces

We avoid discussing HTTP vs CoAP and TCP/IP vs UDP, as this would complicate the description unnecessarily.

REST-based APIs are widely used on the Web, and. This uses the HTTP methods for request/response pairs:

- GET – get a copy of a named resource, in other words, download its state
- PUT – create a named resource using the uploaded data as its state
- PATCH or POST– update part of a named resource's state
- DELETE – delete the named resource

These four operations are sometimes referred to using the abbreviation CRUD, which stands for create, read, update and delete. REST-based APIs use HTTP paths for hierarchical names for resources.

MQTT [22] is an OASIS standard for the IoT. It supports the publish-subscribe pattern in which interested parties register an interest in named topics (i.e. *subscribe* to topics), and then receive messages *published* for those topics. MQTT distributes messages via message brokers. Messages can be used to signal changes in state, and other kinds of events.

MQTT is good for broadcasting events to a large number of clients, but poorly suited for situations where you want to send a message to a specific IoT device, e.g. to control the operation of a sensor or actuator, for which REST is a better solution.

4.2.3 Asynchronous message exchanges

The request/response pattern of REST APIs sometimes needs to be combined with asynchronous notifications, e.g. for different kinds of events. In principle, you could use HTTP requests to poll for recent events, but this introduces problems, e.g. how often do you need to make such requests? How does the server know which events you have already seen?

The WebSocket protocol [23] provides an effective solution and involves a TCP/IP connection for asynchronous exchange of messages between two parties. Applications need to agree on the message format and its interpretation. WebSockets refers to this agreement as a *sub-protocol*. A common approach is to use JSON for messages, exploiting built-in support for parsing and marshalling JSON. It is then straightforward to support both request/response pairs, and asynchronous notifications. A refinement is to use the publish-subscribe pattern for such notifications. Events can then be streamed as either individual messages or as blocks, as appropriate for applications involving high-speed telemetry streams.

HTTP can be used in combination with WebSockets when it comes to authentication and access control. You use HTTP to authenticate a client, and to issue a time-limited security token (e.g. a JSON Web Token). This is then used within WebSocket messages to enable a server to manage who can access what services. The client reauthenticates via HTTP when the token expires. For this both HTTP and WebSockets are used over TLS (transport layer security) for encrypted connections to defend against cyberattackers.

4.3 Semantic Interoperability

Semantic interoperability covers information about shared meaning, e.g. that a particular data value refers to the temperature in Celsius for a given room, and at a given time. This can be addressed through agreements on vocabularies of terms.

This section considers the role of semantic technologies in simplifying the technical challenges for the GATEKEEPER pilots with respect to interoperability in the face of heterogeneous information sources.

When ingesting data into the Graph database, the GATEKEEPER Validator should apply graph shape constraints to ensure that the data conforms to the ontology for a given connector as agreed when the connector was registered. The provenance of data would be recorded to track which sensor/source it came from, and to relate it to models of trust and certification.

4.3.1 RDF and Linked Data

RDF is W3C's framework for metadata. W3C has an extensive suite of associated standards. RDF is used to describe things in terms of graphs composed of vertices and labelled directed edges. RDF focuses on individual edges <subject, label, object> called triples.

RDF makes use of URIs for vertices and edge labels. These URIs act as global identifiers for common concepts, and may be dereferenceable to obtain further metadata. RDF further permits local identifiers called blank nodes that are scoped to a single graph. RDF-based ontologies describe a domain in terms of the concepts and relationships used for that domain. This can be applied to all kinds of things: patients, diseases, medications, treatments, test results, behaviours, sensors, locations, events, etc.

RDF abstracts away from lower-level data formats and APIs, forming a key to simplifying integration across heterogeneous information sources. RDF supports reasoning based upon formal semantics and logical deduction, or rule-based graph traversal.

4.3.2 Labelled Property Graphs

LPG are similar to RDF in being composed of vertices and labelled directed edges. You can further associate both vertices and edges with sets of properties (name/value pairs). LPG can be transformed without loss into RDF, using edges for properties, and reification for annotating edges. However, reification is a rather awkward aspect of RDF along with blank nodes. On the flip side, LPG implementations lack interoperability across vendors with a variety of different query languages and APIs.

4.3.3 Cognitive AI and chunks

Cognitive AI can be defined as AI that seeks to mimic human memory, reasoning and learning to give computing a *human touch*, drawing upon advances in the cognitive sciences, and over 500 million years of neural evolution. Traditional approaches to AI are split into work with symbolic representations of knowledge and work using statistical approaches based upon artificial neural networks. Cognitive AI seeks to address the limitations of traditional AI through the combination of symbolic knowledge with (sub-symbolic) statistics, rules and graph algorithms.

The Gatekeeper project is supporting work on Cognitive AI for the chunks graph data and rules format. This builds upon work over many decades by John Anderson on ACT-R (Adaptive Control of Thought—Rational), a popular cognitive science architecture:

> *"ACT-R is a cognitive architecture: a theory for simulating and understanding human cognition. Researchers working on ACT-R strive to understand how people organize knowledge and produce intelligent behaviour. As the research continues, ACT-R evolves ever closer into a system which can perform the full range of human cognitive tasks: capturing in great detail the way we perceive, think about, and act on the world."*[13]

The combination of symbolic and statistical information is important for machine learning and for many forms of reasoning that rely on the statistics of prior knowledge and past experience, for instance, abductive reasoning that seeks likely explanations for given observations, based on knowledge of causal mechanisms, and their likelihood in a given context. Statistics are also important for inferring potential causal relationships in datasets, e.g. using covariance analysis and more general approaches that can search across multiple overlapping datasets.

The Chunks format has been designed to be easier to work with than traditional approaches to graph data, and includes a means to map to RDF for integration with existing information systems, see the Chunks and Rules draft specification[14] [24]. On-going work is addressing cognitive approaches to natural language processing with a view to enabling summaries of patient data, and as a basis for human-machine collaboration. See Appendix A for more details.

4.3.4 Ontologies

An ontology is a set of concepts and categories for a subject area or domain that shows their properties and the relations that hold between them. Ontologies provide a basis for a shared understanding amongst members of a community. W3C's web ontology language for RDF is called OWL [25].

Ontologies used in the GATEKEEPER project are described in D3.4 [9]. Ontologies to be considered are typically those defined by the W3C (such as SOSA/SSN [26]), ETSI (SAREF [27]), HL7 and terminology developers such as Regenstreif (LOINC, UCUM) and SNOMED Int. (SNOMED CT).

[13] http://act-r.psy.cmu.edu/

[14] https://w3c.github.io/cogai/

4.4 Organisational Interoperability

Organisational interoperability covers agreements on privacy, security and more generally, the terms and conditions agreed between the supplier and consumer of a service.

GATEKEEPER is primarily about providing elderly and frail patients with improved care through the use of consumer and medical-grade devices to monitor the patient's condition and enable appropriate action by the patient, caregivers and healthcare professionals. In respect to the aims for a GATEKEEPER marketplace, GATEKEEPER has yet to clarify who the sellers are, what they are selling, the value proposition for the customers, and who those customers are!

It might be better to start by asking what is the business model for providing the GATEKEEPER platform and associated mobile apps. For instance, are patients or healthcare providers expected to pay a subscription fee according to need, e.g. the number of patients? Likewise, how are the monitoring devices monetized? Can they be purchased for a one-off fee, or is there a subscription fee for their use? How does this vary across state provided healthcare such as the UK's NHS, and free-market approaches such as in the USA?

4.4.1 GATEKEEPER Marketplace

The GATEKEEPER Marketplace is a hub for consortium Members and third parties to publish and monetize WoT Services and for end-users to discover and consume them. The Marketplace will facilitate transactions in all GATEKEEPER spaces including **healthcare** (B2G), **consumer** (B2C) and **business** ecosystem transactions (B2B).

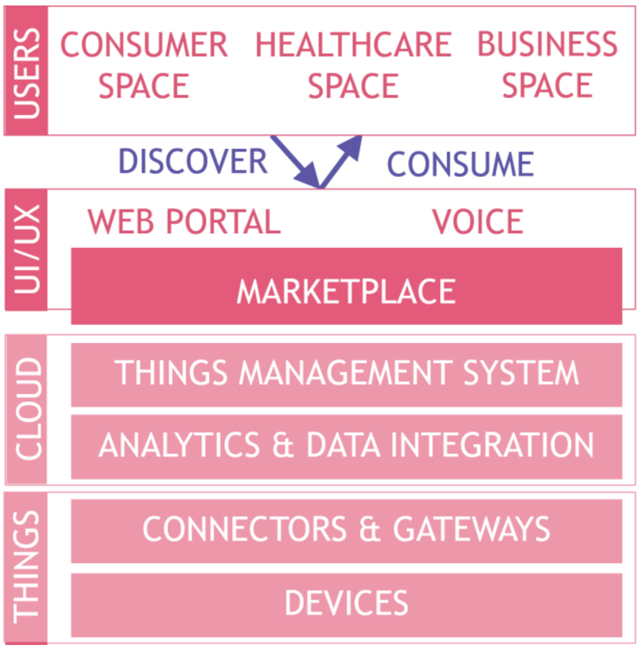


Figure 4 - The Marketplaces relation to other GATEKEEPER components

It can be compared to a "yellow page" directory where users can search for the things hosted in the GATEKEEPER platform. It will provide a single-entry point for all users to explore, conceptualize, test and consume the added value services they are interested in. It will also provide intuitive User Interaction (UI) with modalities such as dialog-based assistants to discover and use services seamlessly, according to user-centred design. Open calls will engage third parties to enrich the Marketplace content and grow the ecosystem.

Interoperability in the Marketplace will be an integral component as all offered services will be coupled with their semantics and interoperability model annotations as defined in WP3. The Marketplace will capitalize on this semantic metadata to offer more effective and rich discovery of what the user seeks.

The Marketplace (based on the current design) will support five categories of Things:

- Web Services
- Sensors
- Medical devices
- Platforms/Closed solutions
- Data.

Figure 4 shows the relation of the Marketplace to other GATEKEEPER components. Essentially, the Marketplaces expose Things (devices, gateways etc.) through the GATEKEEPER platform's Things Management System, Data Integration and Analytics to Users of all spaces, through intuitive UI/UX (Web Portal and Voice modalities). It also functions as a brokerage mechanism as users discover and consume services and Things.

Currently, end-user requirements for all spaces are being collected and coupled with technological specifications of interoperability (WP3) and platform integration (WP5) will shape the Marketplace implementation, which can be followed in future WP4 deliverables.

4.4.2 GATEKEEPER Trust Authority

An example of Organisational Interoperability is the Trust Authority and Open Distributed Ledger of GATEKEEPER which is a component that combines IDSA principles[15] and blockchain technologies in order to support a variety of activities related to privacy and security of the GATEKEEPER ecosystem.

- is responsible for validating "things" according to their conformance with a set of standards related to the GATEKEEPER principles such as WoT, HL7/FHIR, CE marking (for medical devices), etc.
- is responsible for certifying the Things of the GATEKEEPER that achieved a high score in the validation process. This will be used in order to secure the Things that will be included or submitted in the GATEKEEPER Marketplace.
- provides the capabilities for authenticating Things and providing authorisation rules based on the aforementioned levels of certification.
- is responsible for keeping an audit trail of all operations related to things in a privacy preserving way, thus keeping a detailed history of the whole lifecycle of the Thing. This will be used to support the privacy and security of the different spaces of end-users of the GATEKEEPER Platform and Marketplace
- is managing the users' access to the system b configuring their activities according to their roles.

More detailed information about the Trust Authority mechanism is provided and the validation of things will be provided through the task 4.5 and 5.7 respectively.

[15] Reference Architecture Model, International Space Association, https://www.internationaldataspaces.org/wp-content/uploads/2019/03/IDS-Reference-Architecture-Model-3.0.pdf

User Management Module, Certification Authority, Trusted Thing Sharing and Thing Action Tracking are the main components reported in the first version of D4.5 that will support the aforementioned actions.

The Validation mechanism is currently under development and the validation process is combined by four main mechanisms:

- Validation of Things according to the Web of Things (WoT) Thing Description principles
- Validation of Services, including technical and scientific specifications that can support the accurate results of their usage
- Validation of Platforms and Applications, mainly focusing on the secure data management and the integrational capabilities with other systems.
- Validation of Data, including their source, format and semantic interoperability specifications.

The validations function will be both internal, for ensuring that the developed GATEKEEPER components follow the standardisation framework of the project and external in order to certify any type of external "thing" submitted to the Marketplace. Overall, the Validator checks if the "Things" comply with self-defined rules and with general rules specified by the GATEKEEPER ecosystem. Violation of rules can be treated as warnings or errors. If such warnings or errors occur, certification may fail or be rejected.

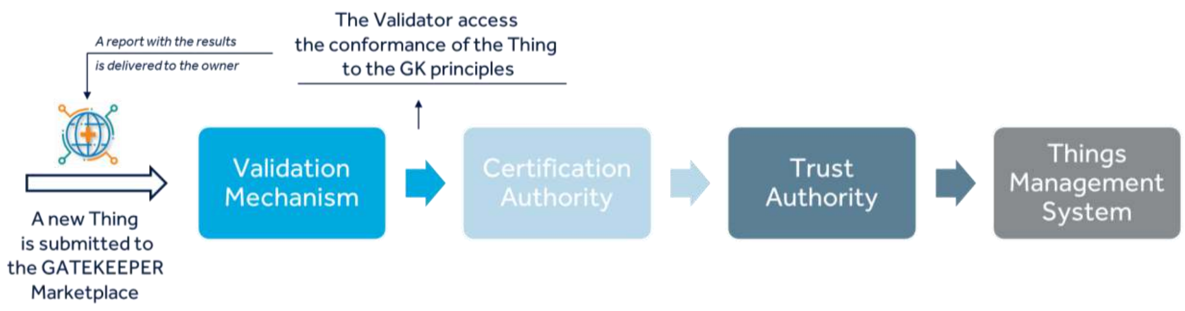

A report with the results is delivered to the owner
The Validator access the conformance of the Thing to the GK principles
A new Thing is submitted to the GATEKEEPER Marketplace
Validation Mechanism
Certification Authority
Trust Authority
Things Management System

Figure 5 – Trust model for the Gatekeeper marketplace

When a "Thing" follows the standardisation guidelines of the GATEKEEPER ecosystem, then a report is delivered to the owners of the "Thing" and a certificated is generated by the Certification Authority. This is stored in the Trust Authority and submitted to the Things Management System as a specification of the new Thing.

4.5 Cross-layer technologies

Some standard technologies span multiple interoperability layers, typically to address technical interoperability, syntactic interoperability and semantic interoperability all at once, and provide hooks to cover organisational interoperability. These standard technologies are of particular interest for the GATEKEEPER project as they restrict the need to specify ad-hoc technical constraints for the platform.

4.5.1 Web of Things

4.5.1.1 Overview

The Web of Things is an abstraction layer for digital twins that seeks to address the fragmentation of the IoT to reduce the costs and risks for all stakeholders.

1. Virtual digital objects that stand for physical and abstract entities
    - *Sensors, actuators, heterogeneous information services,*
2. that are exposed to client applications as local software objects
    - *Clients can interact with the object's properties, actions and events*
    - *Client applications do not see or need to deal with HTTP, Bluetooth, etc.*
        - *those details are handled by the web of things client platform*
3. and used as part of semantic descriptions
    - *The kind of sensor, its physical location, units of measure, …*
    - *Object histories, e.g. EHR records or patient summaries with patient test results*

W3C has been working on the Web of Things for several years and in April 2020 published Recommendations (W3C's term for its standards) for thing descriptions using JSON-LD, and on architectural considerations for the Web of Things. Supplementary notes cover security considerations and a proposed scripting API.

- Web of Things: Architecture [28]
- Web of Things: Thing Descriptions [29]
- Web of Things: Scripting API [30]
- Web of Things: Binding Templates [31]
- Web of Things: Security and Privacy Guidelines [32]
- Web of Things: Current Practices [33]

The W3C Web of Things Interest and Working Groups have also been re-chartered and work is underway on a range of topics including link relations, interoperability profiles, thing description templates, complex interactions, discovery, onboarding, identifier management, security schemes, protocol vocabularies and bindings.

4.5.1.2 Thing Descriptions

This section provides a brief introduction to how JSON-LD is used to describe things in the Web of Things. Thing descriptions cover several aspects:

- The API exposed to client applications in terms of the properties, actions and events, and their associated data types. The Web of Things expresses these in abstract terms that client platforms map to concrete APIs for specific programming languages such as JavaScript, see the W3C WoT Scripting API.
- How the API maps to the underlying protocols supported by the server exposing the service. A declarative representation is supported for REST based APIs with bindings for HTTP, CoAP, as well as for MQTT.
- Security and privacy metadata that details how client platforms can authenticate to servers, and links to privacy policies and service terms & conditions.
- Semantic metadata that relates things to semantic models of services.

The following example is taken from the W3C Recommendation for Thing Descriptions [29]:

```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "id": "urn:dev:ops:32473-WoTLamp-1234",
    "title": "MyLampThing",
    "securityDefinitions": {
        "basic_sc": {"scheme": "basic", "in":"header"}
    },
    "security": ["basic_sc"],
    "properties": {
        "status": {
            "type": "string",
            "forms": [{"href": "https://mylamp.example.com/status"}]
        }
    },
    "actions": {
        "toggle": {
            "forms": [{"href": "https://mylamp.example.com/toggle"}]
        }
    },
    "events": {
        "overheating": {
            "data": {"type": "string"},
            "forms": [{
                "href": "https://mylamp.example.com/oh",
                "subprotocol": "longpoll"
            }]
        }
    }
}
```

This describes a digital twin for a lamp that is monitored and controlled with HTTP. The status property reflects whether the lamp is on or off. The toggle action is used to toggle the lamp's status. In addition, an event named overheating signals when the lamp gets too warm.

The "`@context`" declaration binds names to RDF URIs for the core vocabulary for Thing Descriptions. This enables mapping to RDF for manipulation using the suite of standards for RDF and the Semantic Web.

"`id`" provides a unique identifier for this particular lamp, and acts like a product serial number. "title" provides an informal name for the lamp. "`securityDefinitions`" and "`security`" provide security metadata. This example uses HTTP Basic Authentication over HTTPS.

The data types for Thing Descriptions are expressed using JSON Schema. The status property is a string. The toggle action does not pass any data in the request and associated response. The overheating event passes a parameter named "`data`" which is a string.

The "`forms`" field is used to bind properties, actions and events to the protocols supported by the server. In this example, URLs are provided for a fictitious HTTPS server. The overheating event is monitored using HTTP long polling. This is a technique where the client makes an HTTP request and keeps the connection open until the server responds with a payload. If the connection drops, the client repeats the request, and likewise, after receiving a response. This emulates server-push notifications. More details on protocol bindings for HTTP, CoAP and MQTT are given in the W3C Working Group Note on Binding Templates [31].

HTTP, and HTTP long-polling in particular, are not ideal for applications involving a high frequency for property updates, actions and events. For example, consider a scenario involving real-time medical sensors that stream complex measurements, several hundred times a second. This could be the case where a nurse has applied sensors to a critically ill patient and is seeking urgent advice from a doctor who is many kilometres away.

In such cases, an effective solution is to use asynchronous messaging over WebSockets. This involves defining message payloads for streaming property updates and events as a WebSockets sub-protocol. There is an opportunity for GATEKEEPER to define such a sub-protocol and to propose it to W3C and the IETF as a potential standard. An example of such a messaging scheme is implemented in the open-source project "Arena Web Hub"[16].

The following extends the earlier example of a lamp with semantic annotations using the ETSI SAREF ontology:

```
{
    "@context": [
        "https://www.w3.org/2019/wot/td/v1",
        { "saref": "https://w3id.org/saref#" }
    ],
    "id": "urn:dev:ops:32473-WoTLamp-1234",
    "title": "MyLampThing",
    "@type": "saref:LightSwitch",
    "securityDefinitions": {"basic_sc": {
        "scheme": "basic",
        "in": "header"
    }},
    "security": ["basic_sc"],
    "properties": {
        "status": {
            "@type": "saref:OnOffState",
            "type": "string",
            "forms": [{
                "href": "https://mylamp.example.com/status"
            }]
        }
    },
    "actions": {
        "toggle": {
            "@type": "saref:ToggleCommand",
            "forms": [{
                "href": "https://mylamp.example.com/toggle"
            }]
        }
    },
    "events": {
        "overheating": {
            "data": {"type": "string"},
            "forms": [{
                "href": "https://mylamp.example.com/oh"
            }]
        }
    }
}
```

First, "`@context`" is used to link to the SAREF ontology. Next, "`@type`" is used to indicate the semantics for each property, action and event using terms from that ontology, e.g.

[16] https://github.com/draggett/arena-webhub

"`saref:LightSwitch`" and "`saref:ToggleCommand`". SAREF lacks a suitable term for an overheating lamp.

GATEKEEPER focuses on the healthcare of elderly and frail patients. For this, there are a variety of suitable ontologies that cover medical devices and medical records. These are being studied in Task 3.4 "Definition of the Semantic Models, Vocabularies & Registry".

Note: ***"edi{TD}or"***[17] is an open-source project for a tool for simply designing W3C Thing Descriptions and Thing models.

As illustrated in the above examples, additional contextual definitions, linked to namespaces, can be added to a Thing Description. This mechanism can be used to integrate additional semantics to the content of a Thing Description instance, provided that formal knowledge, such as logic rules for a specific domain of application, exists under the given namespace. The contextual information may also specify configurations and behaviour of the underlying communication protocols declared in the forms field.

4.5.2 FHIR

4.5.2.1 Overview

HL7 FHIR® [34] is a standard developed by HL7[18], based on emerging industry approaches. HL7 FHIR solutions are built on a set of modular components called **resources** that can easily be assembled into working systems to solve real-world clinical and administrative problems at a fraction of the price of existing alternatives. From a model-driven architecture perspective, FHIR resources are notionally equivalent to a physical model implemented in XML, JSON or Turtle RDF.

FHIR is described as a *'RESTful'* specification based on common industry level use of the term REST. The **FHIR API** describes the resources as a set of operations on resources where individual resource instances are managed in collections by their type. FHIR is not however limited to RESTful solutions, but it provides support to other interoperability paradigms as *document*-based, *messaging* or *services*.

At a high level, the FHIR standard can be decomposed in the following parts:

[17] https://github.com/eclipse/editdor

[18] https://hl7.org/

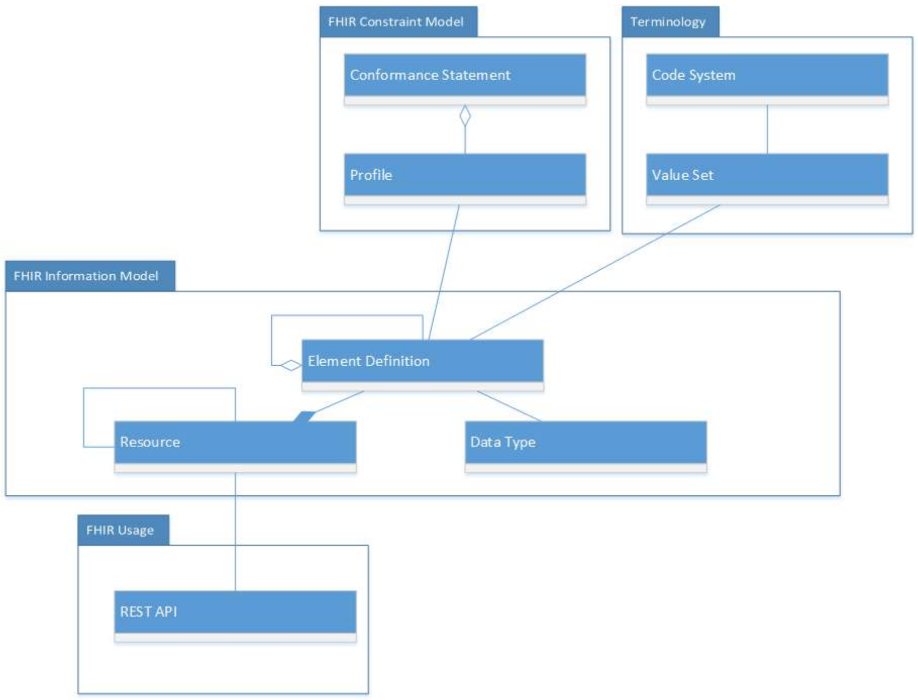


Figure 6 – Overview of the HL7 FHIR standard parts[19]

An Information Model – provides a common ontology for FHIR indicating how things are represented.

Constraints – describes how the FHIR resources can be used to fulfil specific business processes and needs. In the scope of GATEKEEPER, this is specified in the GATEKEEPER Implementation Guide (see GATEKEEPER deliverable D3.5 [10]).

Terminology – Terminology assets are often used by Conformance resources (Constraints) and may also include considerations on how different Value sets are mapped to each other.

Usage – the FHIR component that provides the run-time capacity for using FHIR, typically implemented by an FHIR server. An FHIR REST server is any software that implements the FHIR APIs and uses FHIR resources to exchange data.

Moreover, HL7 FHIR, even if mainly designed for data sharing, improves interoperability in general, for example:

By allowing the formalization of the Logical information Model through the FHIR framework. The FHIR resource used for specifying the "actual" FHIR resources and profiles (i.e. StructureDefinition) may in fact also be used to represent implementation-independent models (logical model) or other standards (as the HL7 CDA).

The FHIR Mapping Language[20] describes how one set of Directed Acyclic Graphs (an instance) is transformed into another set of directed acyclic graphs. It is not necessary for the instances to have formal declarations and/or be strongly typed – just that they have named children that themselves have properties. On the other

[19] Figure from https://hl7.org/fhir

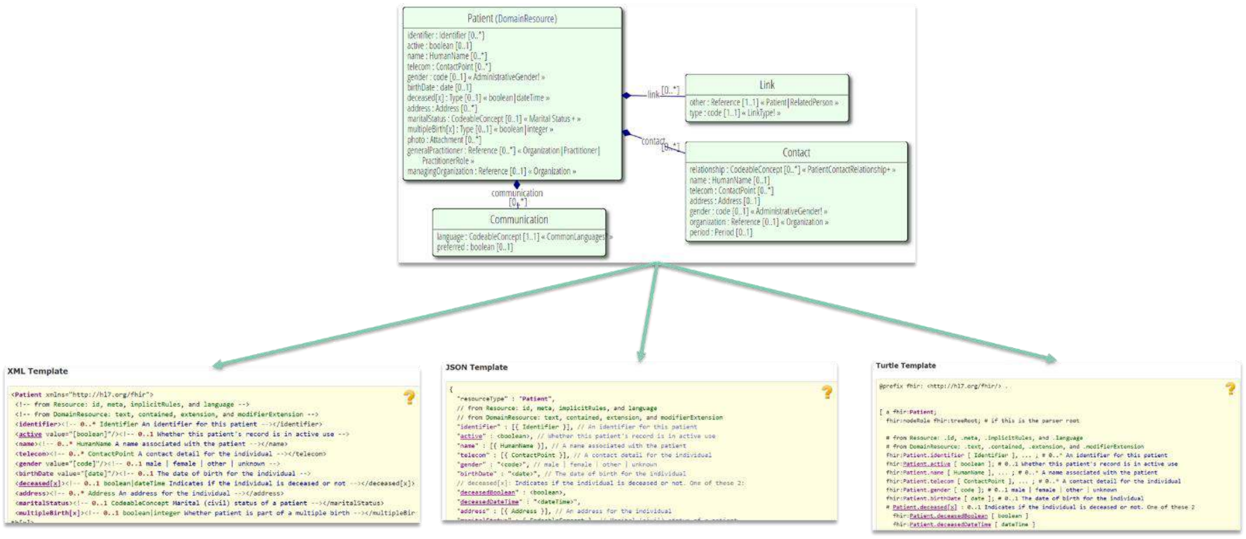
20 http://build.fhir.org/mapping-language.html

hand, when the instances are strongly typed, specifically when they have formal definitions that are represented as Structure Definitions, the mapping language can use additional type-related features.

Path-based navigation and extraction language FHIRPath[21]

4.5.2.2 Physical representation

As mentioned, the current released version (FHIR R4) supports three physical representations: XML, JSON and RDF Turtle.


Patient (DomainResource)
identifier : Identifier [0..*]
active : boolean [0..1]
name : HumanName [0..*]
telecom : ContactPoint [0..*]
gender : code [0..1] « AdministrativeGender! »
birthDate : date [0..1]
deceased[x] : Type [0..1] « boolean|dateTime »
address : Address [0..*]
maritalStatus : CodeableConcept [0..1] « Marital Status + »
multipleBirth[x] : Type [0..1] « boolean|integer »
photo : Attachment [0..*]
generalPractitioner : Reference [0..*] « Organization|Practitioner|PractitionerRole »
managingOrganization : Reference [0..1] « Organization »
link [0..*]
Link
other : Reference [1..1] « Patient|RelatedPerson »
type : code [1..1] « LinkType! »
contact [0..*]
Contact
relationship : CodeableConcept [0..*] « PatientContactRelationship+ »
name : HumanName [0..1]
telecom : ContactPoint [0..*]
address : Address [0..1]
gender : code [0..1] « AdministrativeGender! »
organization : Reference [0..1] « Organization »
period : Period [0..1]
communication [0..*]
Communication
language : CodeableConcept [1..1] « CommonLanguages+ »
preferred : boolean [0..1]
XML Template
JSON Template
Turtle Template

```
<Patient xmlns="http://hl7.org/fhir">
 <!-- from Resource: id, meta, implicitRules, and language -->
 <!-- from DomainResource: text, contained, extension, and modifierExtension -->
 <identifier><!-- 0..* Identifier An identifier for this patient --></identifier>
 <active value="[boolean]"/><!-- 0..1 Whether this patient's record is in active use -->
 <name><!-- 0..* HumanName A name associated with the patient --></name>
 <telecom><!-- 0..* ContactPoint A contact detail for the individual --></telecom>
 <gender value="[code]"/><!-- 0..1 male | female | other | unknown -->
 <birthDate value="[date]"/><!-- 0..1 The date of birth for the individual -->
 <deceased[x]><!-- 0..1 boolean|dateTime Indicates if the individual is deceased or not --></deceased[x]>
 <address><!-- 0..* Address An address for the individual --></address>
 <maritalStatus><!-- 0..1 CodeableConcept Marital (civil) status of a patient --></maritalStatus>
 <multipleBirth[x]><!-- 0..1 boolean|integer Whether patient is part of a multiple birth --></multipleBir
```

```
{
  "resourceType" : "Patient",
  // from Resource: id, meta, implicitRules, and language
  // from DomainResource: text, contained, extension, and modifierExtension
  "identifier" : [{ Identifier }], // An identifier for this patient
  "active" : <boolean>, // Whether this patient's record is in active use
  "name" : [{ HumanName }], // A name associated with the patient
  "telecom" : [{ ContactPoint }], // A contact detail for the individual
  "gender" : "<code>", // male | female | other | unknown
  "birthDate" : "<date>", // The date of birth for the individual
  // deceased[x]: Indicates if the individual is deceased or not. One of these 2:
  "deceasedBoolean" : <boolean>,
  "deceasedDateTime" : "<dateTime>",
  "address" : [{ Address }], // An address for the individual
```

```
@prefix fhir: <http://hl7.org/fhir/> .


[ a fhir:Patient;
  fhir:nodeRole fhir:treeRoot; # if this is the parser root

  # from Resource: .id, .meta, .implicitRules, and .language
  # from DomainResource: .text, .contained, .extension, and .modifierExtension
  fhir:Patient.identifier [ Identifier ], ... ; # 0..* An identifier for this patient
  fhir:Patient.active [ boolean ]; # 0..1 Whether this patient's record is in active use
  fhir:Patient.name [ HumanName ], ... ; # 0..* A name associated with the patient
  fhir:Patient.telecom [ ContactPoint ], ... ; # 0..* A contact detail for the individual
  fhir:Patient.gender [ code ]; # 0..1 male | female | other | unknown
  fhir:Patient.birthDate [ date ]; # 0..1 The date of birth for the individual
  # Patient.deceased[x] : 0..1 Indicates if the individual is deceased or not. One of these 2
    fhir:Patient.deceasedBoolean [ boolean ]
    fhir:Patient.deceasedDateTime [ dateTime ]
```

Figure 7 – Supported serialization formats in FHIR

4.5.2.3 Profiling and mapping

The HL7 FHIR standard provides a "platform specification" with a set of standard building blocks (a set of resources) for interoperability solutions, applicable for the many different contexts of the social and health care.

When these common building blocks are used in practice for a specific context of use, further adaptations of these blocks (i.e. the FHIR resources) are usually needed. This is often implicitly made by the implementers; but it is worth to have them formalized by conformance resources to better describe the actual "model" adopted, and allowing – among other things – validation of the instances. The process of "adapting" resources is usually called **profiling** and may imply constraints applied to the existing elements, datatypes, vocabularies, but also the definition of FHIR extensions where needed.

Usually, all these conformance resources are collected within an FHIR Implementation Guide (which is itself a FHIR resource) that may be used to generate human-readable guidance and computable conformance artefacts. An implementation guide is the result of an adaptive, iterative, and incremental approach starting from business requirements formalized into FHIR logical models and profiles.

21 https://hl7.org/fhirpath

Note: FHIR provides different mechanisms to support mappings, but other mapping languages can be adopted as well:

- a mapping language[22] describing how one set of Directed Acyclic Graphs (an instance) is transformed to another set of directed acyclic graphs is defined.
- A Map of relationships between 2 structures[23] that can be used to transform data.
- A statement of relationships[24] from one set of concepts to one or more other concepts - either concepts in code systems, or data element/data element concepts, or classes in class models.

4.5.2.4 Security Considerations

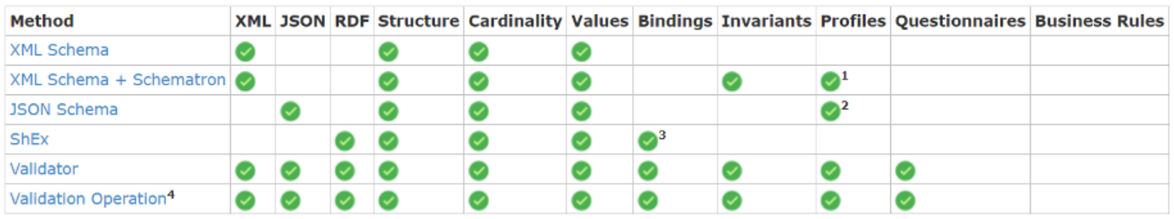
HL7 FHIR is not a security protocol and relies on other standards to realize the security layer. That means security and trust are to be addressed at the platform level in a FHIR based architecture. That said:

- Due to the sensitivity of health data, some considerations on security and privacy are documented in the standard[25], touching Time Keeping; Communications Security; Authentication/Authorization/Access Control; Audit; Digital Signatures; Attachments; Labels; Data Management Policies; Narrative; Input Validation.
- Some security-related resources (Provenance; AuditEvent; Consent) and resource metadata element (security labels) have been specified to facilitate the realization of this layer.
- a FHIR Implementation guide addressing how to use some of these security / privacy standards for FHIR applications has been published and used, see SMART on FHIR[26]

4.5.2.5 Validation

Validation of an FHIR instance may refer to different concepts: validation of the structure, validation of the physical representation; check of invariants; control of used terminologies (binding); compliance with a defined profile; and so on.

Several tools are available to address these different types of validation. The following schema, taken from the FHIR standard[27], summarizes what can be done with what.



Method	XML	JSON	RDF	Structure	Cardinality	Values	Bindings	Invariants	Profiles	Questionnaires	Business Rules
XML Schema	✓			✓	✓	✓					
XML Schema + Schematron	✓			✓	✓	✓		✓	✓[1]		
JSON Schema		✓		✓	✓	✓			✓[2]		
ShEx			✓	✓	✓	✓	✓[3]				
Validator	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Validation Operation[4]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

Figure 8 – Validation options described in the FHIR standard

[22] http://build.fhir.org/mapping-language.html

[23] http://build.fhir.org/structuremap.html

[24] http://build.fhir.org/conceptmap.html

[25] https://www.hl7.org/fhir/security.html

[26] https://smarthealthit.org/

[27] http://build.fhir.org/validation.html

Some notes:

- Schematron generated for a profile can test cardinality and invariants, but not bindings. Slicing is not well supported.
- JSON schema generated for a profile can test cardinality. Slicing is partially supported.
- ShEx can enforce some bindings for well understood terminologies, but this is an ongoing area of development
- How much validation is going to be performed on data is left at the discretion of the server. Most servers reuse the FHIR Java Validator, or code derived from it, and offer similar validation services. Some servers also expose a web interface.

The FHIR Java Validator typically offers the most complete validation solution, including validation against profiles. The latest version can be download from the downloads page[28]; together with the XML and JSON FHIR definitions; XML Validation Schemas (includes support schemas, resource schemas, modular & combined schemas, and Schematrons); JSON Schema[29]; ShEx Schemas (ShEx definitions for validating RDF resources); and GraphQL Schemas.

4.5.3 OpenAPI and other approaches for specifying APIs

There exist other mechanisms to specify APIs that attempt to tackle interoperability issues. A recent survey of network APIs[30] shows that REST-based approaches continue to dominate, with developers using OpenAPI [35] or JSON Schema [36] to specify message payloads. What differentiates the Web of Things approach from these other approaches is that it supports semantic interoperability through the means to describe APIs at the semantic level. Other approaches just cater for syntactic interoperability, leaving semantics to the accompanying human readable documentation. Machine interpretable semantics will be increasingly critical to scaling, both in terms of the numbers of APIs, and in terms of resilience in the face of change in response to evolving needs.

[28] http://build.fhir.org/downloads.html

[29] Needs JSON Schema draft-06 or a more recent version

[30] https://static1.smartbear.co/smartbearbrand/media/pdf/smartbear_state_of_api_2020.pdf

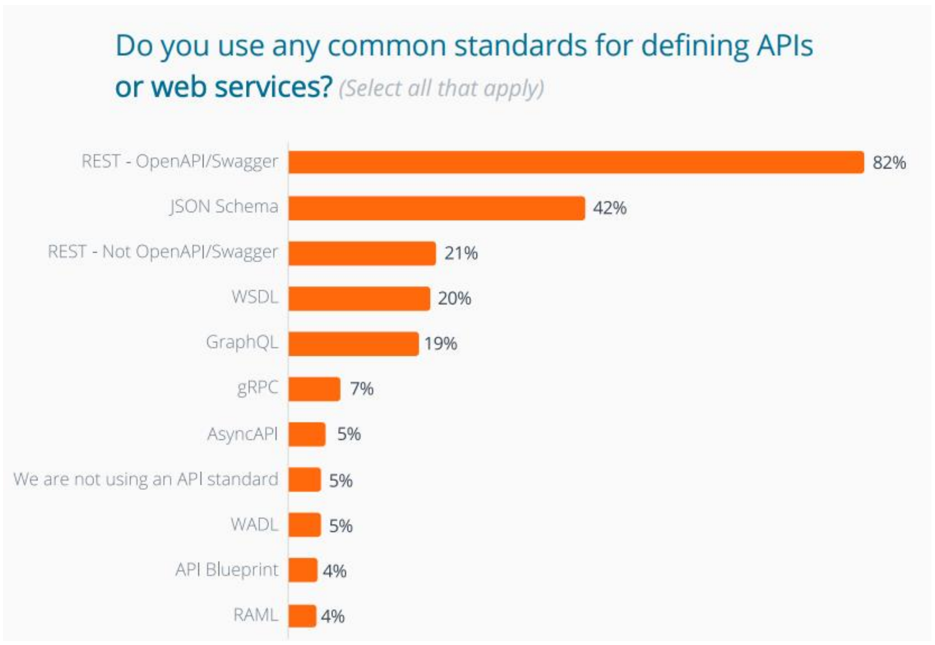


Figure 9: Result of survey used on standard for API description

API standards continue to emerge to meet new challenges, with some showing increased adoption. Year over year, the number of users selecting GraphQL and gRPC have shown dramatic gains, almost doubling for gRPC and just over a 50% gain for GraphQL. Yet, despite growth in these areas, the overwhelming choice of developers continues to be the REST OpenAPI Specification (Figure 9).

GATEKEEPER is betting on the adoption of Thing Descriptions, an open standard developed by the W3C, as it is the only existing approach that caters for machine interpretable semantics. In comparison, OpenAPI does not support semantic annotations through linked data.

That said, the large community around OpenAPI has developed numerous tools and library to simplify developers lives. There are open-source projects for building servers and clients based on OpenAPI in almost every programming language, and rich web testing environments are available.

5 Easing Interoperability in GATEKEEPER

This section provides recommendations for the use of identified in GATEKEEPER to improve interoperability aspects of the GATEKEEPER platform.

5.1 Web of Things (WoT)

5.1.1 Web of Things in GATEKEEPER

The Web of Things is used in GATEKEEPER to describe everything within the platform, building upon the idea behind Web of Things of providing digital twins for physical devices such as sensors, actuators, home appliances, etc. and expanding it to cover services exposed in the GATEKEEPER marketplace.

The GATEKEEPER platform will share information about its public components with Thing Descriptions. Each Thing Description will describe the digital twin related to the component. Within GATEKEEPER platform, digital services and data are also considered to be "physical" things and as such will have an associated Thing Description.

Each public GATEKEEPER component needs to register its Thing Description within the Thing Management System (TMS) [6] that interacts with the GATEKEEPER Trust Authority (GTA) [5] for the validation and certification of the candidate Thing Description. If the candidate Thing Description is valid and certified by the GTA, then it is registered and available through the TMS (Figure 10).

Anyone who wants to interact with the GATEKEEPER platform needs to go through the Thing Management System (TMS), which therefore also acts as a broker agent for the platform. In the Web of Thing terminology, this kind of broker service is referred to as an "intermediary"[31].

[31] See: https://www.w3.org/TR/wot-architecture/#dfn-intermediary

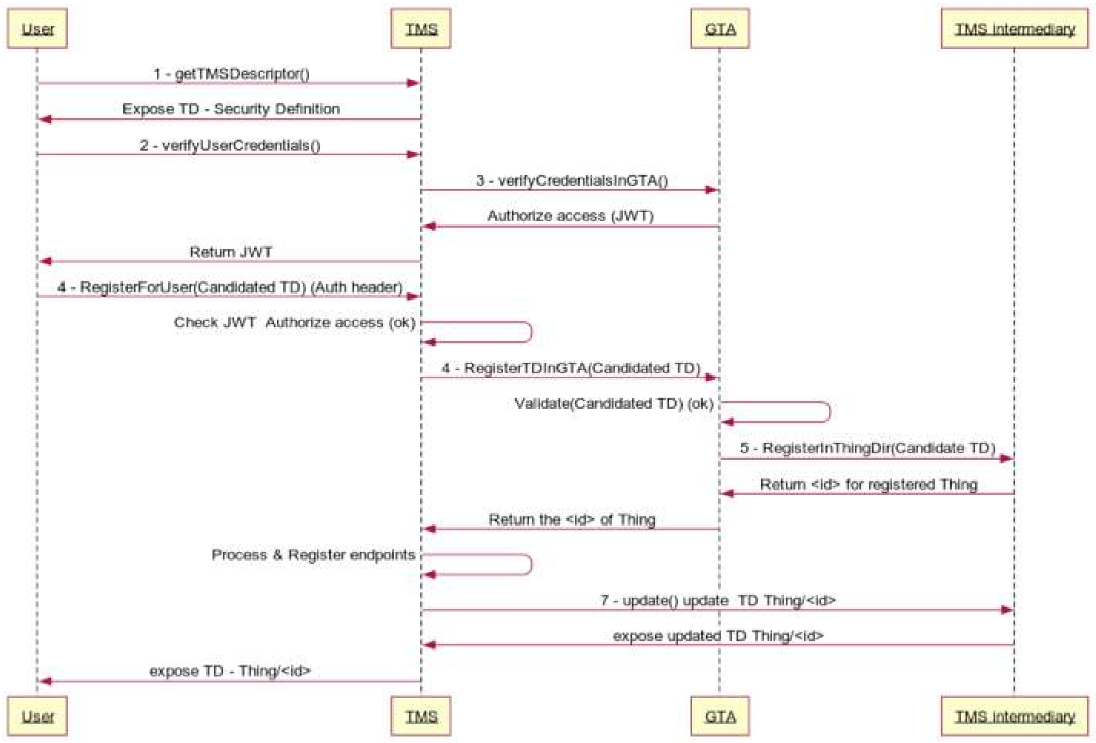

User
TMS
GTA
TMS intermediary
1 - getTMSDescriptor()
Expose TD - Security Definition
2 - verifyUserCredentials()
3 - verifyCredentialsInGTA()
Authorize access (JWT)
Return JWT
4 - RegisterForUser(Candidated TD) (Auth header)
Check JWT Authorize access (ok)
4 - RegisterTDInGTA(Candidated TD)
Validate(Candidated TD) (ok)
5 - RegisterInThingDir(Candidate TD)
Return <id> for registered Thing
Return the <id> of Thing
Process & Register endpoints
7 - update() update TD Thing/<id>
expose updated TD Thing/<id>
expose TD - Thing/<id>
User
TMS
GTA
TMS intermediary

Figure 10: Interaction between TMS and GTA for registration of a Thing Description

The TMS exposes a Thing Description of the GATEKEEPER platform to clients, that serves as entry point for the directory service that collects all the Thing Descriptions of public GATEKEEPER components. This means that, to be part of the GATEKEEPER platform, a core requirement is to register a valid Thing Description, compliant with the Web of Thing TD information Model and the GATEKEEPER information model. Through the TMS' Thing Description, it is possible to navigate and explore registered things in the platform using a web graph structure.

From the above, one can easily realize that GATEKEEPER things are as a subset of the Web of Things (Figure 11), where each thing is certified by the GATEKEEPER Trust Authority (GTA). Certification is an important innovation of the GATEKEEPER project: trust in GATEKEEPER is directly based on the certification process and the certification authority.

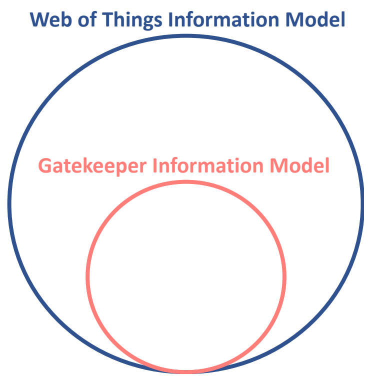


Figure 11: Relation between Web of Things and GATEKEEPER information models

Several aspects related to the capabilities of Thing Descriptions and to the Thing Description information model are provided in the next few sections, along with recommendations for GATEKEEPER platform. These recommendations will be used for the definition of the GATEKEEPER Information model in other WP3 tasks.

5.1.2 WoT and Semantic Descriptions

Recommendations for linking context and using JSON-LD within Thing Descriptions in GATEKEEPER are to:

- Always use https://www.w3.org/2019/wot/td/v1 as the main context for every Thing Description, so that Thing Descriptions can be validated against the Thing Description information model;
- Provide context information in a separate file. In other words, within GATEKEEPER, inline contexts are to be avoided. This will both help to preserve human readability and improve organization and reuse of common JSON-LD contexts;
- Limit the granularity of linked context. For instance, avoid linking atomic properties in multiple domains, and rather try to link entire objects whenever possible. This will keep a clean separation between domains, avoiding cases where one object is composed of fields coming from different ontologies. A multi-domain object should be a composition of concepts.

Regarding WoT and semantic descriptions, the General recommendations for the GATEKEEPER information model are that, for each standard, protocol, specification used in GATEKEEPER, one first needs to look at whether the Thing Description information model already includes it:

- If it is included in Thing Description information model, the existing vocabulary/ontology can be used for validation/certification purposes (for example the htv vocabulary for http messages).
- If there is no standard, protocol, or specification included in the TD information model, a specific vocabulary/ontology needs to be added to the GATEKEEPER information model. Two cases arise:
    - One or more vocabularies/ontologies already exist for the standard, protocol, or specification. If so, one of them needs to be selected and added to the GATEKEEPER information model.
    - No vocabulary/ontology already exists for the standard, protocol, or specification. If so, one needs to be defined and added to the GATEKEEPER information model.

The Thing Description information model describes the "`htv`" vocabulary (in the namespace "`http://www.w3.org/2011/http#`") that standardizes the HTTP protocol in the context of the Web of Things. HTTP is the most common protocol used in the development of REST interface for modelling a request/response communication pattern.

A Thing Description that describes its protocol binding on HTTP with the htv vocabulary can be checked and validated by a machine.

On the other hand, there are no specific recommendations within the Thing Description information model to describe a publish/subscribe messaging pattern (such as an event) for interacting with a thing. The owner of the Thing Description is free to use the vocabulary they want or to describe the binding protocol with plain text. However, this makes it impossible for a machine to understand the kind of interaction that is expected to happen automatically.

Within GATEKEEPER, an existing or novel vocabulary that describes protocols for publish/subscribe messaging pattern such as MQTT, CoAP, AMQP should be supported.

An initial set of vocabulary extensions to the Web of Things Thing Description [31] is under development in the W3C Web of Things Working Group. It is strongly recommended that these MQTT and CoAP binding templates get used within the GATEKEEPER Thing Description information model.

5.1.3 Certification of Thing Descriptions

Certification has an important role to play in enabling a thriving market of services.

- **Trust** – is the service supplier a trustworthy partner?
- **Quality** – does the service fulfil quality requirements, e.g. in respect to accuracy, availability and scalability?
- **Validation**- does the service description conform to the applicable specifications?

The GATEKEEPER platform provides support for this with the Secure Data sharing and Trusted Transactions Component along with well-defined and independently audited certification processes. Digital certificates are used to attest to the certification of services exposed through the GATEKEEPER marketplace.

Validation of Thing Descriptions involves verifying that they are valid JSON-LD documents, and pass tests that apply RDF shape constraints expressed with SHACL [37] or ShEx [38]. These constraints ensure that the Thing Descriptions conform to the W3C Recommendation for Thing Descriptions. Additional checks are used to ensure conformance to GATEKEEPER guidelines, e.g. in respect to JSON-LD contexts and protocol bindings. In principle, deeper checks could be made on the use of ontologies – do the semantic descriptions for services conform to the models described in the ontologies?

5.1.4 WoT and OpenAPI

GATEKEEPER is betting on the adoption of Thing Descriptions as it is the only existing approach that caters for machine-interpretable semantics, but the project also needs to leverage the numerous tools and libraries that exist around the OpenAPI specification.

As such, a key recommendation is that, within GATEKEEPER, the Web of Things approach must be harmonized with OpenAPI to take advantage of the existing available OpenAPI tools. The harmonization may yield novel features to OpenAPI, such as semantic support.

5.1.5 Competing approaches for specifying APIs

The Thing Description of Web of Things is a new standard for specifying APIs that faces competition from existing standards such as GraphQL, gRPC, RAML and OpenAPI. A recent survey of network APIs[32] shows that the REST framework continues to dominate, with developers using OpenAPI [35] or JSON Schema [36] to specify message payloads. What differentiates the Web of Things is that it supports semantic interoperability through the means to describe APIs at the semantic level. Other approaches just cater for syntactic

[32] https://static1.smartbear.co/smartbearbrand/media/pdf/smartbear_state_of_api_2020.pdf

interoperability, leaving semantics to the accompanying human-readable documentation. Machine interpretable semantics will be increasingly critical to scaling, both in terms of the numbers of APIs, and in terms of resilience in the face of change in response to evolving needs.

API standards continue to emerge to meet new challenges, with some showing increased adoption. Year over year, the number of users selecting GraphQL and gRPC have shown dramatic gains, almost doubling for gRPC and just over a 50% gain for GraphQL. Yet, despite growth in these areas, the overwhelming choice of developers continues to be the REST OpenAPI Specification (Figure 9).

GATEKEEPER is betting on the adoption of Thing Descriptions, an open standard developed by the W3C, as it is the only existing approach that caters for machine-interpretable semantics. In comparison, OpenAPI does not support semantic annotations through linked data.

That said, the large community around OpenAPI has developed numerous tools and library to simplify developers lives. There are open-source projects for building servers and clients based on OpenAPI in almost every programming language, and rich web testing environments are available.

As such, a key recommendation is that, within GATEKEEPER, the Web of Things approach must be harmonized with OpenAPI to take advantage of the existing available OpenAPI tools. The harmonization may yield novel features to OpenAPI, such as semantic support.

5.1.6 WoT Service Marketplace

The Marketplace is a one-stop-shop to publish, discover, and monetize offerings of the GATEKEEPER project in the general field of health support services. Offerings in that sense can be Applications (developed in native platforms and ready to be deployed, e.g. on Android, iOS), Devices (wellness, sensors and IoT devices developed in the project) and Services/APIs (published on the TMS following the Web of Things paradigm).

A publisher must be able to describe its offering precisely so that it can easily be discovered by a consumer. In that sense, the need for interoperability arises at the metadata level to improve discoverability. This includes metadata such as:

- Category, field of application, purpose, ailment etc.
- Tools and frameworks used in the implementation
- Platform for deployment
- Current uses in pilot sites, countries etc.

A related ontology, coupled with a way to discover Apps, has previously been proposed in the ACTIVAGE Marketplace [39] where an ontology provides a hierarchy for such metadata and relationships between them and a hybrid algorithm combines logic-based reasoning and text-similarity to provide both closest matches and alternative Apps in response to user queries. The ACTIVAGE Marketplace Ontology is shown on Figure 12.

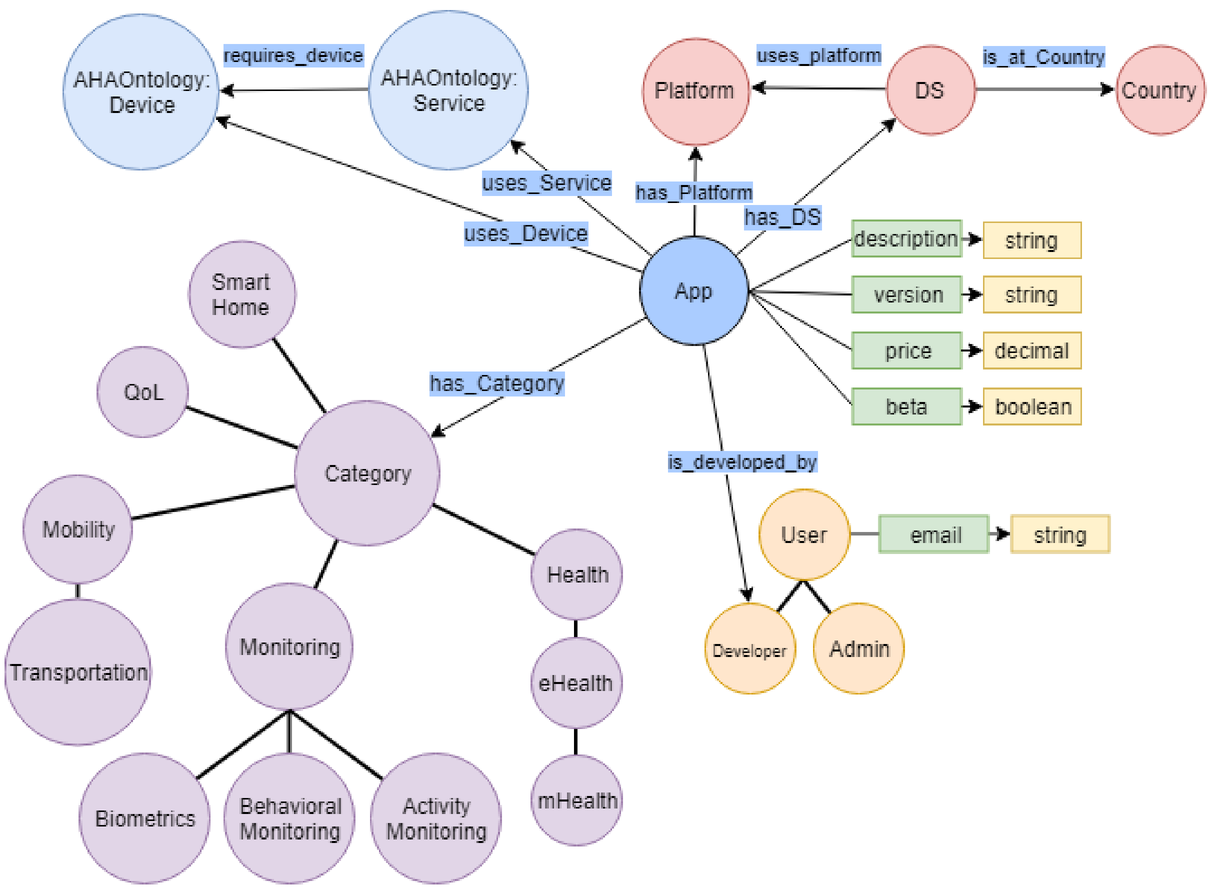


Figure 12. The ACTIVAGE Marketplace ontology: metadata for App discovery

While the ACTIVAGE Marketplace ontology only covers metadata for Apps, it can be extended and enriched to cover a wider spectrum of Apps, Devices and Things/Services in GATEKEEPER. New algorithms for discovery can also be proposed based on the new metadata.

The Marketplace mock-ups, designed in T4.6 Marketplace Services and already being implemented, illustrate initial needs for metadata to publish and monetize offerings. Figure 13 shows the full catalogue of offerings, along with the ability to search and filter them with fixed filters (language, tools, etc.), numeric filters (e.g. price range), and also based on keywords. Figure 14 shows some initial metadata for offerings such as price, comments, ratings, average rating, or number of downloads.

Some of the metadata is intrinsic to the Marketplace platform, meaning that it will be managed by the Marketplace itself:

- Comments
- Downloads
- Ratings, Average Rating, etc.

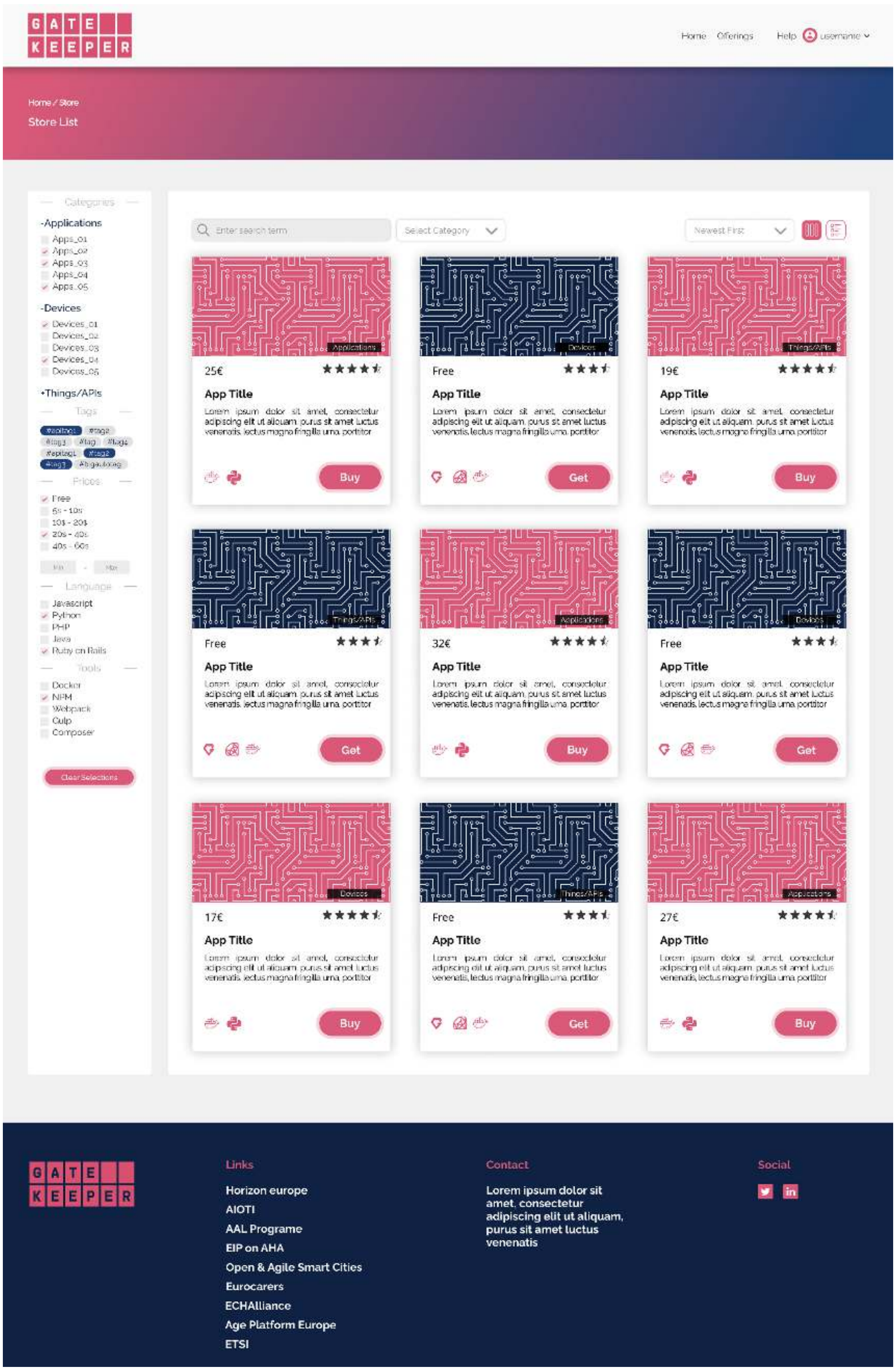
Some of the metadata needs to be specified by the service publisher:

- Price
- Model for access, e.g. purchase (pay once), subscription (recurring), pay by volume, etc.

In some cases, metadata could be handled on either side:

- Category, field of use, purpose, targeted ailment
- Tags, keywords for search

In such cases, as much as practical, the general recommendation for the GATEKEEPER marketplace is to push metadata on the service publisher side. This gives the marketplace less flexibility to innovate in terms of categorization and structure, because a change in mandatory metadata would require all service publishers to update the metadata of their respective services, thereby breaking interoperability of the Thing Descriptions that describe the service. However, it will allow the marketplace to grow organically based on the information provided by the service publisher directly, meaning marketplace administrators can focus on content moderation, instead of having to create the content in the first place. This approach is consistent with existing application stores (e.g. Apple's App store and Google's Play store).



Categories

-Applications
- Apps_01
- ✓ Apps_02
- ✓ Apps_03
- Apps_04
- ✓ Apps_05

-Devices
- ✓ Devices_01
- Devices_02
- Devices_03
- ✓ Devices_04
- Devices_05

+Things/APIs

Tags: #apitag1 #tag2 #tag3 #tag #tag4 #apitag1 #tag2 #tag3 #bigautotag

Prices
- ✓ Free
- 5$ - 10$
- 10$ - 20$
- ✓ 20$ - 40$
- 40$ - 60$

Min - Max

Language
- Javascript
- ✓ Python
- PHP
- Java
- ✓ Ruby on Rails

Tools
- Docker
- ✓ NPM
- Webpack
- Gulp
- Composer

Clear Selections

Category	Price	Title	Description	Button
Applications	25€	App Title	Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor	Buy
Devices	Free	App Title	Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor	Get
Things/APIs	19€	App Title	Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor	Buy
Things/APIs	Free	App Title	Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor	Get
Applications	32€	App Title	Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor	Buy
Devices	Free	App Title	Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor	Get
Devices	17€	App Title	Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor	Buy
Things/APIs	Free	App Title	Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor	Get
Applications	27€	App Title	Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor	Buy

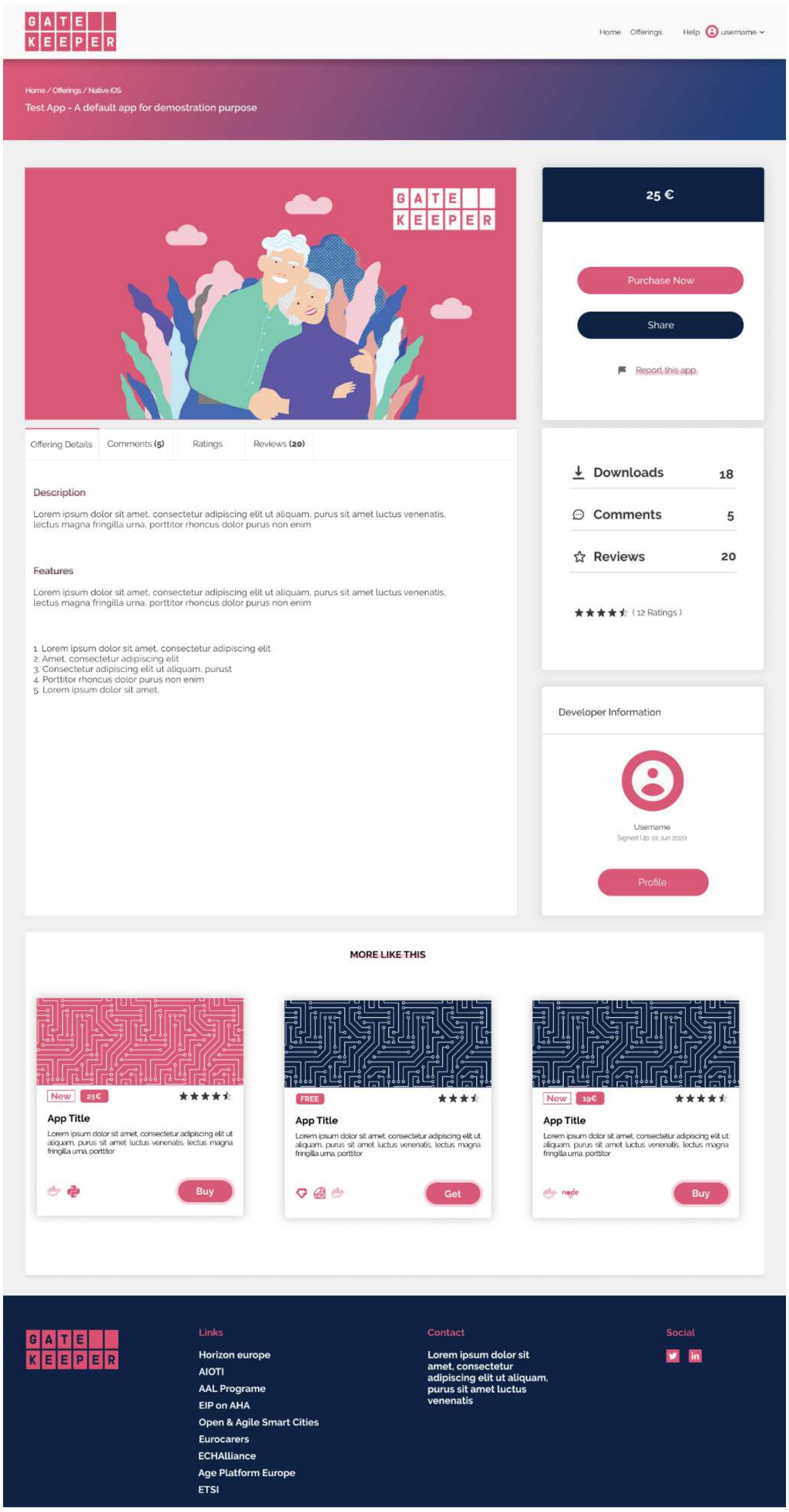
Links
- Horizon europe
- AIOTI
- AAL Programe
- EIP on AHA
- Open & Agile Smart Cities
- Eurocarers
- ECHAlliance
- Age Platform Europe
- ETSI

Contact

Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis

Social

Figure 13. GATEKEEPER Marketplace mockup for Store List Catalogue Search and Discovery



Home / Offerings / Native iOS

Test App - A default app for demostration purpose

25 €

Description

Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor rhoncus dolor purus non enim

Features

Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor rhoncus dolor purus non enim

1. Lorem ipsum dolor sit amet, consectetur adipiscing elit
2. Amet, consectetur adipiscing elit
3. Consectetur adipiscing elit ut aliquam, purust
4. Porttitor rhoncus dolor purus non enim
5. Lorem ipsum dolor sit amet.

Downloads	18
Comments	5
Reviews	20

★★★★½ (12 Ratings)

Developer Information

Username
Signed Up: 01 Jun 2020

MORE LIKE THIS

New 25€ ★★★★½
App Title
Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor

FREE ★★★½
App Title
Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor

New 19€ ★★★★½
App Title
Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor

Horizon europe
AIOTI
AAL Programe
EIP on AHA
Open & Agile Smart Cities
Eurocarers
ECHAlliance
Age Platform Europe
ETSI

Lorem ipsum dolor sit amet, consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis

Figure 14. GATEKEEPER Marketplace mockup for a single Offering View with metadata

5.2 HL7 FHIR

5.2.1 Physical representation

The current released version (FHIR R4) supports three physical representations: XML, JSON and RDF Turtle. The GATEKEEPER FHIR API should accept any of these three representations to support the widest possible range of originator/consumers, noting that JSON, and more significantly XML, are substantially supported by all implementations, while support for RDF Turtle for data exchange is less common.

There is an on-going collaboration activity between the HL7 RDF for Semantic Interoperability group (in the HL7 ITS work group) and the W3C Semantic Web in Health Care and Life Sciences Community Group to specify a JSON-LD 1.1 representation of FHIR resources. However, this activity has not been finalized and is not yet part of the continuous integration FHIR build[33]. Hopefully, it will be part of the future FHIR R5 version.

5.2.2 Persistency

A FHIR server may adopt very different strategies for implementing persistency of FHIR resources: applications can use the resources defined by FHIR by storing them natively in a database or persistent store; or store the information found in the resources in some internal format, and only use resources natively when exchanging with other applications. Some applications take a hybrid approach - storing FHIR resources natively and storing a well-controlled subset of information in some expressly designed persistent scheme, or any other combination of these approaches. The balance between these is driven by the stability of information requirements for different parts of the system. If resources are stored natively, particular attention has to be put in the management of the references: that can be absolute or relative URLs, they can be version specific or not, they might or might not resolve in the local system.

For the scope of the GATEKEEPER solution, without imposing any specific solution for the persistency layer, the *hybrid approach* is suggested. Possibly preserving the received FHIR resources, after having correctly resolved and managed the references, and hopefully remapping the received information into the persistency layer in accordance with the agreed logical model. How it is physically represented may vary depending on the persistency solution adopted.

5.2.3 Profiling and mapping

The key to creating a suitable environment to process FHIR resources in GATEKEEPER is to create a FHIR Implementation Guide that collects conformance resources and enables automatic validation of data. An adaptive, iterative, and incremental developing approach is used to develop this Implementation Guide, starting from the business requirements and formalizing into FHIR logical models and profiles.

[33] http://build.fhir.org

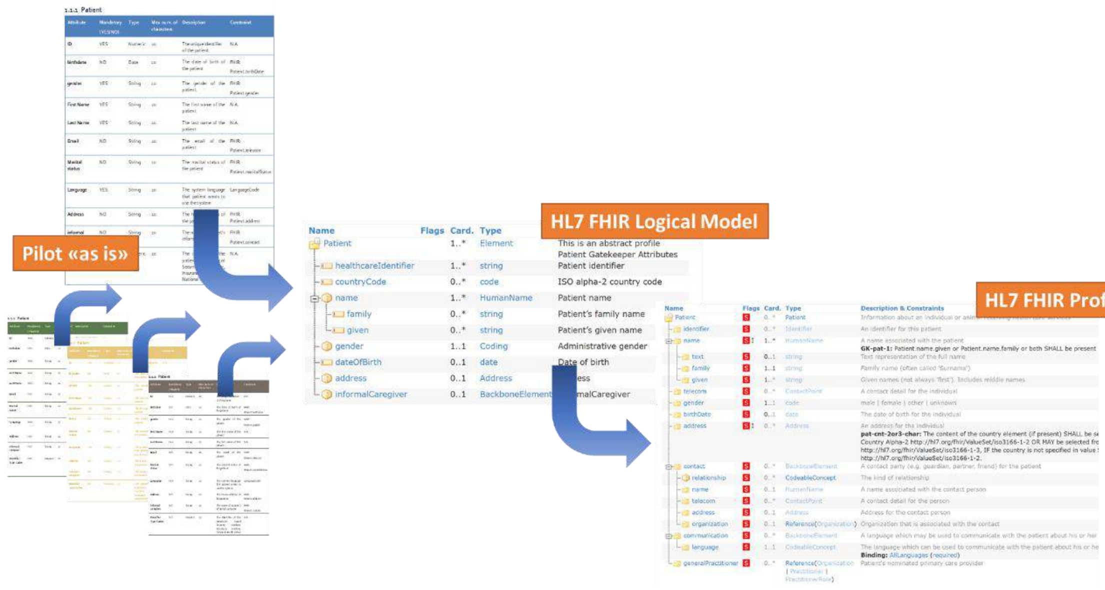
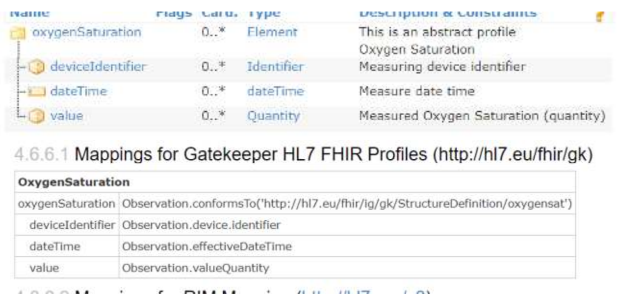


Figure 15 – From requirements to FHIR profiles

As new information needs are collected, these requirements are captured as FHIR logical model and FHIR resources and standardized profiles are selected and further profiled to fulfil GATEKEEPER's business expectations.

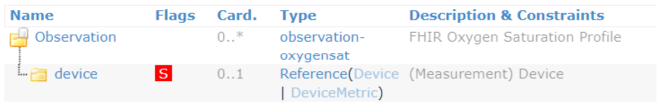


Name	Flags	Card.	Type	Description & Constraints
oxygenSaturation		0..*	Element	This is an abstract profile Oxygen Saturation
deviceIdentifier		0..*	Identifier	Measuring device identifier
dateTime		0..*	dateTime	Measure date time
value		0..*	Quantity	Measured Oxygen Saturation (quantity)

4.6.6.1 Mappings for Gatekeeper HL7 FHIR Profiles (http://hl7.eu/fhir/gk)

OxygenSaturation	
oxygenSaturation	Observation.conformsTo('http://hl7.eu/fhir/ig/gk/StructureDefinition/oxygensat')
deviceIdentifier	Observation.device.identifier
dateTime	Observation.effectiveDateTime
value	Observation.valueQuantity

Figure 16 – FHIR logical model for oxygenSaturation and mapping to the FHIR profile

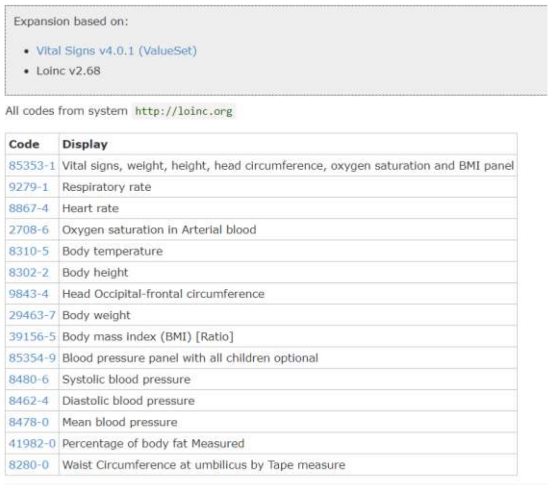


Name	Flags	Card.	Type	Description & Constraints
Observation		0..*	observation-oxygensat	FHIR Oxygen Saturation Profile
device	S	0..1	Reference(Device | DeviceMetric)	(Measurement) Device

Figure 17 – GATEKEEPER FHIR profile for oxygenSaturation

Therefore, these profiles provide a detailed description of how FHIR resources should be used in GATEKEEPER, allowing to validate instances and map models automatically.

To support the project, the GATEKEEPER FHIR Implementation Guide also aims at documenting the list of coded concepts used in the local settings as FHIR value set, providing, where possible, concept maps from these local settings to the commonly agreed value sets. Some examples are provided below.



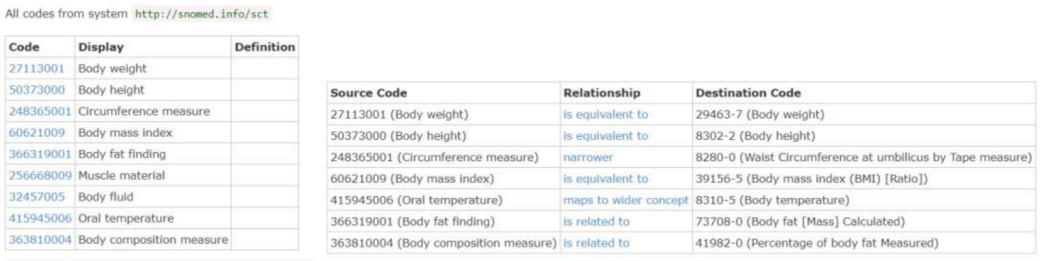
Expansion based on:

- Vital Signs v4.0.1 (ValueSet)
- Loinc v2.68

All codes from system `http://loinc.org`

Code	Display
85353-1	Vital signs, weight, height, head circumference, oxygen saturation and BMI panel
9279-1	Respiratory rate
8867-4	Heart rate
2708-6	Oxygen saturation in Arterial blood
8310-5	Body temperature
8302-2	Body height
9843-4	Head Occipital-frontal circumference
29463-7	Body weight
39156-5	Body mass index (BMI) [Ratio]
85354-9	Blood pressure panel with all children optional
8480-6	Systolic blood pressure
8462-4	Diastolic blood pressure
8478-0	Mean blood pressure
41982-0	Percentage of body fat Measured
8280-0	Waist Circumference at umbilicus by Tape measure

Figure 18 – GATEKEEPER Vital signs value set

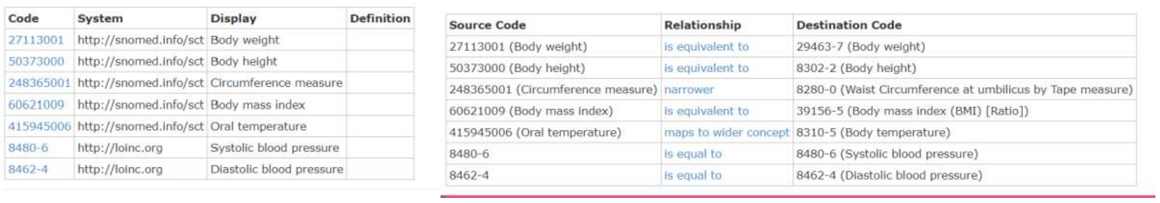


All codes from system `http://snomed.info/sct`

Code	Display	Definition
27113001	Body weight	
50373000	Body height	
248365001	Circumference measure	
60621009	Body mass index	
366319001	Body fat finding	
256668009	Muscle material	
32457005	Body fluid	
415945006	Oral temperature	
363810004	Body composition measure	

Source Code	Relationship	Destination Code
27113001 (Body weight)	is equivalent to	29463-7 (Body weight)
50373000 (Body height)	is equivalent to	8302-2 (Body height)
248365001 (Circumference measure)	narrower	8280-0 (Waist Circumference at umbilicus by Tape measure)
60621009 (Body mass index)	is equivalent to	39156-5 (Body mass index (BMI) [Ratio])
415945006 (Oral temperature)	maps to wider concept	8310-5 (Body temperature)
366319001 (Body fat finding)	is related to	73708-0 (Body fat [Mass] Calculated)
363810004 (Body composition measure)	is related to	41982-0 (Percentage of body fat Measured)

Figure 19 – Vital signs value set for the Greek pilot and associated concept map



Code	System	Display	Definition
27113001	http://snomed.info/sct	Body weight	
50373000	http://snomed.info/sct	Body height	
248365001	http://snomed.info/sct	Circumference measure	
60621009	http://snomed.info/sct	Body mass index	
415945006	http://snomed.info/sct	Oral temperature	
8480-6	http://loinc.org	Systolic blood pressure	
8462-4	http://loinc.org	Diastolic blood pressure	

Source Code	Relationship	Destination Code
27113001 (Body weight)	is equivalent to	29463-7 (Body weight)
50373000 (Body height)	is equivalent to	8302-2 (Body height)
248365001 (Circumference measure)	narrower	8280-0 (Waist Circumference at umbilicus by Tape measure)
60621009 (Body mass index)	is equivalent to	39156-5 (Body mass index (BMI) [Ratio])
415945006 (Oral temperature)	maps to wider concept	8310-5 (Body temperature)
8480-6	is equal to	8480-6 (Systolic blood pressure)
8462-4	is equal to	8462-4 (Diastolic blood pressure)

Figure 20 – Vital signs value set for the Basque country pilot and associated concept map

5.2.4 WoT and FHIR

Mapping FHIR resources to Thing Descriptions should be seen at two levels. The first level is representing things that create/manipulate FHIR resources with a Thing Description. This representation can be kept relatively simple. For instance, to represent a device that produces FHIR Observations, one could describe this generation as an action, referring to the FHIR namespace to describe the resource type:

```
{
   "@context": [
      "https://www.w3.org/2019/wot/td/v1",
      {
         "fhir": "http://hl7.org/fhir/"
      }
   ],
   "actions": {
      "createObservation": {
         "@type": "fhir:Observation",
         "forms": [
            {
                "op": [
                    "invokeaction"
                ],
                "href": "http://hapi.fhir.org/baseR4/Observation",
                "security": […],
                "contentType": "application/fhir+json",
                "htv:methodName": "POST"
            }
        ]
    }
    …
}
}
```

To ingest data from FHIR resources and, more importantly, to reason about such data among other types of data, a second level is needed that requires going deeper in the description of FHIR resource properties using formal ontologies. Services developed for the GATEKEEPER platform should be able to process data ingested from FHIR resources without having to ship code logic dedicated to FHIR resource processing. In other words, ingestion of FHIR resources within the GATEKEEPER platform should create machine-interpretable data.

The FHIR specification defines FHIR resource using a syntactic data model based on a template structure. Interpreting the meaning of a FHIR resource requires human interpretation. To create machine-interpretable representations, one needs to transition from syntactic models based on serialisation formats to formal semantic models that are serialization-agnostic. In other words, FHIR resources need to be described using a formal ontology.

HL7 developed a mapping between the terms used in FHIR resources and RDF URIs. Looking forward, the recommendations for the GATEKEEPER project are to look into formalizing these terms as an OWL ontology [25] and explore the use of SHACL [37] and/or ShEx [38] to express further constraints on the resulting data graph.

5.3 Data federation and Data Space Connectors

The ultimate goal of Data Federation is to achieve both syntactic and semantic interoperability of health data coming from different sources within and across pilots, so that they can be easily consumed and correlated.

The Data Federation process can be logically divided into two steps, corresponding to separate components:

- The Integration Engine, performing the syntactic and semantic mappings that transform input data using the GATEKEEPER FHIR profile defined in the project.
- The platform's FHIR server, where federated data are stored. These two services are exposed, following the Web of Things paradigm, with Thing Descriptions.

Next sub-sections describe the Thing Descriptions of each service and discuss the choices made to enrich the semantic context for each of them. Alternative approaches for the formalization of the FHIR server as a Thing Description are presented.

5.3.1 Thing Description – Integration Engine

The Integration Engine has two main actions that trigger the transformation of data, one for the healthcare related data, the other to deal with IoT devices. The Thing Description that describes the Integration Engine is described in the following figure. This Thing

Description is the result of the translation of the service OpenAPI definition applying the process defined by UPM.

To fully exploit the potential of the Web of Things approach, it would be beneficial to add a more precise semantic meaning to the objects described in the *@context* field.

```
{
    "@context": [
        "https://www.w3.org/2019/wot/td/v1",
        {
            "lst": "https://shop4cf.lst.tfo.upm.es/",
            "xsd": "http://www.w3.org/2001/XMLSchema#",
            "IDataBundle": {
                "@id": "lst:IDataBundle",
                "@context": {
                    "lst": "https://shop4cf.lst.tfo.upm.es/",
                    "xsd": "http://www.w3.org/2001/XMLSchema#"
                }
            }
        }
    ],
    "properties": {},
    "actions": {
        "ehr": {
            "@type": "IDataBundle",
            "uriVariables": {
                "pilot": {
                    "type": "string"
                }
            },
            "forms": [
                {
                    "op": [
                        "invokeaction"
                    ],
                    "href": "http://localhost:8087/gkie/EHR/data/{pilot}",
                    "security": [
                        {
                            "bearerAuth": []
                        }
                    ],
                    "contentType": "application/json",
                    "htv:methodName": "POST"
                }
            ]
        },
        "iot": {
            "@type": "IDataBundle",
            "uriVariables": {
                "pilot": {
                    "type": "string"
                }
            },
            "forms": [
                {
                    "op": [
                        "invokeaction"
                    ],
                    "href": "http://localhost:8087/gkie/IOT/data/{pilot}/{sensorID}",
                    "security": [
                        {
                            "bearerAuth": []
                        }
                    ],
                    "contentType": "application/json",
                    "htv:methodName": "POST"
                }
            ]
        }
    },
    "title": "Data Sources Integration - API",
    "version": {
        "instance": "v1"
    },
    "security": [
        "bearerAuth"
    ],
    "securityDefinitions": {
        "bearerAuth": {
            "scheme": "bearer",
            "format": "JWT"
        }
    }
}
```

Figure 21: Integration Engine Thing Description

To fully exploit the potential of the WoT approach, it would be beneficial to add a more precise semantic meaning to the objects described in the *@context* field of the TD above. To better specify the IDataBundle concept in particular, a proposal could be to use the ontology Linked Health Resources (from now on LHR). [40]

Linked Health Resources is a Web Ontology Language (OWL)-based ontology that models together health data from FHIR standard implemented services, health services and WoT services implemented Web APIs (satisfy certain REST constraints), formal ontology described WoT services and Linked Data: heterogeneous health data are modeled as conceptual information resources. The LHR ontology is shown in Figure 22.

This representation comes from W3C recommendation standards, RDF Schema data modelling vocabulary and the OWL ontology language.

Through the analysis of the graph it is possible to observe the relationship among person, resource and service. In fact service (*lhr:Service*) and person *(lhr:Person)* have resources that are interlinked and that allow to define them as sub-classes of class resource (*lhr:Resource* in the graph). In this way:

- *lhr:Resource* is the superclass of all the involved resources
- *lhr:Service* represents a set of resource in a relatively separated territory to indicate a source of health data
- *lhr:Person* represents a person who owns all the health resources (i.e. *lhr:HealthResource*).
- *lhr:HealthResource* represents all a person are in turn linked from health web services and FHIR implemented services.
- *lhr:EnvResource represents all a person instance's home environment resources that are interlinked from ordinary WoT services and formal description implemented services*

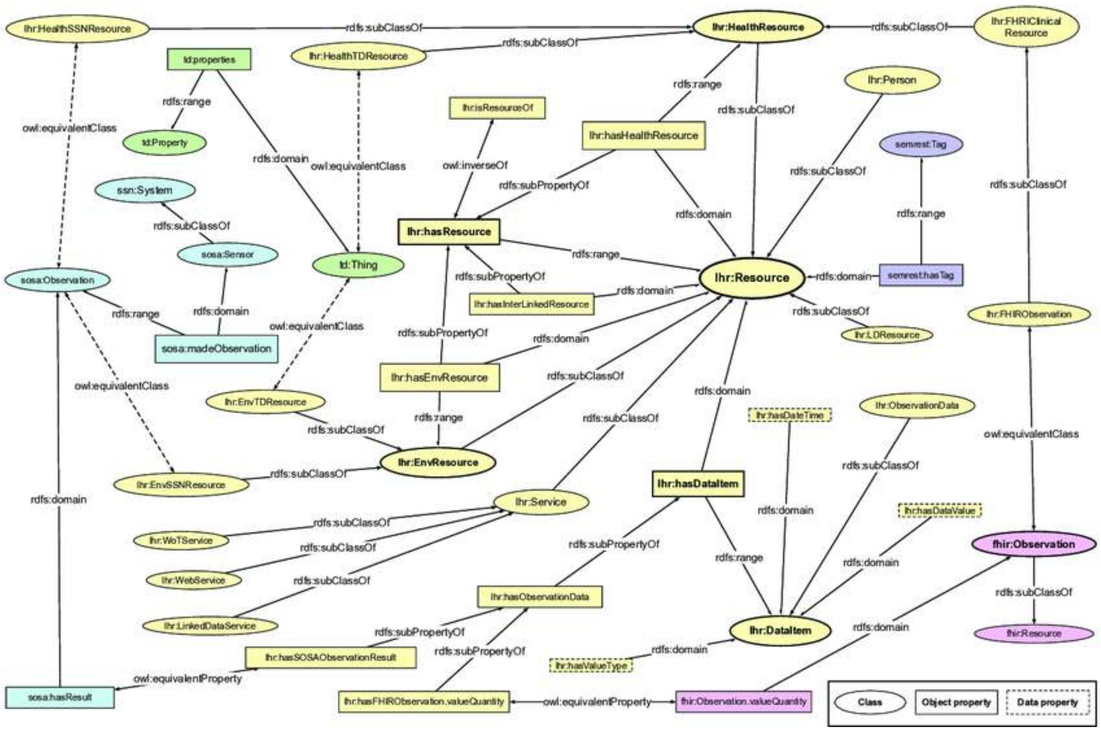


Figure 22: Linked Health Resources Ontology

- *lhr:DataItem* represent the data model of a *lhr:Resource* if it has no predefined model, and class *lhr:ObservationData* is used to represent the health data model of a *lhr:HealthResource*. Resources from FHIR service have predefined data models in RDF, therefore, the object of which is mapped to an instance of *lhr:ObservationData*. The relation is represented by a sub-property of the object property *lhr:hasObservationData*

Therefore, to make consistent the mapping also FHIR has been described according to LHR ontology. For instance, *fhir:Observation* is mapped to *lhr:FHIRObservation*, which is a sub-class of *lhr:HealthResource* (s. items in purple in Figure 22).

IDataBundle concept of the IntegrationEngine TD could thus map to *lhr:HealthResource.* An excerpt of the updated context is the following.

```
"@context": [
        "https://www.w3.org/2019/wot/td/v1",
        {
            "lst": "https://shop4cf.lst.tfo.upm.es/",
            "xsd": "http://www.w3.org/2001/XMLSchema#",
            "IDataBundle": {
                "@id": "lhr:HealthResource",
                "@context": {
                    "lhr": "https://www.example.com/lhr-schema#",
                "xsd": "http://www.w3.org/2001/XMLSchema#"
                }
            }
        }
    ],
```

Figure 23: Integration Engine TD updated context

5.3.2 Thing Description – FHIR Server

A FHIR Server is not a simple database, but a complex object that already carries well defined semantic information. To construct a meaningful Thing Description from the FHIR Server specification, several alternatives were considered, based on the server's REST interface and the embedded FHIR Capability Statement[34].

In the first Thing Description (from D4.2 [6]) below, functionalities are implicit, and are delegated to the capability statement referenced in the *conformance* field.

```
{
    "@context": [
      "https://www.w3.org/2019/wot/td/v1",
      {
          "xsd": "http://www.w3.org/2001/XMLSchema#",
          "td": "https://www.w3.org/2019/wot/td#",
          "FHIRServer":  {"@id": "td:Thing"},
          "conformance": {"@id": "xsd:anyURI"}
      }
    ],
    "@type":"FHIRServer",
    "title": "GATEKEEPER pilot x FHIR server",
    "description" : "A FHIR server implementation",
    "security": ["oauth2_sc"],
    "securityDefinitions": {
        "oauth2_sc": {
            "scheme": "oauth2",
            "authorizationUrl": "http://192.168.23.131:32017/auth/realms/Fhir-
test/account",
            "flow": "code"
        }
    },
    "conformance": "http://192.168.23.131:32021/hapi-fhir-
jpaserver/fhir/metadata"
}
```

Figure 24: FHIR Server TD (from D4.2 [6])

The opposite approach follows the same approach applied to the creation of the Integration Engine TD and starts from the OpenAPI description to define the corresponding properties / actions. This translation has been further refined to add semantic information to the concepts listed in the Thing Description, mapping the terms listed in the properties and actions to the ones from the FHIR specification. For this reason, the `@context` field was extended to add the additional namespace of the FHIR specification and add the domain-specific semantics of FHIR. This addition allows to enrich the vocabulary terms of the Thing Description with all the terms from the FHIR specification. This enriched vocabulary has been used in properties and actions to link the terms. The `@type` field is the place where the association to the FHIR terms has been actually made.

An excerpt of the resulting Thing Description is shown below. Due to the number of operations available in a FHIR Server, the resulting Thing Description is very long. For the sake of readability, the excerpt only shows the actions and operations related to Observations.

[34] https://www.hl7.org/fhir/capabilitystatement.html

```
{
   "@context": [
      "https://www.w3.org/2019/wot/td/v1",
      {
         "lst": "https://shop4cf.lst.tfo.upm.es/",
         "xsd": "http://www.w3.org/2001/XMLSchema#",
         "fhir": "http://hl7.org/fhir/"
      }
   ],
   "properties": {
      …
      "getObservations": {
         "@type": "fhir:Observation",
         "type": "array",
         "readOnly": false,
         "writeOnly": false,
         "forms": [
            {
               "op": [
                  "readproperty"
               ],
           "href": "http://hapi.fhir.org/baseR4/Observation",
           "security": [
              {
                 "fhir_server_auth": [
                    "write:resource",
                    "read:resource"
                 ]
              }
           ],
           "contentType": "application/fhir+json",
           "htv:methodName": "GET"
        }
      ]
   },
   "getObservation": {
      "@type": "fhir:Observation",
      "readOnly": false,
      "writeOnly": false,
      "uriVariables": {
         "id": {
            "type": "string"
         }
      },
      "forms": [
         {
            "op": [
               "readproperty"
            ],
            "href": "http://hapi.fhir.org/baseR4/Observation/{id}",
            "security": [
               {
                  "fhir_server_auth": [
                     "write:resource",
                     "read:resource"
                  ]
               }
            ],
            "contentType": "application/fhir+json",
            "htv:methodName": "GET"
         }
      ]
   },
   "getObservationHistory": {
      "@type": "fhir:Observation",
      "readOnly": false,
      "writeOnly": false,
      "uriVariables": {
         "id": {
            "type": "string"
         }
      },
      "forms": [
         {
            "op": [
               "readproperty"
               ],
               "href": "http://hapi.fhir.org/baseR4/Observation/{id}/_history",
               "security": [
                  {
                     "fhir_server_auth": [
                        "write:resource",
                        "read:resource"
                     ]
                  }
               ],
               "contentType": "application/fhir+json",
               "htv:methodName": "GET"
            }
         ]
      },
      "getObservationsHistory": {
         "@type": "fhir:Observation",
         "readOnly": false,
         "writeOnly": false,
         "forms": [
            {
               "op": [
                  "readproperty"
               ],
               "href": "http://hapi.fhir.org/baseR4/Observation/_history",
               "security": [
                  {
                     "fhir_server_auth": [
                        "write:resource",
                        "read:resource"
                     ]
                  }
               ],
               "contentType": "application/fhir+json",
               "htv:methodName": "GET"
            }
         ]
      },
      "getObservationHistoryVid": {
         "@type": "fhir:Observation",
         "readOnly": false,
         "writeOnly": false,
         "uriVariables": {
            "id": {
               "type": "string"
            }
         },
         "forms": [
            {
               "op": [
                  "readproperty"
               ],
               "href": "http://hapi.fhir.org/baseR4/Observation/{id}/_history/{vid}",
               "security": [
                  {
                     "fhir_server_auth": [
                        "write:resource",
                        "read:resource"
                     ]
                  }
               ],
               "contentType": "application/fhir+json",
               "htv:methodName": "GET"
            }
         ]
      }
      …
      },
    "actions": {
   …
           "createObservation": {
               "@type": "fhir:Observation",
               "forms": [
                   {
                       "op": [
                           "invokeaction"
                       ],
                       "href": "http://hapi.fhir.org/baseR4/Observation",
                       "security": [
                           {
                               "petstore_auth": [
                                   "write:resource",
                                   "read:resource"
                               ]
                           }
                       ],
                       "contentType": "application/fhir+json",
                       "htv:methodName": "POST"
                   }
               ]
           },
           "updateObservation": {
               "@type": "fhir:Observation",
               "uriVariables": {
                   "id": {
                       "type": "string"
                   }
               },
               "forms": [
                   {
                       "op": [
                           "invokeaction"
                       ],
                       "href": "http://hapi.fhir.org/baseR4/Observation/{id}",
                       "security": [
                           {
                               "petstore_auth": [
                                   "write:resource",
                                   "read:resource"
                               ]
                           }
                       ],
                       "contentType": "application/fhir+json",
                       "htv:methodName": "PUT"
                   }
```

```
                ]
            },
            "deleteObservation": {
                "@type": "fhir:Observation",
                "uriVariables": {
                    "id": {
                        "type": "string"
                    }
                },
                "forms": [
                    {
                        "op": [
                            "invokeaction"
                        ],
                        "href": "http://hapi.fhir.org/baseR4/Observation/{id}",
                        "security": [
                            {
                                "petstore_auth": [
                                    "write:resource",
                                    "read:resource"
                                ]
                            }
                        ],
                        "contentType": "application/fhir+json",
                        "htv:methodName": "DELETE"
                    }
                ]
            }
    …
        },
        "title": "HAPI FHIR R4",
        "description": "Fast Healthcare Interoperability Resources (FHIR, pronounced
    \"Fire\") defines a set of \"Resources\" that represent granular clinical concepts.
    The resources can be managed in isolation, or aggregated into complex documents.
    Technically, FHIR is designed for the web; the resources are based on simple XML or
    JSON structures, with an http-based RESTful protocol where each resource has
    predictable URL. Where possible, open internet standards are used for data
    representation. \n",
        "version": {
            "instance": "1.0-oas3"
        },
        "security": [
            "fhir_server_auth"
        ],
        "securityDefinitions": {
            "fhir_server_auth": {
                "scheme": "oauth2",
                "authorization": "https://gk.eng.it/gk-fhir-server/auth/realms/Fhir-
    test/account",
                "flow": "code",
                "scopes": [
                    "write:resource",
                    "read:resource"
                ]
            }
        }
    }
```

The limitation of this approach is that the result is complete from the point of view of the functionalities exposed by the server but does not exploit the expressivity of FHIR and in particular the Capability Statement.

For this reason, the project is currently exploring a third approach, based on the capability statement of the FHIR server, that mediates the two previous approaches by using part of the information carried in the capability statement to define the Thing Description's properties and actions, so that the resulting Thing Description is more self-contained than in the first option.

Starting from the existing definitions of protocol bindings[35], we tried to define a mapping from the FHIR capability statement concepts to the Thing Description.

The main investigation is focused on the Instance level interactions available for the resources of the FHIR server to specify a mapping with the operation value of properties / actions of the WoT TD.

Moreover, the FHIR capability statement lists the type-specific search parameters for each resource. They have been mapped to URI variables, together with standard variables such as id or vid. In the table resource-specific, search parameters are noted as *parameters*.

Restful API	Description	WoT op value	URI variables
Instance Level Interactions			
read	Read the current state of the resource	readproperty	id
vread	Read the state of a specific version of the resource	readproperty	id, vid
update	Update an existing resource by its id (or create it if it is new)	writeproperty	id
patch	Update an existing resource by posting a set of changes to it	writeproperty	Id
delete	Delete a resource	invokeaction	id
history	Retrieve the change history for a particular resource	readmultipleproperties	Id, parameters
Type Level Interactions			
create	Create a new resource with a server assigned id	invokeaction	-
search	Search the resource type based on some filter criteria	readmultipleproperties	parameters[36]
history	Retrieve the change history for a particular resource type	readmultipleproperties	parameters
Whole System Interactions			
capabilities	Get a capability statement for the system	readproperty	-

[35] https://www.w3.org/TR/wot-binding-templates/

36 Name / type pairs from http://hl7.org/fhir/capabilitystatement-definitions.html#CapabilityStatement.rest.resource.searchParam

batch/transaction	Update, create or delete a set of resources in a single interaction	writemultipleproperties	-
history	Retrieve the change history for all resources	readmultipleproperties	parameters
search	Search across all resource types based on some filter criteria	readmultipleproperties	parameters

Using the proposed association, the capability statement of the GATEKEEPER FHIR Server would be represented by the following TD. For readability, we report here only parts of the complete Thing Description, focused on properties and actions for the type Observation:

```
{
    "@context": [
        "https://www.w3.org/2019/wot/td/v1",
        {
            "lst": "https://shop4cf.lst.tfo.upm.es/",
            "xsd": "http://www.w3.org/2001/XMLSchema#",
            "fhir": "http://hl7.org/fhir/",
            "reference"; {
             "type": "string",
             "description": "The reference to the id of the referenced resource in the form [type]/[ID]"
            },
            "ObservationSearchParam": {
                "type": "array",
                "items": {
                    "_id": {
                        "type": "number",
                        "description": "The ID of the resource"
                    },
                    "_language": {
                        "type": "string",
                        "description": "The language of the resource"
                    },
                    "date": {
                        "type": "date",
                        "description": "Obtained date/time. If the obtained element is a period, a date that falls in the period"
                    },
                    ...
                    "subject": {
                        "type": "string",
                        “@type”: “reference”,
                        "description": "The subject that the observation is about"
                    },
                    "value-concept": {
                        "type": "number",
                        "description": "The value of the observation, if the value is a CodeableConcept"
                    },
                    "value-date": {
                        "type": "date",
                        "description": "The value of the observation, if the value is a date or period of time"
                    },
                    "derived-from": {
                        "type": "string",
                        “@type”: “reference”,
                        "description": "Related measurements the observation is made from"
                    },
                    "focus": {
                        "type": "string",
                        “@type”: “reference”,
                        "description": "The focus of an observation when the focus is not the patient of record."
                    },
                    "part-of": {
                        "type": "string",
                        “@type”: “reference”,
                        "description": "Part of referenced event"
                    },
                    ...
                    "patient": {
                         "type": "string",
                        “@type”: “reference”,
                        "description": "The subject that the observation is about (if patient)"
                    },
                    "specimen": {
                        "type": "string",
                        “@type”: “reference”,
                        "description": "Specimen used for this observation"
                    },
...
                    "category": {
                        "type": "number",
                        "description": "The classification of the type of observation"
                    },
                    "device": {
                        "type": "string",
                        “@type”: “reference”,
                        "description": "The Device that generated the observation data."
                    },
                    "combo-value-concept": {
                        "type": "token",
                        "description": "The value or component value of the observation, if the value is a CodeableConcept"
                    },
                    "status": {
                        "type": "token",
                        "description": "The status of the observation"
                    }
                }
            }
        }
    ],
    "properties": {
        "Observation": {
            "@type": "fhir:Observation",
            "type": "array",
            "readOnly": false,
            "writeOnly": false,
            "uriVariables": {
                "id": {
                    "type": "string",
                    "description": "The ID of the resource"
                 },
                "vid": {
                       "type": "string",
                        "description": "The version number of the resource"
                    },
                    "param": {
                        "$ref": "#/ObservationSearchParam"
                    }
                },
                "forms": [
                    {
                        "href": "/Observation/{id}",
                        "contentType": "application/fhir+json",
                        "op": "readproperty",
                        "fhir:interaction": "read",
                        "htv:methodName": "GET"
                    },
                    {
                        "href": "/Observation/{id}/{vid}",
                        "contentType": "application/fhir+json",
                        "op": "readproperty",
                        "fhir:interaction": "vread",
                        "htv:methodName": "GET"
                    },
                    {
                        "href": "/Observation/{id}",
                        "contentType": "application/fhir+json",
                        "op": "writeproperty",
                        "fhir:interaction": "update",
                        "htv:methodName": "PUT"
                    },
                    {
                        "href": "/Observation/{id}",
                        "contentType": "application/fhir+json",
                        "op": "writeproperty",
                        "fhir:interaction": "patch",
                        "htv:methodName": "PATCH"
                    },
                    {
                        "href": "/Observation/{id}",
                        "contentType": "application/fhir+json",
                        "op": "writeproperty",
                        "fhir:interaction": "search",
                        "htv:methodName": "PATCH"
                    },
                    {
                        "op": "readmultipleproperties",
                        "href": "Observation/{id}/_history{?param}",
                        "contentType": "application/fhir+json",
                        "fhir:interaction": "history",
                        "htv:methodName": "GET"
                    }
                ]
            }
        },
        "actions": {
            "historyObservation": {
                "uriVariables": {
                    "param": {
                        "$ref": "#/ObservationSearchParam"
                    }
                },
                "output": {
                    "type":"array",
                    "@type": "fhir:Observation"
                },
                "forms": [
                    {
                        "op": "invokeaction",
                        "href": "Observation/_history{?param}",
                        "contentType": "application/fhir+json",
                        "fhir:interaction": "history",
                        "htv:methodName": "GET"
                    }
                ]
            },
            "createObservation": {
                "input": {
                    "type":"object",
                    "@type": "fhir:Observation"
                },
                "output": {
                    "type":"object",
                    "@type": "fhir:Observation"
                },
                "forms": [
                    {
                        "op":  [
                            "invokeaction"
                        ],
                        "href": "/Observation",
                        "security": [
                            {
                                "petstore_auth": [
                                    "write:resource",
                                    "read:resource"
                                ]
                            }
                        ],
                        "contentType": "application/fhir+json",
                        "fhir:interaction": "create",
                        "htv:methodName": "POST"
                    }
                ]
            },
            "deleteObservation": {
                "uriVariables": {
                    "id": {
                        "type": "string"
                    }
                },
                "forms": [
                    {
                        "op": [
                            "invokeaction"
                        ],
                        "href": "/Observation/{id}",
                        "contentType": "application/fhir+json",
                        "htv:methodName": "DELETE",
                        "fhir:interaction": "delete"
                    }
                ]
            }
        },
    ...
    }
```

Figure 25: FHIR Server TD, based on the Capability Statement

The three Thing Descriptions examples described in this section highlight the fact that there does not exist a single approach to map a complex service such as a FHIR server to a Thing Description.

The main goal of a Thing Description is to create machine-interpretable representation of the semantics of a Thing in terms of properties, actions and events. The third approach presented above allows to create a more direct mapping between the FHIR Capability Statement that describes the functionalities supported by the FHIR Server and the Thing Description that exposes that information to WoT clients. A more direct mapping facilitates generation, review and validation. As such, the recommendation for GATEKEEPER is to prefer that approach to describe a FHIR servers as a Thing.

5.4 Big Data Analysis using AI/ML

The GATEKEEPER AI/ML strategy, as it has been elaborated in D6.3.1, aims at generating a set of trustworthy AI/ML-based computational models on top of the GATEKEEPER research hypotheses. Special emphasis has been placed on ensuring the transparency of the associated methods and the reproducibility of the derived results by: (1) tapping into good AI/ML practices related to problem definition and formulation, data selection and management, model optimisation, clinical evaluation, and real-world monitoring and continuous evaluation, (2) methodically treating the key requirements defining Trustworthy AI ("Ethics guidelines for trustworthy AI" by the High-Level Expert Group on Artificial Intelligence[37,38]), and (3) ensuring the completeness and transparency in reporting the methods and results drawing upon established reporting guidelines (Transparent Reporting of a multivariate prediction model for Individual Prognosis or Diagnosis: The TRIPOD Statement[39]). In this direction, the aspect of AI/ML interoperability in GATEKEEPER will be precisely defined through the representation of the building components of an AI/ML predictive model; starting from the training and test datasets, continuing with the AI/ML learning and evaluation process (i.e. model selection, model training, model testing), and ending with the employed ML frameworks. The precise representation of the syntactic and semantic properties of the training/test datasets and the GATEKEEPER AI/ML models, as part of the GATEKEEPER Information Model, will allow their representation as solid entities on the Web of Things via the specification of a GATEKEEPER DCAT Profile (Figure 26).

[37] https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai

[38] https://ec.europa.eu/digital-single-market/en/news/assessment-list-trustworthy-artificial-intelligence-altai-self-assessment

[39] https://www.tripod-statement.org/

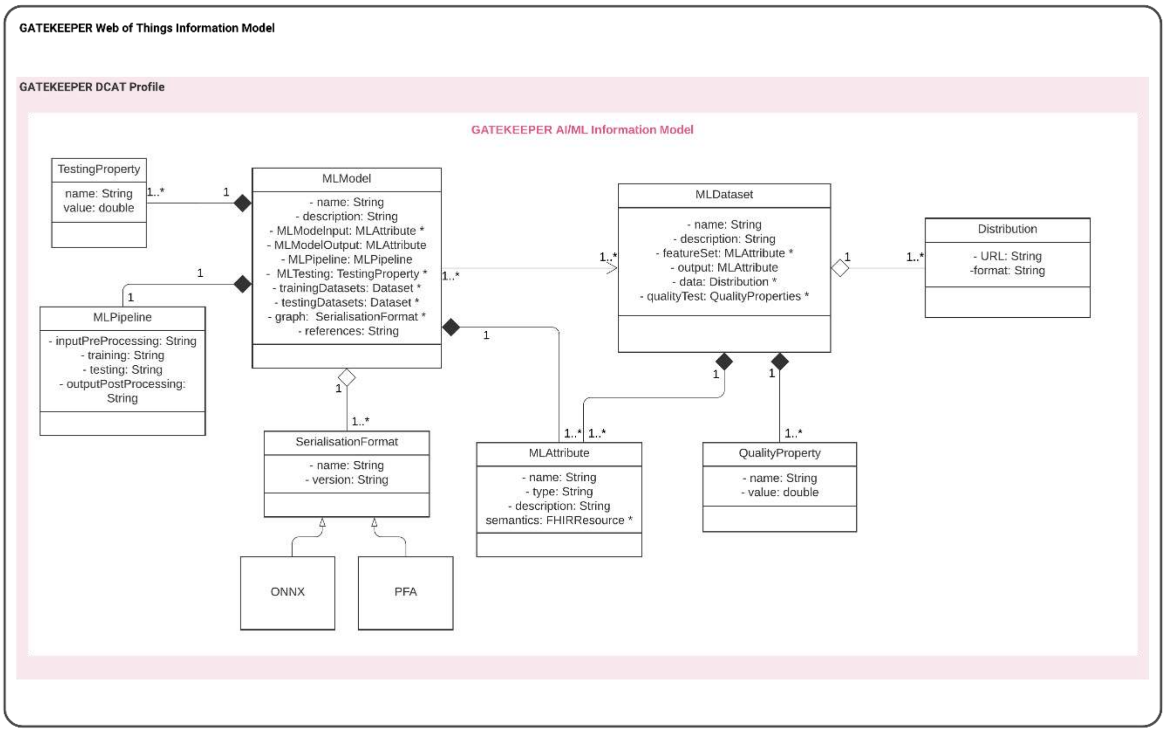


Figure 26 – The GATEKEEPER AI/ML Information Model and its encapsulation into the GATEKEEPER DCAT Profile

The GATEKEEPER AI/ML Information Model, a view of which is shown in Figure 26, will define the properties of the 'MLModel' and the 'MLDataset' entities along with their relationships. The feature set ('featureSet') and, in the case of supervised learning tasks, the output ('output') comprising a training or test dataset, are described via the 'MLAttribute' class. The latter class, via the 'semantics' property, links a dataset's attributes with their domain specific semantics of FHIR. The quality and integrity of an 'MLDataset', which highly affects the quality of the generated models, is quantified by a set of properties represented by the 'QualityProperty' class; the data integrity and quality properties along with the respective assessment methods are being specified in Task 6.3 and will be defined in the 2nd version of D6.3.

A dataset may be accessible in multiple serialisations represented by the 'Distribution' class. Similarly, a GATEKEEPER AI/ML model encapsulates a set of properties: (i) the 'MLModelInput': the set of features comprising the input of the model, with their semantic information being provided the 'MLAttribute' class; (ii) the 'MLModelOutput': the output of a predictive, diagnostic or prognostic, model (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), (iii) the 'MLPipeline': the processes of data selection and management, model optimisation, model testing accompanied by post-processing operations of the output of the models, (iv) the 'MLTesting': the relevant indices pertaining to the technical and clinical performance of the developed AI/ML models [41] (i.e. correctness, model relevance, robustness and security, efficiency, fairness, interpretability and privacy), which are represented by the 'TestingProperty' class; the ML testing properties along with the respective assessment methods are being specified in Task 6.3 and will be defined in the 2nd version of D6.3, (v) the 'trainingDatasets': the utilised training datasets, (vi) the 'testDatasets': the utilised test datasets, (vii) the 'graph': the representation of the AI/ML models as ONNX or PFA objects, and (viii) the ‘references’: references to the scientific publications supporting the evidence generated by the AI/ML models.

The GATEKEEPER AI/ML models will be represented (serialised) in the Open Neural Network eXchange (ONNX) format; “ONNX defines a common set of operators - the building blocks of machine learning and deep learning models - and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers”[40]. In addition to ONNX, the Portable Format for Analytics (PFA) can be additionally considered; “a PFA document is a string of JSON-formatted text that describes an executable called a scoring engine. Each engine has a well-defined input, a well-defined output, and functions for combining inputs to construct the output in an expression-centric syntax tree”[41]. Both ONNX and PFA render the GATEKEEPER AI/ML models interoperable, being ML framework-agnostic, enabling, in turn, model sharing with relevant stakeholders.

Each GATEKEEPER AI/ML model along with the associated training and test datasets can be published on the Web as a Data Catalog (DCAT) resource [42]; “DCAT is an RDF vocabulary designed to facilitate interoperability between data catalogs published on the Web.” A dataset in DCAT is defined as a "collection of data, published or curated by a single agent, and available for access or download in one or more serializations or formats". We ought to mention that DCAT allows the documentation of different aspects of the quality of DCAT first class entities (e.g., datasets, distributions) and the conformance to stated quality standards through the Data Quality Vocabulary [43]. Properties of the GATEKEEPER AI/ML Information Model not be covered by the DCAT vocabulary will be specifid in a GATEKEEPER DCAT Profile, built upon the base standard.

[40] https://onnx.ai/index.html

[41] http://dmg.org/pfa/docs/motivation/

6 Additional Interoperability Considerations

6.1 Comparison of relativist and reductionist approaches to semantics

According to the Cambridge Dictionary, *semantics* is the study of meanings in a language, that is, the study of the relationship between words and how we draw meaning from those words. A reductionist approach to meaning is based upon the Aristotelian tradition of logic, formal semantics and model theory. This is essentially about what is provably true given a set of assumptions and sound inference rules. The Semantic Web is founded on this approach, including the use of description logics and tableaux proof procedures.

Work in psychology, e.g. by Philip Johnson-Laird, has shown that humans don't rely on the laws of logic and probability, but rather by thinking about what is possible. We use mental models of examples, as well as metaphors and analogies. Meaning is relative to the context and the tasks we are trying to carry out. This is a relativist approach in which meaning is fuzzy rather than absolute. The Cambridge dictionary and others like it define words in terms of other words, and examples of how words are used in practice.

Statistical analysis of large text corpora can provide a mapping of words to vectors, such that proximity in the vector space signifies a measure of semantic distance [44]. This can be used to find documents matching a query string, but not for substantive reasoning over meanings. Statistics are thus only a weak surrogate for semantics.

For restricted applications, the semantics can be implicit in the application data and code. More generally, where interoperability is important, it is possible to manually construct semantic models as ontologies, but these require lots of communal effort to devise and maintain. In the longer term, we need to find ways to teach cognitive agents how to understand and reason with medical knowledge. This will involve cognitive approaches to natural language processing, learning and reasoning.

6.2 Challenges for managing semantics at scale

Whilst it is relatively easy to reach agreement in a small community, it is much harder to do so across larger communities and across multiple communities, each of which are likely to have somewhat different requirements, and to be accustomed to using different terminologies. This makes it unwise and impractical to force everyone to use the same ontologies. A further challenge is posed by the continuing evolution of requirements, along with the need for backwards compatibility to avoid breaking existing applications.

One way to map between different ontologies is through a mapping to a shared model of concepts at a more abstract level. The shared models are expressed in what is commonly referred to as an "upper ontology". A drawback is that the level of abstraction can make such ontologies harder to understand, and complicate the mappings of domain ontologies to upper ontologies.

Another approach is to define mappings between ontologies at a peer to peer level. This has the drawback of requiring an impractical number of peer to peer mappings when there are more than a few domain ontologies to map between. That can be addressed by mapping through a single ontology, analogous to translating from Tamil to Swahili by first translating to English.

A further challenge is due to different domain ontologies providing more or less levels of specificity – how to deal with situations where the details needed in the target ontology are lacking in the source ontology. The mapping may be dependent on contextual information that isn't readily available. A potential work around is to use defaults. A more general solution is to task specific mappings that are good enough for the task in hand.

6.3 Auditability and Provenance

The user data collected by pilots is highly sensitive. Informed consent is needed for the end-user (the patient) to give healthcare professionals the right to use data collected from the patient. Where patients lack the mental capacity for informed consent[42], other people will have to make the decision for them. In a federated architecture, access to data coming from medical devices may be subject to contractual agreements between parties and associated legal constraints, including the EU General Data Protection Regulation[43]. Parties may for instance be required to track access to data and actions done with and/or on it. This may warrant the use of a distributed ledger that would provide an auditable log. Knowing also where the data comes from is important if we wish to correctly interpret the data and that is where data provenance comes into play.

6.4 Pull-based privacy business models

The Web has thrived on free services supported by advertising. In essence, the end-users are the product, and the emphasis is on tracking user behaviour to support more effective advertising. Users have become habituated to this and tend to see the cookie permission requests as nuisances to be clicked away, to get to the services they want to access.

User tracking may feel harmless, but could easily be abused to charge some users more for the same products or services, or to discriminate against users based upon their race, gender, sexual preferences or religious beliefs. Companies could charge higher premiums for medical insurance based upon tracking data that suggests poor health or behaviours likely to result in medical problems later in life.

We are already seeing problems for some people in respect to access to finance for large purchases due to poor credit ratings that are based upon bad information that the people affected are unable to correct. Consumers may find targeted advertising spooky if it suggests that the advertisers appear to know a great deal about them.

Medical data is especially sensitive, requiring very careful attention to privacy and security. Just by holding medical data, companies put themselves at risk of fines and expensive settlements to litigation on behalf of patients following data breaches. At the same time, there are many potential benefits to patients from companies being able to offer valuable services based upon access to medical data.

This points to opportunities for business models in which the end-user's personal data is provided to certified service providers on an as-needed basis, and subject to restrictive terms and conditions, and audit trails, along with strong recourse in case of abuse. End-users are typically not legal or privacy experts, and unable or unwilling to deal with the details. The solution is to involve a trusted party that looks after the user's privacy based upon an assessment of the user's attitude to risk, something that can be determined based upon the user's personal history, and that of others like him or her. There are plenty of challenges to be identified and discussed, and the organisation of a W3C Workshop is being investigated to explore this space.

[42] See the advice on mental capacity and informed consent issued by the BMA

[43] See the ICO Guide to the General Data Protection Regulation (GDPR).

6.5 A systems perspective for GATEKEEPER

This section attempts to provide a systems perspective for GATEKEEPER and summarise the choices the project will need to make and the interoperability implications thereof.

The overall aim of GATEKEEPER is to provide for improved well-being for elderly and frail patients along with improvements on the state of the art for caregivers and clinical staff. We would like to combine a broad range of capabilities to offer a unified integrated approach for monitoring patients that build upon clinical readings with medical instruments, information covering test results, existing medical conditions, medications and treatments, along with continuous monitoring in so far as it is practical, e.g. fall detection, geolocation, pulse rate and oxygenation, etc. Patients themselves can be encouraged to take regular readings, e.g. of their weight, glucose levels, and to report their individual sense of well-being.

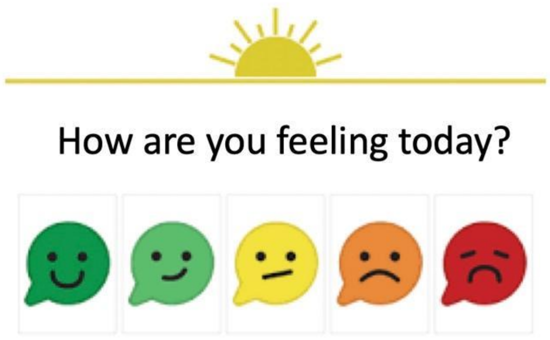


Figure 27 - Contentment reporting

The devices involved use a heterogeneous mix of technologies, and the complexity that this presents should be dealt with via forwarding and transforming data into a uniform graph data framework that simplifies the development and maintenance of application services. This involves the use of gateways that collect data from the devices at the network edge, using whatever technologies are needed, and then forward this data to servers hosting graph databases via secure connections using HTTP over TLS (HTTPS). One exception is where doctors wish to remotely monitor ECG traces in real-time, for which Web Sockets over TLS (WSS) is likely to be more effective.

The GATEKEEPER platform would include the following major components:

- HTTP server for uploading data to the GATEKEEPER platform. This may involve the use of specific connector modules that transform data before ingesting it, subject to validation against the ontology agreed when the connector was registered.
- Graph database for storing data and locally applying efficient and scalable graph algorithms, including those needed to support machine learning.
- Data relating to different patients is isolated into separate graphs.
- An application server that hosts multiple GATEKEEPER applications.
- Security module that supports authentication, access control and logging, as well as being able to summon security staff and take remedial action on detecting attacks.
- Notifications module for sending alerts to patients, caregivers and clinical staff.

- HTTP server that provides an API for remote applications, allowing for federated architectures that compartmentalise patient data for improved resilience to cyber-attacks.
- HTTP client library for accessing remote information sources, e.g. REST APIs for electronic health records using HL7 FHIR. This data is transformed upon ingesting into the graph database.
- The GATEKEEPER platform could also support a pull-model for other kind of information sources as needed, using connector modules that use the HTTP client library to pull data from other servers, and transform it before adding it to the database.
- A stream processing system capable of processing very large amounts of data. This could be used to handle data a) as it is ingested into GATEKEEPER, and b) as a means to batch process data, e.g. for machine learning across data for many patients, subject to strict controls to safeguard patient privacy.
- HTTP server for web applications designed for patients, caregivers and clinical staff, this would for instance allow clinical staff to set alarm thresholds, and to view an integrated dashboard for the patient's well-being.

The main interoperability challenges include:

- Obtaining the technical information needed to interoperate with a broad variety of medical and consumer devices. Device vendors may be unwilling to provide this information, thereby limiting the choice of devices that GATEKEEPER can utilise.
- Mapping data formats, identifiers and concepts when ingesting information from external sources. Where practical this can rely on existing mappings to RDF.
- Supporting patient aids and clinical decision support tools with respect to applying analytics and AI on the collected data safely.

Some design challenges include:

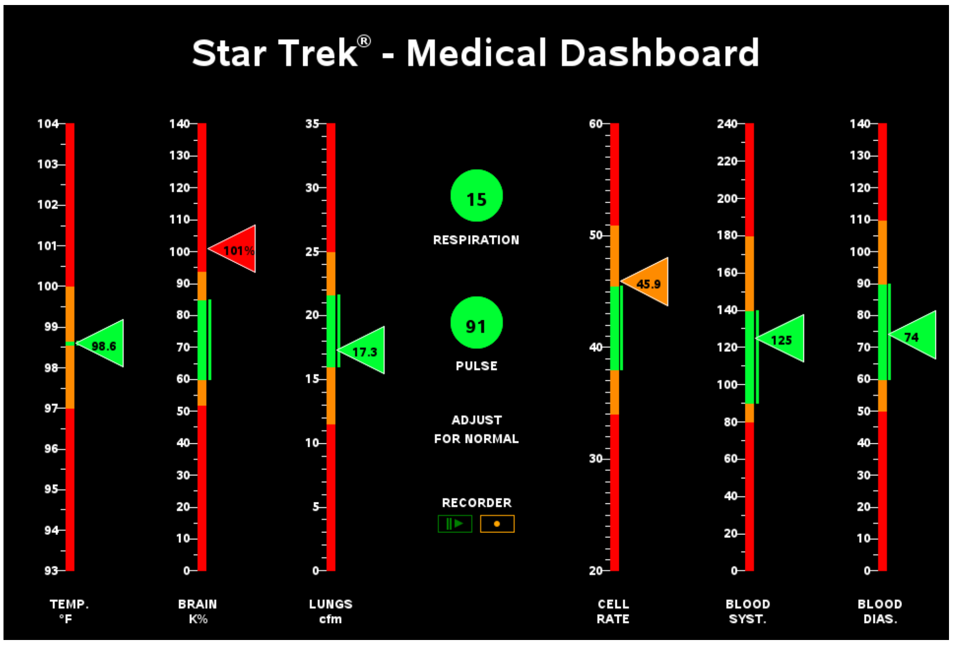
- Whether to use an existing RDF triple store, or to develop a new graph database engine for "chunks" as an amalgam of RDF and Property Graphs.
- Whether to integrate a rule engine or to rely on a graph traversal and manipulation API? Rules would make for a higher level of abstraction when designing services. Example rules would be those that raise alarms when particular readings exceed upper or lower thresholds set by clinical staff. Rules could infer hidden states from a combination of information sources. Rules could further be used to make suggestions to patients, e.g. reminders to take medications, to weigh themselves, etc.
- The potential role of SPARQL, OWL and SHACL as RDF standards for respectively, queries, ontologies and constraints?
- Understanding which problems for caregivers and clinical staff could be addressed by GATEKEEPER.
- Understanding the functional requirements for the web user interface for patients, caregivers and clinical staff.

Applying those requirements to create the corresponding client and server-side resources, e.g. JavaScript libraries, images, style sheets and media files, as needed to work effectively on mobile and desktop devices.

7 Conclusions

Gatekeeper seeks to improve the well-being of elderly and frail patients, starting from existing approaches and integrating a broad variety of techniques for monitoring physical and mental health. The Star Trek medical dashboard provides one vision for what that could mean, but is lacking in many respects for the practical needs of a real-world solution. Can we identify some equally compelling ways to present the patient's physical and mental health, and how it is changing over time, that provides an effective tool to support modern practices for home healthcare?

Figure 28 - Medical Dashboard courtesy of Robert Allison



The above fictional dashboard shows current values for some signs of health. However, caregivers will also want to understand how a patient has been over a given period of time, e.g. during the last hour, day or week. This is an opportunity for combining automatic text summarisation with notes taken by caregivers and medical staff. Other requirements include the means to deliver notifications to caregivers and medical staff, as well as encouragement and reminders to the patients themselves, e.g. for exercise and medications.

This vision calls for a uniform framework for storing and processing information, so that application services are decoupled from the complexity of the heterogeneous information sources, data formats and protocols. This framework will facilitate work on machine reasoning over patient data to suggest appropriate interventions.

The GATEKEEPER platform builds on top of the Web of Things to create that uniform framework. In GATEKEEPER, all resources are represented with a Thing Description that leverages JSON-LD and ontologies to expose resource semantics in a machine-interpretable way. The GATEKEEPER platform integrates a graph database, statistics, rule engine and graph algorithms.

Using Thing Descriptions goes a long way towards enabling interoperability between heterogeneous sources and services but further steps are needed to facilitate reuse and reasoning over available data points. This document details recommendations on key components of the GATEKEEPER architecture to further address interoperability issues. These include:

1. Provisions to create homogeneous Thing Descriptions across the platform
2. Recommendations to certify resources
3. Need to import/export data built using framework such as the OpenAPI or FHIR APIs, recasting them in terms of Thing Descriptions where appropriate in combination with data models and ontologies.
4. Considerations for the semantic modelling of the GATEKEEPER service marketplace
5. Recommendations to ease certification of FHIR resources
6. Approaches to developing Thing Descriptions to represent core data federation components of GATEKEEPER (Integration Engine and FHIR Server)
7. Considerations on Thing Description modelling for big data analysis based on DCAT.

The project also needs to keep an eye on additional interoperability considerations, including challenges for managing semantics at scale and mechanisms to make and present meaningful (from a human perspective) interpretations of the data.

The existence of multiple technologies, formats and ontologies dedicated to specific domains and the relative reluctance of device vendors to provide the information needed to implement gateways for ingesting data from sensor devices, means that the heterogeneity of data sources is here to stay. A uniform framework for storing and manipulating information will make it much easier to create services that combine multiple information sources.

Appendix A Cognitive AI – mimicking human thought

Whilst AI is currently dominated by Deep Learning with multi-layer artificial neural networks, the limitations of current approaches to Deep Learning are increasingly apparent, e.g. single rather than general purpose solutions, a lack of transparent explanations, a lack of saliency that results in brittleness and exposure to attacks, difficulties with out-of-distribution generalisation, and the need for very large datasets compared to human learning.

Symbolic approaches address some of these concerns, but the emphasis on manual development can make solutions difficult to design and maintain. Symbolic approaches can also be difficult to apply in the presence of uncertainty, incompleteness and inconsistency. In addition, approaches based upon formal semantics, model theory and logical deduction are often a poor fit to human concepts and reasoning.

Cognitive AI seeks to blend symbolic graph-based approaches with sub-symbolic statistical techniques, inspired by over 500 million years of neural evolution and decades of advances in the cognitive sciences. The newly formed W3C Cognitive AI Community Group[44] is incubating Cognitive AI. This has included work on cognitive architecture, the chunks data and rules language, and a series of demos on autonomous vehicles, smart homes, industrial robots and ongoing work on machine learning, natural language processing and conversational interfaces. The Community Group is preparing a formal specification for the chunks data and rules language[45] with a view to its standardisation.

Work on cognitive natural language processing is exploring the use of incremental word by word shift-reduce parsing, and rules for mapping between syntactic and semantic structures, that are applied to both natural language understanding and generation. Backtracking is avoided through concurrent asynchronous processing of syntax and semantics that takes the context into account. Natural language is key to both human-machine collaboration and to teaching skills as a more scalable approach than direct manual knowledge engineering.

A robot demo was prepared in support of a keynote talk on Cognitive AI by Dr. Dave Raggett at the Summer School on AI for Industry 4.0,[46] Saint-Étienne, France – 27th to 31st July 2020. The scenario depicts a bottling factory in which empty wine bottles are fed to a robot arm via a conveyor belt. The robot then moves bottles to the filling and capping stations before packing them into boxes, that each hold six bottles, on a second conveyor belt.

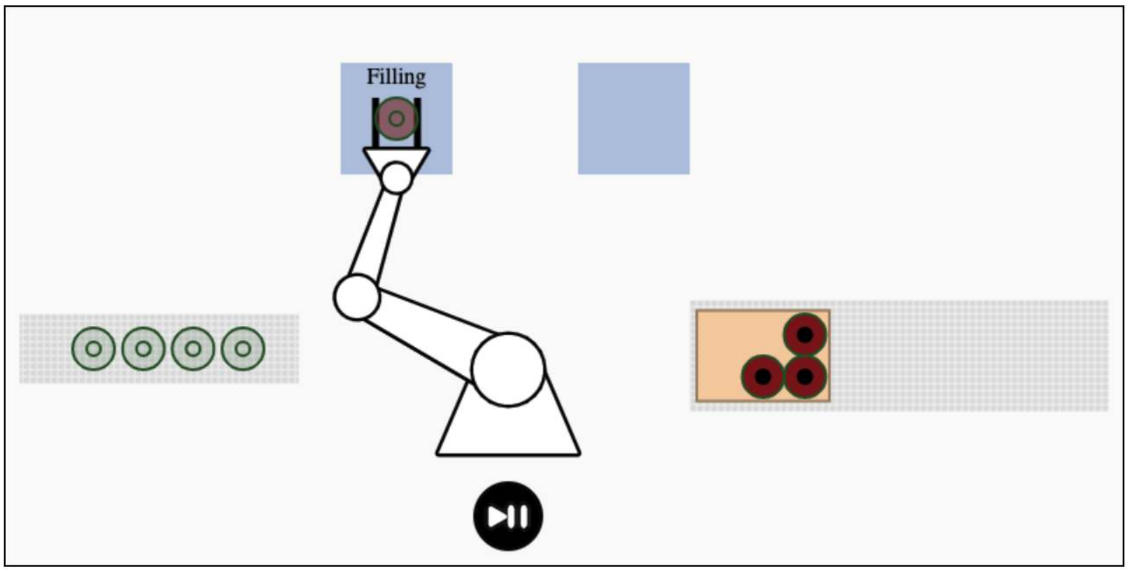
The demo involves real-time concurrent control over the two conveyor belts, the robot arm, the filling and capping stations. The control behaviour is expressed as 20 rules that express what to do when something happens, e.g. when space becomes available at the beginning of the first conveyor belt then an empty bottle should be added. If the space is already available, this action takes place immediately, otherwise it is triggered at some time in the future when space becomes available. A screen shot of the web-based demo is given below:

[44] https://www.w3.org/community/cogai/

[45] https://w3c.github.io/cogai/

[46] https://ai4industry.sciencesconf.org/

Robot demo: https://www.w3.org/Data/demos/chunks/robot/



The following box shows a selection of some of the rules from the demo:

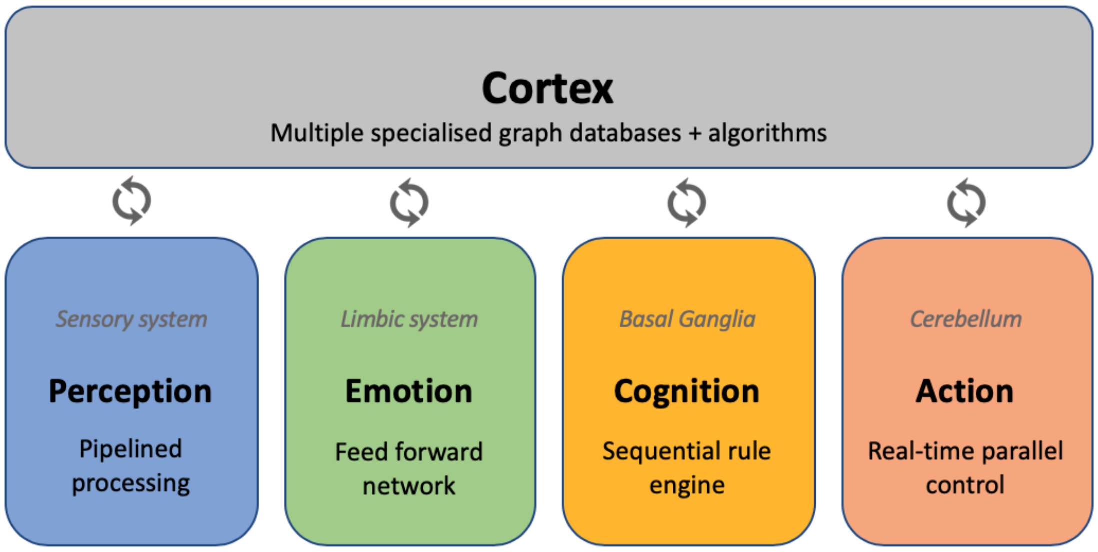
```
# add bottle when belt1 has space and wait afresh
space {thing belt1} =>
        action {@do addBottle; thing belt1},
        space {@do wait; thing belt1; space 30}

# stop belt1 when it is full and move arm
full {thing belt1} =>
        action {@do stop; thing belt1},
        action {@do move; x -120; y -75; angle -180; gap 40; step 1}

# move robot arm into position to grasp empty bottle
after {step 1} => robot {@do move; x -170; y -75; angle -180; gap 30;
step 2}

# grasp bottle and move it to the filling station
after {step 2} =>  goal {@do clear}, robot {@do grasp},
       robot {@do move; x -80; y -240; angle -90; gap 30; step 3}
```

Rule actions include starting and stopping the conveyor belts, moving the robot arm, grasping and releasing bottles, filling and capping. Actions are asynchronous and associated with a continuation to be executed when the action completes. The robot has three rotatory joints and a gripper. The commands specify the target position, spacing and bearing of the gripper. The robot computes a joint trajectory from the current state to the target state, and then smoothly accelerates and decelerates each joint according to its individual specification. The demo simulates this behaviour using a high precision timer for computing functions over time.

Another demo simulates a smart home with support for automated and manual control over heating and lighting depending on the time of day, the personal preferences of people currently in the room, and who has precedence for what things. This involves default reasoning expressed as a combination of declarative and procedural knowledge.

The approach emulates the human brain, inspired by the work of John Anderson at CMU on *ACT-R*[47], a popular architecture for Cognitive Science, involving a functional model of the cortex and basal ganglia.

The high-level architecture is given in the following figure. The cortex acts like a blackboard for sharing information across different cognitive systems. Lobes in the human cortex are modelled as cognitive databases, each of which support a set of generic and task specific graph algorithms that are executed locally to the data.

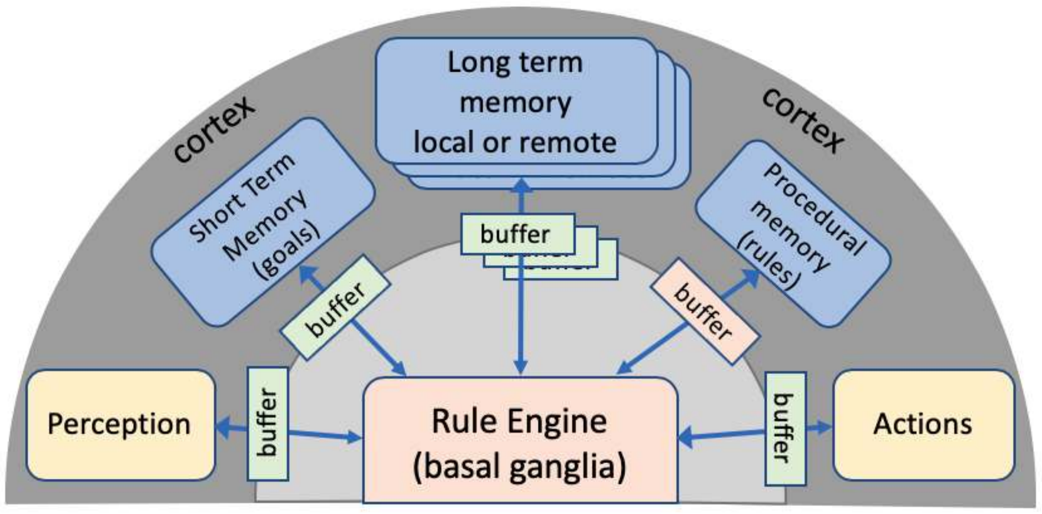


- *Perception* interprets sensory data and places the resulting models into the cortex. Cognitive rules can set the context for perception, and direct attention as needed. Events are signalled by queuing chunks to cognitive buffers to trigger rules describing the appropriate behaviour. A prioritised first-in-first-out queue is used to avoid missing events that are closely spaced in time.
- *Emotion* is about cognitive control and prioritising what's important. The limbic system provides a rapid assessment of situations without the delays incurred in deliberative thought. This is sometimes referred to as System 1 vs System 2.
- *Cognition* involves sequential execution of rules for deliberative thought, including the means to invoke graph algorithms in the cortex, and to invoke operations involving other cognitive systems.
- *Action* is about real-time control over actuators using sensory information placed into the cortex by perception.

The cortex is modelled as a set of cortical modules that each hold facts as a collection of chunks, where each chunk is a typed collection of properties whose values are literals or references to other chunks. Each module is associated with a single chunk buffer that represents the concurrent firing pattern of the bundle of nerve fibres connecting to a particular cortical region. Cognitive rules are expressed as chunks, where the conditions match the module buffers, and the actions either directly update the buffers, or invoke operations associated with each module.

[47] http://act-r.psy.cmu.edu/about/

The following figure zooms in on cognition and the cortico-basal ganglia circuit:


cortex
Long term memory local or remote
cortex
Short Term Memory (goals)
buffer
Procedural memory (rules)
buffer
buffer
Perception
buffer
Rule Engine (basal ganglia)
buffer
Actions

Cognitive Architecture - Cognition

Cognitive rules initiate actions, which are delegated to the cortico-cerebellar circuit and executed asynchronously. The Cerebellum can be likened to a flight controller using sensory input from the Cortex for real-time control over myriad muscles. When you decide to reach out to grab a mug of coffee, your cognitive system passes an instruction with a rough indication of the mug's position. The cortico-cerebellar circuit uses visual input to smoothly guide the motion of the hand to the mug, whilst taking account of proprioception – awareness of the joint positions and loads on the muscles throughout your body.

An ontology for the bottling scenario presented above[48] can be expressed as chunks, and used for validating facts and rules, e.g. to check that actions don't pass invalid values, that actions only use declared operations, and that action responses are handled correctly. Declarative models can be used to update the robot's behaviour as needed to adapt to changing needs. This points to a bright future for cognitive approaches, along with the role of natural language for human-computer collaboration and skill acquisition.

A webinar was given by Dr. Raggett on 6th November 2020 to the Knowledge Media Institute at the Open University. The institute is the Gatekeeper project partner responsible for work on applying robots in support of home healthcare. A second webinar was arranged on 25th November 2020 for the Centre for Artificial Intelligence, Robotics and Human-Machine Systems at Cardiff University. The centre focuses on human-like AI, ethical and explainable AI, human-centred technologies and society, humans and robots.

Further work is envisaged in 2021 on exploring how Cognitive AI can be applied to the scenarios relevant to the Gatekeeper project pilot studies.

Some common questions about Cognitive AI are discussed in the FAQ:

- https://github.com/w3c/cogai/blob/master/faq.md

[48] https://www.w3.org/Data/demos/chunks/robot/facts.chk

Appendix B References

1. **GATEKEEPER Consortium.** *D6.1 Medical use cases specification and implementation guide.*

2. —. *D6.2 Early detection and interventions operational planning.*

3. —. *D3.2 Overall GATEKEEPER architecture.*

4. —. *D3.1 Functional and technical requirements of GATEKEEPER platform.* 2020.

5. —. *D4.5 GATEKEEPER Trust Authority.*

6. —. *D4.2 Thing Management System.*

7. —. *D8.1 Overview of relevant standards in smart living environments and gap analysis.*

8. —. *D6.3 GATEKEEPER Big Data and Data Analytics Strategies.*

9. —. *D3.4 Semantic Models, Vocabularies & Registry.*

10. —. *D3.5 GATEKEEPER HL7 FHIR optimization for IoT and Smart and healthy living environments.*

11. **Jeni, Tennison.** *CSV on the Web: A Primer.* s.l. : W3C Note, https://www.w3.org/TR/tabular-data-primer/, 2016.

12. **Jeni, Tennison and Gregg, Kellogg.** *Metadata Vocabulary for Tabular Data.* s.l. : W3C Recommendation, https://www.w3.org/TR/tabular-metadata/, 2015.

13. **Tim, Bray, Jean, Paoli and C. M., Sperberg-McQueen.** *Extensible Markup Language (XML) 1.0.* s.l. : W3C Recommendation, https://www.w3.org/TR/1998/REC-xml-19980210.html, 1998.

14. **Tim, Bray, et al.** *Extensible Markup Language (XML) 1.0 (Fifth Edition).* s.l. : W3C Recommendation, https://www.w3.org/TR/xml/, 2008.

15. **Shudi, Gao, C. M., Sperberg-McQueen et Henry S., Thompson.** *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures.* s.l. : W3C Recommendation, https://www.w3.org/TR/xmlschema11-1/, 2012.

16. **David, Peterson, et al.** *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes.* s.l. : W3C Recommendation, https://www.w3.org/TR/xmlschema11-2/, 2012.

17. **Tim, Bray.** *The JavaScript Object Notation (JSON) Data Interchange Format.* s.l. : IETF, https://tools.ietf.org/html/std90, 2017.

18. **Greg, Kellogg, Pierre-Antoine, Champin and Dave, Longley.** *JSON-LD 1.1 - A JSON-based Serialization for Linked Data.* s.l. : W3C Recommendation, https://www.w3.org/TR/json-ld11/, 2020.

19. **Savas, Parastatidis, Jim, Webber and Ian, Robinson.** *REST in Practice Hypermedia and Systems Architecture.* s.l. : O'Reilly Media, 2010.

20. **E., Rescorla.** *The Transport Layer Security (TLS) Protocol Version 1.3.* s.l. : IETF, https://tools.ietf.org/html/rfc8446, 2018.

21. **M., Jones, J., Bradley and N., Sakimura.** *JSON Web Token (JWT).* s.l. : IETF, https://tools.ietf.org/html/rfc7519, 2015.

22. **Andrew, Banks, et al.** *MQTT Version 5.0.* s.l. : OASIS, https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, 2019.

23. **I., Fette and A., Melnikov.** *The WebSocket Protocol.* s.l. : IETF, https://tools.ietf.org/html/rfc6455, 2011.

24. **François, Daoust and Dave, Raggett.** *Chunks and Rules.* s.l. : W3C Editor's Draft, https://w3c.github.io/cogai/.

25. **W3C OWL Working Group.** *OWL 2 Web Ontology Language (Second Edition).* s.l. : W3C Recommendation; https://www.w3.org/TR/owl2-overview/, 2012.

26. **Armin, Haller, et al.** *Semantic Sensor Network Ontology (SSN/SOSA).* s.l. : W3C Recommendation, https://www.w3.org/TR/vocab-ssn/, 2017. Recommendation.

27. **Laura, Daniele, et al.** *SAREF: the Smart Applications REFerence ontology.* s.l. : ETSI, https://saref.etsi.org/core/v3.1.1/, 2020.

28. **Matthias, Kovatsch, et al.** *Web of Things (WoT) Architecture.* s.l. : W3C Recommendation, https://www.w3.org/TR/wot-architecture/, 2020.

29. **Sebastian, Kaebisch, et al.** *Web of Things (WoT) Thing Description.* s.l. : W3C Recommendation, https://www.w3.org/TR/wot-thing-description/, 2020.

30. **Zoltan, Kis, et al.** *Web of Things (WoT) Scripting API.* s.l. : W3C Note, https://www.w3.org/TR/wot-scripting-api/, 2020.

31. **Michael, Koster and Ege, Korkan.** *Web of Things (WoT) Binding Templates.* s.l. : W3C Note, https://www.w3.org/TR/wot-binding-templates/, 2020.

32. **Elena, Reshetova and Michael, McCool.** *Web of Things (WoT) Security and Privacy Guidelines.* s.l. : W3C Note, https://www.w3.org/TR/wot-security/, 2019.

33. **Victor, Charpenay, et al.** *Web of Things (WoT) Current Practices.* s.l. : W3C Editor's Draft, http://w3c.github.io/wot/current-practices/wot-practices.html, 2020.

34. **HL7.** *Fast Healthcare Interoperability Resources (FHIR®).* s.l. : https://hl7.org/fhir/.

35. **Darrel, Miller, et al.** *OpenAPI Specification, Version 3.0.3.* s.l. : OpenAPI initiative, 2020.

36. **A., Wright, et al.** *JSON Schema: A Media Type for Describing JSON Documents.* s.l. : IETF, https://tools.ietf.org/html/draft-handrews-json-schema-02, 2019.

37. **Holger, Knublauch et Dimitris, Kontokostas.** *Shapes Constraint Language (SHACL).* s.l. : W3C Recommendation; https://www.w3.org/TR/shacl/, 2017.

38. **Eric, Prud'hommeaux, et al.** *Shape Expressions Language 2.next.* s.l. : W3C Draft; https://shexspec.github.io/spec/, 2020.

39. **ACTIVAGE Consortium.** *D4.3 Marketplace Components.* 2018.

40. **Cong, Peng et Prashant, Goswami.** *Meaningful Integration of Data from Heterogeneous Health Services and Home Environment Based on Ontology.* s.l. : Sensors. 19. 1747. 10.3390/s19081747, 2019.

41. **J.M., Zhang, et al.** *Machine learning testing: Survey, landscapes and horizons.* s.l. : IEEE Transactions on Software Engineering, 2020.

42. **Riccardo, Albertoni, et al.** *Data Catalog Vocabulary (DCAT) - Version 2.* s.l. : W3C Recommendation; https://www.w3.org/TR/vocab-dcat-2/, 2020.

43. **Jeremy, Debattista, et al.** *Data on the Web Best Practices: Data Quality Vocabulary.* s.l. : W3C Note; https://www.w3.org/TR/vocab-dqv/, 2016.

44. **Mikolov, Tomas et al.** *Efficient Estimation of Word Representations in Vector Space.* 2013. https://arxiv.org/abs/1301.3781.

45. **European Commission.** *Medical devices: Guidance document - Classification of medical devices (MEDDEV).* s.l. : http://ec.europa.eu/DocsRoom/documents/10337/attachments/1/translations, 2010.