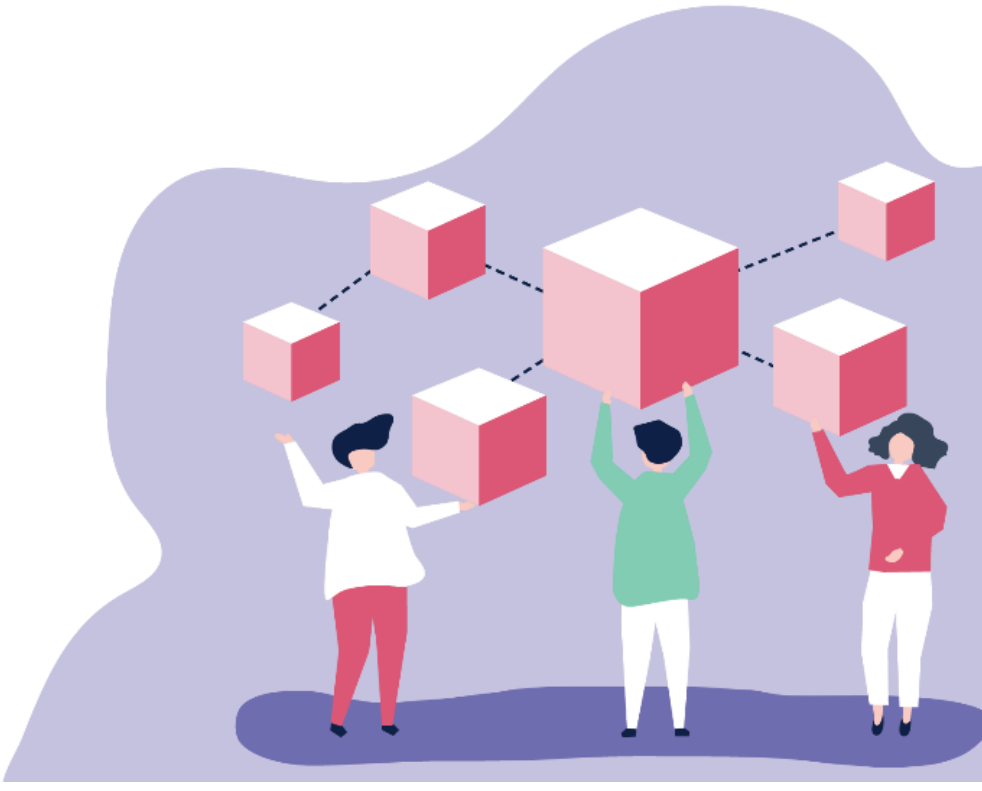
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 857223

D3.3 Interoperability within Gatekeeper

Deliverable No.	D3.3	**Due Date**	31/03/2020
Description	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**	Final



Authors

Name and surname	Partner name	e-mail
Francois Daoust	W3C	fd@w3.org
Dave Raggett	W3C	dsr@w3.org
Gabriel Galeote	UPM	ggaleote@lst.tfo.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

History

Date	Version	Change
02/03/2020	0.1	Initial draft
04/03/2020	0.2	Expanded prior to request for contributions
12/03/2020	0.4	Interoperability for Gatekeeper & restructuring
16/03/2020	0.5	Appendix on graph databases
17/03/2020	0.6	Architectural Patterns and DMCoach
18/03/2020	0.7	Expanded thing lifecycle
19/03/2020	0.8	Revised Architectural Patterns and DMCoach
23/03/2020	0.9	Incorporating feedback from S4C

26/03/2020	0.9.1	Incorporating feedback from STM
26/03/2020	0.9.2	Major restructuring following telecon
30/03/2020	0.9.3	Moved up section of web of things
14/04/2020	0.9.4	With further feedback from STM and SC
20/04/2020	1.0	Deliverable ready for submission

Key data

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

Abstract

This deliverable explores the interoperability challenges facing the Gatekeeper pilots 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)?

Statement of originality

This deliverable contains original unpublished work except where clearly indicated otherwise.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

Table of contents

Table of tables

Table 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 seeks to explore the possibilities via a series of pilots and the development of an open source platform.

This report on interoperability within Gatekeeper reviews the requirements for the Gatekeeper pilots to the extent they are known at the time of writing this report, and matches them to the solutions that the technology partners are able to contribute, along with the requirements for interoperability. The analysis leads to the proposal for defining a uniform framework for storing and manipulating information, decoupling application services from the complexity of the heterogeneous information sources, data formats and protocols.

The proposed Gatekeeper platform would integrate a graph database, statistics, rule engine and graph algorithms. HTTP would be used for uploading data to the platform from sensors, for access to electronic health records, and for the means to provide Web based tools for patients, caregivers and clinical staff, inspired by the Star Trek medical dashboard, though that itself is not the solution. We will seek to provide equally compelling ways to present the patient's physical and mental health and how it is changing over time, as an effective tool to support modern best practices for home healthcare.

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:

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

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

[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 2 describes the relationship of this report to other Gatekeeper reports. Section 3 examines the project's aims in more detail and introduces the Web of Things. Section 4 looks at what is currently known about the technology requirements for the Gatekeeper pilots. Section 5 discusses architectural patterns and their implications for privacy and security. Section 6 discusses interoperability layers. Sections 7-10 delve into further details for each layer. Section 11 discusses additional considerations. Section 12 lists the conclusions. The appendices list the solutions available from the technology partners, provide an introduction to cognitive databases, and finish with a glossary of terms.

2 Relationship to other Gatekeeper deliverables

This report (D3.3) will be followed by deliverable D3.4 "Semantic Models, Vocabularies and Registry" in two months' time, and D3.2 "Overall Gatekeeper architecture" in a further two months. D3.5 "Gatekeeper binary FHIR optimisation for IoT" will follow in six months. The work on the design of the Gatekeeper architecture will use the interoperability requirements defined in this deliverable to adopt the appropriate architectural patterns ensuring interoperability at all levels. As such, this report seeks to inform work that feeds into those reports.

This report relies upon D6.1 "Medical use cases specification and implementation guide" for the preliminary assessment of the requirements for each pilot, as well as a survey conducted to identify the solutions that the Gatekeeper technical partners can provide. D6.2 "Early detection and interventions, operational planning" provides further background. This report complements deliverable D8.1 "Overview of relevant standards in smart living environments and gap analysis".

3 Monitoring Elderly and Frail Patients

This section expands the ideas from the introduction, and explains why Gatekeeper needs to focus on a uniform framework for data and metadata, rather than the Web of Things. The following section looks at the technologies the Gatekeeper Pilots plan to use for the reference use cases and how they can be integrated using a common approach.

3.1 Framing the 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?

As will be shown in Section 6, 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 technology 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.

It is also clear even now that well defined business models are 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.

The project proposal was written on the assumption that monitoring services can be conveniently built around W3C's Web of Things. 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. This will be explained in more details in the following subsection. These things will form part of a uniform framework for data and metadata. Gatekeeper needs to work with medical experts to adopt and in some cases create vocabularies of terms for a knowledge graph that can be used by all of the reference use cases described in D6.1 and D6.2.

What is now less clear is whether monitoring services should be built on top of software interfaces for digital twins (i.e. object properties, actions and events), as assumed in the project proposal, or whether it would make more sense for services to be built on top of APIs for knowledge graphs. Such APIs would support traversal and manipulation of knowledge graphs, condition-action rules, high performance graph algorithms and event driven processes, including notifying caregivers and medical staff when alarm criteria are met. This latter approach represents an evolution from the Web of Things to the Sentient Web as a synthesis of the IoT, cognition and machine learning.

3.2 Web of Things

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 don't 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 has recently published proposed 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
- Web of Things: Thing Descriptions
- Web of Things: Scripting API
- Web of Things: Binding Templates
- Web of Things: Security and Privacy Guidelines
- Web of Things: Current Practices

The question that has still to be addressed is how the Web of Things can be applied to monitoring the condition of elderly and frail patients at home. 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[3] integrates a 3G modem and sends the measurement via the mobile

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

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.

In principle, we could define a Thing with a URL that identifies the particular weighing machine. An HTTP GET request on the URL would return the Thing Description as a JSON-LD resource. The Thing has a numeric property for the weight. The unit of measure would be 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 even 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 patient. The use of a Thing Description is hidden from services which interface with the graph database. It would thus make no difference to the services if measurements were pushed to the Gatekeeper cloud via HTTP PUT/POST requests from a device gateway.

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, as for the previous example, it would be simpler for the device gateway to push the buffered measurements to the Gatekeeper cloud.

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. There is no role in this for the Web of Things.

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 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, 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[4] 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. Once again, there is little benefit for using a Thing Description, particularly as 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.

Trying to model an HL7 FHIR resource as a Thing Description would only complicate matters. A much better idea is to implement a driver that pulls the XML or JSON resource from 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 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, e.g. "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).

3.3 Web of Things and Open API

Open API (formerly known as "Swagger") is a text-based format for describing REST APIs. At this point, we have yet to establish just how important Open API will be to Gatekeeper. Nonetheless, it seems reasonable to ask how Open API services could be exposed as "things" as a means to decouple client application code from having to deal directly with the REST APIs.

The Web of Things is more general and describes things at three levels:

1. The kinds of things and their inter-relationships as needed to support semantic interoperability.
2. The local object model exposed to client applications in terms of the data model for the object's properties, actions and events.
3. Protocol details for use by Web of Things client platforms to communicate with servers that expose things. This enables the Web of Things to work with a variety of ecosystems using different standards.

To expose Open API services and Things you will need to:

- Assign a URI for the Thing Description,
- Transform the Open API description to the Web of Things communication metadata.

[4] e.g. MediLynx PocketECG

- Design and describe the object model by which the service will be exposed to client applications.
- Add other metadata as needed to enable semantic interoperability.

You further need to check that your Web of Things client library supports Open API. The library would be responsible for mapping the object model exposed to client code to the REST API exposed by the Open API service.

4 Technologies for Gatekeeper Pilots

This section reviews the emerging technical requirements for the Gatekeeper pilots as understood from the working version of D6.1 "Medical use cases specification and implementation guide" that was available when this report was being written.

The following table summarises the technologies that are cited by each of the seven reference use cases described in D6.1:

Table 1 - Technologies for reference use cases

Technology	RUC1	RUC2	RUC3	RUC4	RUC5	RUC6	RUC7
Mobile app	✔	✔	✔	✔	✔	✔	✔
Chatbot	✔	✔	✔		✔		
User's website with questionnaires	✔			✔			
Professional's website	✔	✔	✔	✔	✔	✔	✔
Wearables and other medical devices	✔	✔	✔	✔	✔	✔	✔
Home sensors and robots	✔		✔				✔
External medical records	?	?	?	?	?	?	?
Miscellaneous.						VR & AR	

At the time of writing the requirements for the reference use cases in respect to external medical records was unknown. This report therefore considers a range of plausible possibilities and their interoperability mechanisms.

The following subsections provide further details about each of the technology groups.

4.1 Mobile apps

This covers applications that run on mobile phones and tablets, possibly in association with consumer wearables such as smart watches. Mobile apps can be used as gateways for ingesting data from monitoring devices, as well as for providing user interfaces for monitoring, alarm notifications and enrolling/un-enrolling devices. Mobile apps can be divided into native applications and Web applications that run within a web browser.

Native applications are executed directly by the device's operating system. The two most popular are: Google's Android and Apple's iOS. On Android applications are implemented using the Java programming language, whilst on iOS, applications can be implemented in either Objective-C or Apple's Swift programming language. Native apps are typically available from a centralised app store system, e.g. Google Play and Apple's App Store.

Mobile Web applications are implemented in HTML and JavaScript, using a wide variety of application frameworks. To minimise privacy concerns, Web browsers offer a more

constrained set of APIs compared to device native applications. This could be an issue for Gatekeeper pilots, but on the other hand, Web applications are easier to develop compared to device native applications.

4.2 Chatbots

Chatbots offer the ability for users to interact with computer systems using constrained natural language dialogues via text or speech. The growing popularity of smart speakers for home use, such as *Amazon Alexa* and *Google Home* with well-established frameworks for third-party developers, make them an attractive option for Gatekeeper pilots. Chatbots can also be integrated as part of native mobile apps or browser-based Web applications.

Chatbots can be written in regular programming languages, such as JavaScript, Python, and Java, or using higher level frameworks, such as AIML or Dialogflow. One approach involves pattern matching to extract the user's intent and any associated parameters, for example, if the user says: "what is the weather right now?", the intent is a request for the current weather at the user's location. This intent can be passed on to an application that looks up the required information and formulates a suitable response. Other intents may result in invoking specific actions, e.g. to dim the room's lights, or to increase the volume of a smart speaker.

Another approach is to use a goal directed dialogue involving a sequence of questions to the user that gathers the information needed to perform the desired task. This may involve starting with an initial open-ended question to determine what the user wishes to talk about, and to offer suggestions if the user gets stuck. For spoken dialogues, it may be necessary to ask for a confirmation when the user's spoken utterance can't be recognised with a high certainty.

The development process involves work on the natural language dialogues and work on backend services. Amazon Alexa, for instance, allows applications known as "skills" to be written in JavaScript with the ASK software development kit for NodeJS, and to use HTTP to access external services via REST APIs.

Chatbots may be developed using machine learning from a large corpus of training examples, which makes for more natural dialogues. However, there are risks for applying machine learning post deployment, as Microsoft found with its "Tay" chatbot for Twitter, which was designed to mimic the language patterns of its users, and had to be shut down after mimicking racist and sexually-charged utterances by other Twitter users.

To support improvements, it may be necessary to ask users for permission to record dialogues for analysis, e.g. to extend the vocabulary, language usage and response patterns. This raises serious privacy concerns and requires very careful consideration as how to provide appropriate safeguards. Special processing may help to anonymise logged dialogues prior to storage.

4.3 User Questionnaires

Native apps and Web apps can be used to collect information from users via form filling. This also relates to information gathered from family members and informal caregivers. User questionnaires define the questions to be asked, how they are ordered and grouped, any intervening instructional text and what the constraints are on the allowed answers. Questionnaires can follow an interoperable protocol since they can be communicated using

the Resource Questionnaire - Content and the Questionnaire Response resource of HL7/FHIR.

4.4 Professional Websites

For doctors, nurses and other caregivers, pilots should be able to provide websites designed for access from desktop computers, laptops as well as mobile devices (smartphones and tablets). The websites should be designed to be secure, easily usable and accessible to people with disabilities. The usability needs to be tested with different users prior to launch. Each pilot will need to identify what information is needed by healthcare professionals, caregivers and patients.

4.5 Wearables and other Medical Devices

Medical grade devices are certified to provide clinically actionable data. There are four broad classes ranging from low to high risk (Class I, Class IIa, Class IIb and Class III). The classification of medical devices is a 'risk based' system based on the vulnerability of the human body taking account of the potential risks associated with the devices. For further background, see:

- Guidance document – classification of medical devices (MEDDEV)

Consumer devices such as smartphones, wristbands and smart watches, can indicate that there is something to explore further. Both kinds of devices can play a valuable role for home healthcare.

Devices can be standalone, or connected wirelessly, e.g. via Bluetooth or WiFi, or using a physical cable, e.g. via a USB connection.

Personal alarms and security systems are devices that can summon help if the user falls, wanders off or has a problem at home, including intercom systems that allow family members to see who is knocking at the door. Alarms can be triggered if:

- The user falls over or out of bed
- The user has a fit
- The user wanders off or gets lost
- The room is either too hot or too cold

The following shows examples of commercially available devices that are worn around the neck or wrist, and can summon help on falling or when the wearer presses the panic button. These typically alert family members/caregivers and may allow for two-way voice conversation. The devices vary considerably in their range. Those using Bluetooth are very short range and are unlikely to work from a different room. Other devices may be designed to work even when the wearer is outside in the garden.

Figure 1 - Examples of alert devices

BlueStar Home Medical Alert	SureSafeGO 2 Anywhere Alarm	GreatCall Lively	Buddi wristband
Fall detector and panic alert with 200m range, connects to landline phone with two-way voice conversation.	Fall detector, panic alert and GPS and integrated mobile phone for two-way voice conversation.	Fall detector and panic alert, connects via Bluetooth to wearer's nearby mobile phone which needs to be within 10m	Fall detector and panic alert, connects via Bluetooth to wearer's nearby mobile phone or Buddy clip device (<10m)

Other kinds of devices include monitors, data loggers and single-reading devices, such as a weighing machine to monitor body weight. See section 5 for devices of interest to the Gatekeeper pilots.

4.6 External Medical Records

This involves access to existing systems, either local to a clinic or hospital, or remote. HL7's FHIR involves secure REST based APIs with XML or JSON as data formats. Recent work on the HL7 FHIR International Patient Summary (IPS), facilitates access to a snapshot of health data or electronic health record extract with key information such as conditions, medications, allergies, recent operations, implantable devices, etc. in an interoperable format. The means to store and analyse data over time is key to evaluating the evolution of a disease.

In principle, Gatekeeper could also make use of remote access to pharmacological databases, e.g. to check for possible problems when users are prescribed multiple drugs. The ability to connect to external services such as drug to drug interaction databases and information regarding safe medication use through web services can help advance patient and citizen empowerment.

4.7 Miscellaneous

One pilot intends to explore the use of virtual reality and augmented reality. How will this be interfaced to the software systems and services? How will it use health data of the user of the technology? The answers to these questions are not yet known at the time of writing this report.

4.8 Sensors and Actuators

The pilots will use either consumer or medical grade devices, such as:

- Motion sensors (accelerometers, gyro sensors)
- Magnetic door/window sensors
- Environmental sensors e.g. temperature and humidity
- Blood pressure sensors

- Pulse oximeters (heart rate and oxygenation)
- ECG sensors
- Glucometers
- GPS based geofencing
- Electric power usage (e.g. TV)

For the most part this will be connected by Bluetooth, but other choices include Z-Wave, WiFi, 3G/4G cellular networks, and USB cables. As yet, the pilots foresee little need for actuators, apart from smart home devices for controlling the temperature and lighting, as well as smart speakers for voice control of radio, television etc.

The use of actuators for medical purposes has regulatory implications for the design and conduct of pilots, involving higher costs, longer audits and extra documentation requirements.

4.9 Data Collection

Sensor data will mostly be collected via mobile apps on a smartphone or tablet. In some cases, the sensors will connect to a home hub/gateway, such as the Samsung SmartThings Hub. In other cases, sensors will use cellular networks to connect to remote gateways.

Some data is collected at the time of measurement, e.g. when standing on weighing scales, or when taking a blood pressure reading. Some devices generate live streams of data, e.g. an Electrocardiogram (ECG). Other devices may need to buffer data for periodic upload. This may further involve pre-processing to reduce the amount of data that needs to be buffered and transmitted, as well as to transform it into a more useful form.

An example is a solid-state accelerometer that generates large amounts of raw data. This is processed locally to measure the number of steps walked in a day, whether the patient has walked up and down stairs, the patient's sleeping patterns, and to call for help as a matter of urgency if the patient has fallen and is unable to get up again. Fall detectors sense movement followed by an abrupt stop.

Most portable devices such as a wristband or smart phone need to be regularly charged. This is likely to mean gaps in data collection. In addition, Bluetooth doesn't go far, especially in some buildings. A solution is for devices to buffer data until they are able to reconnect. We therefore can't rely on a live stream of data.

Sensors are but one source of data that an application may want to process. Additional sources may need to be considered. For instance, most pilot scenarios require access to the user's healthcare information. This information may come from multiple sources:

- The patient himself via an application questionnaire and by keeping a diary
- The caregiver via some application questionnaire
- Medical records accessed remotely (medical history and test results)
- Pharmacological databases
- etc.

The need to combine data from different sources obviously creates interoperability constraints on mechanisms to retrieve that data (APIs and formats), and on semantic vocabularies, also known as ontologies, used to represent and make sense of the data to support decision making. However, devices and information systems are likely to use different data models and data formats. The solution is to use a common data format and ontology that these different data models and data formats could be mapped to. That is

easier said than done. Nonetheless, this is a critical part of exposing data from monitoring devices as digital twins in an integrated dynamically updated knowledge graph.

Ontologies used in the Gatekeeper project will be described in "D3.4 – Semantic Models, Vocabularies & Registry". Ontologies to be considered typically those defined by the W3C (such as SOSA/SSN[5]), ETSI (SAREF), and those used by HL7 FHIR.

4.10 Application Execution Environments

The technology inventory conducted by Pila Sala further shows that the pilots will use a combination of mobile apps, hubs and gateways, and cloud servers. In many cases Gatekeeper partners have already developed mobile applications for Android and iOS, exploiting the native APIs and programming languages. The Samsung SmartThings Hub supports the Groovy programming language, but another possibility for hubs/gateways would be to use NodeJS, e.g. on a Raspberry Pi.

Cloud servers can be used for analytics, machine learning, event detection and notification and long-term storage. The technology inventory shows that most solutions provide REST APIs for external access. The pilots will also need Web servers to support web applications that can be accessed using smart phones, tablets or desktop computers.

In principle, mobile apps, hubs/gateways and cloud servers could all involve libraries for exposing and consuming "things", decoupling application code from the underlying details for the communication protocols and data formats.

Apple has for years prohibited mobile apps from downloading and running executable code and interpreted code. For Apple phones and tablets, this means that either Gatekeeper services need to be pre-built as part of the mobile app, or that the service needs to run in the cloud with the mobile app acting as a gateway for forwarding data. For more details, see section 3.3.2 of the Apple Developer Program License Agreement and App Store Review Guideline 2.5.2.

This suggests that Gatekeeper pilots may wish to focus on the cloud as a basis for dynamically installing and managing services, with native mobile apps used in a restricted role as data gateways, and built-in user interfaces.

[5] https://www.w3.org/TR/vocab-ssn/

5 Gatekeeper Architectural Patterns

A systems architecture provides a conceptual model that defines the structure, behaviour and other views of a system. This can include physical and logical components and their interrelationships.

Interoperability requirements depend on the choice of reference architecture that the project envisions and on the scenarios that pilots will explore. All cases allow for the storage and analysis of data to track changes in the patient's condition.

The reference architecture for the project was not settled when this document was written. It will be described in "D3.2 – Gatekeeper Web of Things (WOT) Reference Architecture". To discuss interoperability mechanisms, three main directions can be envisioned.

5.1 Cloud-based architecture

One approach is to use a **cloud-first** architecture. In this architecture, all data and all services are hosted in the cloud, and sensors and actuators are duplicated as digital twins hosted on servers under the control of the service provider.

The main benefit of this approach is that it allows service providers to retain complete control over data and services. Service providers can design the service with a purely centralized perspective, that is both easier to deploy and to maintain. For instance, it can implement data retention policies that suit the service, and instantiate more servers as needed to meet the computational and network demand in real-time.

One drawback is that this approach exposes service providers to privacy and security risks. With the cloud-first approach, service providers are responsible for the security of medical and personal data held on their servers. Security breaches risk exposing user data that is inherently personal and private given the health scenarios addressed by Gatekeeper. From a technical perspective, this liability creates strong requirements on security mechanisms. It also creates significant financial and reputational risks from a business perspective.

5.2 Edge-based architecture

A second possibility is to take a completely opposite approach and focus on an **edge-based** architecture. In this model, user data remains on the user's premises as far as is practical. Applications that need to process that data also run close to the user as well, either locally or on the edge.

The main benefit of this approach is that it greatly reduces the surface of exposure for the users' private data, as it is not held in the cloud. A security breach on a particular user's system can only expose that user's data, reducing the liability for service providers. Also, users remain in control of their personal data, and can choose and control which services can access their data. Interestingly as well, services can continue to run locally when the user's Internet connection is lost, unlike the cloud-first approach. Benefits could be reduced consumption budget due to the connectivity switch-off, quality of Service to patient even disconnected.

The main drawback is that the implementation of this model requires a decentralized approach that may be tricky to implement, deploy and maintain. Service providers may not have real-time direct control over the devices on which their applications run, and no longer

control data storage. This creates the need to agree on interfaces and representations for data, and on the definition of runtimes for applications.

Local runtime may also not be suitable for running complex Artificial Intelligence (AI) algorithms on data and are de facto not suitable to run analyses that span multiple end users. A work around is to use federated learning algorithms in which models are downloaded from the cloud, improved with local data, and uploaded back to the cloud, without the need to transmit personal data to the cloud.

Another drawback is that the definition of the home network may not match that of a simple Local Area Network (LAN). It may span various telecommunication mechanisms such as Ethernet, WiFi, Bluetooth, 4G/5G, or Low-power Wide-Area Network (LPWAN), whose interconnection may require going through the cloud. Some scenarios may be difficult to achieve using a pure edge-based approach as a result. On the other hand, other use case scenarios may take advantage on edge-computing insuring autonomous local operation.

5.3 Federated architecture

The third approach is a **federated** architecture, involving keeping some information at the network edge and other information on different cloud-based systems. Here the idea is to minimise the amount of personal information held on any one server. An example is the approach taken by Gatekeeper partner Medisanté, in which they provide cloud-based access to data collected from remote medical devices. Each reading is associated with an anonymous identifier for the device it derives from. However, it is up to clients of the cloud API to relate these readings to particular patients.

With such a federated approach, clients can access information from multiple sources on an as needed basis, subject to certification, pre-agreed terms and conditions, and logged to an auditable distributed ledger. The approach can also be used for federated learning across edge systems and multiple cloud-based systems that each hold a limited set of data.

5.4 Gatekeeper Platform

The challenges relating to syntactic and technical interoperability can be delegated to the "connectors" that feed data into the Gatekeeper cloud platform. One example involves a wearable step sensor that uses Bluetooth to talk to a native app on a mobile phone. That app then uses HTTPS to upload the patient's step count, e.g. on an hourly or daily basis.

The Gatekeeper platform should support a uniform means for uploading data using HTTPS. For live streaming it would make sense to support Web Sockets. Data would be transferred in a common format, e.g. Chunks (see appendix B), JSON or JSON-LD.

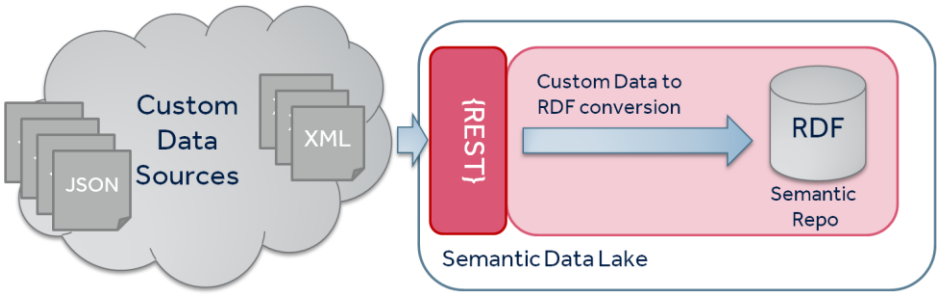
Another example involves a weighing scale that uses 3G mobile to transmit the patient's body weight to a connector in the cloud. The connector then uses HTTPS to forward the reading to Gatekeeper.

A further example is where data is pulled to the Gatekeeper platform via an HTTPS request to an external source, e.g. an electronic health record accessed via HL7 FHIR. In this case the "connector" can be considered as an app hosted by Gatekeeper.

Connectors hide the details of how they obtained the data, and are responsible for transforming data into the form expected by the Gatekeeper platform.

The following figure depicts the role of *connectors* for ingesting data from external sources:

Figure 2 - Ingesting data



Semantic Interoperability is guaranteed by the Gatekeeper data Integration components. Those components collect data in different data formats and convert them to HL7 FHIR ontologies (for health data) and to IoT ontologies (e.g. SAREF) for IoT sensors data so they can be used by the analytics services in Gatekeeper, as well as by the Pilot Applications.

The Gatekeeper platform will be expected to provide a management API for installing application services on behalf of the Pilots. The execution framework should ensure best security practices, for example, applications monitoring a given patient should be prevented from accessing data for other patients. At one level this can be implemented using access control for operations on the database, however, that by itself is insufficient.

To limit damage where a cyber-attacker has compromised a particular instance of an application, applications should be executed in separate address spaces, so that they can't access physical memory used for another patient.

This security model can be implemented by executing each application instance in its own operating system process, and relying on the operating system to segregate memory for each process. Interprocess communication is then used to support the APIs exposed to the applications.

Gatekeeper needs to provide an easy to use means for enrolling new devices and associating them with a given patient. The following explores some ideas for how that could be done for a wearable such as a wristband that uses Bluetooth to connect to a mobile phone:

1. Install the Gatekeeper app on the patient's smartphone, after finding it on Google Play or the Apple App Store.

2. Register the phone with the Gatekeeper system for the given patient. This could involve typing in the patient identifier or using the phone to scan a QRcode issued by the health service on behalf of that patient. It is likely that some second factor will be needed to ensure stronger security.

   This could for instance take the form of an easy to enter alphanumeric one-time code issued by the health service for that purpose. At this point Gatekeeper could set up public key based authentication to allow the mobile app to securely authenticate itself to the Gatekeeper system when needed.

3. The Gatekeeper mobile app can now be paired with the wristband. This may involve pressing a button on the wristband. The user will be given a clear visible confirmation by the mobile app that this step has been successful.

4. The mobile app can now initialise a "connector" for the wristband and register it with the Gatekeeper system for the given patient, along with the ontology to be used for the wristband's data.

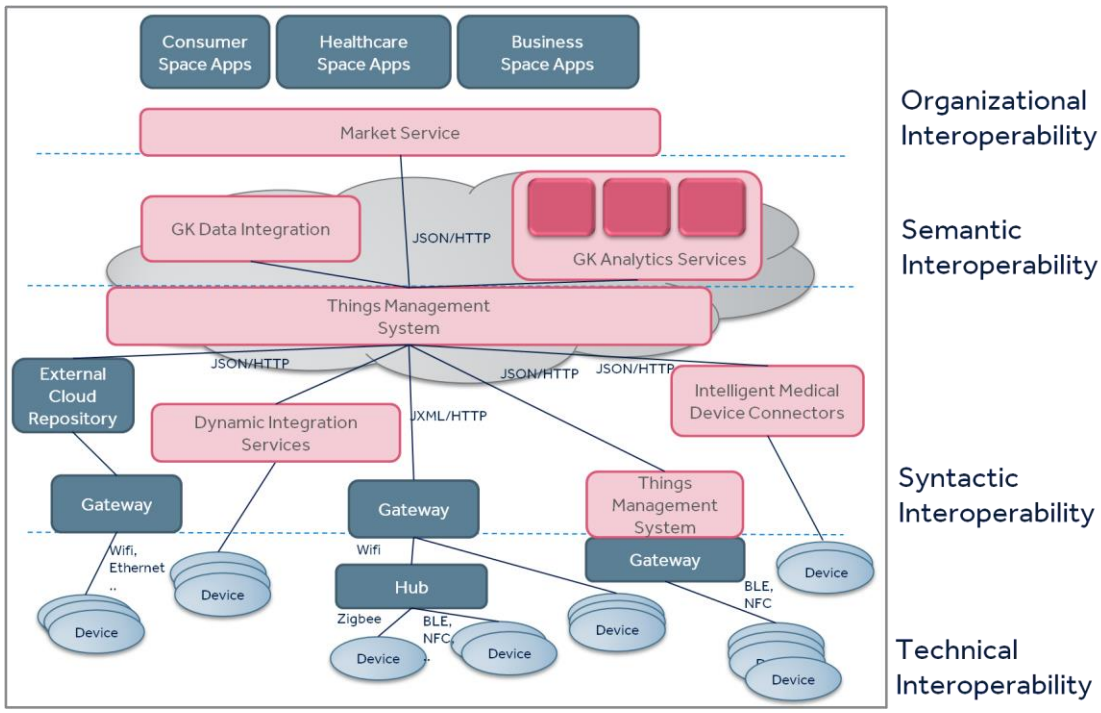
5. The wristband periodically sends data to the mobile app, which forwards it to Gatekeeper for ingestion into the graph database for that patient.
6. When it comes time to unenroll the wristband, this can be done using either the mobile app or by using a web application that serves as a management interface for caregivers and healthcare professionals.

6 Interoperability Layers

We consider interoperability at different layers of abstraction, where each layer depends on the layer below: Organisational, Semantic, Syntactic and Technical. This section introduces how each of these layers relate to Gatekeeper, and is followed by separate sections that provide more detail for each layer.

The following figure shows a high-level architecture schema for the Gatekeeper Platform, highlighting the interoperability challenge that each subsystem must fulfil.

Figure 3 - 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. (see Appendix A).

  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.

The Thing 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.

- **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 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.

7 Technical Interoperability in Gatekeeper

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.

If we adopt the Web of Things, the 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.

A simpler 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: XML, JSON and JSON-LD. An additional format is "Chunks", see Appendix B. This is a simple amalgam of RDF and Property Graphs that should be easier to use by the average developer.

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 under the hood technologies used for marshalling and unmarshalling serialization of data, in contrast to other formats such POJO or JavaBeans that are only for Java.

More details on the standards for HL7 FHIR, XML, JSON and JSON-LD will be given in deliverable D8.1. The following is provided as interim measure before D8.1 is released.

7.1 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[6] and was updated in 2010.[7]

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

[6] https://www.w3.org/TR/1998/REC-xml-19980210.html

[7] https://www.w3.org/TR/REC-xml/

focuses on simplicity, generality, and usability across the Internet, but it does not do anything to the data that is carried.

To understand XML, it is needed 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. It became a recommendation of W3C since February 1998 due to the numerous incompatible formats from many systems that was a time-consuming for web developers because they had to program converters to use those formats. Since then, this format is very extended and used along the community.

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

7.2 JSON

The JavaScript Object Notation (JSON]) 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.[8]

The JSON format is agnostic about the semantics of numbers, that means that it does not type of 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:

[8] See: https://tools.ietf.org/html/std90, http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf and https://www.iso.org/standard/71616.html

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

7.3 JSON-LD

This format is a modification of 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.

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. 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.

```
{
  "@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
}
```

7.3 Chunks

Chunks is a simple amalgam of RDF and Labelled Property Graphs that was inspired by work 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."*
>
> See: http://act-r.psy.cmu.edu/

Chunks makes it easy to express entities with multiple properties and labelled directed relationships to other entities. Chunks is further designed to support a blend of symbolic and sub-symbolic (statistical) information, that facilitates reasoning and machine learning, as well as the challenges posed by the uncertainty, incompleteness and inconsistency often found in the real world, and a major challenge for Data Science. As such Chunks enables Cognitive AI, that is, the next generation of Artificial Intelligence inspired by hundreds of millions of years of evolution and the means to give computing a *human* touch.

For more information see Appendix B and the W3C Cognitive AI Community Group.

8 Syntactic Interoperability in Gatekeeper

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 API's 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 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 will provide a detailed survey of standards and standardisation gaps.

8.1 REST interface

REpresentational State Transfer (REST) is an architectural style inspired by the Web. 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 doesn't 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 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 isn't 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 doesn't 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[9], sometimes abbreviated to HATEOAS, was coined to describe a core tenet of the REST architectural style. 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[10] is the Flickr web service[11]. Despite the "rest" in the URI, this was clearly designed as an RPC-style service, one that uses

[9] REST in Practice Hypermedia and Systems Architecture, By Savas Parastatidis, Jim Webber, Ian Robinson, Publisher: O'Reilly Media, Release Date: September 2010

[10] RESTful Web Services, By Leonard Richardson, Sam Ruby, Publisher: O'Reilly Media, Release Date: December 2008

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

HTTP as its envelope format. It's 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.

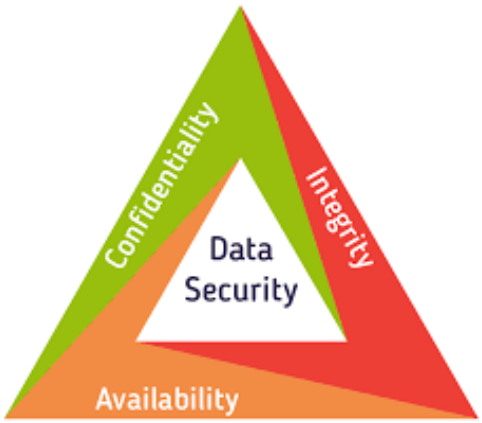
This optical illusion happens when an RPC-style service uses plain old HTTP as its envelope format, and when both the method and the scoping information happen to live in the URI portion of the HTTP request. If the HTTP method is GET, and the point of the web service request is to "get" information, it's hard to tell whether the method information is in the HTTP method or in the URI. Look at the HTTP requests that go across the wire and you see the requests you'd 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 doesn't 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's 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's 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.

8.1.1 Stateless and security

Both RESTful and Hybrid RESTful service rely on HTTP. This means that they are missing a security. 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 4 - 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 5246) 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 1).

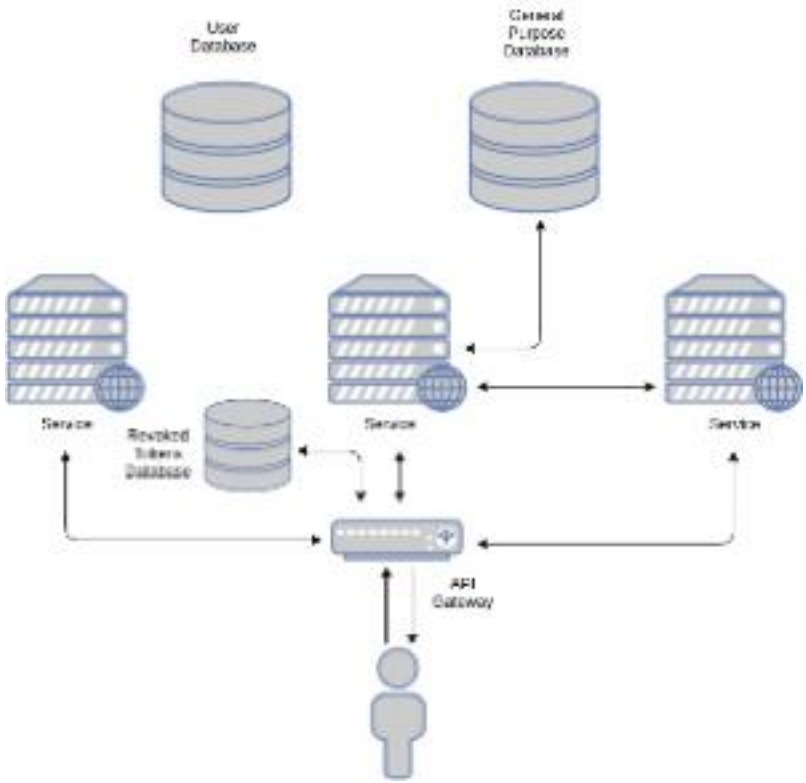
TLS can also be used for mutual authentication where both the server and the client can be assured of the other party's identity. For instance, you can use basic authentication to authenticate users but rely on TLS to authenticate the client and the server.

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[12] is an application that needs no knowledge of previous interactions and stores no session information, it is usually based on an architecture that doesn't need user data (see Figure 2). 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).

Figure 5 - A stateless service infrastructure

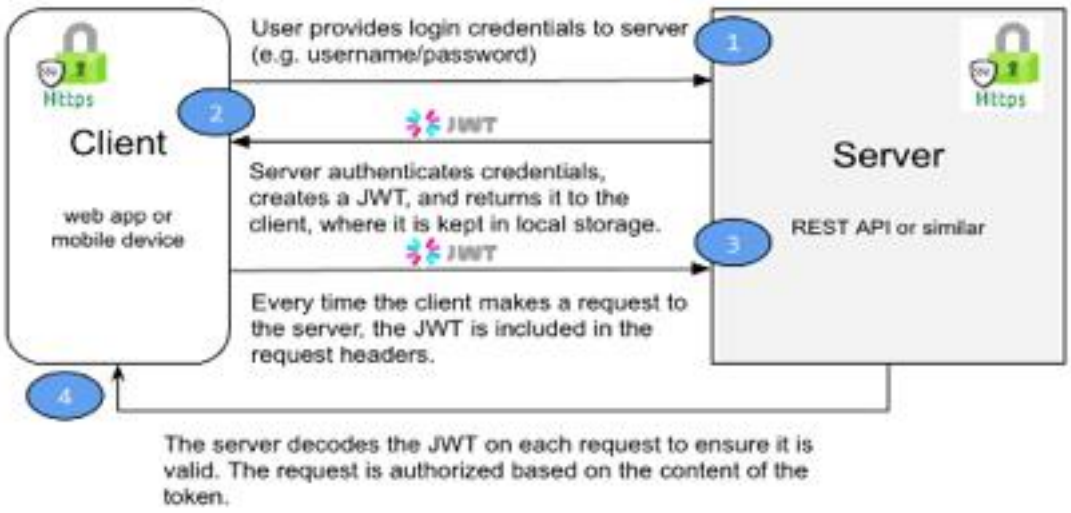


[12] Architecting for the Cloud AWS Best Practices February 2016, https://d0.awsstatic.com/whitepapers/AWS_Cloud_Best_Practices.pdf.

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. This means, at server side we don't maintain the state of a user. The server is completely unaware of who sends the request as we don't maintain the state. We can achieve the stateless authentication by using JWT (JSON Web Token). Token based approach solves problem of traditional approach in which server has to store Ids of Session and relevant data for each individual. One of the token based approach is JSON-based Open Standard (RFC 7519)[13] known as JWT (Figure 3).

Figure 6 - JSON Web Token authentication for a stateless service



[13] https://tools.ietf.org/html/rfc7519, JSON-based Open Standard (RFC 7519]

9 Semantic Interoperability in Gatekeeper

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 in respect to interoperability in the face of heterogeneous information sources, then gives an overview of the approach that Gatekeeper expects to take..

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.

9.1 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 from 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.

Ontologies used in the Gatekeeper project will be described in "D3.4 – Semantic Models, Vocabularies & Registry". Ontologies to be considered are typically those defined by the W3C (such as SOSA/SSN[14]), ETSI (SAREF), HL7 and terminology developers such as Regenstreif (LOINC, UCUM) and SNOMED Int. (SNOMED CT).

9.2 Labelled Property Graphs (LPG)

LPG are similar to RDF in being composed from 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.

[14] https://www.w3.org/TR/vocab-ssn/

9.3 Chunks and Cognitive AI

Chunks is an amalgam of RDF and LPG, where each chunk has a type, an identifier, and set of properties, whose values name other chunks to form graphs. In more detail, property values can be Booleans (true or false), numbers, names, string literals (in double quotes) or lists there-of. Property names themselves can act as chunk identifiers.

The term "chunk" is borrowed from Psychology. Chunks are used to express declarative and procedural knowledge. Cognitive AI combines symbolic knowledge (chunks) with statistics, rules and graph algorithms, inspired by advances in the cognitive sciences, that point the way to giving computer systems a more human touch.

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.

For more details see: https://www.w3.org/community/cogai/

9.4 SAREF Ontologies

ETSI started with the SAREF (Smart Applications REFerence ontology) specifications for energy, environment and buildings, and have extended this with extensions for smart cities, manufacturing, and smart agriculture and food chain domains. SAREF provides building blocks that allow separation and recombination of different parts of the ontology according to specific needs. The SAREF4CITY specification use cases include eHealth and smart parking, air quality monitoring, mobility and street lighting.

- Smart Applications Reference Ontology and extensions

ETSI TC SmartM2M is working to include more activity sectors and to complete the development of an open portal to gather direct contributions to SAREF by 2020. The stakeholders' evolving data model inputs can then be directly reflected in the ETSI SAREF and oneM2M specifications.

9.5 Ontologies in Healthcare

Gatekeeper needs to be aware of existing work on ontologies. Here are a few pointers:

- LOINC (Logical Observation Identifier Names and Codes) defines a common terminology for laboratory and clinical observations, and seeks to replace the idiosyncratic internal code values for identifiers as used by most laboratories and clinical services.
- HL7 FHIR defines a set of resources describing the social and health care domain, utilising existing terminologies (e.g. SNOMED CT and LOINC).
- Unified Medical Language System (UMLS) integrates and distributes key terminology, classification and coding standards, and associated resources to

promote creation of more effective and interoperable biomedical information systems and services, including electronic health records.
- Open Clinical lists some current work on medical ontologies
- ITEMAS healthcare ontology developed by a network of 66 healthcare centres in Spain.
- W3C Semantic Sensor Network (SSN) ontology for describing sensors and their observations, the involved procedures, the studied features of interest, the samples used to do so, and the observed properties, as well as actuators. SSN includes SOSA as a lightweight self-contained ontology (based upon IoT-Lite) and defining elementary classes and properties.

Gatekeeper will seek to re-use existing ontologies where practical.

10 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.

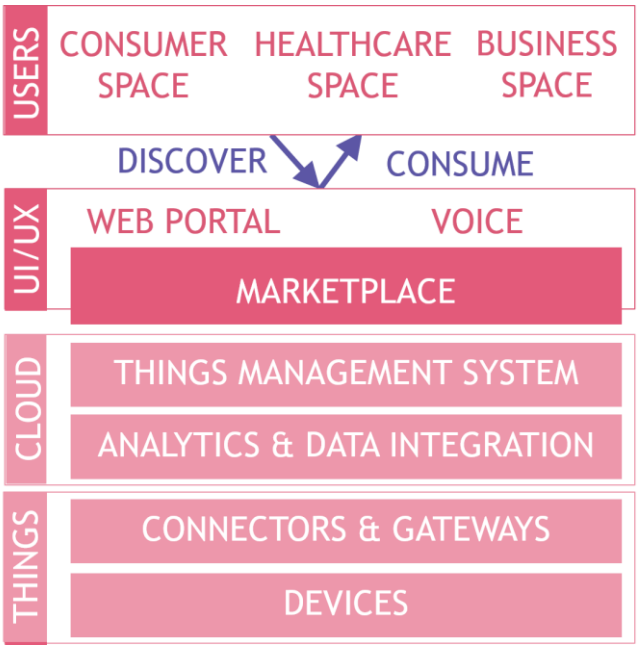
Gatekeeper is primarily about providing elderly and frail patients with improved care through the use of consumer and medical grade devices to monitor the patient's condition and enable appropriate action by the patient, caregivers and healthcare professionals. In respect to the aims for a Gatekeeper 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?

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 7 - 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 GK 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: a) Web Services, b) Sensors, c) Medical devices, d) Platforms/Closed solutions and e) Data.

Figure 4 shows the relation of the Marketplace to other GK components. Essentially, the Marketplaces expose Things (devices, gateways etc.) through the GK 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.

Gatekeeper Trust Authority

An example of Organisational Interoperability is the Trust Authority and Open Distributed Ledger of Gatekeeper which is a blockchain-based platform that

- is responsible for certifying the Things of the Gatekeeper platform based on a set of standards and for calculating the corresponding levels of certification for these Things. This will be used in order to secure the Things that will be included or submitted in the Gatekeeper Marketplace.
- it 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

The solution will be based on Hyperledger Fabric chaincode for the calculation of the levels of certification and for keeping the audit trail. It will also provide a connection with a Fabric CA server in order to issue and manage certificates for the Things.

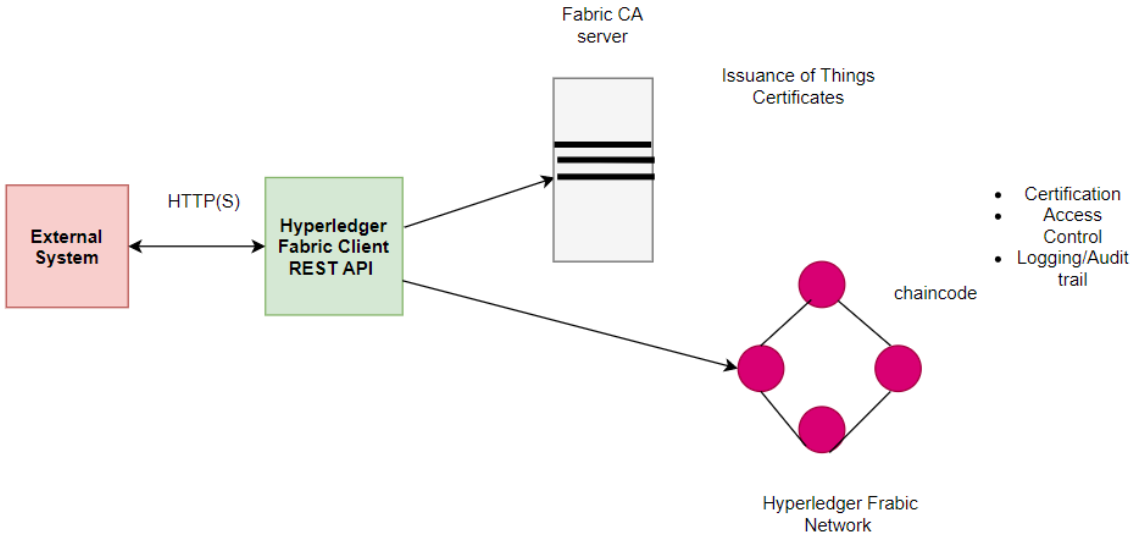
Interoperability mechanisms

The solution will be able to communicate with any infrastructure that is able to communicate over HTTP(s) by exposing a RESTful API and by using JSON as the data format. The component diagram depicting the communication between this platform and any compatible external system is depicted in the Figure below.

An example of Organisational but also Syntactic Interoperability is the Trust Authority Validator. A subcomponent of the Trust Authority and Open Distributed Ledger component, which will be developed in frames of the Task 4.5, will be responsible for certifying the Things after validating them against a set of standards.

This subcomponent, which will be called 'Validator' hereafter, is related with the Interoperability aspects of the Gatekeeper platform since it will validate Things that comply with the W3C WoT standards as described in T3.3. Furthermore, the Validator will validate Things and assign to them levels of certification/trustiness based on the set of standards considered in frames of T8.1.

Figure 8 - Gatekeeper will use Hyperledger Fabric for security logs



11 Additional Considerations

This is a collection of additional considerations that impact interoperability.

11.1 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[15], 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[16]. 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.

11.2 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

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

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

user's personal history, and that of others like him or her. There are plenty of challenges to be identified and discussed, and a W3C Workshop is now at an early stage of planning to address this.

11.3 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 builds 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 9 - 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 in respect to applying analytics and AI on the collected data in a safe manner.

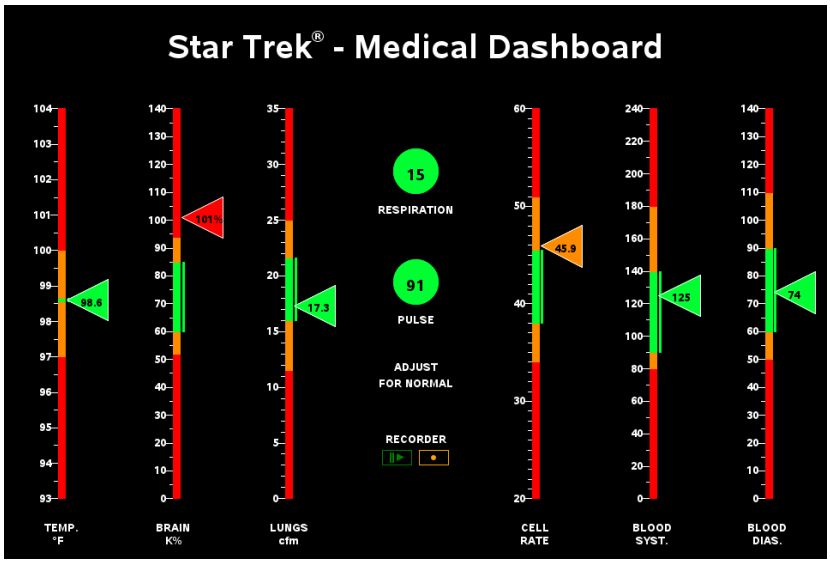
Some design challenges include:

- 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.

12 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 10 - Medical dashboard



Can Gatekeeper provide the real-world equivalent of the Star Trek sick bay monitor in respect to the well-being of elderly and frail patients? With grateful thanks to Robert Allison.

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. The proposed Gatekeeper platform integrates a graph database, statistics, rule engine and graph algorithms. HTTP is used for uploading data to the platform from sensors, for access to electronic health records, and for the means to provide Web based tools for patients, caregivers and clinical staff.

A uniform framework for storing and manipulating information will make it much easier to create services that combine multiple information sources. The main barriers to achieving this include: a) having the imagination and ambition to fully understand what is needed to support modern practices for home healthcare, b) the reluctance of device vendors to provide the information needed to implement gateways for ingesting data from sensor devices, and c) challenges for integrating different information sources using different data formats, data models and underlying concepts. All of these barriers can be overcome if we work together!

Appendix A Gatekeeper technologies

This section describes the solutions to be provided by each of the Gatekeeper technology partners, along with the associated interoperability considerations

- Short description of the technology, its purpose and relevance to the pilots
- Status of the implementation, e.g. maturity, availability and level of support
- What are the interoperability mechanisms for integrating this technology, including the list of standards used
- Where to find more details, e.g. online documentation
- Point of contact for any questions (email address and/or website link)

A.1 HL7

HL7 is a standards development organization that provides a comprehensive framework and related standards for the exchange, integration, sharing, and retrieval of electronic health information. HL7 standards cover different aspects of interoperability (e.g. information models; service and system functional specifications; API) at different abstraction levels (e.g. conceptual, logical, implementable).

The most relevant standard for the purpose of this project is HL7 FHIR. HL7 FHIR is a fully computable standard that combines the best features of HL7's v2, HL7 v3 and CDA product lines while leveraging the latest web standards and applying a tight focus on implementability. FHIR solutions are built from a set of modular components called "Resources". FHIR is mainly designed for REST applications, but it can also support document-based, messaging and services-based interoperability paradigms. FHIR resources are typically accessed through HTTP-based REST APIs and can be represented with XML, JSON or RDF turtle. The RDF turtle representation will be likely superseded in the next release by JSON-LD 1.1. The last published HL7 FHIR release is R4, including normative content.

FHIR is widely adopted at the global level and it is supported by a large community of practice.

FHIR profiles and implementation guides play a relevant role in the adoption and usage of the base standard, allowing for validation and increasing interoperability. They define, by means of conformance resources, how FHIR should be used in specific contexts and scopes. They also specify which terminologies (e.g. LOINC, SNOMED CT) to use and how. For the scope of this project is worth to mention the International Patient Summary FHIR Implementation Guide. The International Patient Summary (see overview) is a cross-SDOs initiative (e.g. HL7, ISO, CEN, IHE), that is also funded by the EC, and specifies a minimal set of health information that can be used everywhere by anyone.

For more details, please contact: Giorgio Cangioli <giorgio.cangioli@hl7europe.org>.

A.2 Mysphera

Mysphera can contribute monitoring software they previously developed for use with elderly patients at home. This includes a mobile app for Android and iOS phones, and tablets that collects battery level and GPS location data for geofencing, as well as using Bluetooth Low

Energy to collect data from smart home sensors for temperature, humidity, a magnetic sensor for door opening and closing, and presence in rooms.

- What kinds of sensors specifically and how are they to be installed and configured?
    - The devices for the home monitoring are developed by Mysphera, under the brand of LOCS system. There is one device that includes a presence sensor, based on Texas Instrument PIR sensor, as well as temperature and humidity sensors also from Texas Instruments, HDC1010. The magnetic sensor used for open/close door (or window, etc.) is also based on Texas Instrument sensors.
    - The configuration is done by linking the MAC addresses of BLE chip to a code of the installation, conforming a kit, and configuring the gateway to collect and process only the data coming from those MAC addresses. Also, each device is assigned a label to indicate in which room is going to be installed, and all those fields are kept in the central database of the solution.
    - Installation is done by a technician that visits the designated home with the code that corresponds to the kit, places the sensors, switches them on and checks with the gateway that the signal from all the sensors is well received. In occasions adjustment on sensor placement or gateway placement need to be done to ensure proper reception of the signals from all devices.
- Do the elderly patients need to keep the phones with them at all times?
    - The smartphones of the elderly patients are only providing the outdoor functionality, so it is only needed when the elderly leave the home to go outside. At home, they don't need to carry any device, the home monitoring system works totally unobtrusively, without any interaction of the elderly user.
- Is there a home gateway other than the patient's phone? If so please give details.
    - For the home monitoring there is gateway that currently is using an Android Tablet (7.x+) as platform and is executing LOCS gateway software, this tablet is placed in the home at a fixed point where it can receive the Bluetooth transmission from all the devices at home, and it incorporates a 3G SIM card to send the raw and processed data to the cloud server

The mobile application uses the 3G/4G cellular network to upload data to a cloud server running FIWARE. This data is then exposed to Android smartphone apps (one designed for elderly patients, the other for caregivers, e.g. family members and relatives). A desktop web application is used to provide access to clinical staff.

- How is geofencing configured and who by?
    - Geofencing is configured in the caregivers app and in the professional web portal and by the informal caregiver or the professional caregiver. This functionality is only activated if the outdoor monitoring application is installed and assigned to the installation code of the elderly person. That means that an elderly person can have the home monitoring alone, the outdoor monitoring alone or both solutions combined.
- How are alerts notified to the patients, caregivers and clinical staff? This question seeks to understand the technical means, e.g. SMS, polling, or a push-mechanism over HTTP or Web Sockets, along with the standards used to achieve this.

- Alerts are shown in the app and in the web portal, as well as by push mechanism in the smartphone of the informal caregiver that is using the app. Other mechanisms, such as email and SMS are in the roadmap for future improvements of the solution.

The software was developed in a previous European project (ACTIVAGE) and has been deployed in 530 homes with over 1500 users for over 12 months. MYSPHERA commits to supporting the software for the lifetime of the Gatekeeper project.

The mobile apps are available on Google Play and the Apple App Store.

- What OS versions are supported for Android and IoS?
  - Available in the stores are the following apps:
    - LOCS Outdoor: only supported in Android 8 or above
    - LOCS Family: it supports Android 6 or above, and iOS 8 or above
- How are the applications installed and configured?
  - The applications that are used by the elderly person (i.e. home monitoring gateway or outdoor monitoring) are installed and configured by the technical team in Mysphera in charge of preparing the installation, and final configurations may be done also by the care service providers once they decide which user receives the installation.
  - The informal caregiver app is installed directly by the user himself, but he/she receives training and support for configuration during the installation of the home solution and through the use of video tutorials or support calls.

The cloud server uses HTTPS along with user ID and password, together with a device-based JWT token for authentication. The server is deployed as a Docker container for easier installation. It further supports the ETSI NGSI v2 based context information manager. Location data is exposed using GeoJSON.

- Is the HTTPS server for the mobile apps and web application integrated as part of FIWARE or is it a separate component?

The HTTPS server is integrated as part of FIWARE that on one hand is managing authentication and role access, and on the other hand offers an API to access and sending data, and as such it could be used to develop new applications that use the data collected by the solution.

For more details, please contact: Pila Sala <psala@mysphera.com>.

A.3 Samsung

Samsung are contributing a variety of technologies:

Samsung Health – a mobile application pre-installed on the Galaxy S3 phone and downloadable from the Samsung Galaxy Store. Versions are available for Android 6+, iOS9+ and Tizen (for Samsung Gear and Galaxy watches). The application collects data from user interaction with the app, on-phone sensors, wristbands and watches, smart scales and other smart home devices. Data is viewable within the mobile app, and is also uploaded to the Samsung cloud.

ACTIVAGE – is an application developed by Samsung (for the ACTIVAGE project?) that runs as a mobile web app in a browser or as a native app on Android 6+ and Tizen. It relies on the user wearing a smartwatch to detect movement (e.g. to count steps, monitor sleep patterns), and transfers this data to a smart phone app via Bluetooth Low Energy. The phone uploads the data to a cloud server for further analysis, and monitoring by clinical staff.

- How are hydration, blood glucose, blood pressure, weight, caffeine intake monitored, TV viewing, bathroom usage etc. monitored? What are the standards and data formats?

Bixby Voice – a virtual assistant implemented as a mobile app on Samsung phones. The assistant can invoke the Android API to implement a variety of services, e.g. to take a photograph, to set alarms as reminders, to show a movie on a connected smart TV, etc.

- How is Bixby programmed, and what would this entail for Gatekeeper pilots?

Samsung SmartThings Hub – this is a data relay that transfers data collected from SmartThing sensors (using Bluetooth, Z-Wave or WiFi) and uploads it to a cloud server. Likewise, applications running in the cloud can invoke SmartThing actuators via the hub. The available sensors include motion sensors, energy consumption sensors, magnetic door/window opening and closing sensors.

- How can Gatekeeper pilots interface to this data, either via the cloud or directly via the hub?
- Could Gatekeeper pilots develop applications for execution on the hub?
- Could Gatekeeper pilots develop applications for execution on the cloud server?
- How would Gatekeeper pilots utilise the security and device management capabilities, e.g. as exposed by Samsung Knox?

Samsung Galaxy Watch – a smartwatch with Bluetooth, NFC, WiFi, GPS, accelerometer, barometer, gyro sensor, heart-rate sensor, light sensor, microphone and vibrator.

- How does the need to regularly recharge the device's battery effect its potential use for monitoring the user's sleep patterns, heart rate, fall detection etc?

For more details, please contact: Carlo Allocca <c.allocca@samsung.com>.

A.5 BioAssist

BioAssist has developed the HeartAround platform as a means to monitor elderly patients who are living independently at home. Data is collected from wearables and other sensors and relayed via a tablet-based mobile app to the cloud where it is stored and used for analytics, generation of alarms, and web-based user interfaces for patients, relatives, doctors and other caregivers.

Sensors are connected to the tablet via Bluetooth and include a pulse oximeter, blood pressure meter, glucometer, spirometer, weighing scale, and physical activity tracker. Supported devices are from manufacturers such as manufacturers such as iHealth, Beurer, Phillips, Ascensia, MIR, Jumper. Patients are directed to take daily readings which are automatically transmitted to the cloud server.

Each patient is associated with a cloud-based electronic health record (EHR) which includes medical test results, medications and allergies. When the measurements taken by the patient

exceeds given thresholds, the doctor is notified by a preselected means, e.g. push notification, email or SMS. The doctor is able to have a video chat with the patient on a weekly basis via the tablet.

For more details, please contact: Ilias Maglogiannis <imaglo@bioassist.gr>.

A.6 Biobeat

Figure 11 - Biobeat sensor and monitor

Biobeat has developed a monitoring platform consisting of wearable wireless optical sensors. We have two configurations – a patch monitor attached to the patients' chest or a wristwatch monitor. The wristwatch works for 3 days and then is recharged for up to 2 hours, and is intended mainly for prolonged monitoring, i.e. monitoring of chronic patients at home or home care facilities. The patch works for 5-7 days, it is disposable, and intended for pre-hospital and in-hospital use. They transmit 15 parameters in real time and in several measurement rates through BT to a gateway (or an individual's cell phone if at home) and from the gateway/cell phone to the company's cloud (AWS-based) and from there to the health care providers.

As it is cloud based there is no limit to how many people can be monitored at any given time and for how long. The data can be displayed on a tablet, laptop, monitor screen, etc. The data includes non-invasive cuffless blood pressure, pulse pressure, mean arterial pressure, heart rate, heart rate variability, respiratory rate, blood oxygen saturation, stroke volume, cardiac output, cardiac index, systemic vascular resistance, temperature, sweat, movement, and one-lead ECG (in the patch only), all are continuously measured in all patients. For each parameter an alert is set, and the caregiver can look at the trend from the moment the patient is monitored.

We have FDA clearance for non-invasive cuffless blood pressure, heart rate and saturation, and CE Mark approval. We follow strict HIPAA, GDPR and privacy protection regulations. The data in our cloud is de-identified. Another feature is based on our advanced algorithms. We can analyze the big data collected, which is comprised of hundreds of millions of measurement points.

A.7 Medisanté

Medisanté provides a cloud-based hub solution (ELIOT) to collect patient data via the 3G/4G cellular network direct from medical IoT devices, e.g. glucometer, weighing scale, blood pressure meter, ECG event recorder, spirometer and pulse oximeter. Medisanté themselves

can supply blood pressure meters, glucometers and weighing scales. Data is exposed by the cloud server via a REST API as JSON structured data following the HL7 FHIR v1 or v4 format.

For more details, please contact: Imad Ahmed <imad.ahmed@medisante-group.com>.

A.8 UPM

UPM has previously developed monitoring software in support of the treatment of Parkinson's disease. This includes a mobile app for Android phones that uses the phone's built-in accelerometer to monitor movement, tremors and falls. The raw sensor data is pre-processed and periodically transmitted to a cloud-based server (HOOP) on Ubuntu Linux, which makes the processed data available to the web application used by clinical staff to monitor the patient's well-being.

The software was developed in a previous European project (ACTIVAGE) and has been deployed in a pilot with 50 patients over a period of many months. UPM commits to supporting the software for the lifetime of the Gatekeeper project.

The mobile app is available on Google Play and supports Android version 6+. UPM is considering extending the app to work with Bluetooth-based wristbands and insole detectors, as a means to gather data when the end user has put the phone down. These devices would be paired with the user's phone and cloud server by clinical staff during the enrolment process.

UPM is also looking at the potential for integrating with the Mozilla software stack for IoT gateways, as a means to collect data and forward it to the cloud and to local web applications.

The cloud server provides a REST API over HTTPS using JSON Web tokens (JWT, RFC 7519[17]) as OAuth 2.0 Bearer Tokens for access control, and a JSON based format for motion data that was developed by UPM. Data can be provided as per HL7's specifications for integration with electronic health records.

Short description of the API:

- HTTPS POST to upload a block of JSON data
- HTTPS GET to query for a block of JSON data

The Web application is designed to work with both desktop and mobile phones and tablets, and uses a conventional user ID and password pair over HTTPS for authentication. Clinical staff can register to be notified by SMS messages to their phone when the system infers a strong likelihood that the patient has fallen down and needs attention.

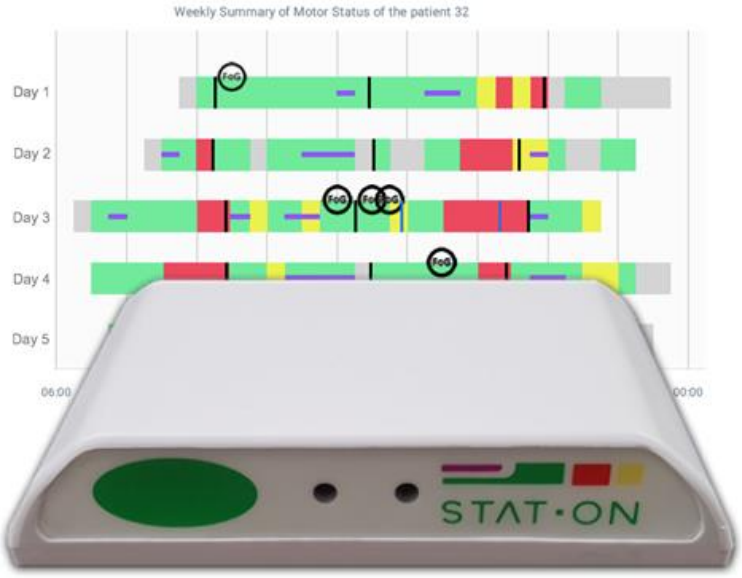
For more details, please contact: Eugenio Gaeta <eugenio.gaeta@lst.tfo.upm.es>.

A.9 Sense4Care

Sense4Care provides a wearable medical device, called STAT-ON, for monitoring symptoms of Parkinson's disease. Motion data is transferred via Bluetooth for processing by a mobile app, which provides access to the data in PDF and CSV formats. The data is not sent to the cloud.

[17] https://oauth.net/2/jwt/

Figure 122 – STAT-ON sensor by Sense4Care



The neurologist accesses to data once they get the sensor after a week of monitoring. Then, they download the information to a smartphone that generates a PDF report.

- How would Gatekeeper pilots be able to integrate with this solution?

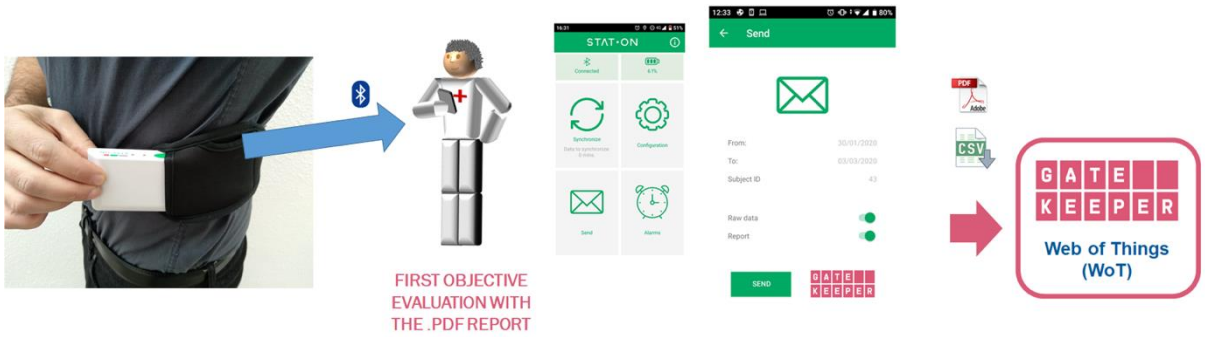
The Gatekeeper project provides a solution that enables the possibility of:

- Having data more accessible to healthcare staff
- Standardised data
- Organised digital data
- Repository of patients
    - Evaluation of disease evolution
    - Data analytics
    - Patients clustering by treatment

In GATEKEEPER, the STAT-ON app will have a new characteristic that will enable to upload the data obtained to the Gatekeeper system. It will also upload the PDF and the CSV file which is the raw data for specific analysis.

The uploaded information will be integrated in the Gatekeeper platform via Web of Things that will enable the healthcare professional and to the patient access easily to objective data of the motor state of the patient in order to adjust better a tailored medication.

Figure 132 – STAT-ON sensor by Sense4Care



For more details, please contact: Daniel Rodriguez <daniel.rodriguez@sense4care.com>.

A.10 University of Ioannina

The University of Ioannina has developed algorithms for personalised, adaptive care of advanced type 2 diabetes. This is based upon machine learning, and can be embedded as part of mobile apps that are connected to glucometers, e.g. the Medtronic Guardian Sensor 3, or Dexcom G6 or Menarini GlucoMen Day CGM.

For more details, please contact: E. I. Georga <egeorga@gmail.com>.

A.11 Tecnalia

Tecnalia can contribute with a Virtual Reality (VR) training software for stroke detection and mitigation. This will be provided through a web application combined with VR headset (Oculus or HTC Vive) based on WebXR technology. This application is addressing the Stroke Use Case in Basque Country and it will allow to simulate different real situations when stroke happens and help to learn families and patients how to identify them and how to react to minimize the risks.

The design and development will be done together with Osakidetza. The TRL of the VR headsets and the SW to be used is 9.

In order to support a wide variety of hardware form factors, we will use WebXR, which is a group of W3C standards used together to provide the interfaces necessary to enable developers to build compelling, comfortable, and safe immersive applications on the web[18].

For more details, please contact: Leire Bastida <Leire.Bastida@tecnalia.com>.

A.12 Engineering

The DMCoach solution is a "digital therapeutic" tool intended for patients or citizens at risk of Type 2 Diabetes Mellitus (T2D) to unobtrusively monitor (physical activities, nutrition) and to

[18] https://www.w3.org/TR/webxr/

coach them with personalized feedback towards a healthier lifestyle. The coaching is enabled by healthcare professionals who "prescribe" a mobile application to the patient to:

1. view the information collected by the patient;
2. easily tailor and configure the app on patient profile defining plans and goals and
3. contact (one-way) the patient to coach, (re)plan, meet by keeping at minimum the time-consuming activities.

Status of the implementation, e.g. maturity, availability and level of support

The implementation has been validated in 2 healthcare institutions and in 4 companies. It is currently at TRL7.

Interoperability mechanisms

The solution, as a whole, is strongly based on the adoption of the HL7 FHIR standard used to exchange information between patients and professionals. As such, DMCoach is ready to be integrated with existing EMR systems compatible with such standards.

From a more technical point-of-view DMCoach comes with a back-end APIs used for data storage enabled by web services RESTful and fully supporting the FHIR interoperability specification.

For data collection it relies on Google Fit and Apple Health Kit.

Technological Classification

Following the classification provided in Figure 1, DMCoach exploits data from devices and sensors compliant with Google Fit and Apple health, that use short range protocols to sync with their proprietary applications, while local Google and Apple mobile apps act as a gateway to uniform data representation at syntactic level.

Table 2 - Data types (Technical Interoperability)

DEVICE	PROTOCOLS	BRAND
Smart bands (steps)	Android devices can sync with Google Fit. iOS devices can likewise sync with Apple HealthKit. Both companies provide APIs for access from trusted apps	Any Google Fit or Apple HealthKit compliant device
Smart scales (Weight)		
Workout		
HR Monitors (Heart Rate)		

Table 3 - Cloud Gateways (Syntactic Interoperability)

Cloud Data Sever	PROTOCOLS	BRAND
GoogleFit Cloud server	HTML\JSON	Google

Apple Health Server	HTML\JSON	Apple

DMCoach takes care also of ingesting data from these sources at a semantic level, and translating and storing data for export to a FHIR server.

Table 4 - Data Repository (Semantic Interoperability)

NAME	Solution	GK reference ontology
DMCoach FHIR Server	HAPI-FHIR	FHIR

More information about the DMCoach solution can be found at: www.dmcoach.eu

Or contact: Valentina Di Giacomo <valentina.digiacomo@eng.it>.

A.13 CERTH

Diabetes Management Platform (DMP)

Diabetes Type 2 Management Platform is proposing a novel mHealth management system based on software and hardware (commercial devices) solutions that are part of a comprehensive approach to managing and supporting patients with diabetes. Realizing the multi-faceted nature of the management of diabetes, a systematic, multi-pronged and an integrated approach like the DMP is required for promoting self-care practices among diabetic patients to avert any long-term complications. The proposed solution focuses on monitoring patients' adherence to medical treatment, physiological and environmental variables but also on providing a personalised guidance platform transmitting all the measurements to a prediction engine for giving appropriate feedback to the user on how to manage diabetes

The types of parameters monitored by the component are: General information about the patient (e.g. name, birth date, gender), a medical summary consisting of the most important clinical patient data (e.g. allergies, current medical problems, medical implants, or major surgical procedures during the last six months), a list of the current medication including all prescribed medicines that the patient is currently taking, data from the Integrated Medical Devices Data from the Care Plan, community Data and educational Activities

Interoperability mechanisms

The solution, as a whole, is strongly based on the adoption of the HL7 FHIR standard used to exchange information between patients, informal caregivers and healthcare professionals.

Furthermore, the system is built on microservices. Thus, each component has an API communication protocol that controls data exchange. The platform can be integrated with any device/infrastructure that provides open connectivity.

Table 5 - Data Repository (Semantic Interoperability)

NAME	Solution	GK reference ontology

DMP FHIR Server	HAPI-FHIR	FHIR

Table 6 - Data types (Technical Interoperability)

DEVICE	PROTOCOLS	BRAND
Smart bands (Activity)	Proprietary and vendor dependent	Medisante (A.6) Garmin, Fitbit Samsung (A.3)
Smart scales (BMI)		
Smart sensors (Sleep/Stress)		
BG/BP,/Purse Monitors		

Table 7 - Cloud Gateways (Syntactic Interoperability)

Cloud Data Sever	PROTOCOLS	BRAND
Amazon Cloud server	HTML\JSON	Medisante
Garmin Health Server	HTML\JSON	Garmin

Trust Authority and Open Distributed Ledger

Trust Authority and Open Distributed Ledger is a blockchain-based platform that

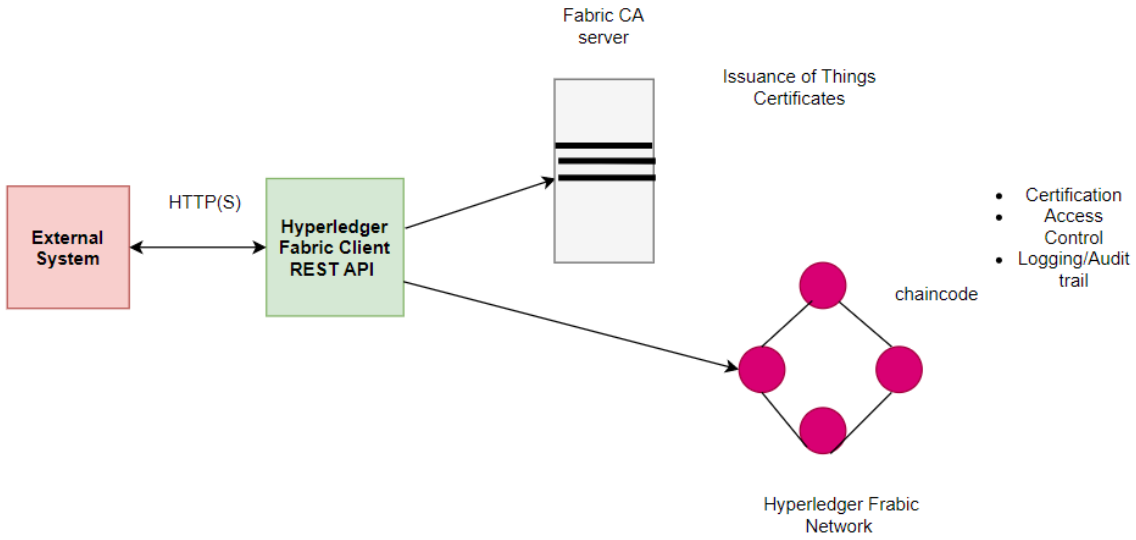
- is responsible for certifying the Things of the Gatekeeper platform based on a set of standards and for calculating the corresponding levels of certification for these Things.
- it provides the capabilities for authenticating Things and providing authorisation rules based on the aforementioned levels 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.

The solution will be based on Hyperledger Fabric chaincode for the calculation of the levels of certification and for keeping the audit trail. It will also provide a connection with a Fabric CA server in order to issue and manage certificates for the Things.

Interoperability mechanisms

The solution will be able to communicate with any infrastructure that is able to communicate over HTTP(s) by exposing a RESTful API and by using JSON as the data format. The component diagram depicting the communication between this platform and any compatible external system is depicted in the Figure below.

Figure 14 - Hyperledger Fabric



Appendix B Graph Databases

A uniform framework for data, metadata and rules will simplify the development of application services by decoupling them from the details of heterogeneous information sources, devices, protocols, data formats and data models. This section explores the potential for simplifying working with graph data and rules and sketches an architecture for remote access.

Gatekeeper could integrate with an existing graph database such as an RDF triple store, see, e.g. the long list on Wikipedia. This would have the advantage of starting from a mature implementation.

One such database is the open source Eclipse RDF4J project, which offers a Java API for traversing and manipulating RDF graphs. This level of API could then be used to support a higher-level framework than raw triples, which would be easier for the average developer.

Another approach would be to implement our own database engine that directly supports a high-level framework for traversing and manipulating graphs.

For Gatekeeper, we would also seek to create a high-level rule language that makes it easier to describe common behaviours, e.g. reminders to take medications, upper and lower thresholds for alarms, and what to do when they are breached. Such rules could be expressed as chunks (see below) and integrate with the stream processing system.

B.1 Cognitive Databases

A cognitive database holds chunks: col<lections of properties that include references to other chunks. Chunks can be associated with statistical information reflecting prior knowledge and past experience. Cognitive databases have the potential to hold vast amounts of information similar to the cerebral cortex. Cognitive databases can be local or remote and shared with multiple cognitive agents, subject to access control policies.

Memory retrieval fits Web architecture, supporting remote invocation of graph algorithms in a request/response pattern rather like HTTP. Retrieval is analogous to Web search engines where results are computed based upon what is likely to be most relevant to the user. It is often impractical and inappropriate to try to return the complete set of matches.

Cognitive databases support a variety of graph algorithms that are executed local to the data, and capable of scaling to Big Data. These algorithms include:

- Basic storage and retrieval
- Specialised algorithms for natural language, spatial and temporal reasoning
- Algorithms for data analytics and machine learning

B.2 Chunks

Chunks is an amalgam of RDF and LPG, that makes it easy to work with entities that have multiple properties, and has a simpler syntax compared to JSON-LD. Each chunk has a type, an identifier, and set of properties, whose values name other chunks to form graphs. In more detail, property values can be Booleans (true or false), numbers, names, string literals (in double quotes) or comma separated lists there-of. Property names themselves can act as chunk identifiers.

Here are some example chunks:

```
friend f34 {
    name Joan
}

friend {
    name Jenny
    likes f34
}
```

- Where friend is a chunk type, f34 is a chunk identifier, name and likes are property names, Joan and Jenny are also names.
- likes f34 signifies that Jenny likes Joan via the link to the chunk for Joan.
- Missing chunk identifiers are automatically assigned when inserting a chunk into a graph
- Uses line breaks as punctuation

You can also use a short form for simple directed labelled relationships, e.g.

```
dog kindof mammal
```

which is equivalent to:

```
kindof {
    subject dog
    object mammal
}
```

The latter form is better if you want to annotate the relationship with additional properties, something that is awkward in RDF as it requires the use of reification.

To relate chunks to RDF you can use @rdfmap, for instance:

```
@rdfmap {
    dog http://example.com/ns/dog
    cat http://example.com/ns/car
}
```

You can use @base to set a default base URI for names that are not declared explicitly, e.g.

```
@rdfmap {
    @base http://example.com/ns
    dog http://example.com/ns/dog
    cat http://example.com/ns/cat
}
```

Which would map the mouse to http://example.com/ns/mouse.

You can use @prefix for defining URI prefixes, e.g.

```
@prefix p1 {
    ex: http://example.com/ns/
}

@rdfmap {
    @prefix p1
    dog ex:dog
    cat ex:cat
}
```

It may often be more convenient to refer to an external collection of @rdfmap and @prefix declarations, rather than inlining them e.g.

@rdfmap from http://example.com/mappings

If there are multiple conflicting definitions, the most recent will override earlier ones.

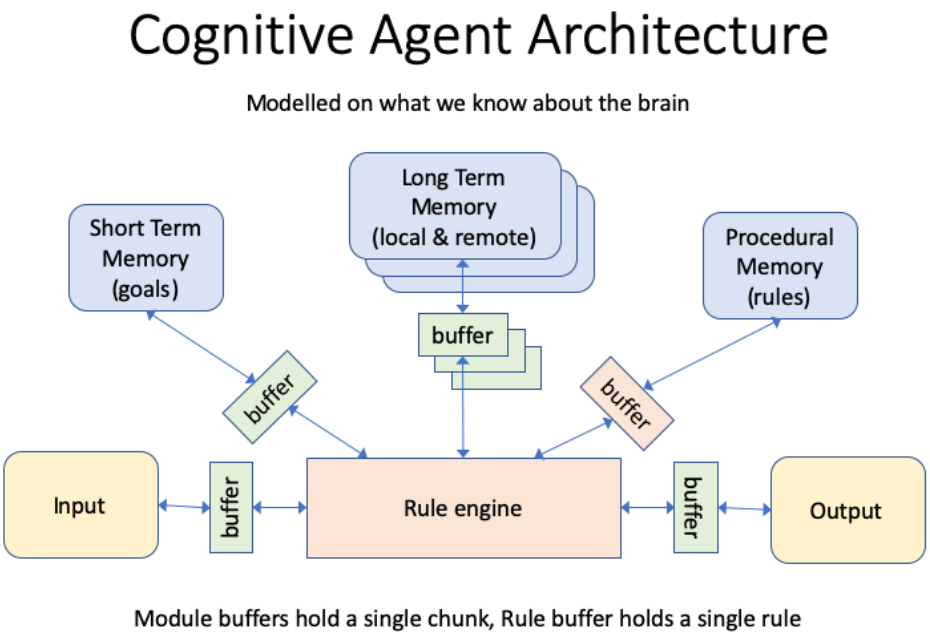
Note: people familiar with JSON-LD would probably suggest using @context instead of @rdfmap, however, that would be confusing given that we want to use the @context in respect to reasoning in multiple contexts.

B.3 Chunk API

This section describes an API developed for a JavaScript library for chunk graphs along with a condition-action rule engine. JavaScript is a dynamically typed programming language, and the API may need some adjustments for use with statically typed languages such as Java.

The JavaScript library includes a rule engine in which the rules are themselves expressed as chunks. Each rule has one or more conditions and one or more actions. The conditions are matched to module buffers which hold a single chunk. The actions either directly update the module buffers or invoke graph algorithms that may indirectly update the module buffers. This design was inspired by the popular ACT-R cognitive science architecture. For more details, see introduction to chunks and rules.

Figure 15 - Cognitive architecture



Agents make use of modules that act as chunk databases and are accessed via buffers. These buffers correspond to bundles of nerve fibres that connect the basal ganglia to the cerebral cortex.

Here are some operations you can perform on a chunk graph:

new ChunkGraph(source)

Create a new graph from a text string containing the chunks and links.

graph.chunks[id]

Find a chunk given its id.

graph.types[type]

Find the list of chunks with a given type.

graph.forall(kind, handler, context)

Apply a function to all chunks whose type has the kindof relationship to the given kind. This applies recursively to chains of kindof relationships. The handler is a function that is passed the chunk and the context.

graph.recall(type, values)

Recall a chunk with a given type, and matching values as denoted by a JavaScript object with a set of named properties. Note that this is stochastic and returns the 'best' chunk when there are multiple matches.

graph.remember(type, values, id)

Remember (i.e. create and store) a chunk with a given type, and matching values as denoted by a JavaScript object with a set of named properties. The chunk id will be assigned automatically if not supplied.

graph.parse(source)

Parse source as chunks and add to this graph.

graph.add(chunk)

Adds a chunk or link to the graph, see below for ways to create chunks and links. If the chunk is currently part of another graph, it will be removed from that graph before being added to this one.

graph.remove(chunk)

Remove a chunk or link from the graph.

Here are some operations you can perform on a chunk:

new Chunk(type, id)

Create a new chunk for a given type and id. The id is optional and will be assigned automatically when the chunk is added to a graph if not supplied.

new Link(subject, predicate, object)

Create a new Link as a subclass of chunk where the chunk type is given by the predicate. The chunk id will be assigned automatically when the Link is added to a graph.

chunk.id

Access the chunk's id.

chunk.type

Access the chunk's type.

chunk.properties[name]

Access a chunk property value given the property's name.

chunk.setValue(name, value)

Overwrite the value of a named property

chunk.addValue(name, value)

Add a value for named property. An array is used only if the property value has multiple values.

chunk.removeValue(name)

Remove a value from the named property - this is the inverse of addValue.

chunk.hasValue(name, value)

Returns true or false according to whether the named property contains the given value, i.e. the property is either that value or it is a list, one of whose items is that value. If the property is undefined for this chunk, then the return value is false.

chunk.toString()

Returns a pretty printed version of the chunk.

The following describes the API for rule engines for condition-action rules expressed as chunks:

new RuleEngine()

Create a new rule engine.

engine.addModule(name, graph[, backend])

Registers a new local module with its name, graph and an optional backend for graph algorithms.

The backend is declared as an object whose property values are functions that implement the algorithm identified by the property's name. The algorithm's name can then be used with @invoke in rule actions for this module. The action is passed a single argument that is an object whose property values are the bindings for the variables identified by the object's property names.

The backend functions can be used to override the default actions for recall, remember and update. Note that "rules" and "goals" are required modules. The rules module is used to hold procedural knowledge as a set of rules. By default, the "goals" module is initialised to an empty graph. A separate method is envisaged for adding remote modules.

engine.getModule(name)

Return the module created by calling engine.addModule.

engine.start(rules, facts[, initial_goal])

A convenience function to set the rules and facts modules, along with an optional initial chunk, that is provided as a chunk, i.e. as source text. Note: rules is a graph containing the rules that define procedural knowledge, and facts is a graph containing a set of chunks that define declarative knowledge. The goal is not used until you call engine.next().

engine.next()

Find and execute the next matching rule.

engine.setGoal(source)

Parse the source for a chunk and load it into the goal module's buffer.

engine.setBuffer(name, source)

Parse the source for a chunk and add it to the named module's graph, then load it into the module's buffer.

engine.setBuffer(name, chunk)

Load the chunk into the named module's buffer.

engine.getBuffer(name)

Return the chunk in the named module's buffer.

engine.addListener(listener)

Register a listener function that will be called when any of the module buffers are updated. The listener is passed a single argument identifying the module by name.

Appendix C Glossary of Terms

Action	An Interaction Affordance that allows to invoke a function of the Thing, which manipulates state (e.g., toggling a lamp on or off) or triggers a process on the Thing (e.g., dim a lamp over time).
Binding Templates	A re-usable collection of blueprints for the communication with different IoT platforms. The blueprints provide information to map Interaction Affordances to platform-specific messages through WoT Thing Description as well as implementation notes for the required protocol stacks or dedicated communication drivers.
Code System	A managed collection of Concept Representations, including codes, but sometimes more complex sets of rules and references, optionally including additional Concept Representations playing various roles including identifiers of the concepts, designations, etc.
Cognitive AI	An approach to artificial intelligence inspired by advances in the cognitive sciences.
Connector	A module used to ingest data into the Gatekeeper platform, transforming the data formats, identifiers and data models as needed.
Consumed Thing	A software abstraction that represents a remote Thing used by the local application. The abstraction might be created by a native WoT Runtime or instantiated as an object through the WoT Scripting API.
Consuming a Thing	To parse and process a TD document and from it create a Consumed Thing software abstraction as interface for the application in the local runtime environment.
Consumer	An entity that can process WoT Thing Descriptions (including its JSON-based representation format) and interact with Things (i.e., consume Things).
Data Schema	A data schema describes the information model and the related payload structure and corresponding data items that are passed between Things and Consumers during interactions.
Digital Twin	A digital twin is a virtual representation of a device or a group of devices that resides on a cloud or edge node. It can be used to represent real-world devices which may not be continuously online, or to run simulations of new applications and services, before they get deployed to the real devices.
Domain-specific Vocabulary	Linked Data vocabulary that can be used in the WoT Thing Description but is not defined by W3C WoT.
Edge Device	A device that provides an entry point into enterprise or service provider core networks. Examples include gateways, routers, switches, multiplexers, and a variety of other access devices.
Event	An Interaction Affordance that describes an event source, which asynchronously pushes event data to Consumers (e.g., overheating alerts).
Exposed Thing	A software abstraction that represents a locally hosted Thing that can be accessed over the network by remote Consumers. The

	abstraction might be created by a native WoT Runtime or instantiated as an object through the WoT Scripting API.
Exposing a Thing	To create an Exposed Thing software abstraction in the local runtime environment to manage the state of a Thing and interface with the behavior implementation.
Graph Data	An approach to modelling data in terms of vertices and labelled directed edges.
HL7 FHIR Conformance Resource	A single resource in a package that makes rules about how an implementation works.
HL7 FHIR Implementation Guide	A coherent and bounded set of adaptations that are published as a single unit. Validation occurs within the context of the Implementation Guide.
HL7 FHIR Resource	Resource that identifies itself as one of the types of resource defined in the FHIR specification; contains a set of structured data items as described by the definition of the resource type; has an identified version that changes if the contents of the resource change. They share the following set of characteristics: ● A common way to define and represent them, building them from data types that define common reusable patterns of elements ● A common set of metadata ● A human readable part
HL7 FHIR Profile	A set of constraints on a resource represented as a structure definition with kind = constraint.
Hypermedia Control	A serialization of a Protocol Binding in hypermedia, that is, either a Web link [RFC8288] for navigation or a Web form for performing other operations. Forms can be seen as request templates provided by the Thing to be completed and sent by the Consumer.
Interaction Affordance	Metadata of a Thing that shows and describes the possible choices to Consumers, thereby suggesting how Consumers may interact with the Thing. There are many types of potential affordances, but W3C WoT defines three types of Interaction Affordances: Properties, Actions, and Events. A fourth Interaction Affordance is navigation, which is already available on the Web through linking.
Interaction Model	An intermediate abstraction that formalizes and narrows the mapping from application intent to concrete protocol operations. In W3C WoT, the defined set of Interaction Affordances constitutes the Interaction Model.
Intermediary	An entity between Consumers and Things that can proxy, augment, or compose Things and republish a WoT Thing Description that points to the WoT Interface on the Intermediary instead of the original Thing. For Consumers, an Intermediary may be indistinguishable from a Thing, following the Layered System constraint of REST.
IoT Platform	A specific IoT ecosystem such as OCF, oneM2M, or Mozilla Project Things with its own specifications for application-facing APIs, data model, and protocols or protocol configurations.

Metadata	Data that provides a description of an entity's abstract characteristics. For example, a Thing Description is Metadata for a Thing.
Ontology	A formal collection of concepts and relationships used to describe some area of interest
Personally Identifiable Information (PII)	Any information that can be used to identify the natural person to whom such information relates or is or might be directly or indirectly linked to a natural person. We use the same definition as [ISO-IEC-29100].
Privacy	Freedom from intrusion into the private life or affairs of an individual when that intrusion results from undue or illegal gathering and use of data about that individual. We use the same definition as [ISO-IEC-2382]. See also Personally Identifiable Information and Security, as well as other related definitions in [ISO-IEC-29100].
Private Security Data	Private Security Data is that component of a Thing's Security Configuration that is kept secret and is not shared with other devices or users. An example would be private keys in a PKI system. Ideally such data is stored in a separate memory inaccessible to the application and is only used via abstract operations, such as signing, that do not reveal the secret information even to the application using it.
Property	An Interaction Affordance that exposes state of the Thing. This state can then be retrieved (read) and optionally updated (write). Things can also choose to make Properties observable by pushing the new state after a change.
Protocol Binding	The mapping from an Interaction Affordance to concrete messages of a specific protocol, thereby informing Consumers how to activate the Interaction Affordance. W3C WoT serializes Protocol Bindings as hypermedia controls.
Public Security Metadata	Public Security Metadata is that component of a Thing's Security Configuration which describes the security mechanisms and access rights necessary to access a Thing. It does not include any secret information or concrete data (including public keys), and does not by itself, provide access to the Thing. Instead, it describes the mechanisms by which access may be obtained by authorized users, including how they must authenticate themselves.
Reasoning	The process of inferring new information from old information in a logical, sensible way. This may involve formal semantics and logical deduction, e.g. the use of formal ontologies for logical entailments, or it may involve the application of rules to graph representations of information, possibly in combination with statistics that reflect prior knowledge and past experience.
Resource	Any identifiable thing, whether digital, physical, or abstract.
Rules	Application logic expressed as rules to specify behaviour, e.g. to send notifications when defined thresholds are exceeded, or to send patients reminders to take medications, …
Security	Preservation of the confidentiality, integrity and availability of information. Properties such as authenticity, accountability, non-

	repudiation, and reliability may also be involved. This definition is adapted from the definition of Information Security in [ISO-IEC-27000], which also includes additional definitions of each of the more specific properties mentioned. Please refer to this document for other related definitions. We additionally note that it is desirable that these properties be maintained both in normal operation and when the system is subject to attack.
Security Configuration	The combination of Public Security Metadata, Private Security Data, and any other configuration information (such as public keys) necessary to operationally configure the security mechanisms of a Thing.
Servient	A software stack that implements the WoT building blocks. A Servient can host and expose Things and/or host Consumers that consume Things. Servients can support multiple Protocol Bindings to enable interaction with different IoT platforms.
Stream Processing	Application logic applied to data streams as a means to handle large amounts of data.
Subprotocol	An extension mechanism to a transfer protocol that must be known to interact successfully. An example is long polling for HTTP.
TD	Short for WoT Thing Description.
TD Vocabulary	A controlled Linked Data vocabulary by W3C WoT to tag the metadata of Things in the WoT Thing Description including communication metadata of WoT Binding Templates.
Thing or Web Thing	An abstraction of a physical or a virtual entity whose metadata and interfaces are described by a WoT Thing Description, whereas a virtual entity is the composition of one or more Things.
Thing Directory	A directory service for TDs that provides a Web interface to register TDs (similar to [CoRE-RD]) and look them up (e.g., using SPARQL queries or the CoRE RD lookup interface [CoRE-RD]).
Transfer Protocol	The underlying, standardized application layer protocol without application-specific requirements or constraints on options or subprotocol mechanisms. Examples are HTTP, CoAP, or MQTT.
Value Set	A set of codes defined by code systems that can be used in a specific context.
Virtual Thing	An instance of a Thing that represents a Thing that is located on another system component.
Vocabulary	A collection of terms used to describe concepts and relationships, and often used to embrace both formal and informal collections of terms.
WoT Interface	The network-facing interface of a Thing that is described by a WoT Thing Description.
WoT Runtime	A runtime system that maintains an execution environment for applications and is able to expose and/or consume Things, to process WoT Thing Descriptions, to maintain Security Configurations, and to interface with Protocol Binding implementations. A WoT Runtime may have a custom API or use the optional WoT Scripting API.
WoT Scripting API	The application-facing programming interface provided by a Servient in order to ease the implementation of behavior or

	applications running in a WoT Runtime. It is comparable to the Web browser APIs. The WoT Scripting API is an optional building block for W3C WoT.
WoT Servient	Synonym for Servient.