D4.1

Specification of remote and D2D protocol and APIs for HR exchange - V1

ABSTRACT

This report describes the initial version of the open specification of the device-to-device (D2D) and the remote-to-device (R2D) protocols, defined (and to be experimented) in the context of the InteropEHRate project. These protocols will be used by software applications facilitating health data exchange between the citizens and the Healthcare professionals. The D2D protocol defines the technology, data structures and operations to be offered by the interacting applications, allowing the exchange of health data between the aforementioned stakeholders without the usage of internet. On the other side, the R2D protocol defines the technology, operations and structure of data used for enabling the exchange of health data between an Electronic Health Record (EHR) system of a single organization, either a National EHR or to a S-EHR Cloud (a remote cloud system for the storage of encrypted personal health data) and the citizen with the usage of the internet.

Delivery Date	11th July 2019
Work Package	WP4
Task	4.1
Dissemination Level	Public
Type of Deliverable	Report
Lead partner	UPRC

InteropEHRate project is co-funded by the European Union (EU) Horizon 2020 program under Grant number 826106.

CONTRIBUTORS

	Name	Partner
Contributors	Thanos Kiourtis, Argyro Mavrogiorgou, Dimitris Laliotis, Aikaterini Aggeli	UPRC
	Alessio Graziani	ENG
	Martin Marot, Julien Henrard	A7
	Nicu Jalba	SIVECO
Reviewers	Gábor Bella, Simone Bocca	UNITN
	Francesco Torelli	ENG

LOGTABLE

Version	Date	Change	Author	Partner
0.1	2019-05-06	First draft of ToC	Thanos Kiourtis, Argyro Mavrogiorgou	UPRC
0.2	2019-05-10	Finalized ToC	Thanos Kiourtis, Argyro Mavrogiorgou	UPRC
0.3	2019-05-21	First version of diagrams, state of the art analysis and description of D2D protocol	Thanos Kiourtis, Argyro Mavrogiorgou	UPRC
0.4	2019-05-25	First version of diagrams, state of the art analysis and description of R2D protocol	Alessio Graziani, Francesco Torelli	ENG
0.5	2019-06-02	Description of S-EHR Application	Martin Marot, Julien Henrard	A7
0.6	2019-06-05	Description of HCP Application	Nicu Jalba	SIVECO
0.7	2019-06-07	Updates on R2D protocol	Alessio Graziani, Francesco Torelli	ENG
0.8	2019-06-14	Updates on D2D protocol and finalization of the overall	Thanos Kiourtis, Argyro	UPRC

		context of the deliverable	Mavrogiorgou	
0.9	2019-06-16	Deliverable sent for internal review	Thanos Kiourtis, Argyro Mavrogiorgou	UPRC
1.0	2019-06-21	First version of internal review	Francesco Torelli	ENG
1.1	2019-06-22	Deliverable sent for second internal review	Thanos Kiourtis, Argyro Mavrogiorgou	UPRC
1.2	2019-07-10	Finalized and sent for Quality Review	Thanos Kiourtis, Argyro Mavrogiorgou, Alessio Graziani, Francesco Torelli	UPRC, ENG
1.3	2019-07-10	Completed Quality Review	Argyro Mavrogiorgou	UPRC
Vfinal	2019-07-11	Final check and version for submission	Laura Pucci	ENG

ACRONYMS

Acronym	**Description**
D2D	Device-to-Device
R2D	Remote-to-Device
HER	Electronic Health Record
S-HER	Smart Electronic Health Record
NFC	Near Field Communications
RFID	Radio Frequency Identification
WBAN	Wireless Body Area Networks
WPAN	Wireless Personal Area Networks
WLAN	Wireless Local Area Networks
VAN	Vehicle Area Networks

WSN	Wireless Sensor Networks
EMR	Electronic Medical Record
BLE	Bluetooth Low Energy
IEEE	Institute of Electrical and Electronics Engineers
EDR	Enhanced Data Rate
GATT	Generic Attribute Profile
LTE	Long-Term Evolution
AoA	Angle of Arrival
AoD	Angle of Departure
P2P	Peer-to-Peer
ISM	Industrial, Scientific, and Medical
GUI	Graphical User Interface
UI	User Interface
UX	User Experience
ITU	International Telecommunication Union
MD2D	Mobile Device-to-Device
TD2D	Terminal Device-to-Device
MD2DI	Mobile Device-to-Device Interface
TD2DI	Terminal Device-to-Device Interface
HCO	Healthcare Organization
CEF	Connecting Europe Facility
eHDSI	eHealth Digital Service Infrastructure
NCP	National Contact Points
API	Application Programming Interface
FHIR	Fast Healthcare Interoperability Resources
UML	Unified Modeling Language

SQL	Structured Query Language
R2DI	Remote-to-Device Interface
UUID	Universally Unique Identifier
SDDB	Service Discovery DataBase
SPP	Serial Port Profile
RFCOMM	Radio Frequency Communication
LMP	Link Manager Protocol
L2CAP	Logical Link Control and Adaptation Protocol
OSI	Open Systems Interconnection
GSM	Global System for Mobile Communication
SDP	Service Discovery Protocol

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Scope of the document

The main goal of this document is to describe the initial version of the specification of both the device-to-device (D2D) and the remote-to-device (R2D) protocols. In essence, the D2D protocol specifies a series of specified bluetooth messages regarding the information that is being exchanged (e.g. in terms of successful or failed data exchange) and healthcare related data between a healthcare practitioner (utilizing a web application (HCP app)) and a citizen (utilizing a mobile application (S-EHR app)), without the usage of internet connection. The R2D protocol defines a set of operations in terms of which its interface depends on, as well as the structure of the data that can be used for the exchange of health data between any electronic health record (EHR) or S-EHR Cloud (a remote cloud system used by citizens for the storage of their encrypted health data) and a S-EHR (a kind of mobile application of a citizen), with the usage of internet. Henceforth, both the D2D and the R2D protocols will be used for standardizing how software applications exchange health data with the mobile application (S-EHR app) of the citizen.

The current document outlines for both protocols (i.e. D2D and R2D will be realized as a library enabling its exploitation in various implementation contexts for health data exchange) a detailed description of their functionality, accompanied by an explanation of their separate purposes of existence. Moreover, the deliverable describes the research that was undertaken for each different field regarding short range and remote, data - and health data - exchange, for designing each corresponding protocol.

Having in mind that there exist several short-range communication techniques that could be used for this data exchange without using internet connection (e.g. Wi-Fi Direct, Bluetooth Low Energy, Bluetooth Classic), an initial version of the D2D protocol is currently specified based on the Bluetooth Classic. The reason for this choice is that we wanted to first identify the messages and the way that they would be exchanged, in the platform of Android mobile devices that is currently used in most of the mobile devices, and for that reason the specification of the D2D protocol is currently based on the Android Bluetooth API. In the next version of the D2D protocol specification (March 2020), the D2D protocol specification will be independent from any API or platform (i.e. Operating System), while it might be based upon a different short-range communication technique - based on our research, in order for any developer that would like to use this protocol to be able to use the listed operations or develop her own operations, in order to implement the corresponding data exchange.

To this end, it should also be mentioned that the current document received as input the reference architecture as elaborated in deliverable D2.4 - InteropEHRate Architecture - V1 [D2.4], and the user requirements that were analysed and presented in deliverable D2.1 - User Requirements for cross-border HR integration - V1 [D2.1].

1.2. Intended audience

The current document is mainly intended for developers, and manufacturers that are interested in designing and building either mobile or web applications that need to exploit and reuse either the D2D or the R2D or both these two functionalities of the InteropEHRate framework, in the context of their applications. As a result, this audience will be able to offer the aforementioned separate functionalities in their developed applications, since both the D2D and the R2D protocols can be easily integrated and reused by other systems and applications. Apart from this, the document is intended to researchers as well, as they may be interested in understanding the way that those two libraries work, influenced by them, and possibly extending and updating them.

1.3. Structure of the document

The current document is organized in the following Sections:

- **Section 1** (the current section) introduces the overall concept of the document, defining its scope, intended audience, and relation to the other project tasks and reports.
- **Section 2** outlines the short range health data exchange protocol (D2D), describing in deep detail the purpose of the existence of the protocol, whereas stating the research that was undertaken for concluding into the design of the protocol, and the overall description of it.
- **Section 3** describes the remote health data exchange protocol (R2D), explaining in deep detail the purpose of the existence of the protocol, whereas stating the research that was undertaken for concluding into the design of the protocol, and the overall description of it.
- **Section 4** outlines the conclusions of the current document, including the future developments and updates of the two libraries.

1.4. Updated with respect to previous version (if any)

Since this document is the initial version of the design and the specification of the D2D and R2D protocols, there does not exist any update with regards to previous related documents.

2. D2D PROTOCOL: SHORT RANGE HEALTH DATA EXCHANGE

The D2D protocol defines the set of operations that allow the exchange of health data between a S-EHR App and a near HCP App (i.e. in the context of a distance of up to 10m long) without the usage of internet (in opposition to the R2D that is the protocol to exchange health data remotely, including the usage of internet).

The next section of this document, provides a detailed description on the technologies that have been studied and implemented towards the realization of the D2D protocol, the context in which the D2D will be used, and of all the conditions that have influenced the decisions taken in the process carried out to define the D2D specifications.

2.1. D2D Protocol Scope

The purpose of the D2D protocol is to propose a series of specified bluetooth messages regarding the information that is being exchanged (e.g. in terms of successful or failed data exchange) and healthcare related data between a healthcare practitioner and a citizen, without the usage of internet connection. This protocol is based on short-range wireless technologies and in particular Bluetooth, to be adopted at EU level, for secure exchange of health records between a smart mobile device and a health information system in the form of a web application. The smart device will use the S-EHR app (i.e. application that serves the purposes of the D2D from the side of the citizen), while the health information system will use the HCP app (i.e. application that serves the purposes of the D2D from the side of the healthcare practitioner), that will be used by the citizen and the healthcare practitioner accordingly. According to the objectives defined by the InteropEHRate consortium, the first version of the D2D protocol focuses on exchanging medical data (e.g. vital signs, evaluation data), from a citizen's S-EHR app to an HCP app using Bluetooth technology.

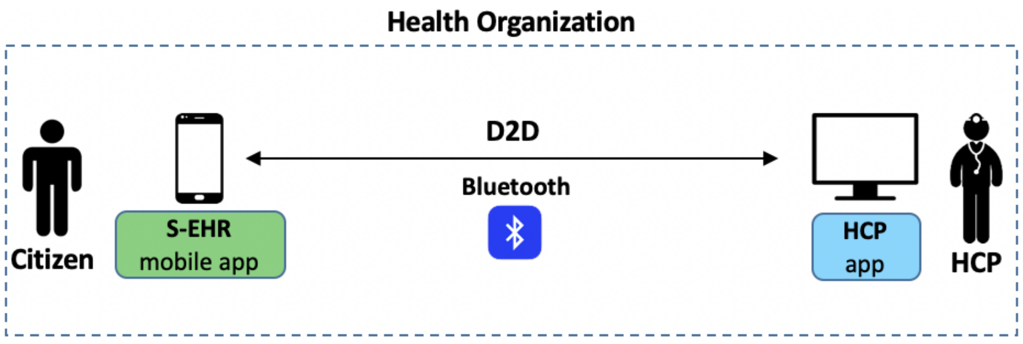

Figure 1 - D2D protocol scope

Bluetooth technology is most commonly associated with exchanging data between two bluetooth enabled devices in short distance (±10 meters), through which a bluetooth-enabled device as soon as it receives the initialization advertisement message from another bluetooth-enabled device, it establishes a connection to it, being thus able to exchange and display information between them, without needing any other technologies or types of connection (e.g. internet connection). Adopting a similar paradigm, the proposed D2D protocol will facilitate the information exchange between patients (i.e. through smartphones) and healthcare practitioners (i.e. through a computer with a bluetooth adapter), without the usage of central cloud services or any other parties.

In order to accomplish all the aforementioned tasks, the D2D protocol will define the Bluetooth services (represented by the interface TD2DI) to be offered by a healthcare organization to share health data contained in their EHR with the S-EHR app, as well as the Bluetooth services (represented by the interface MD2DI) to be offered by the S-EHR app for receiving requests from the HCP app of the Healthcare Organization Information System. The D2D protocol will also exploit D2D Security protocol (which interfaces are part of the R2DI and MD2DI) for performing Identity Management, Consent Management, and Authorization Management, that will be integrated into the overall functionality of the D2D protocol.

2.2. Related Work

2.2.1. Short Range Wireless Communication

Short-range wireless communications systems have to do with a wide range of scenarios, technologies and requirements. There is not any specific definition of such systems though one can always classify short-range wireless systems according to their typical reach or coverage. Henceforth, short-range wireless communications can be defined as the systems that aim to provide wireless connectivity within a local sphere of interaction. Short-range wireless systems deal with transferring of data from millimeters to a few hundreds of meters, which are not only providing wireless connectivity in the immediate proximity, but in a broader area they can be defined as technologies that are used to construct service access in local areas. Together with wide/metropolitan area cellular systems, short-range wireless systems represent the two main developing directions in today’s wireless communications scene having certain common parts as well as various differences from cellular systems. The main goal is to maximize the supported data throughput for both types of wireless networks, and as a result a detailed comparison between them is not easily implemented. There is a great deal of cooperation between short-range and cellular networks, and in many cases exploiting their full characteristics results in very efficient solutions.

The short-range wireless systems include NFC for very close connectivity, RFID ranging from centimeters up to a few hundred meters, WBAN providing wireless access in the close vicinity of a person, WPAN offering users in their surroundings of up to ten meters or so, WLAN, the de facto local connectivity for indoor scenarios covering typically up to a hundred meters around the access point, VAN involving distances of up to several hundred meters and WSN, reaching even further. Short-range wireless communication systems include a great diversity in possible air interface solutions, topologies as well as achievable data throughput and supported mobility. In general, short-range networks have ad hoc distributed architectures allowing direct and multi-hop connectivity among nodes. Centralized access is also possible, as in WLAN, which in fact supports both centralized and distributed topologies. Data throughput requirements for short-range wireless systems are very broad, from some hundred bits per second in simple RFID systems up to 10 Gbps and beyond in WLAN systems, for instance.

One of the key requirements for short-range wireless systems is low power consumption , particularly taking into account that transceivers are in many cases battery-driven. Particular attention is necessary when designing short-range wireless devices while minimizing the power consumption at the three basic conditions of the transceiver, namely transmitting, receiving and idle states. Low cost is another important factor to take into consideration when designing short-range wireless systems. Furthermore, minimizing size and weight are also quite often imperative engineering principles that need to be applied by the designer of such communication systems. Another important aspect of short-range wireless communications networks is their relationship to other existing wireless networks and their possible interaction.

These days, there is more wireless technology in use than ever before. Wireless technology is portable, easy to install, flexible and eliminates the cost of expensive wiring. With the explosion of available wireless devices, there has also been a big increase of wireless protocols and standards to support all of that technology. These include several short-range wireless communication technologies that transmit shorter distances than other long range technologies, making them great for certain applications. Below is a short list of the most commonly used short-range wireless communication standards and technologies.

ANT+ [ANT+]
ANT and ANT+ are sensor network technologies used for collecting and transferring sensor data. This short-range wireless communication technology is a type of personal-area network (PAN) that features low power consumption and long battery life. It divides the 2.4 GHz band into 1 MHz channels and accommodates multiple sensors. It is primarily used for short-range, low-data-rate sensor applications such as sports monitors, wearables, wellness products, home health monitoring, vehicle tire pressure sensing and in household items that can be controlled remotely such as TVs, lights and appliances. It can be configured to spend long periods in a low-power “sleep” mode (consuming of the order of microamps of current), wake up briefly to communicate (when consumption rises to a peak of 22mA (at -5dB) during reception and 13.5mA (at -5 dB) during transmission) and return to sleep mode. Average current consumption for low message rates is less than 60 microamps on some devices. Each ANT channel consists of one or more transmitting nodes and one or more receiving nodes, depending on the network topology. Any node can transmit or receive, so the channels are bi-directional. ANT accommodates three types of messaging: broadcast, acknowledged, and burst.

- Broadcast is a one-way communication from one node to another (or many). The receiving node(s) transmit no acknowledgment, but the receiving node may still send messages back to the transmitting node. This technique is suited to sensor applications and is the most economical method of operation.
- Acknowledged messaging confirms receipt of data packets. The transmitter is informed of success or failure, although there are no retransmissions. This technique is suited to control applications.
- Burst messaging is a multi-message transmission technique using the full data bandwidth and running to completion. The receiving node acknowledges receipt and informs of corrupted packets that the transmitter then re-sends. The packets are sequence numbered for traceability. This technique is suited to data block transfer where the integrity of the data is paramount.

Bluetooth/ BLE [BLUETOOTH]
Bluetooth is covered by the IEEE 802.15.1 standard. Originally created as an alternative to cabled RS-232, Bluetooth is now used to send data from PANs and fixed and mobile devices. This plug-and-play technology utilizes the 2.4 -2.485 GHz band and has a standard range of 10 meters, but it can be extended to 100 meters at maximum power with a clear path. BLE has a simpler design and is a direct competitor of ANT+, focusing on health and medical applications.

Bluetooth is a packet-based short-range wireless communication technology with a master/slave architecture, where the devices can switch roles, by agreement, and the slave can become the master. One master may communicate with up to seven slaves in a piconet. All devices share the master's clock. Packet exchange is based on the basic clock, defined by the master, which ticks at 312.5 µs intervals. Two clock ticks make up a slot of 625 µs, and two slots make up a slot pair of 1250 µs. In the simple case of single-slot packets, the master transmits in even slots and receives in odd slots. The slave, conversely, receives in even slots and transmits in odd slots. Packets may be 1, 3 or 5 slots long, but in all cases the master's transmission begins in even slots and the slave's in odd slots. The master chooses which slave device to address by switching rapidly from one device to another in a round-robin fashion. Since it is the master that chooses which slave to address, whereas a slave is supposed to listen in each receive slot, being a master is a lighter burden than being a slave. Being a master of seven slaves is possible; being a slave of more than one master is possible. The specification is vague as to required behavior in scatternets. To use Bluetooth wireless technology, a device must be able to interpret certain Bluetooth profiles, which are definitions of possible applications and specify general behaviors that Bluetooth-enabled devices use to communicate with other Bluetooth devices. These profiles include settings to parameterize and to control the communication from the start. Adherence to profiles saves the time for transmitting the parameters anew before the bi-directional link becomes effective. There are a wide range of Bluetooth profiles that describe many different types of applications or use cases for devices. Bluetooth has different versions, with either minor or major differences among them, based on their implementation and usage. The version history of Bluetooth is depicted below:

- **Bluetooth 1.0 and 1.0B**: This version had many problems, and manufacturers had difficulty making their products interoperable. They included mandatory Bluetooth hardware device address transmission in the Connecting process, which was a major setback for certain services planned for use in Bluetooth environments
- **Bluetooth 1.1**: This version was ratified as an IEEE Standard 802.15.1–2002. It included many error fixes from the v1.0B specifications, including the possibility of non-encrypted channels, and Signal Strength Indicator for the received data.
- **Bluetooth 1.2**: This version included faster Connection and Discovery, adaptive frequency-hopping spread spectrum, which improved the resistance to radio frequency interference by avoiding the use of crowded frequencies in the hopping sequence. It also included higher transmission speeds than in v1.1, and improved quality of audio links by allowing retransmissions of corrupted packets, and increasing audio latency to provide better concurrent data transfer.
- **Bluetooth 2.0**: This version of the Bluetooth Core Specification was released in 2004. The main difference is the introduction of an EDR for faster data transfer. The bit rate of EDR is 3 Mbit/s, although the maximum data transfer rate is 2.1 Mbit/s. EDR can provide a lower power consumption through a reduced duty cycle. The specification is published as Bluetooth v2.0 + EDR, which implies that EDR is an optional feature. Aside from EDR, the v2.0 specification contains other minor improvements, and products may claim compliance to "Bluetooth v2.0" without supporting the higher data rate.
- **Bluetooth 2.1**: This version of Bluetooth Core Specification includes secure simple pairing, improving the pairing experience for Bluetooth devices, while increasing the use and strength of security. It also allows improvements, including extended inquiry response, which provides more information during the inquiry procedure to allow better filtering of devices before connection, and sniff subrating that reduces the power consumption in low-power mode.

- **Bluetooth 3.0**: This version of the Bluetooth Core Specification was adopted in 2009. Bluetooth v3.0 provides theoretical data transfer speeds of up to 24 Mbit/s, though not over the Bluetooth link itself. Instead, the Bluetooth link is used for negotiation and establishment, and the high data rate traffic is carried over a colocated 802.11 link. The main new feature is Alternative MAC/PHY, the addition of 802.11 as a high-speed transport. The high-speed part of the specification is not mandatory, and hence only devices that display the "+HS" logo actually support Bluetooth over 802.11 high-speed data transfer. A Bluetooth v3.0 device without the "+HS" suffix is only required to support features introduced in Core Specification Version 3.0 or earlier Core Specification Addendum 1.
- **Bluetooth 4.0**: This version of the Bluetooth Core Specification version 4.0 (Bluetooth Smart) has been adopted as of 2010. It includes Classic Bluetooth, Bluetooth high speed and BLE short-range wireless communication technologies. Bluetooth high speed is based on Wi-Fi, and Classic Bluetooth consists of legacy Bluetooth short-range wireless communication technologies. BLE, previously known as Wibree, is a subset of Bluetooth v4.0 with an entirely new short-range wireless communication technology stack for rapid build-up of simple links. As an alternative to the Bluetooth standard short-range wireless communication technologies that were introduced in Bluetooth v1.0 to v3.0, it is aimed at very low power applications powered by a coin cell. Chip designs allow for two types of implementation, dual-mode, single-mode and enhanced past versions. Compared to Classic Bluetooth, BLE is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range. In a single-mode implementation, only the low energy short-range wireless communication technology stack is implemented. The compliant architecture shares all of Classic Bluetooth's existing radio and functionality resulting in a negligible cost increase compared to Classic Bluetooth. Cost-reduced single-mode chips, which enable highly integrated and compact devices, feature a lightweight Link Layer providing ultra-low power idle mode operation, simple device discovery, and reliable point-to-multipoint data transfer with advanced power-save and secure encrypted connections at the lowest possible cost. General improvements in version 4.0 include the changes necessary to facilitate BLE modes, as well as the GATT and Security Manager services with Advanced Encryption Standard.
- **Bluetooth 4.1**: This version is an incremental software update to Bluetooth Specification v4.0, and not a hardware update. The update incorporates some Bluetooth Core Specifications and adds new features that improve consumer usability. These include increased co-existence support for LTE, bulk data exchange rates, and aid developer innovation by allowing devices to support multiple roles simultaneously. New features of this specification include mobile Wireless Service Coexistence Signaling, Train Nudging and Generalized Interlaced Scanning, Low Duty Cycle Directed Advertising, L2CAP Connection Oriented and Dedicated Channels with Credit-Based Flow Control, Dual Mode and Topology, LE Link Layer Topology, 802.11n PAL, Audio Architecture Updates for Wide Band Speech, Fast Data Advertising Interval, and Limited Discovery Time.
- **Bluetooth 4.2**: This version includes major areas of improvement in the domain of Low Energy Secure Connection with Data Packet Length Extension. It also includes Link Layer Privacy with Extended Scanner Filter Policies, Internet Protocol Support Profile version 6 ready for Bluetooth Smart things to support connected home, and supports the fact that older Bluetooth hardware may receive 4.2 features such as Data Packet Length Extension and improved privacy via firmware updates.

- **Bluetooth 5.0**: This version has new features mainly focusing on emerging Internet of Things technology. Bluetooth 5.0 provides for BLE, options that can double the speed (2 Mbit/s burst) at the expense of range, or up to fourfold the range at the expense of data rate, and eightfold the data broadcasting capacity of transmissions, by increasing the packet lengths. The increase in transmissions could be important for Internet of Things devices, where many nodes connect throughout the whole house. Bluetooth 5 adds functionality for connectionless services such as location-relevant navigation of low-energy Bluetooth connections.
- **Bluetooth 5.1**: This version supports major areas of improvement including AoA and AoD which are used for location and tracking of devices, Advertising Channel Index, GATT Caching.

EnOcean [ENOCEAN]
This system is self-powered and able to wirelessly transmit data by using ultra-low power consumption and energy collecting technology. Instead of a power supply, EnOcean's wireless sensor technology collects energy from the air. Energy from the environment, such as light, pressure, kinetic motion and temperature differences, is harvested and used to transmit a signal up to 30 meters indoors using a very small amount of energy. In the US, EnOcean runs on the 315 MHz and 902 MHz bands. In Europe, it uses the 868 MHz frequency band and in Japan, it operates on the 315 MHz and 928 MHz frequency bands. EnOcean technology is based on the energetically efficient exploitation of slight mechanical motion and other potentials from the environment, such as indoor light and temperature differences, using the principles of energy harvesting. In order to transform such energy fluctuations into usable electrical energy, electromagnetic, solar, and thermoelectric energy converters are used. EnOcean-based products (such as sensors and light switches) perform without batteries and are engineered to operate maintenance-free. The radio signals from these sensors and switches can be transmitted wirelessly over a distance of up to 300 meters in the open and up to 30 meters inside buildings. Early designs from the company used piezo generators, but were later replaced with electromagnetic energy sources to reduce the operating force (3.5 newtons), and increase the service life to 100 operations a day for more than 25 years. EnOcean wireless data packets are relatively small, with the packet being only 14 bytes long and are transmitted at 125 kbit/s. RF energy is only transmitted for the 1's of the binary data, reducing the amount of power required. Three packets are sent at pseudo-random intervals reducing the possibility of RF packet collisions. Modules optimized for switching applications transmit additional data packets on release of push-button switches, enabling other features such as light dimming to be implemented. The transmission frequencies used for the devices are 902 MHz, 928.35 MHz, 868.3 MHz and 315 MHz. On 2017, EnOcean unveiled a series of light switches utilizing BLE radio (2.4 GHz).

NFC [NFC]
NFC is an ultra-short-range technology created for contactless communication between devices. It is often used for secure payment applications, fast passes and similar applications. Operating on the 13.56 MHz ISM frequency, NFC has a maximum range of around 20 cm, which provides a more secure connection that is usually encrypted. Many smart phones already include an NFC tag. NFC short-range wireless communication technologies established a generally supported standard. When one of the connected devices has Internet connectivity, the other can exchange data with online services. NFC-enabled portable devices can be provided with application software, for example, to read electronic tags or make payments when connected to an NFC-compliant apparatus. Earlier close-range communication used technology that was proprietary to the manufacturer for applications such as stock tickets, access control and payment

readers. Like other "proximity card" technologies, NFC employs electromagnetic induction between two loop antennas when NFC-enabled devices—for example a smartphone and a printer—exchange information, operating within the globally available unlicensed radio frequency ISM band of 13.56 MHz on ISO/IEC 18000-3 air interface at rates ranging from 106 to 424 kbit/s. NFC tags contain data and are typically read-only, but may be writable. They can be custom-encoded by their manufacturers or use NFC Forum specifications. The tags can securely store personal data such as debit and credit card information, loyalty program data, PINs and networking contacts, among other information. The NFC Forum defines four types of tags that provide different communication speeds and capabilities in terms of configurability, memory, security, data retention and write endurance. Tags currently offer between 96 and 4,096 bytes of memory.

The two modes for NFC are:

- **Passive**: The initiator device provides a carrier field and the target device answers by modulating the existing field. In this mode, the target device may draw its operating power from the initiator-provided electromagnetic field, thus making the target device a transponder.
- **Active**: Both initiator and target device communicate by alternately generating their own fields. A device deactivates its RF field while it is waiting for data. In this mode, both devices typically have power supplies.

It should be noted that each full NFC device can work in three modes:

- **NFC card emulation:** It deals with NFC-enabled devices such as smartphones to act like smart cards, allowing users to perform transactions such as payment or ticketing.
- **NFC reader/writer**: It enables NFC-enabled devices to read information stored on inexpensive NFC tags embedded in labels or smart posters.
- **NFC peer-to-peer**: It enables two NFC-enabled devices to communicate with each other to exchange information in an ad hoc fashion.

RFID [RFID]

RFID uses small, flat, cheap tags that can be attached to anything and used for identification, location, tracking and inventory management. When a reader unit is nearby, it transmits a high-power RF signal to the tags and reads the data stored in their memory. Low frequency RFID uses the 125-134 kHz band, high frequency RFID uses the 13.56 MHz ISM band and Ultra-high frequency RFID uses the 125-134 kHz band. With multiple ISO/IEC standards available for RFID, this technology has replaced bar codes in some industries. It uses electromagnetic fields to automatically identify and track tags attached to objects. The tags contain electronically stored information. Passive tags collect energy from a nearby RFID reader's interrogating radio waves. Active tags have a local power source (such as a battery) and may operate hundreds of meters from the RFID reader. Unlike a barcode, the tag need not be within the line of sight of the reader, so it may be embedded in the tracked object. RFID is one method of automatic identification and data capture.

ZigBee [ZIGBEE]

ZigBee is the standard of the ZigBee Alliance. The path of a message in this network zig-zags like a bee, hence the name. It is a software short-range wireless communication technology that uses the 802.15.4 transceiver as a base and is meant to be cheaper and simpler than other WPANs, like Wi-Fi or Bluetooth. ZigBee is able to build large mesh networks for sensor monitoring, handling up to 65,000 nodes, and it can also support multiple types of radio networks such as point-to-point and point-to-multipoint. It has a data

rate of 250 kB/s and can transfer wireless data over a distance of up to 100m. ZigBee can be used for a range of applications including remote patient monitoring, wireless lighting and electrical meters, traffic management systems, consumer TV and factory automation, to name a few.
Zigbee devices are of three kinds:

- **Zigbee Coordinator:** The most capable device, the Coordinator forms the root of the network tree and might bridge to other networks. There is precisely one Zigbee Coordinator in each network since it is the device that started the network originally (the Zigbee LightLink specification also allows operation without a Zigbee Coordinator, making it more usable for off-the-shelf home products). It stores information about the network, including acting as the Trust Center & repository for security keys.
- **Zigbee Router:** As well as running an application function, a Router can act as an intermediate router, passing on data from other devices.
- **Zigbee End Device:** Contains just enough functionality to talk to the parent node (either the Coordinator or a Router); it cannot relay data from other devices. This relationship allows the node to be asleep a significant amount of the time thereby giving long battery life. A Zigbee End Devicerequires the least amount of memory, and, therefore, can be less expensive to manufacture than a Zigbee Router or Zigbee Coordinator.

The current Zigbee short-range wireless communication technologies support beacon and non-beacon enabled networks.

- In non-beacon-enabled networks, an unslotted CSMA/CA channel access mechanism is used. In this type of network, Zigbee Routers typically have their receivers continuously active, requiring a more robust power supply. However, this allows for heterogeneous networks in which some devices receive continuously, while others only transmit when an external stimulus is detected. The typical example of a heterogeneous network is a wireless light switch: The Zigbee node at the lamp may constantly receive, since it is connected to the mains supply, while a battery-powered light switch would remain asleep until the switch is thrown. The switch then wakes up, sends a command to the lamp, receives an acknowledgment, and returns to sleep. In such a network the lamp node will be at least a Zigbee Router, if not the Zigbee Coordinator; the switch node is typically a Zigbee End Device.
- In beacon-enabled networks, the special network nodes called Zigbee Routers transmit periodic beacons to confirm their presence to other network nodes. Nodes may sleep between beacons, thus lowering their duty cycle and extending their battery life. Beacon intervals depend on data rate; they may range from 15.36 milliseconds to 251.65824 seconds at 250 kbit/s, from 24 milliseconds to 393.216 seconds at 40 kbit/s and from 48 milliseconds to 786.432 seconds at 20 kbit/s. However, low duty cycle operation with long beacon intervals requires precise timing, which can conflict with the need for low product cost.

In general, the Zigbee short-range wireless communication technologies minimize the time the radio is on, so as to reduce power use. In beaconing networks, nodes only need to be active while a beacon is being transmitted. In non-beacon-enabled networks, power consumption is decidedly asymmetrical: Some devices are always active while others spend most of their time sleeping.

Wi-Fi Direct [WIFIDIRECT]

Wi-Fi Direct, initially called Wi-Fi P2P, is a Wi-Fi standard enabling devices to easily connect with each other without requiring a wireless access point. Wi-Fi Direct allows two devices to establish a direct Wi-Fi connection without requiring a wireless router. Hence, Wi-Fi Direct is single radio hop communication, not multihop wireless communication, unlike wireless ad hoc networks and mobile ad hoc networks. Wi-Fi ad hoc mode, however, supports multi-hop radio communications, with intermediate Wi-Fi nodes as packet relays. One advantage of Wi-Fi Direct is the ability to connect devices even if they are from different manufacturers. Only one of the Wi-Fi devices needs to be compliant with Wi-Fi Direct to establish a peer-to-peer connection that transfers data directly between them with greatly reduced setup. Wi-Fi Direct negotiates the link with a Wi-Fi Protected Setup system that assigns each device a limited wireless access point. The "pairing" of Wi-Fi Direct devices can be set up to require the proximity of a near field communication, a Bluetooth signal, or a button press on one or all the devices. When a device enters the range of the Wi-Fi Direct host, it can connect to it, and then gather setup information using a Protected Setup-style transfer. Connection and setup is so simplified that it may replace Bluetooth in some situations. Wi-Fi Direct-certified devices can connect one-to-one or one-to-many and not all connected products need to be Wi-Fi Direct-certified. One Wi-Fi Direct enabled device can connect to legacy Wi-Fi certified devices. The Wi-Fi Direct certification program is developed and administered by the Wi-Fi Alliance, the industry group that owns the "Wi-Fi" trademark. The specification is available for purchase from the Wi-Fi Alliance

Z-Wave [ZWAVE]

Z-Wave is a wireless communications short-range wireless communication technology used primarily for home automation. It is a mesh network using low-energy radio waves to communicate from appliance to appliance, allowing for wireless control of residential appliances and other devices. A Z-Wave system can be controlled via the Internet from a smartphone, tablet or computer, and locally through a smart speaker, wireless keyfob, or wall-mounted panel with a Z-Wave gateway or central control device serving as both the hub controller and portal to the outside. Z-Wave is designed to provide reliable, low-latency transmission of small data packets at data rates up to 100kbit/s. The throughput is 40kbit/s (9.6kbit/s using old chips) and suitable for control and sensor applications, unlike Wi-Fi and other IEEE 802.11-based wireless LAN systems that are designed primarily for high data rates. Communication distance between two nodes is about 30 meters (40 meters with 500 series chip), and with message ability to hop up to four times between nodes, it gives enough coverage for most residential houses. Modulation is by Manchester channel encoding. Z-Wave uses the Part 15 unlicensed ISM band. It operates at 868.42 MHz in Europe, at 908.42 MHz in the North America and uses other frequencies in other countries depending on their regulations. This band competes with some cordless telephones and other consumer electronics devices, but avoids interference with Wi-Fi, Bluetooth and other systems that operate on the crowded 2.4 GHz band. The lower layers, MAC and PHY, are described by ITU-T G.9959 and fully backwards compatible. In 2012, the ITU included the Z-Wave PHY and MAC layers as an option in its G.9959 standard for wireless devices under 1 GHz. Data rates include 9600 bps and 40 kbps, with output power at 1 mW or 0 dBm. Devices can communicate to one another by using intermediate nodes to actively route around and circumvent household obstacles or radio dead spots that might occur in the multipath environment of a house. A message from node A to node C can be successfully delivered even if the two nodes are not within range, providing that a third node B can communicate with nodes A and C. If the preferred route is unavailable, the message originator will attempt other routes until a path is found to the C node. Therefore, a Z-Wave network can span much farther than the radio range of a single unit; however, with several of these hops a slight delay may be introduced between the control command and the desired result.

2.2.2. InteropEHRate D2D Approach

A comparison study took place with specified criteria, for identifying the most ideal short-range wireless communication technology as a basis to realize the communication aspects of the envisioned D2D protocol. The short-range wireless communication technologies that have been considered include: Wi-Fi direct, Bluetooth v4.0, BLE, and NFC. These criteria included all of our needs and necessities as described in D2.1 deliverable **[D2.1]**, in order to be compliant with what the objectives of the D2D protocol, for the interoperable exchange of healthcare data between a mobile application (i.e. S-EHR app) and an application on the healthcare practitioner's personal computer (i.e. HCP App).

In more detail, the comparison criteria were the following:

- Range (in centimeters/ meters)
- Data Rate (in bps)
- Security (Low, Medium, High based on the short-range wireless communication technology specification)
- Platform Applicability (referring to the mostly used platform operating systems (iOS, Android))
- Ease of use (Low, Medium, High based on the short-range wireless communication technology specification)
- Power Consumption (Low, Medium, High based on the short-range wireless communication technology specification)

	Wi-Fi Direct	Bluetooth v4.0	BLE	NFC
Range	Up to 180 m	Up to 10m	Up to 10m	Up to 4cm
Data Rate	Up to 250Mbps	Up to 25 Mbps	Up to 200kbps	Up to 424kbps
Security	High	High	High	Medium
Platform Applicability	**Android**: Host Wi-Fi network by Android or PC **iOS**: Manually join hosted network by the PC	**Android**: Applicable **iOS**: Needs certification under MFI program	**Android**: Applicable **iOS**: GATT-based API	**Android**: Applicable **iOS**: Applicable (data transfer by NFC packets to be understood)
Ease of Use	High	High	High	High
Power Consumption	High	Medium	Low	Low

Table 1 - Comparison between short-range wireless communication technologies

In terms of process, a list of requirements was circulated to the hospitals of the InteropEHRate consortium (i.e. users), in order to identify their needs and specifications. This list (including the users' answers) can be seen in the ANNEX. Based on the results that are displayed in Table 1, as well as the needs and the criteria that were set by the users, the two most likely candidate short-range wireless communication technologies for the D2D protocol were Bluetooth v4.0, and BLE. In general, one of the primary advantages of Bluetooth is that it is enabled in almost any Android or Apple iOS device, allowing these devices to transmit data wirelessly. This advantage can be translated to more specific benefits including wirelessly connecting or "pairing" devices to create a wireless personal area network or WPAN, wireless synchronization, as well as conveniently sending and/or receiving files without the trouble of carrying and using cables or other hardware interfacing technology such as the USB standard or Thunderbolt technology. That is why complementary devices have been developed and marketed because Bluetooth has seemingly become a standard feature of modern computers, specifically laptops and mobile devices. These devices included wireless speakers and headphones, smart devices such as smartwatches and other wearable technologies for monitoring activities, and Bluetooth-enabled smart home appliances and office equipment, among others. Moreover, pairing devices with built-in Bluetooth radio is considerably easy. There is no need to install additional software or driver to establish communication between Bluetooth-enabled devices. There is also no rigorous setup process for two devices to communicate. The technology simplifies the entire pairing process by making enabled devices readily discoverable to one another as long as that their Bluetooth radios are turned on, and they are within the coverage radius. In addition, the technology also includes a protocol for identifying services using the Service Discovery Protocol and Universal Unique Identifier to list down specific services or features of a particular device. These protocols allow another device to readily determine and display the name and class of a device it intends to pair with, as well as its services or features and technical information. Furthermore, Bluetooth technology is relatively energy efficient, thus promoting further the benefits and convenience that come with wireless data transmission. This is particularly true for the BLE or BLE standard. The ultra-low power requirement of BLE makes it ideal for small devices, including wearable technologies, in which minimal battery life requirement and small form factor are critical design and engineering considerations.

However, since both Bluetooth v4.0 and BLE were still two different candidates, this research was continued for identifying the most efficient solution for the InteropEHRate project. It was identified that BLE could be considered ideal for devices that must operate for long periods of time on small energy sources. In general, devices that utilize BLE today range in intelligence from heart rate monitors to smart watches, where BLE is particularly well suited for receiving small data updates, such as the current heart rate, every few seconds. In that case, the "pairing" process is also very simple since BLE devices can advertise themselves (at Peripheral state) and multiple BLE devices can be connected to another device (at Central state), such as a smartphone. However, BLE cannot be considered as the answer for every IoT device or scenario, especially in the case that our goal is to transmit moderately sized files (including texts of a few kilobytes, or images of a few megabytes (Mbytes)). As it can be seen in Table 2, the users demanded from the D2D protocol to exchange - among others, image files of a few megabytes, for the identified types of examinations.

Type of Examination	Weight evaluation by category (Mbytes)
Radiology	30
Ultrasound	10

Mammography	100
TC	300
RM	200
Angiographies	150
Positron Emission Tomography – PET	30
Other Nuclear Medicine examinations	30
Radiotherapy	30
Other	38
Reports	0.1

Table 2 - Size of files that will be exchanged through the D2D protocol

In more detail, with Bluetooth v4.0, throughput is as high as 25 Mbps, so once two devices are paired, sending 100 Mbytes of data would take a very small amount of time to be transmitted. In order to develop applications requiring Bluetooth access in Apple iOS devices, this can be cost-prohibitive, as Apple collects licensing and other fees in exchange for this functionality. It should be noted that this is not an issue on Android, but what was needed was the used devices to be compatible with both major mobile operating systems. In that scenario, BLE could be considered as the most convenient solution, since Apple iOS devices support BLE out of the box, without the need to enroll into the MFI program, and many Android devices are being manufactured with BLE as well. In the case of BLE, in order to send large data files, one needs to break apart the payload into 20-byte chunks, where on the receiving end, these chunks are recombined. Four such chunks can be sent per connection interval, which can vary from one device to the next. Android devices support intervals as low as 7.5 ms, while Apple iOS devices support connection intervals in the 30 ms range. At this rate, with BLE a 500KB image file would take over three minutes to transfer. Some workarounds for reducing this amount of data transfer time could be the usage of data partitioning, serialization, and compression which could help in reducing payload sizes, but in the case of InteropEHRate, this could not satisfy the users' needs and requirements who demand low transfer times and high data transfer rates. Consequently, due to its specifications, Bluetooth v4.0 has been identified as the most convenient and suitable candidate for short-range wireless communication data exchange, for being implemented in the D2D protocol in the InteropEHRate project.

2.3. D2D Protocol Description

The D2D protocol will define the bluetooth operations represented by the interfaces that will be offered by the mobile application and the healthcare's organization application speaking of the S-EHR app and the HCP app accordingly. These interfaces will be included in both applications accordingly, and will be used by the two main actors of the D2D protocol, the citizens and the HCPs. These two actors will be the only involved ones in the overall interaction, for exchanging the consent of accessing each one's personal data, the healthcare related data, and the evaluation data accordingly.

In order for the D2D protocol to realize the communication and interaction with the two parties, two different interfaces will be designed and realized. The first interface (MD2DI) will be responsible for offering the bluetooth operations and services by the S-EHR app for interconnecting, exporting messages and receiving requests from the HCP app, while the second interface (TD2DI) will be responsible for offering the bluetooth operations and services by the HCP app for similar types of tasks from the S-EHR app. As discussed in Section 2.2, the overall communication will be based on the Bluetooth short-range wireless communication technology, hence the initial step of the overall D2D protocol will be the two involved actors to pair and bond their devices, prior to exchanging any messages. Figure 2 displays the overall connection between a citizen and an HCP, including the aforementioned interfaces, as well as the involved applications.

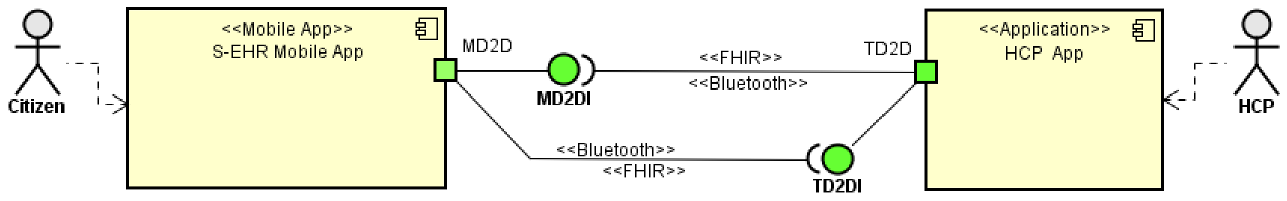

Figure 2 - D2D protocol interfaces

The rest of this section provides technical specifications (using UML language) of the D2D protocol. The section is structured in two sub-sections, each one focused on a specific topic:

- **Conceptual D2D**: This includes the conceptual description of the D2D protocol, defining the basic operations that this protocol should contain.
- **Involved Applications**: This includes a short description of the involved applications (S-EHR app and the HCP app), referring to the purpose and to a high-level description of their architecture.
- **D2D Interface Interactions**: This includes the overall interactions between the involved interfaces for both the S-EHR app and the HCP app through a Sequence Diagram.
- **D2D over Bluetooth**: This includes the concrete description of the D2D protocol, regarding the Bluetooth profiles used for the connection and pairing process of the involved applications, and the messages that are exchanged according to the applications' interactions.

2.3.1. Conceptual D2D

The Conceptual D2D defines the basic operations that the protocol should contain, for the exchange of data between the HCP and the citizen through the HCP app and the S-EHR app accordingly. The following figures show two UML class diagrams representing the main interfaces of D2D, named MD2DI and TD2DI, from the side of the S-EHR App and the side of the HCP Web App.

MD2DI

MD2DI and MD2DI-Security is the name of the interface that is offered by the S-EHR app regarding the D2D protocol, containing the operations for letting the HCP app to perform tasks related to the S-EHR app, by invoking these operations.

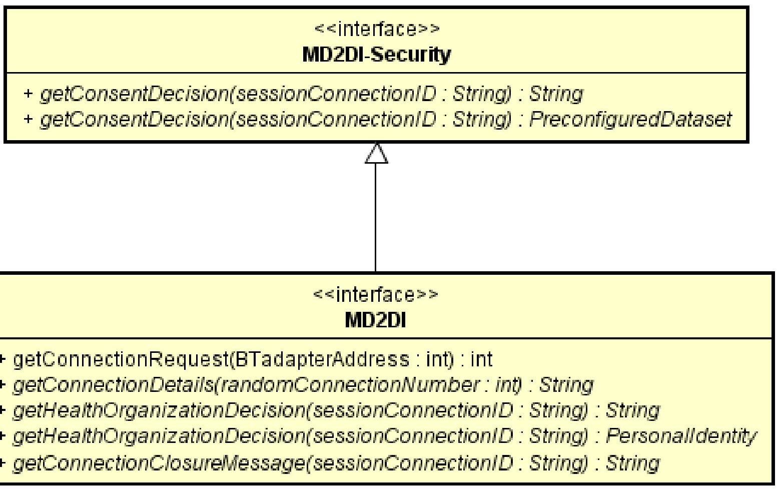
<<interface>>
MD2DI-Security
+ getConsentDecision(sessionConnectionID : String) : String
+ getConsentDecision(sessionConnectionID : String) : PreconfiguredDataset
<<interface>>
MD2DI
+ getConnectionRequest(BTadapterAddress : int) : int
+ getConnectionDetails(randomConnectionNumber : int) : String
+ getHealthOrganizationDecision(sessionConnectionID : String) : String
+ getHealthOrganizationDecision(sessionConnectionID : String) : PersonalIdentity
+ getConnectionClosureMessage(sessionConnectionID : String) : String

Figure 3 - MD2D interface operations

The following tables provide a complete description of all the operations of the MD2DI.

Operation getConnectionRequest

Name	getConnectionRequest
Description	This operation is invoked by the HCP app for getting the advertised connection request from the S-EHR app for performing the pairing process between them, and getting a random connection number that will be used for pairing the two applications.
Arguments	● **BTadapterAddress**: a string that contains the MAC address of the Bluetooth adapter that is used from the side of the HCP
Return Value	This operation will return a number (**randomConnectionNumber**) in the form of an integer, that will be used for pairing the two applications.
Exceptions	● Security exceptions related to the Bluetooth connection. ● Network exceptions related to Bluetooth connection failure.
Preconditions	● The Bluetooth connection exists in the mobile phone that includes the S-EHR app.

Operation getConnectionDetails

Name	getConnectionDetails
Description	This operation is invoked by the HCP app for getting the connection unique session identifier, in the form of a random String. This String will be used by both sides (S-EHR app and HCP app), for current connection identification purposes.
Arguments	● **randomConnectionNumber**: the random number that will be provided to the S-EHR app and has to be approved from both sides of the S-EHR app and the HCP app for the pairing instantiation.
Return Value	This operation will return a unique session identifier (sessionConnectionID), in the form of a random String that will be used by both sides (S-EHR app and HCP app), for current connection identification purposes.
Exceptions	● Security exceptions related to the Bluetooth connection. ● Network exceptions related to Bluetooth connection failure.
Preconditions	● The Bluetooth connection exists in the mobile phone that includes the S-EHR app.

Operation getHealthOrganizationDecision

Name	getHealthOrganizationDecision
Description	This operation is invoked by the HCP app for getting the decision from the side of the S-EHR app, regarding whether the provided Health Organization identity is approved or not
Arguments	● **sessionConnectionID**: a string that identifies the connection between the two involved applications.
Return Value	This operation will return, one of the two following options: ● A connection closure message (connectionClosureMessage) in the form of a String, indicating that the Health Organization identity was not approved, hence the connection will be closed.

	• The personal data of the S-EHR app owner (i.e. name, surname, date of birth, location of birth, gender, country of residence, and social security number) in the form of an object (PersonalData).
Exceptions	• Security exceptions related to the Bluetooth connection. • Network exceptions related to Bluetooth connection failure.
Preconditions	• The Bluetooth connection exists in the mobile phone that includes the S-EHR app. • The smartphone is enabled with Bluetooth v4.0 and above • The session is still valid.

Operation getConsentDecision

Name	getConsentDecision
Description	This operation is invoked by the HCP app for getting the decision from the side of the S-EHR app, regarding whether the consent for getting the S-EHR app owner's data has been approved or not.
Arguments	• **sessionConnectionID**: a string that identifies the connection between the two involved applications.
Return Value	This operation will return, one of the two following options: • A connection closure message (connectionClosureMessage) in the form of a String, indicating that the Health Organization identity was not approved, hence the connection will be closed. • The already preconfigured dataset of the S-EHR application owner (i.e. data that has been approved for sharing) in the form of an object (PreconfiguredDataset).
Exceptions	• Security exceptions related to the Bluetooth connection. • Network exceptions related to Bluetooth connection failure.
Preconditions	• The Bluetooth connection exists in the mobile phone that includes the S-EHR app. • The smartphone is enabled with Bluetooth v4.0 and above • The session is still valid.

Operation getConnectionClosureMessage

Name	getConnectionClosureMessage
Description	This operation is invoked by the HCP app for getting the final message of connection closure, after the overall interaction has successfully ended with the provision of the evaluation data.
Arguments	● **sessionConnectionID**: a string that identifies the connection between the two involved applications.
Return Value	This operation will return a connection closure message (connectionClosureMessage) in the form of a String, indicating that the overall interaction has successfully completed.
Exceptions	● Security exceptions related to the Bluetooth connection. ● Network exceptions related to Bluetooth connection failure.
Preconditions	● The citizen has already opened through her smartphone the Bluetooth connection. ● The smartphone is enabled with Bluetooth v4.0 and above ● The session is still valid.

TD2DI

TD2DI and TD2DI-Security is the name of the interface that is offered by the HCP app regarding the D2D protocol, containing the operations for facilitating the S-EHR app to perform tasks related to the HCP app, by invoking these operations, as defined in the InteropEHRate architecture.

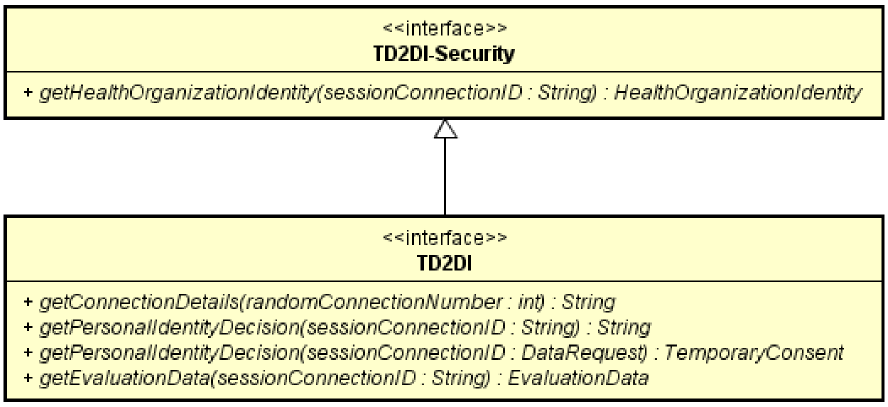
<<interface>>
TD2DI-Security
+ getHealthOrganizationIdentity(sessionConnectionID : String) : HealthOrganizationIdentity
<<interface>>
TD2DI
+ getConnectionDetails(randomConnectionNumber : int) : String
+ getPersonalIdentityDecision(sessionConnectionID : String) : String
+ getPersonalIdentityDecision(sessionConnectionID : DataRequest) : TemporaryConsent
+ getEvaluationData(sessionConnectionID : String) : EvaluationData

Figure 4 - TD2D interface operations

The following tables provide a complete description of all the operations of the TD2DI.

Operation getConnectionDetails

Name	getConnectionDetails
Description	This operation is invoked by the S-EHR app for getting the connection unique session identifier, in the form of a random String. This String will be used by both sides (S-EHR app and HCP app), for current connection identification purposes.
Arguments	● **randomConnectionNumber**: the random number that will be provided to the S-EHR app and has to be approved from both sides of the S-EHR app and the HCP app for the pairing instantiation.
Return Value	This operation will return a unique session identifier (sessionConnectionID), in the form of a random String that will be used by both sides (S-EHR app and HCP app), for current connection identification purposes.
Exceptions	● Security exceptions related to the Bluetooth connection. ● Network exceptions related to Bluetooth connection failure.
Preconditions	● The HCP app is enabled with Bluetooth v4.0 and above

Operation getHealthOrganizationIdentity

Name	getHealthOrganizationIdentity
Description	This operation is invoked by the S-EHR app for getting the Healthcare Organization personal identity, for identifying whether the identity is valid or not.
Arguments	● **sessionConnectionID**: a string that identifies the connection between the two involved applications.
Return Value	This operation will return the Health Organization identity in the form of an object (HealthOrganizationIdentity) containing specific details that identify the Organization.
Exceptions	● Security exceptions related to the Bluetooth connection. ● Network exceptions related to Bluetooth connection failure.

Preconditions	● The HCP app is enabled with Bluetooth v4.0 and above ● The session is still valid.

Operation getPersonalIdentityDecision

Name	getPersonalIdentityDecision
Description	This operation is invoked by the S-EHR app for getting the decision from the side of the HCP app, regarding whether the provided Personal identity is approved or not
Arguments	● **sessionConnectionID**: a string that identifies the connection between the two involved applications.
Return Value	This operation will return, one of the two following options: ● A connection closure message (connectionClosureMessage) in the form of a String, indicating that the Personal identity was not approved, hence the connection will be closed. ● The temporary consent request of the HCP app owner in the form of an object (TemporaryConsent), requesting access to specific types of data stored in the S-EHR app.
Exceptions	● Security exceptions related to the Bluetooth connection. ● Network exceptions related to Bluetooth connection failure.
Preconditions	● The HCP app is enabled with Bluetooth v4.0 and above ● The session is still valid.

Operation getEvaluationData

Name	getEvaluationData
Description	This operation is invoked by the S-EHR app for getting the evaluation data from the side of the HCP app (after the examination), in order to be stored to the S-EHR app.
Arguments	● **sessionConnectionID**: a string that identifies the connection between the two involved applications.

Return Value	This operation will return the evaluation data of the examination to the S-EHR app, in the form of an object (EvaluationData), containing the results of the evaluation in various formats (e.g. textual data, images).
Exceptions	● Security exceptions related to the Bluetooth connection. ● Network exceptions related to Bluetooth connection failure. ● Access exceptions related to the ownership of data (only evaluation data of the current S-EHR app owner can be provided).
Preconditions	● The HCP app is enabled with Bluetooth v4.0 and above ● The session is still valid.

2.3.2. Involved Applications (A7, SIVECO)

This section describes and lists the involved applications from the side of the S-EHR app and the HCP app.

2.3.2.1. S-EHR Application

A S-EHR app is any application installed on a personal mobile device, that is able to store the personal health data of a user in a secure (encrypted) way according to the constraints specified by [D3.1] and that supports the InteropeEHRate protocols defined in the current deliverable and in [D4.8]. Different vendors may develop different S-EHRs apps. A S-EHR app contains health data of the user, produced and signed (for traceability and trustability) by the healthcare organization that produces them, but can also contain data directly stored and produced by citizens or by sensors. The provenance and author of each health data is unambiguously persisted on the S-EHR and the principles of integrity and non repudiation are guaranteed. More details on what is supported by a S-EHR app can be found in [D2.4].

2.3.2.2. HCP Application (SIVECO)

HCP App is a software application designed to provide medical staff with the ability to access and operate patients' data from S-EHR Mobile App, S-EHR Cloud and EHR of the Healthcare Organization. In other words, the HCP App is an application used by the HCPs to securely exchange health data of their EHRs with any S-EHR Mobile App and to read health data stored in S-EHR Cloud using the InteropEHRate protocols. More details on what is supported by an HCP app can be found in [D2.4].

2.3.3. D2D Interface Interactions

The interactions between the interfaces of the MD2D and the TD2D are explained and visualized through the Sequence Diagram of Figure 5.

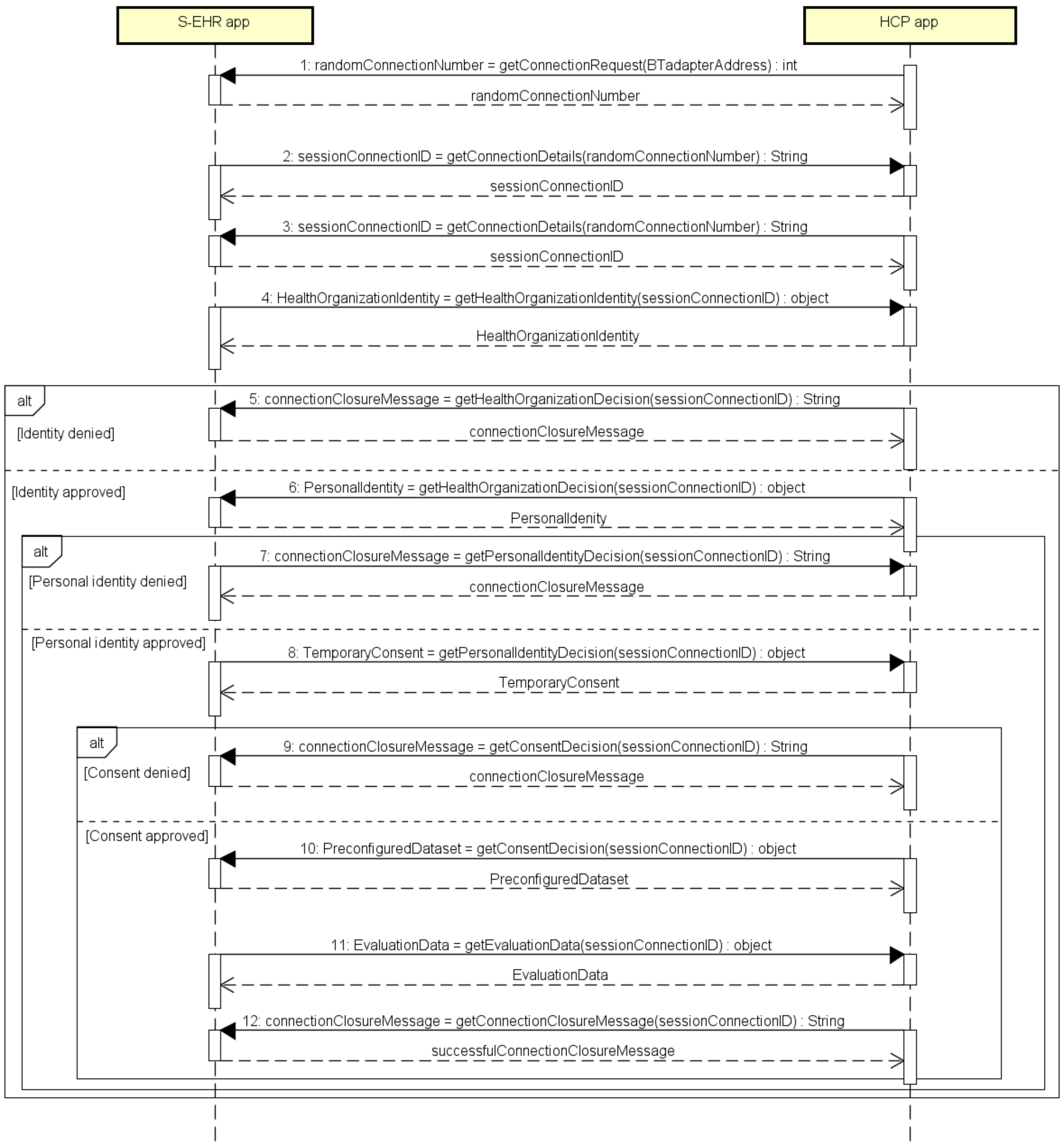
S-EHR app
HCP app
1: randomConnectionNumber = getConnectionRequest(BTadapterAddress) : int
randomConnectionNumber
2: sessionConnectionID = getConnectionDetails(randomConnectionNumber) : String
sessionConnectionID
3: sessionConnectionID = getConnectionDetails(randomConnectionNumber) : String
sessionConnectionID
4: HealthOrganizationIdentity = getHealthOrganizationIdentity(sessionConnectionID) : object
HealthOrganizationIdentity
alt
[Identity denied]
5: connectionClosureMessage = getHealthOrganizationDecision(sessionConnectionID) : String
connectionClosureMessage
[Identity approved]
6: PersonalIdentity = getHealthOrganizationDecision(sessionConnectionID) : object
PersonalIdenity
alt
[Personal identity denied]
7: connectionClosureMessage = getPersonalIdentityDecision(sessionConnectionID) : String
connectionClosureMessage
[Personal identity approved]
8: TemporaryConsent = getPersonalIdentityDecision(sessionConnectionID) : object
TemporaryConsent
alt
[Consent denied]
9: connectionClosureMessage = getConsentDecision(sessionConnectionID) : String
connectionClosureMessage
[Consent approved]
10: PreconfiguredDataset = getConsentDecision(sessionConnectionID) : object
PreconfiguredDataset
11: EvaluationData = getEvaluationData(sessionConnectionID) : object
EvaluationData
12: connectionClosureMessage = getConnectionClosureMessage(sessionConnectionID) : String
successfulConnectionClosureMessage

Figure 5 - D2D protocol interactions

Following, a detailed description of the sequence diagram takes place. It is assumed that the pairing and connection process has been already performed (as described in **Section 2.3.4**):

- **Step 1** - **getConnectionRequest**: This operation is invoked by the HCP app for getting the advertised connection request from the S-EHR app for performing the pairing process between them, and getting a random connection number that will be used for pairing the two applications.
- **Step 2 - getConnectionDetails**: The S-EHR app invokes this operation for getting the connection's unique session identifier, in the form of a random String. This String will be used by both sides (S-EHR app and HCP app), for the current's connection identification purposes.

- **Step 3 - getConnectionDetails**: The HCP app invokes this operation for getting the connection's unique session identifier, in the form of a random String (for acknowledgment reasons). This String will be used by both sides (S-EHR app and HCP app), for the current's connection identification purposes.
- **Step 4 - getHealthOrganizationIdentity**: The next step is for the S-EHR app to invoke this operation for getting the Healthcare Organization personal identity, for identifying whether the identity is valid or not.
- **Step 5-6 - getHealthOrganizationDecision**: As soon as the decision has been made about the Healthcare Organization identity, this operation is invoked by the HCP app for getting the decision from the side of the S-EHR app, regarding whether the provided Health Organization identity is approved or not. This operation will return, one of the two following options:
  - A connection closure message (connectionClosureMessage) in the form of a String, indicating that the Health Organization identity was not approved, hence the connection will be closed.
  - The personal data of the S-EHR app owner (i.e. name, surname, date of birth, location of birth, gender, country of residence, and social security number) in the form of an object (PersonalData).
- **Step 7-8 - getPersonalIdentityDecision**: In the case that the Healthcare Organization identity has been approved, this operation is invoked by the S-EHR app for getting the decision from the side of the HCP app, regarding whether the provided Personal identity is approved or not. As in the previous cases, this operation will return, one of the two following options:
  - A connection closure message (connectionClosureMessage) in the form of a String, indicating that the Personal identity was not approved, hence the connection will be closed.
  - The temporary consent request of the HCP app owner in the form of an object (TemporaryConsent), requesting access to specific types of data stored in the S-EHR app.
- **Step 9-10 - getConsentDecision**: In the case that the temporary consent has been requested, this operation is invoked by the HCP app for getting the decision from the side of the S-EHR app, regarding whether the consent for getting the S-EHR app owner's data has been approved or not. As in the previous cases, this operation will return, one of the two following options:
  - A connection closure message (connectionClosureMessage) in the form of a String, indicating that the Health Organization identity was not approved, hence the connection will be closed.
  - The already preconfigured dataset of the S-EHR application owner (i.e. data that has been approved for sharing) in the form of an object (PreconfiguredDataset).
- **Step 11 - getEvaluationData**: In the case that the preconfigured dataset has been provided, the S-EHR app invokes this method for getting the evaluation data from the side of the HCP app (after the examination), in order to be stored to the S-EHR app.
- **Step 12 - getConnectionClosureMessage**: The last steps includes this operation that has to be invoked by the HCP app for getting the final message of connection closure, after the overall interaction has successfully ended with the provision of the evaluation data.

2.3.4. D2D over Bluetooth

In order for the Bluetooth pairing and connection between the devices that host the S-EHR app and the HCP app to be realized, there must be followed a specific process between the client and the server application. In our case, both the S-EHR app and the HCP app will behave as a true peer-to-peer endpoint by exposing both server and client behavior. Typically, the pairing process that is being followed is based on the following figure (Figure 6).

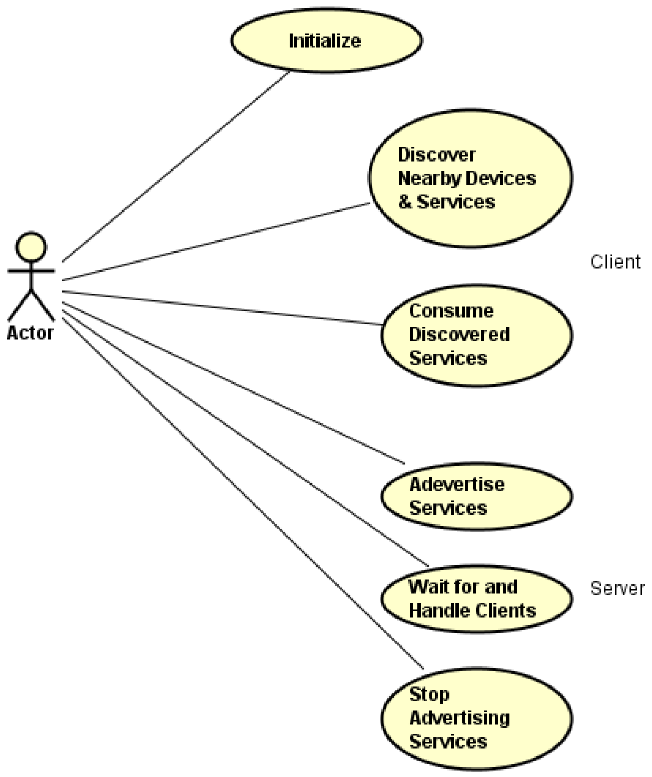
Initialize
Discover Nearby Devices & Services
Client
Actor
Consume Discovered Services
Adevertise Services
Wait for and Handle Clients
Server
Stop Advertising Services

Figure 6 - Bluetooth pairing process

In general, in order for Bluetooth-enabled devices to transmit data between each other, they must first form a channel of communication using a pairing process. One device, a discoverable device, makes itself available for incoming connection requests. Another device finds the discoverable device using a service discovery process. After the discoverable device accepts the pairing request, the two devices complete a bonding process where they exchange security keys. The devices cache these keys for later use. After the pairing and bonding processes are complete, the two devices exchange information. When the session is complete, the device that initiated the pairing request releases the channel that had linked it to the discoverable device. The two devices remain bonded, however, so they can reconnect automatically during a future session as long as they're in range of each other and neither device has removed the bond.

This process can be summarized as follows:

1. **Initialization**: All bluetooth enabled applications must first initialize the Bluetooth stack.
2. **Client**: A client consumes the remote services. This is done by first discovering any nearby devices, and afterwards for each discovered device it searches for services of interest.
3. **Server**: A server makes the services available to clients. It registers them in the SDDB **[SDDB]**, in effect advertising them. It then waits for incoming connections, accepts them as they come in, and

serves the clients that make them. Finally, when the service is no longer needed, the application removes it from the SDDB.

In general, in order to use Bluetooth, a device must be compatible with the subset of Bluetooth profiles, meaning a set of rules, which are necessary to use the desired services defined by these rules. A Bluetooth profile is a specification regarding an aspect of Bluetooth-based wireless communication between devices that resides on top of the Bluetooth Core Specification and (optionally) additional protocols [BCS]. The way a device uses Bluetooth depends on its profile capabilities. The profiles provide standards which manufacturers follow to allow devices to use Bluetooth in the intended manner.

Since the D2D protocol will be based on the Bluetooth v4.0, a research was performed to the different Bluetooth profiles that are supported by the Bluetooth Core Specification version 4.0 [BSCv4.0], where it was identified that the Bluetooth [SPP] was the most suitable Bluetooth profile to be used for the purposes of the D2D protocol. Shortly, the Bluetooth SPP profile defines the requirements for Bluetooth devices that are necessary for setting up emulated serial cable connections using the RFCOMM protocol [RFCOMM] between the two peer devices. The requirements are expressed in terms of services provided to applications, and by defining the features and procedures that are required for interoperability between Bluetooth devices.

In the Bluetooth SPP profile, the following roles are defined:

- Device A (DevA) takes initiative to form a connection to another device (Initiator).
- Device B (DevB) waits for another device to take initiative to connect (Acceptor).

Figure 7 shows the protocols and entities used in the Bluetooth SPP profile in the case of the S-EHR and the HCP app, where the S-EHR app is considered as the Initiator while the HCP app is considered as the Acceptor. These roles can change between the two involved applications. Shortly, the Baseband [Baseband], LMP [LMP] and L2CAP [L2CAP] are the OSI layer [OSI] of the Bluetooth protocols. RFCOMM is the Bluetooth adaptation of GSM TS 07.10 [GSM], providing a transport protocol for serial port emulation, while SDP [SDP] is the Bluetooth Service Discovery Protocol.

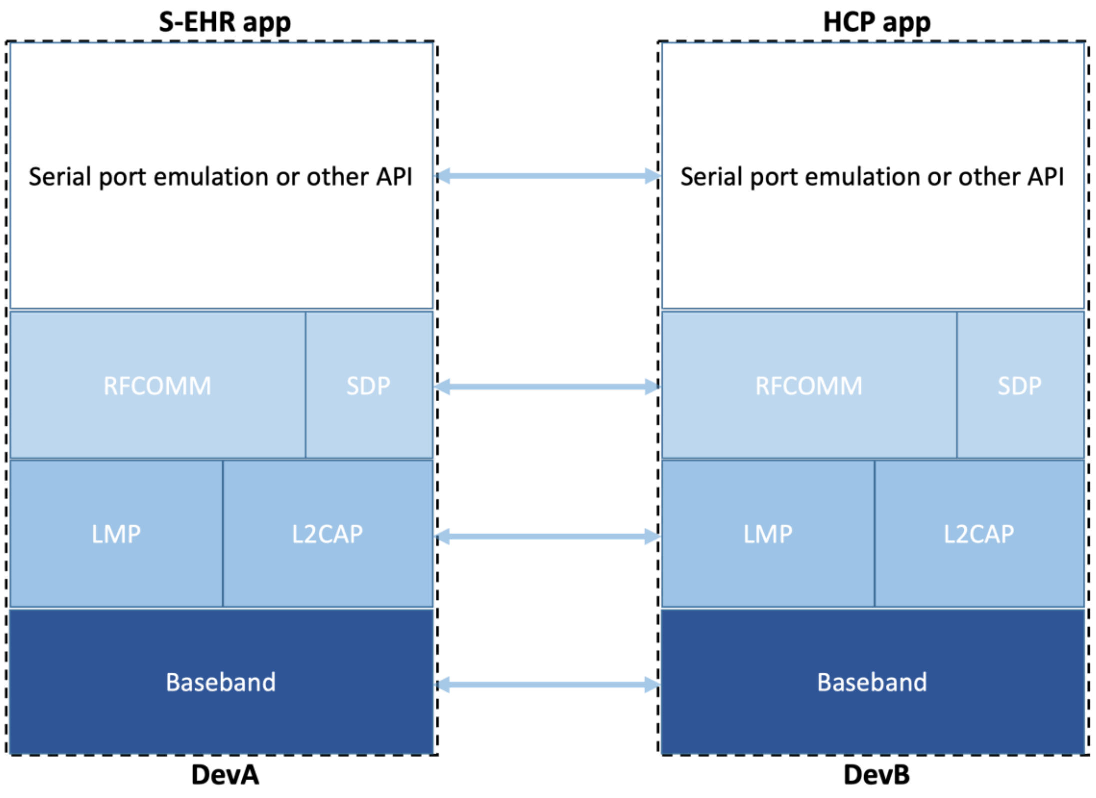

Figure 7 - Protocols and entities of the Bluetooth SPP profile

The required procedures that should be defined in the Bluetooth SPP profile are divided in three different steps, referring to the (i) establishment of a link and the set up of a Virtual Serial Connection, (ii) the acceptance of the link and the establishment of a Virtual Serial Connection, and (iii) the registration of the Service Record in a local SDP database.

(i) Establish Link and Set up Virtual Serial Connection

These steps are defined below:

1. Submit a query using SDP to find out the RFCOMM Server channel number of the desired application in the remote device.
2. Optionally, require authentication of the remote device to be performed. Also optionally, require encryption to be turned on.
3. Request a new L2CAP channel to the remote RFCOMM entity.
4. Initiate an RFCOMM session on the L2CAP channel.
5. Start a new data link connection on the RFCOMM session, using the aforementioned server channel number. After step 5, the virtual serial cable connection is ready to be used for communication between applications on both sides.

In the case that there already exists an RFCOMM session between the devices when setting up a new data link connection, the new connection must be established on the existing RFCOMM session. (This is equivalent to skipping over steps 3 and 4 above.)

(ii) Accept Link and Establish Virtual Serial Connection

These steps are defined below:

1. If requested by the remote device, take part in authentication procedure and, upon further request, turn on encryption.
2. Accept a new channel establishment indication from L2CAP.
3. Accept an RFCOMM session establishment on that channel.
4. Accept a new data link connection on the RFCOMM session. This may trigger a local request to authenticate the remote device and turn on encryption, if the user has required that for the emulated serial port being connected to (and authentication/encryption procedures have not already been carried out).

(iii) Register Service Record in Local SDP Database

This procedure refers to registration of a service record for an emulated serial port (or equivalent) in the SDP database. This implies the existence of a Service Database, and the ability to respond to SDP queries. All services/applications reachable through RFCOMM need to provide an SDP service record that includes the parameters necessary to reach the corresponding service/application.

2.3.4.1. D2D over Android Bluetooth API

In the 1st version of the D2D protocol specification, we are exploiting the Bluetooth SPP profile, while the specification is based on the Android Bluetooth API [ANDROIDBT] since the D2D protocol is currently in a very immature phase, and we would like to perform additional research on the different platforms for identifying the Bluetooth profile that can satisfy all the requirements as specified in D2.1 [D2.1]. In the next version of the D2D protocol specification that will be released in March 2020, the D2D protocol specification will be independent from any API in order for any developer that would like to use this protocol to be able to use the listed operations or develop her own operations, so as to implement the corresponding data exchange between a S-EHR and an HCP application.

As a result, based on what has been described in the Bluetooth SPP profile and the Android Bluetooth API that is based upon this profile, the following steps are followed for enabling the connection and the exchange of data between the involved applications. These steps are classified in five different categories: (a) find devices, (b) query paired devices, (c) discover devices, (d) connect devices (either as a server or as a client), and (e) manage connection.

(a) Find devices

In D2D, the BluetoothAdapter is being used for finding remote Bluetooth devices either through device discovery or by querying the list of paired devices. In general, device discovery is considered as the scanning procedure that searches the local area for Bluetooth-enabled devices and requests some information about each one. This process is sometimes referred to as discovering, inquiring, or scanning. However, a nearby Bluetooth device responds to a discovery request only if it is currently accepting information requests by being discoverable. If a device is discoverable, it responds to the discovery request by sharing some information, such as the device's name, its class, and its unique MAC address. Using this information, the device that is performing the discovery process can then choose to initiate a connection to the discovered device. Once a connection is made with a remote device for the first time, a pairing request is automatically presented to the user. When a device is paired, the basic information about that device - such as the device's name, class, and MAC address - is saved and can be read using the Bluetooth APIs.

Using the known MAC address for a remote device, a connection can be initiated with it at any time without performing discovery, assuming the device is still within range.

- In order for the devices to be paired, the two devices have to be aware of each other's existence, have a shared link-key that can be used for authentication, and are capable of establishing an encrypted connection with each other.
- In order for the devices to be connected, the two devices have to share an RFCOMM channel and to transmit data with each other, meaning that they have to be already paired with each other. Pairing is automatically performed when it is initiated an encrypted connection with the Bluetooth APIs.

(b) Query paired devices

Before performing device discovery, a query is performed to the set of paired devices to see if the desired device is already known. This is done through calling **getBondedDevices()**. This returns a set of BluetoothDevice objects representing paired devices.

(c) Discover devices

To start discovering devices, **startDiscovery()** is called. The process is asynchronous and returns a boolean value indicating whether discovery has successfully started. The discovery process usually involves an inquiry scan of about 12 seconds, followed by a page scan of each device found to retrieve its Bluetooth name. In order to receive information about each device discovered, the involved applications must register a BroadcastReceiver for the ACTION_FOUND intent. The system broadcasts this intent for each device. The intent contains the extra fields EXTRA_DEVICE and EXTRA_CLASS, which in turn contain a BluetoothDevice and a BluetoothClass, respectively.

(d) Connect devices

In order to create a connection between the involved applications, both the server-side and client-side mechanisms have to be implemented, because one device must open a server socket, and the other one must initiate the connection using the server device's MAC address. The server device and the client device each obtain the required BluetoothSocket in different ways. The server receives socket information when an incoming connection is accepted. The client provides socket information when it opens an RFCOMM channel to the server. The server and client are considered connected to each other when they each have a connected BluetoothSocket on the same RFCOMM channel. At this point, each device can obtain input and output streams, and data transfer can then begin.

- **Connect as a Server**

  The server should hold an open BluetoothServerSocket. The purpose of the server socket is to listen for incoming connection requests and provide a connected BluetoothSocket after a request is accepted. When the BluetoothSocket is acquired from the BluetoothServerSocket, the BluetoothServerSocket can be discarded. In order to set up a server socket and accept a connection, the following sequence of steps are followed:

  - Get a BluetoothServerSocket by calling **listenUsingRfcommWithServiceRecord()**.
  The string is an identifiable name of the service, which the system automatically writes to a new SDP database entry on the device. The name is arbitrary and can simply be the application's name.

The UUID is also included in the SDP entry and forms the basis for the connection agreement with the client device. That is, when the client attempts to connect with this device, it carries a UUID that uniquely identifies the service with which it wants to connect. These UUIDs must match in order for the connection to be accepted.

- Start listening for connection requests by calling **accept()**.
This is a blocking call that returns when either a connection has been accepted or an exception has occurred. A connection is accepted only when a remote device has sent a connection request containing a UUID that matches the one registered with this listening server socket. When successful, accept() returns a connected BluetoothSocket.

- Close connections by calling **close()**.
This operation releases the server socket and all its resources, but doesn't close the connected BluetoothSocket that has been returned by accept(). RFCOMM allows only one connected client per channel at a time, so in most cases, it makes sense to call close() on the BluetoothServerSocket immediately after accepting a connected socket.

- **Connect as a Client**
  In order to initiate a connection with the application that is accepting connections on an open server socket, it must first be obtained a BluetoothDevice object that represents the remote device. The basic procedure is as follows:

  - Using the BluetoothDevice, get a BluetoothSocket by calling **createRfcommSocketToServiceRecord(UUID)**.
  This operation initializes a BluetoothSocket object that allows the client to connect to a BluetoothDevice. The UUID passed here must match the UUID used by the server device when it called **listenUsingRfcommWithServiceRecord(String, UUID)** to open its BluetoothServerSocket.

  - Initiate the connection by calling **connect()**.
  This is a blocking call, where after the client calls this method, the system performs an SDP lookup to find the remote device with the matching UUID. If the lookup is successful and the remote device accepts the connection, it shares the RFCOMM channel to use during the connection, and the **connect()** method returns. If the connection fails, or if the **connect()** method times out (after about 12 seconds), then the method throws an IOException.

(e) Manage a connection

After the applications have successfully connected, each one has a connected BluetoothSocket. Using the BluetoothSocket, the general procedure to transfer data is as follows:

- Get the InputStream and OutputStream that handle transmissions through the socket using **getInputStream()** and **getOutputStream()**, respectively.

- Read and write data to the streams using **read(byte[])** and **write(byte[])**.

It should be mentioned that it is used a dedicated thread for reading from the stream and writing to it. This is important because both the **read(byte[])** and **write(byte[])** methods are blocking calls. The **read(byte[])** method blocks until there is something to read from the stream. The **write(byte[])** method does not usually block, but it can block for flow control if the remote device is not calling **read(byte[])** quickly enough and the intermediate buffers become full as a result. As a result, the main loop in the thread is dedicated to reading from the InputStream. A separate public method in the thread can be used to initiate writes to the OutputStream.

- After the constructor acquires the necessary streams, the thread waits for data to come through the InputStream. When **read(byte[])** returns with data from the stream, the data is provided using a member Handler from the parent class. The thread then waits for more bytes to be read from the InputStream.
- Sending outgoing data is performed through calling the thread's **write()** method and passing in the bytes to be sent. This method calls **write(byte[])** to send the data to the remote device. If an IOException is thrown when calling **write(byte[])**, the thread notifies that the device could not send the given bytes to the other (connected) device.

- The thread's **cancel()** method allows the connection to be terminated at any time by closing the BluetoothSocket. This method should always be called when the Bluetooth connection has to be finished.

Based on the D2D over Android Bluetooth API and the Conceptual D2D descriptions, the conceptual operations that are invoked by the S-EHR app and the HCP app, can be mapped with the methods of the Android Bluetooth API as follows:

For the operations of the MD2DI which are invoked by the HCP app,

- getConnectionRequest can be mapped with **listenUsingRfcommWithServiceRecord()** that is responsible for getting a Bluetooth Server Socket, while also being replaced by the total of the methods that are offered in the stages of finding devices, querying paired devices and discovering devices, as described above.
- getConnectionDetails can be mapped with the methods offered by the stage of querying paired devices, regarding the method of **getBondedDevices()** for getting the bonded devices, while it can also be mapped with the connected devices stages, regarding the **accept()** and **close()** methods for either accepting or closing the specific connection.
- getHealthOrganizationDecision can be mapped with the methods offered by the stage of managing a specific connection, regarding the methods of **getInputStream(), getOutputStream(), read(byte[])** and **write(byte[])**, for reading or writing specific data accordingly.
- getConsentDecision can be mapped with the methods offered by the stage of managing a specific connection, regarding the methods of **getInputStream(), getOutputStream(), read(byte[])** and **write(byte[])**, for reading or writing specific data accordingly.
- getConnectionClosureMessage can be mapped with the methods offered by the stage of managing a specific connection, regarding the methods of **getInputStream(), getOutputStream(), read(byte[])** and **write(byte[])**, for reading or writing specific data accordingly.

The operations of the TD2DI which are invoked by the S-EHR app,

- getConnectionDetails can be mapped with the methods offered by the stage of querying paired devices, regarding the method of **getBondedDevices()** for getting the bonded devices, while it can also be mapped with the connected devices stages, regarding the **accept()** and **close()** methods for either accepting or closing the specific connection.
- getHealthOrganizationIdentity can be mapped with the methods offered by the stage of managing a specific connection, regarding the methods of **getInputStream(), getOutputStream(), read(byte[])** and **write(byte[])**, for reading or writing specific data accordingly.
- getPersonalIdentityDecision can be mapped with the methods offered by the stage of managing a specific connection, regarding the methods of **getInputStream(), getOutputStream(), read(byte[])** and **write(byte[])**, for reading or writing specific data accordingly.
- getEvaluationData can be mapped with the methods offered by the stage of managing a specific connection, regarding the methods of **getInputStream(), getOutputStream(), read(byte[])** and **write(byte[])**, for reading or writing specific data accordingly.

2.3.4.2. D2D Commands & Replies Specification

As soon as the devices have been paired and the connection has been established, the messages that will be exchanged within the context of the D2D between the involved applications, should follow the process defined below, being also consistent to the provided rules, as described in the following tables. Figure 8 illustrates the commands and replies that will be exchanged in the case of the D2D protocol, between any involved applications.

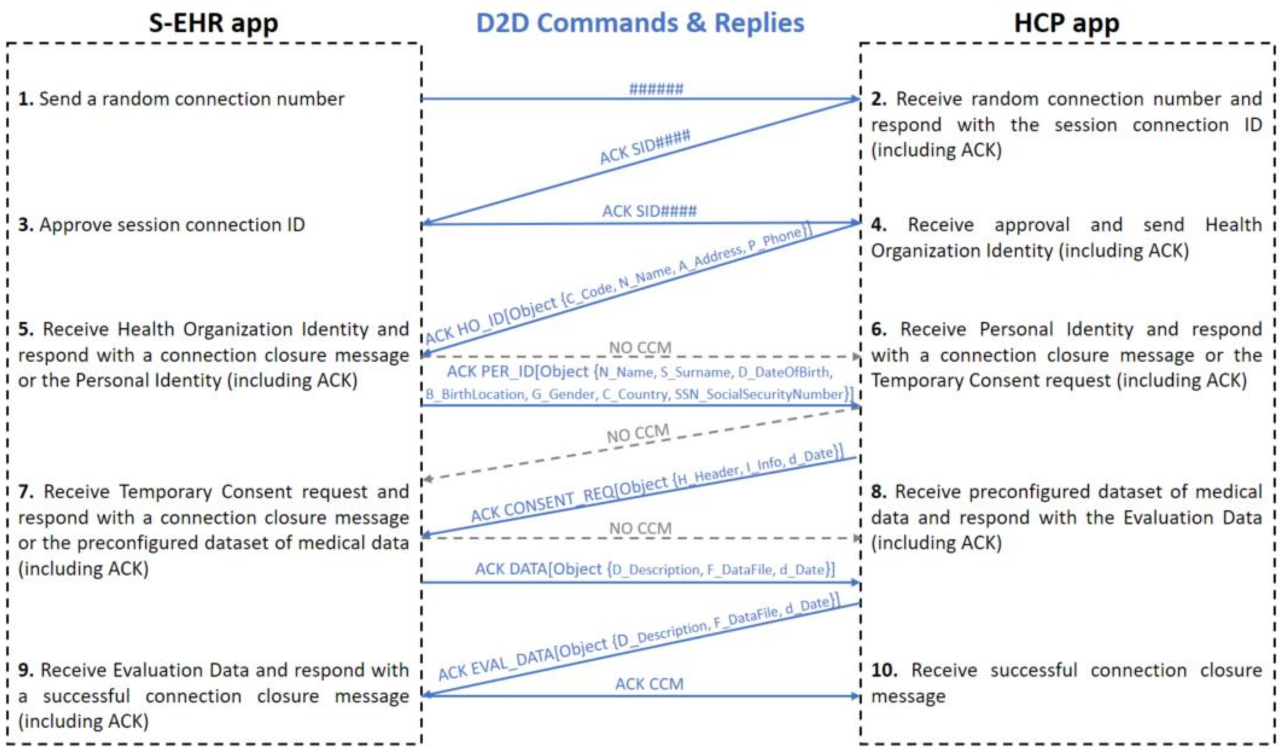
S-EHR app
D2D Commands & Replies
HCP app
1. Send a random connection number
######
2. Receive random connection number and respond with the session connection ID (including ACK)
ACK SID####
3. Approve session connection ID
ACK SID####
4. Receive approval and send Health Organization Identity (including ACK)
ACK HO_ID[Object {C_Code, N_Name, A_Address, P_Phone}]
5. Receive Health Organization Identity and respond with a connection closure message or the Personal Identity (including ACK)
NO CCM
ACK PER_ID[Object {N_Name, S_Surname, D_DateOfBirth, B_BirthLocation, G_Gender, C_Country, SSN_SocialSecurityNumber}]
6. Receive Personal Identity and respond with a connection closure message or the Temporary Consent request (including ACK)
NO CCM
ACK CONSENT_REQ[Object {H_Header, I_Info, d_Date}]
7. Receive Temporary Consent request and respond with a connection closure message or the preconfigured dataset of medical data (including ACK)
NO CCM
ACK DATA[Object {D_Description, F_DataFile, d_Date}]
8. Receive preconfigured dataset of medical data and respond with the Evaluation Data (including ACK)
ACK EVAL_DATA[Object {D_Description, F_DataFile, d_Date}]
9. Receive Evaluation Data and respond with a successful connection closure message (including ACK)
ACK CCM
10. Receive successful connection closure message

Figure 8 - D2D exchanged messages

A description of the steps of Figure 8 follows, accompanied with the rules that each command and reply should follow. All these steps are part of the D2D protocol that are provided by operations which are mapped with the methods offered by the stage of managing a specific connection in the Android Bluetooth API, regarding the methods of **getInputStream(), getOutputStream(), read(byte[])** and **write(byte[])**, for reading or writing specific data accordingly.

1. **S-EHR app**: The S-EHR app will send to the HCP app a random connection number in the form of 6 numerical digits following the format of: [######]

randomConnectionNumber Rules

Name	randomConnectionNumber
Description	The random number that will be generated for establishing the Bluetooth connection between the involved application in the D2D protocol.
Accepted Format	This message should follow the format of an integer of length of 6 digits`
Example	990045

2. **HCP app**: The HCP app will send to the S-EHR app an acknowledgement message, followed by the current's session connection identifier, following the format of: [ACK SID####]

sessionConnectionID Rules

Name	sessionConnectionID
Description	The identifier of the current Bluetooth connection session between the involved applications
Accepted Format	This message should follow the format of "ACK SID" followed by any String of length of 4 digits (alphanumerical)
Example	ACK SID3564

3. **S-EHR app**: The S-EHR app will send to the HCP app an acknowledgment message for approving the current's session connection identifier, following the format of: [ACK SID####]

sessionConnectionIDAcceptance Rules

Name	sessionConnectionIDAcceptance
Description	The identifier of the current Bluetooth connection session between the involved applications
Accepted Format	This message should follow the format of “ACK SID” followed by any String of length of 4 digits (alphanumerical)
Example	ACK SID3564

4. **HCP app**: The HCP app will send to the S-EHR app an acknowledgement message, followed by the object identifying the Health Organization identity, including information about the current coded name, name, address and phone number, following the format of: [ACK HO_ID[*Object*{C_*Code*, N_*Name*, A_*Address*, P_*Phone*}]

HealthOrganizationIdentity Rules

Name	HealthOrganizationIdentity
Description	The acknowledgment message and the that corresponds to the Healthcare Organization identity.
Accepted Format	This message should follow the format of “ACK HO_ID” followed by an object within brackets that specifies the code, the name, the address and the phone number of the Healthcare Organization. These 4 values can be of the types defined below, where each one of them should have the following initials prior to their description, separated by commas. ● The Code should start with C_<CODE>: String ● The Name should start with N_<NAME>: String ● The Address should start with A_<ADDRESS>: String ● The Phone should start with P_<PHONE>: String
Example	ACK HO_ID[C_FTGM, N_Fondazione Toscana "Gabriele Monasterio”, A_Via Giuseppe Moruzzi, 1, 56124 Pisa PI, Italy, P_+39 050 315 3229]

5. **S-EHR app**: The S-EHR app will send to the HCP app an acknowledgement message, followed by the object identifying the Personal Identity of the S-EHR app owner, following the format of: [ACK PER_ID[*Object* {N_*Name*, S_*Surname*, D_*DateOfBirth*, B_*BirthLocation*, G_*Gender*, C_*Country*, SSN_*SocialSecurityNumber*}]. In the case that the Health Organization identity will not be approved,

the message that will be sent will be a one way message, informing about the connection closure, following the format of: [NO CCM].

PersonalIdentity Rules

Name	PersonalIdentity
Description	The acknowledgment message and the that corresponds to the Personal identity.
Accepted Format	This message should follow the format of "ACK PER_ID" followed by an object within brackets that specifies the name, surname, date of birth, birth location, gender, country and social security number of the owner of the S-EHR app. These 7 values can be of the types defined below, where each one of them should have the following initials prior to their description, separated by commas. ● The Name should start with N_<NAME>: String ● The Surname should start with S_<SURNAME>: String ● The Date of birth should start with D_<DATEOFBIRTH>: date (accepted format dd/mm/yyyy) ● The Birth location should start with B_<BIRTHLOCATION>: String ● The Gender should start with G_<GENDER>: String (accepted format for female, male or undefine are F OR M or U accordingly) ● The Country should start with C_<COUNTRY>: String ● The Social Security Number (SSN) should start with S_<SSN>: String In the case that a connection closure message, this should follow the format of "NO CMM" indicating the connection closure.
Example	ACK PER_ID[N_Peter, S_Brown, D_19/12/1974, B_London, G_Male, C_UnitedKingdom, SSN_35763BB]

6. **HCP app**: The HCP app will send to the S-EHR app an acknowledgement message, followed by the object of the Temporary Consent towards the S-EHR app owner, following the format of: [ACK CONSENT_REQ[*Object* {H_*Header*, I_*Info*, d_*Date*}]. In the case that the Consent Request will not be approved, the message that will be sent will be a one way message, informing about the connection closure, following the format of: [NO CCM].

TemporaryConsent Rules

Name	TemporaryConsent

Description	The acknowledgment message and the that corresponds to the Consent Request
Accepted Format	This message should follow the format of "ACK CONSENT_REQ" followed by an object within brackets, that specifies the header, information and the date of the provided consent. These 3 values can be of the types defined below, where each one of them should have the following initials prior to their description, separated by commas. ● The Header should start with H_<HEADER>: String ● The Information should start with I_<INFORMATION>: String ● The date should start with d_<DATE>: date (accepted format dd/mm/yyyy) In the case that a connection closure message, this should follow the format of "NO CMM" indicating the connection closure.
Example	ACK CONSENT_REQ[H_AccessData, I_Request accessing personal data for viewing the latest examination's results, d_21/06/2019]

7. **S-EHR app**: The S-EHR app will send to the HCP app an acknowledgement message, followed by the object identifying the preconfigured dataset to be provided by the S-EHR app owner, following the format of: [ACK DATA[*Object* {D_*Description*, F_*DataFile,* d_*Date*}]. In the case that the Consent Request will not be approved, the message that will be sent will be a one way message, informing about the connection closure, following the format of: [NO CCM].

PreconfiguredDataset Rules

Name	PreconfiguredDataset
Description	The acknowledgment message and the that corresponds to the Preconfigured Dataset
Accepted Format	This message should follow the format of "ACK DATA" followed by an object within brackets, that specifies the description, file and the date of the preconfigured dataset. These 3 values can be of the types defined below, where each one of them should have the following initials prior to their description, separated by commas. ● The Description should start with D_<DESCRIPTION>: String ● The file should start with F_<DATAFILE>: File ● The date should start with d_<DATE>: date (accepted format

	dd/mm/yyyy) In the case that a connection closure message, this should follow the format of "NO CMM" indicating the connection closure.
Example	ACK DATA[D_Last Examination Results, F_ExamResults.csv, d_21/06/2019]

8. **HCP app**: The HCP app will send to the S-EHR app an acknowledgement message, followed by the object of the Evaluation Data towards the S-EHR app owner, following the format of: [ACK EVAL_DATA[*Object* {D_*Description*, F_*DataFile,* d_*Date*}].

EvaluationData Rules

Name	EvaluationData
Description	The acknowledgment message and the object that corresponds to the Evaluation Data
Accepted Format	This message should follow the format of "ACK EVAL_DATA" followed by an object within brackets, that specifies the description, file and the date of the evaluation data. These 3 values can be of the types defined below, where each one of them should have the following initials prior to their description, separated by commas. ● The Description should start with D_<DESCRIPTION>: String ● The file should start with F_<DATAFILE>: File ● The date should start with d_<DATE>: date (accepted format dd/mm/yyyy)
Example	ACK EVAL_DATA[D_Last Examination Evaluation Data, F_EvaluationResults.csv, d_21/06/2019

9. **S-EHR app**: The S-EHR app will send to the HCP app a one way message, informing about the successful connection closure (upon receiving the evaluation data), following the format of: [ACK CCM].

successfulConnectionClosureMessage Rules

Name	successfulConnectionClosureMessage

Description	The message that informs about the success of the receival of the evaluation data, and the connection closure between the involved applications
Accepted Format	This message should follow the format of “ACK CMM” indicating the connection closure.
Example	ACK CCM

3. R2D PROTOCOL: REMOTE HEALTH DATA EXCHANGE

The R2D protocol, defines the set of operations and structure of data used for enabling (in a standard way) the exchange of health data between a local or National EHR or a S-EHR Cloud and the S-EHR App with the usage of the internet (in opposition to the D2D that is the protocol to exchange health data without the usage of internet).

This chapter provides a detailed description of the context in which the R2D will be used, and of all the conditions that have influenced the decisions taken in the process carried out to define the R2D specifications.

3.1. R2D Protocol Scope

The most characteristic aspect of InteropEHRate project is the fact that an EHR resides on a citizen's mobile device. Citizen's (health) data transfer is realized by using the two protocols described in this deliverable: D2D for short range transmission, R2D for remote transmission over the internet.

According to the objectives defined by the InteropEHRate consortium for the first year of the project, the first version of the R2D protocol focuses on acquiring medical data from a National EHR, in order to import them in the citizen's device.

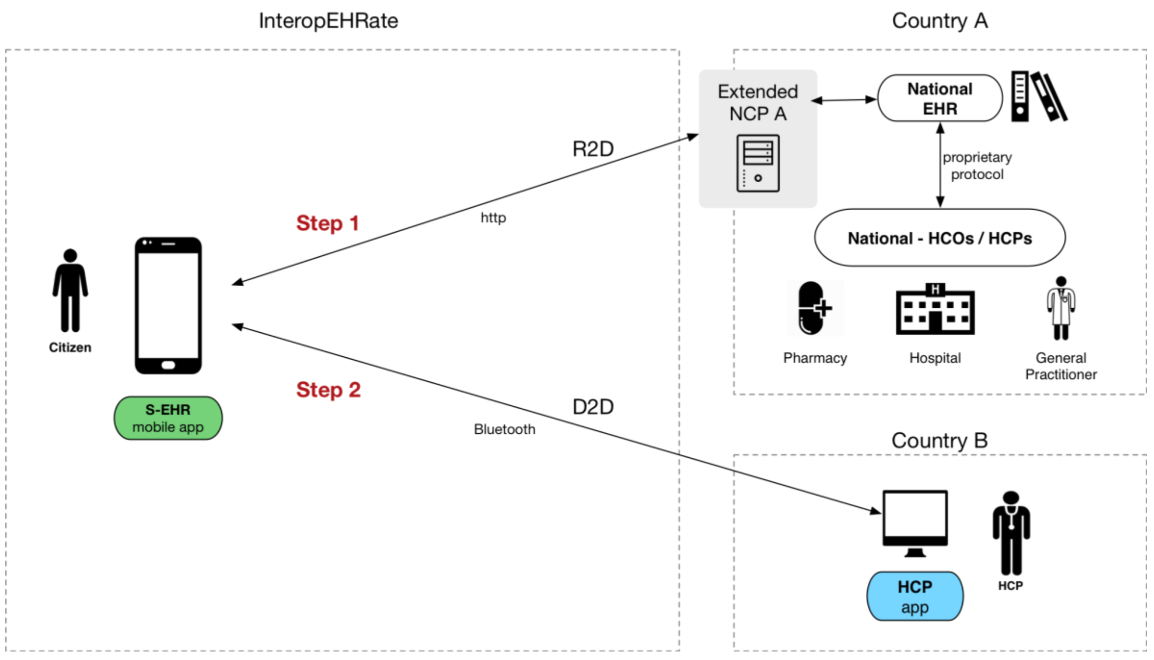

Figure 9 - R2D protocol scope

In the InteropEHRate standard architecture (i.e. the European architecture for health data exchange proposed by the project), data flows from an EHR system of Country A (the country of the citizen) to the Health Organization Information System of an HCP operating in Country B, with the intermediation of the citizen. Alla data transfers are performed over the two InteropEHRate's protocols: R2D and D2D. The two operations depicted in the above figure, do not happen at the same time: first of all the citizen has imported (using R2D) his EHR from the National EHR system of his Country to his smartphone, only after this operation has been executed, he would be able to transfer his EHR (using D2D) to an HCP in Country B.

However, importing health data in a standard way from EHRs of several European countries is a very challenging subject. At the moment each Member State has its own EHR, based on proprietary APIs, proprietary data model and proprietary data representation. These EHRs have been designed in order to satisfy only requirements coming from stakeholders of the Member State itself and not to be interoperable with each other. The architecture shown in the previous figure represents the evolution of the actual situation, and the application of the idea behind the InteropEHRate.

Many steps have been performed by the EU in order to foster interoperability between European eHealth systems, especially through the activities of the eHealth Network (established under Article 14 of Directive 2011/24/EU of the European Parliament and of the Council) and the CEF Programme.

3.2. Related Work

The most important project regarding cross border health data exchange financed by EU is eHDSI or eHealth DSI, whose objectives are the initial deployment and operation of services for cross-border health data exchange under the CEF. The architecture of eHDSI (Figure 10) is based on a closed and trusted federation of NCP one for each Member State named eHDSI Circle of Trust; an NCP is a software component representing a Member State's EHR. An NCP provides a reduced-but-common API designed for allowing EHRs of EU countries to interoperate in order to exchange health data.

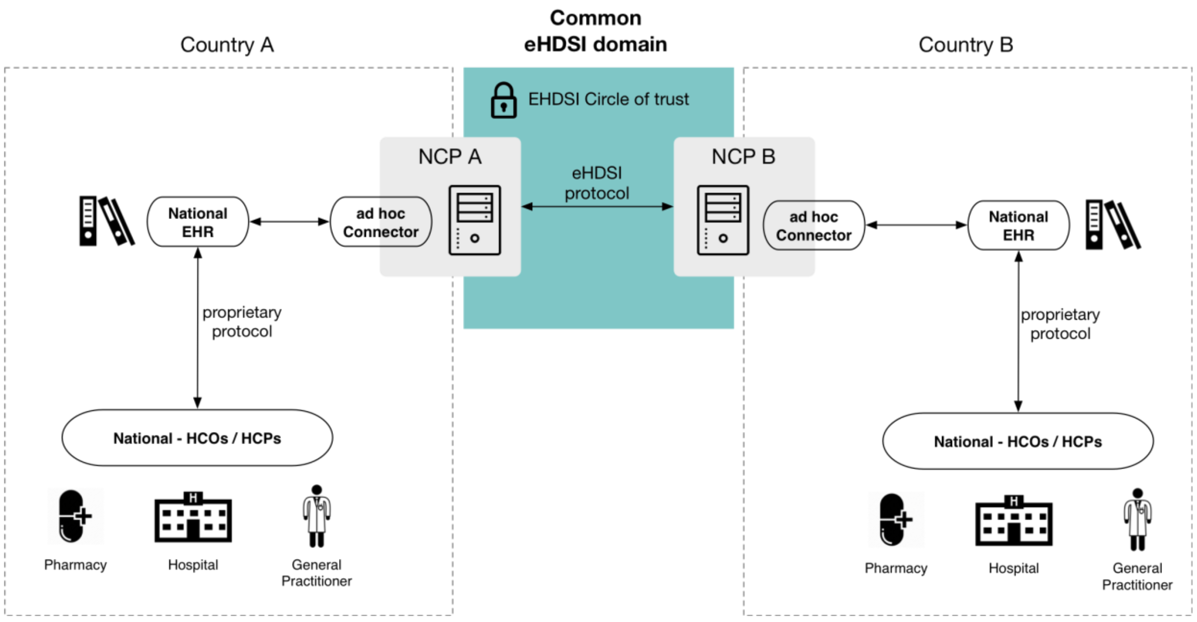

Figure 10 - eHDSI system architecture

Figure 10 shows the eHDSI System Architecture [EHDSI SYS SPECS], stressing the role of the NCPs federation as a common integration layer between the different EHRs of Member States. An NCP has the following responsibilities:

- forwarding requests coming from its country to other NCPs;
- handling incoming requests coming from other NCPs;
- converting data from eHDSI data representation to owner country data representation.

Differently from InteropEHRate, in eHDSI a citizen's EHR, despite the ownership of the citizen itself, resides in the IT National Infrastructure. In this context, an HCP of Country A, in order to request health data of a citizen of Country B (in that specific moment in Country A in front of the HCP), needs to access to National Infrastructure of his country (Country A) and use the provided cross border health data exchange functionalities to submit requests to Country B via NCP (requests are delegated to NCP of Country A, that forwards them to NCP of Country B). Transactions in eHDSI always work following this schema, the HCP interacts only with his National Infrastructure, and in the background the two involved national systems exchange data using eHDSI as common integration layer (the citizen does not have an active role in eHDSI transactions). According to [EUCBEHR REC] (EU Commission Recommendation of the sixth of February 2019 on a European Electronic Health Record exchange format), the baseline for the exchange contains the following health data:

- Patient Summary;
- ePrescription / eDispensation;
- Laboratory results;
- Medical imaging and reports;
- Hospital discharge reports.

R2D and eHDSI share many objectives (mainly the exchange of health data in a standard way), however, they show also some relevant differences:

- eHDSI is a system available only to HCOs and HCP, it has not been designed considering the citizen as a primary actor. In eHDSI, a citizen cannot submit requests to his reference NCP (NCP of his country), because NCPs are configured to receive requests only from other NCPs (closed and trusted federation). Differently, in InteropEHRate the citizen is the actor at the centre of the platform, he periodically downloads his data from the National EHR to his smartphone, and he handles directly his data when in front of an HCP.
- eHDSI uses a standard representation of data (HL7/CDA), but it does not adopt any of the existing eHealth standard API, like for instance the API proposed by OpenEHR project [OPENEHR]. Differently, R2D will be based upon a reduced version of FHIR, it will support FHIR API language and FHIR data model and data representation.

This last point is very important both for the future of eHDSI and for a possible convergence between eHDSI and R2D. Also the European Commission outlined this issue in [EUCBEHR ANNEX] (ANNEX to the Commission Recommendation on a European Electronic Health Record exchange format): "The refinement of the exchange format should consider the possibility offered by resource driven information models (such as Health Level Seven Fast Healthcare Interoperability Resources (HL7 FHIR©)".

The following figure shows how eHDSI and InteropEHRate could coexist, if eHDSI would adopt R2D / FHIR and would allow a citizen to authenticate to the NCP of his Country:

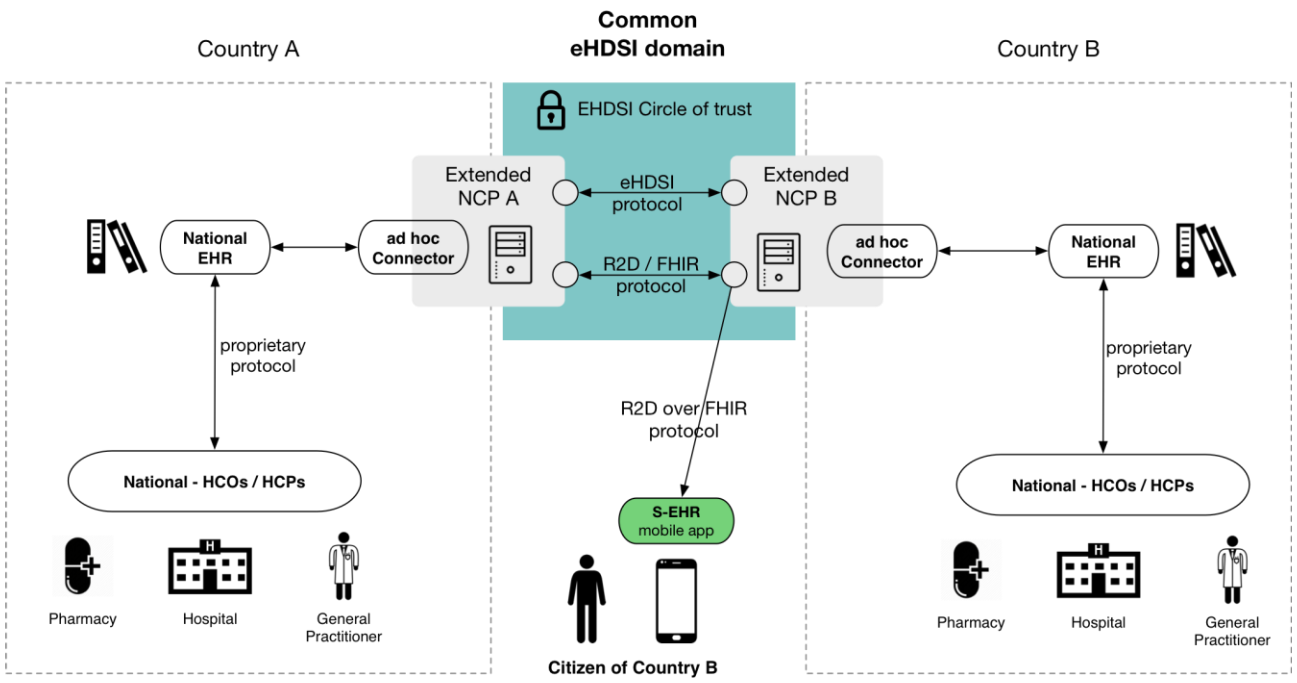

Figure 11 - Common eHDSI domain

R2D protocol will be specified in two versions, one based on FHIR and the other one based on the current eHDSI API specifications. R2D protocol will be specified first at a conceptual level and then translated into two concrete specifications: i) one named R2D over FHIR, ii) the other one named R2D over eHDSI (extended with the security protocol defined in [D3.3]). The following figure shows another possible way in which eHDSI and InteropEHRate could coexist:

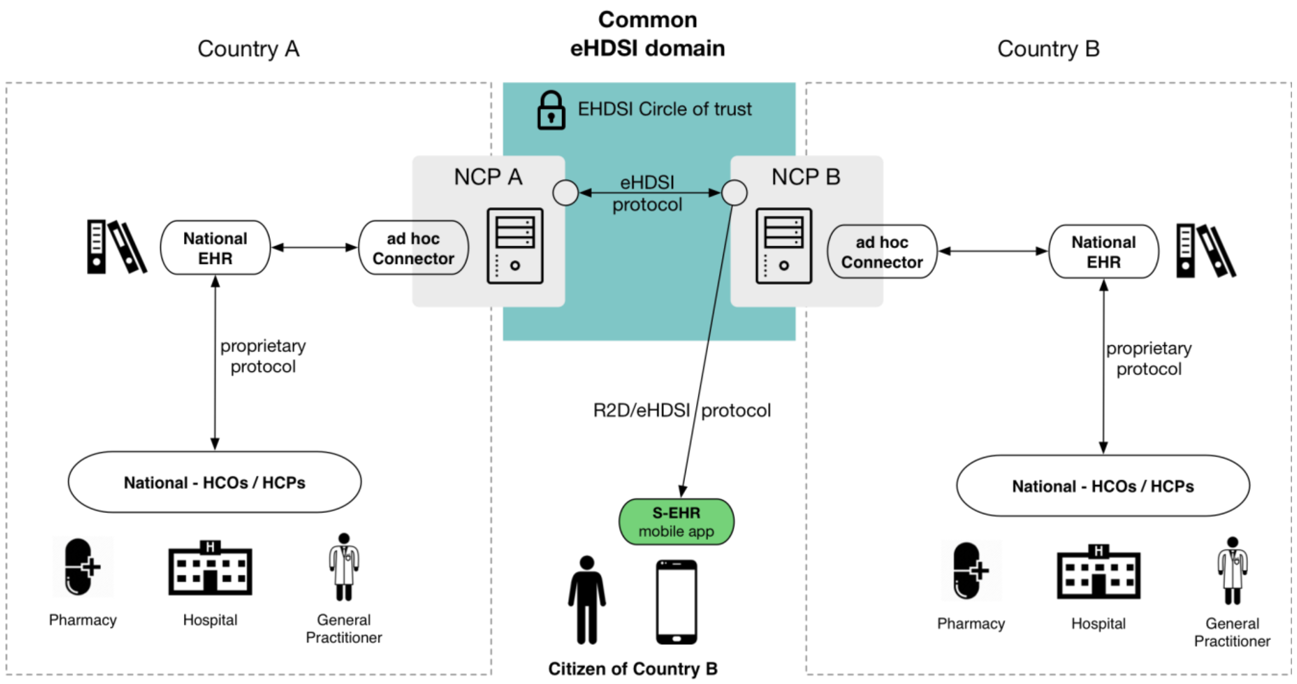

Figure 12 - Common eHDSI domain

3.3. R2D Protocol Description

This section provides technical specifications (using UML language) of the R2D protocol, but before going into technical details, it is better to recall some general concepts behind R2D design. The main objective of the current version of R2D is to define a standard protocol for importing data from an EHR. Beyond this objective there are also secondary objectives that it is good to explain as they allow to understand the reason behind some choices taken during the protocol design:

- be compliant to FHIR API and data model: FHIR is a well accepted standard supported by HL7 its importance and adoption is increasing constantly. Also UE Commission in an official document [EUCBEHR ANNEX] recommends the adoption of FHIR for cross border health data exchange.
- Simplify adoption by EHR providers: R2D will be implemented and deployed by EHR providers, so one important issue to take into account is to simplify the adoption process to them. If R2D had been defined only in FHIR it could have been an impediment to its adoption, especially for those EHR providers that have already supported the effort for adopting NCP / eHDSI. Thus the decision to define R2D also as an extension of eHDSI, defining a more limited version of the R2D (eHDSI API do not offer the expression capabilities provided by the FHIR API) but easier to be adopted by EHR providers already compliant to NCP / eHDSI.

In practical terms it means that R2D (as said in the previous chapter) will be defined in two versions providing the same conceptual operations. In order to improve the comprehension of the operations provided by the two versions, a conceptual version of the R2D will be first introduced. The following part of this section is structured in three subsections, each focused on a specific topic:

- Conceptual R2D: conceptual descriptions of the R2D protocol. It defines the basic operations that a protocol for the exchange of medical data should contain, without any explicit reference to a concrete realization.
- R2D over FHIR: defines how the conceptual R2D protocol is realized as a set of operations compliant to the FHIR API.
- R2D over eHDSI: defines how the conceptual R2D protocol is realized as a set of operations compliant to the current eHDSI API.

3.3.1. Conceptual R2D

The Conceptual R2D defines the basic operations that a protocol for the exchange of medical data should support. In accordance with the InteropEHRate objectives of Year 1, the protocol defines only operations for reading medical data from a remote EHR source.

Although R2D is focused on exchange of medical data, it requires some security operations to authenticate a citizen (see report [D3.1] for further details). These security operations are the only operations that does not exchange medical data, and must be considered as preliminary operations invoked to enable a secure exchange of medical data. Authenticating to the InteropEHRate platform, is the only way to gain access to the core R2D functionalities for exchange of medical data. R2D has two main states:

- AUTHENTICATED: is the state that allows the use of the operations for the exchange of medical data.
- NOT_AUTHENTICATED: is the state that does not allow the use of the operations for the exchange of medical data, but allows only the use of the operations for the authentication of a citizen.

Once a citizen is authenticated and the protocol is in the AUTHENTICATED state, all the R2D operations for the exchange of medical data can be used, without any restrictions and without the need of following restrictions in the sequence of operations (indeed, a part of the security operations that changes the state of the communication, the rest of the R2D protocol is just a restful a service).

The following figure shows an UML class diagrams representing the main interface of R2D named R2DI and implemented by the HTTP service providing access to the underlying EHR:

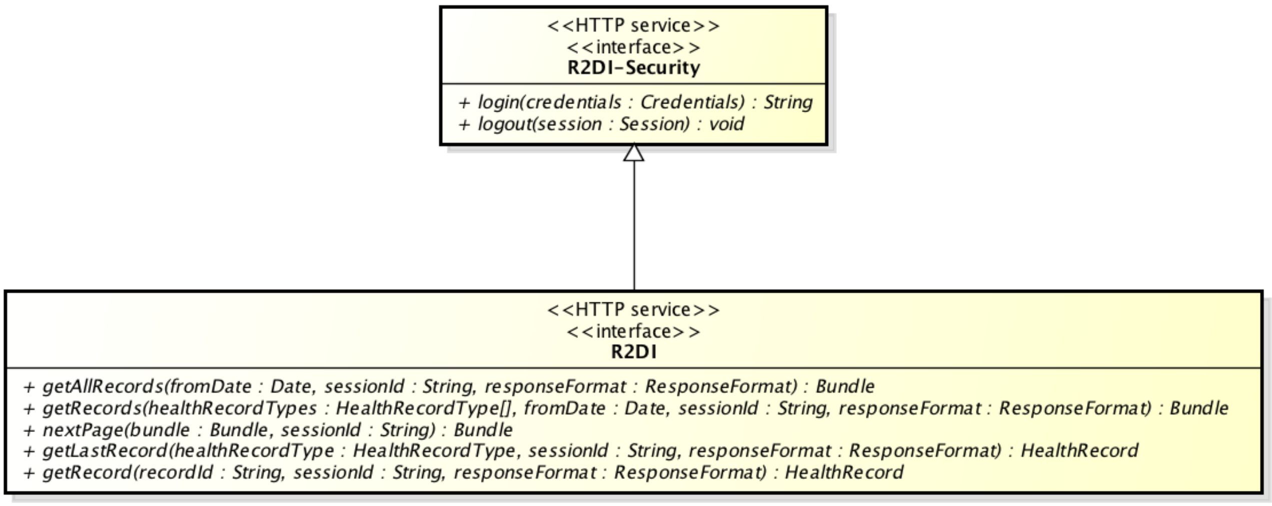
<<HTTP service>>
<<interface>>
R2DI-Security
+ login(credentials : Credentials) : String
+ logout(session : Session) : void
<<HTTP service>>
<<interface>>
R2DI
+ getAllRecords(fromDate : Date, sessionId : String, responseFormat : ResponseFormat) : Bundle
+ getRecords(healthRecordTypes : HealthRecordType[], fromDate : Date, sessionId : String, responseFormat : ResponseFormat) : Bundle
+ nextPage(bundle : Bundle, sessionId : String) : Bundle
+ getLastRecord(healthRecordType : HealthRecordType, sessionId : String, responseFormat : ResponseFormat) : HealthRecord
+ getRecord(recordId : String, sessionId : String, responseFormat : ResponseFormat) : HealthRecord

Figure 13 - R2D interface operations

Before going into detailed description of R2DI interface, it is preferable to have an overview of the reference (conceptual) data model on which the R2DI operations depends on (Figure 14).

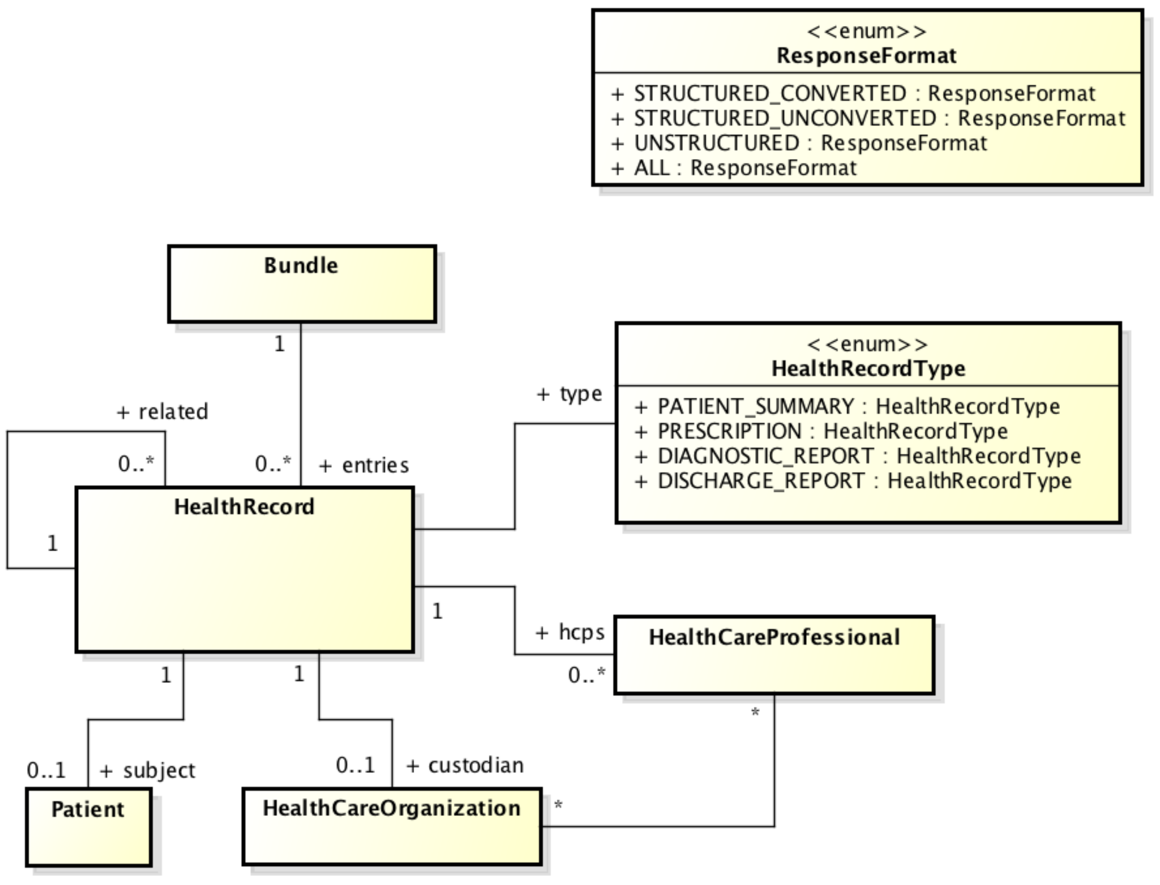
<<enum>>
ResponseFormat
+ STRUCTURED_CONVERTED : ResponseFormat
+ STRUCTURED_UNCONVERTED : ResponseFormat
+ UNSTRUCTURED : ResponseFormat
+ ALL : ResponseFormat
Bundle
1
+ related
0..*
0..*
+ entries
1
HealthRecord
+ type
<<enum>>
HealthRecordType
+ PATIENT_SUMMARY : HealthRecordType
+ PRESCRIPTION : HealthRecordType
+ DIAGNOSTIC_REPORT : HealthRecordType
+ DISCHARGE_REPORT : HealthRecordType
1
+ hcps
0..*
HealthCareProfessional
*
1
1
0..1
+ subject
Patient
0..1
+ custodian
HealthCareOrganization
*

Figure 14 - Conceptual data model

In the following table, a brief description of each class of the data model:

Name	Description
HealthRecord	It represents any health data that can be exchanged.
HealthRecordType	Enumeration for the classification of some specific kind of health data. Values of HealthRecordType have been defined in order to match baseline of EU in Cross Border Health Data Exchange recommendations. Its values are: ● PATIENT_SUMMARY ● PRESCRIPTION ● DIAGNOSTIC_REPORT (including Laboratory Results and Medical Imaging and Reports) ● DISCHARGE_REPORT It will likely be extended in the future to support any health data that may also be represented using the FHIR standard.
Patient	It represents an individual receiving care or other health-related services.
HealthCareProfessional	It represents an individual providing care or other health-related services to a patient.

HealthCareOrganization	It represents any organization offering health-related services.
Bundle	It is a container for sets of instances of HealthRecord class.
ResponseFormat	It represents the format required by the client S-EHR of health data to be returned: ● STRUCTURED_CONVERTED: returned data are represented only in the common format (FHIR based or CDA based depending if the FHIR realization or the eHDSI realization is chosen) and using only semantic codes prescribed by the InteropEHRate profile [D2.7] (possibly obtained from the conversion of the codes(if any) in the original source data or by means of a semi automatic information extraction process). Semantic codes belonging to the InteropEHRate profile will allow the client S-EHR to show the semantically annotated content in the natural language of the user. ● STRUCTURED_UNCONVERTED: returned data are represented in FHIR (without any specific restriction), using the same semantic codes (if any) as the original source data (no conversion of semantic codes is performed). ● UNSTRUCTURED: returned data are represented in a human readable document format (e.g. PDF). No information extraction or conversion of semantic code is performed. The exact content of the returned document is not defined by the R2D protocol. ● ALL: data are returned both in the same format returned by UNSTRUCTURED value and in the format STRUCTURED_CONVERTED (if the server is able to convert), or STRUCTURED_UNCONVERTED (if the server is not able to convert the semantic codes, but is able to transform the content in FHIR format). If the server is not able to perform any data transformations, only UNSTRUCTURED values are returned.

The main requirement that drove the design of the R2DI interface is to support incremental download. It is very likely that the first time the user will perform the import, a large amount of data will be transferred, so the R2D protocol has been designed in order to well support such circumstances. All the R2D methods that performs a search and whose results may be very large, will firstly return only a limited portion of this data (page), providing information about the total size of the results and the number of pages needed to download them all. The method nextPage() allows to access (incrementally) all the pages in which a search result has been segmented, under the complete control of the client, that is able to interrupt fetching of the results in whatever moment.

The following tables provide a complete description of all the operations of the R2DI.

Operation getAllRecords

Name	getAllRecords
Description	This method allows the client to retrieve all kinds of health data of a citizen starting from a certain date.
Arguments	● Date fromDate: a date indicating the day after which the requested health data must have been produced. ● String sessionId: a valid session token, representing the user who successfully executed the login. This sessionId is obtained directly from the platform after successful execution of login method. ● ResponseFormat responseFormat: one of the predefined ResponseFormat enumeration values identifying the output format of the requested health data.
Return Value	An instance of Bundle containing the first page of the returned data (next pages can be accessed using methods nextPage described below).
Exceptions	● Security exceptions related to the validation of the session. ● Network exceptions related to failure during remote communication.
Preconditions	● The citizen has successfully executed the authentication to the InteropEHRate infrastructure. ● The session is still valid.

Operation getRecords

Name	getRecords
Description	This method allows the client to retrieve some kind of health data (of a citizen) starting from a certain date. The kind of health data to be retrieved are passed passed as argument.
Arguments	● HealthRecordType[] healtRecordTypes: an array of predefined values containing the types of health data requested by the client. ● Date fromDate: a date indicating the day after which the requested health data must have been produced. ● String sessionId: a valid session token, representing the user who successfully executed the login. This sessionId is obtained directly from the platform after successful execution of login method. ● ResponseFormat responseFormat: one of the predefined ResponseFormat enumeration's value identifying the output format of the requested health data.
Return Value	An instance of Bundle containing the first page of the returned data (next pages can be accessed using methods nextPage described below).

Exceptions	● Security exceptions related to the validation of the session. ● Network exceptions related to failure during remote communication.
Preconditions	● The citizen has successfully executed the authentication to the InteropEHRate infrastructure. ● The session is still valid.

Operation nextPage

Name	nextPage
Description	This method allows the client to obtain the next page of a bundle instantiated in a previous search (methods getAllRecords() and / or getRecords()).
Arguments	● Bundle bundle: the bundle whose next page is requested. ● String sessionId: a valid session token, representing the user who successfully executed the login. This sessionId is obtained directly from the platform after successful execution of login method.
Return Value	An instance of Bundle containing the next page (in reference to the bundle provided as argument) of health data of the citizen. The next page is obtained using the same search criteria used in the search method that originated the first page of the bundle.
Exceptions	● Security exceptions related to the validation of the session. ● Network exceptions related to failure during remote communication.
Preconditions	● The citizen has successfully executed the authentication to the InteropEHRate infrastructure. ● The session is still valid. ● The provided bundle is valid and has a next page

Operation getLastResource

Name	getLastRecord
Description	This method allows the client to obtain the most recent instance of a certain type of health data (of a citizen) provided by the client itself as an argument.
Arguments	● HealthRecordType healtRecordType: an instance of one of the predefined values of HealthRecordType indicating the type of health data requested by the client. ● String sessionId: a valid session token, representing the user who successfully executed the login. This sessionId is obtained directly from the platform after successful execution of login method. ● ResponseFormat responseFormat: one of the predefined ResponseFormat enumeration values identifying the output format of the requested health data.

Return Value	An instance of HealthRecord containing the most recent instance of a specific health data type.
Exceptions	● Security exceptions related to the validation of the session. ● Network exceptions related to failure during remote communication.
Preconditions	● The citizen has successfully executed the authentication to the InteropEHRate infrastructure. ● The session is still valid.

Operation getRecord

Name	getRecord
Description	This method allows the client to obtain a specific instance of health data identified by its unique id.
Arguments	● String recordId: a valid id of a health data. ● String sessionId: a valid session token, representing the user who successfully executed the login. This sessionId is obtained directly from the platform after successful execution of login method. ● ResponseFormat responseFormat: one of the predefined ResponseFormat enumeration values identifying the output format of the requested health data.
Return Value	An instance of Bundle containing the first page of the returned data (next pages can be accessed using methods nextPage described below).
Exceptions	● Security exceptions related to the validation of the session. ● Network exceptions related to failure during remote communication. ● Access exception related to the ownership of data (only health data of the authenticated citizen can be accessed).
Preconditions	● The citizen has successfully executed the authentication to the InteropEHRate infrastructure. ● The session is still valid.

3.3.2. Involved Applications (A7, ENG)

This section provides a brief description of the two applications involved in the use of R2D, the S-EHR app and the National EHR.

3.3.2.1. *S-EHR Application*

S-EHR Application is also described in **Section 2.3.2.1** of the current document.

3.3.2.2. *Extended NCP*

As described in sections 3.1 and 3.2, the main objective of the eHealth Network and the CEF Programme were to foster interoperability between European eHealth systems in order to allow European citizens to access to health care centres and be treated (in case of emergency or not) all over Europe. eHDSI project defined the EU software architecture to support interoperability between European eHealth systems, introducing the NCP, a web service acting as a proxy for a National EHR and providing a common interface and a common view of whichever European EHR belonging to the eHDSI federation. The Extended NCP described in this paragraph refers to an extension of the base NCP defined in eHDSI, the extensions regards two specific points:

- Authentication policy: eHDSI does not consider the citizen as an active actor, requests to an NCP can be submitted only by another NCP and not by a software acting on behalf of a citizen. Differently the Extended NCP allows a citizen to authenticate (and access) to the NCP of his country.
- Adoption of R2D protocol: the Extended NCP adopts R2D as standard for the exchange of health data.

3.3.3. R2D Interface Interactions

The following UML sequence diagram shows how the R2D API, described in previous section, may be used to retrieve health data of a citizen. As already said, the R2D is not a rigid protocol and the same results may be obtained using different sequences of methods invocation; the objective of this sequence diagram is to show one of the possible sequences of invocations in order to perform a search and browse all pages of data (in which total results have been segmented).

The sequence diagram shows the usage of the following four methods: getLastRecord(), getRecords(), getRecord(), nextPage(), in the diagram, the invocation of these methods are highlighted in red. The actors of this sequence diagram are the following:

- Citizen: a generic citizen interested in accessing his health data. The citizen is an active actor, he is the one that starts the import of data and that defines what kind of data must be imported.
- S-EHR App: the app used by the citizen to send requests to the NCP Node. The app represents a client of the R2D protocol.
- NCP Node: A National Contact Point as described in the eHDSI platform, but also implementing the R2D protocol (represents the server side of the protocol).

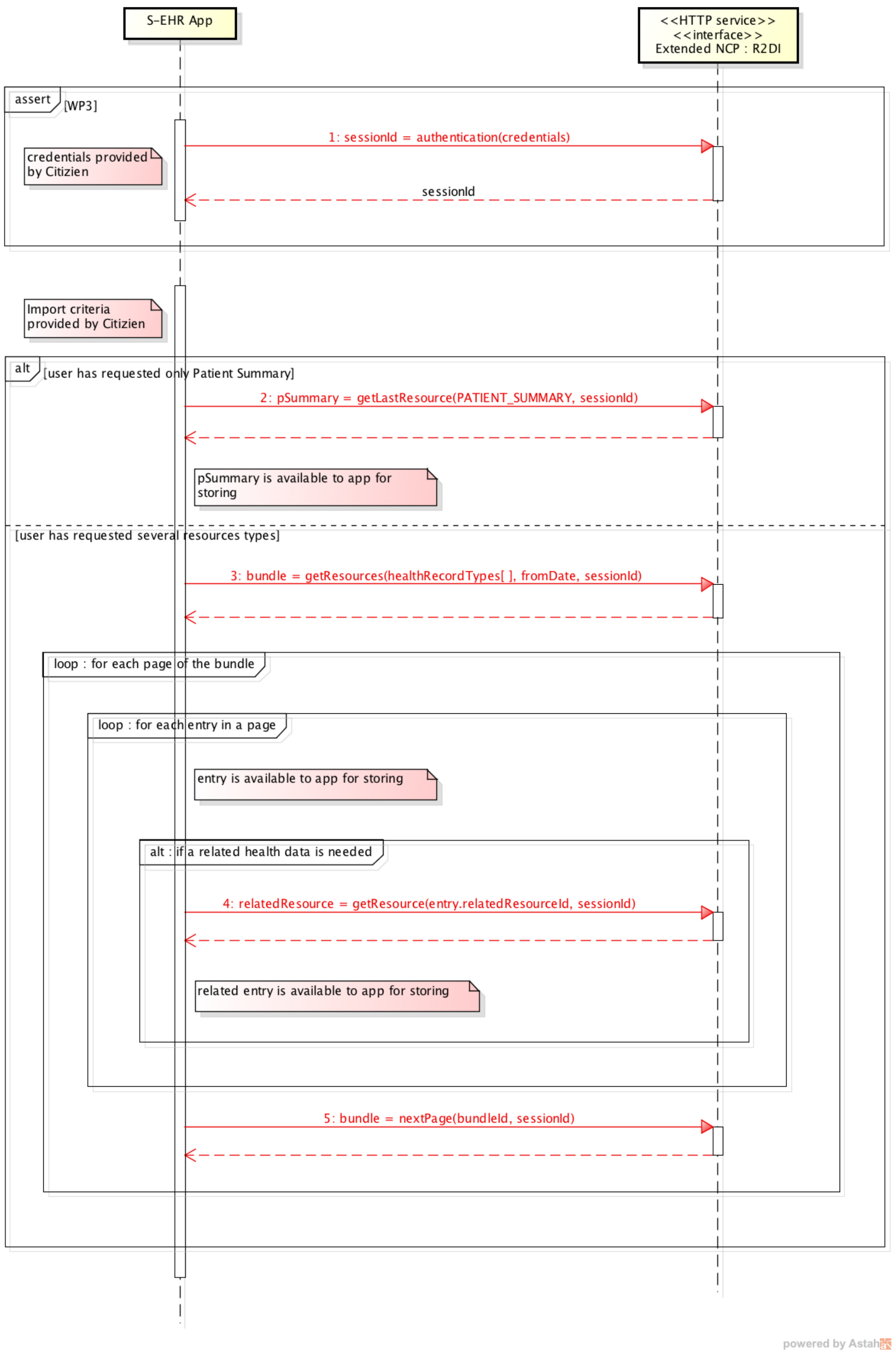
S-EHR App
<<HTTP service>>
<<interface>>
Extended NCP : R2DI
assert
[WP3]
credentials provided by Citizien
1: sessionId = authentication(credentials)
sessionId
Import criteria provided by Citizien
alt
[user has requested only Patient Summary]
2: pSummary = getLastResource(PATIENT_SUMMARY, sessionId)
pSummary is available to app for storing
[user has requested several resources types]
3: bundle = getResources(healthRecordTypes[ ], fromDate, sessionId)
loop : for each page of the bundle
loop : for each entry in a page
entry is available to app for storing
alt : if a related health data is needed
4: relatedResource = getResource(entry.relatedResourceId, sessionId)
related entry is available to app for storing
5: bundle = nextPage(bundleId, sessionId)
powered by Astah

Figure 15 - R2D protocol interactions

Following a detailed description of the sequence diagram:

- **Step 1 - authentication**: this is a preliminary step, that moves the R2D state from NOT_AUTHENTICATED to AUTHENTICATED and allows the app citizen to gain access to all the services of the R2D protocol. A detailed description of this specific interaction is provided in the deliverables about security of work package 3.
- **Step 2 - import**: this is the step started by the citizen that triggers the import of health data from the remote repository. This step is composed by the following sub steps and its execution foresees the provisioning of some parameters by the citizen, in order to drive the data download. The parameters are the following: i) healthRecordTypes[]: an array of predefined values containing the types of health data requested by the client; ii) fromDate: a date indicating the day after which the requested health data must have been produced.
    - **Step 2.1 - getLastResource**: optional step executed only if the citizen has requested only to download his / her patient summary. In this case the client invokes the protocol method named getLastResource(), providing as input parameter the enum value PATIENT_SUMMARY. The method returns to the client the most recent instance of HealthRecord of type HealthRecordType.PATIENT_SUMMARY. The patient summary is now available to the client for storing.
    - **Step 2.2 - getAllRecords**: optional step executed only if the citizen has requested more than one type of HealthRecord. In this case the client invokes the protocol method named getRecords(), passing as input parameters the array of types provided by the citizen and the reference date. This method returns an instance of Bundle containing only the first of the overall pages that compose the entire result.
    - **Step 2.3 - getRecord**: at this point, the client has obtained the first page of the overall results and starts looping the items of the current page (entries of the bundle) processing each one of them. If during this processing, the client needs to download a health data related to the one under processing, the client will invoke the protocol method named getRecord(), passing the id of the related resource as input. After this invocation, the client is able to process both the primary health data and its related resource.
    - **Step 2.4 - nextPage**: when the client has fetched all the entries in the bundle, it requests the next bundle by invoking the method nextPage() and providing as input the current bundle. In case there is another page to be read, the client gets back the next bundle and continues processing it, otherwise it gets back a null value indicating that all results have been fetched.

So far, R2D has been described only by a conceptual point of view, while the following sections provide concrete specifications of R2D realized using FHIR and eHDSI.

3.3.4. R2D over FHIR

This section describes a specific realization of the R2D based on the FHIR API query language [FHIR API SPEC], operations of conceptual R2D (described in previous section) are mapped to specific RESTFul FHIR invocations. In particular for each conceptual operation will be specified:

- the HTTP method to invoke;
- the type of FHIR resource involved;
- The allowed parameters and, if needed, the constraints defined over parameter values.

Furthermore, this section shows how the R2D protocol may be offered by an EHR compliant to the FHIR standard.

Operation getAllRecords(fromDate : Date, sessionId : String, responseFormat : ResponseFormat) : Bundle
Realization of method getAllRecords() follows the same rules of method getRecords() when passed all values of the enumeration HealthRecordType. For this reason both methods are described in the following paragraph.

Operation getRecords(healthRecordTypes : HealthRecordType[], fromDate : Date, sessionId : String, responseFormat : ResponseFormat) : Bundle
The method getRecords() allows a client to retrieve different kinds of health data related to the same patient, the client specifies what are the kind of health data to be retrieved by using the input parameter named healthRecordTypes : HealthRecordType[] (every value passed as input in the array of HealthRecordType corresponds to retrieve a specific type of FHIR Resource). Although the FHIR specifications allow executing queries over different kind of Resources (simulating the SQL UNION operator), these kind of queries are strongly limited by restrictions to the applicable set of search parameters that must be common to all involved Resource types, reducing the capabilities provided by single-resource query. Moreover, these restrictions do not allow to execute correctly the queries needed by the protocol.

So, for each value passed in the input array of HealthRecordType a specific FHIR query must be executed, and the overall result of this method is the logical union of all results obtained by the individual query performed. The following paragraphs describes the query to be executed (i.e. by invoking a specific FHIR service) for each of the possible values. Note that for some combination of values, several queries have to be executed (for instance, two queries have to be executed when the parameter HealthRecordTypes contains the value PATIENT_SUMARY and the parameter ResponseFormat contains the value ALL)

If healthRecordTypes contains the value PATIENT_SUMMARY and ResponseFormat is one of STRUCTURED_CONVERTED, STRUCTURED_UNCONVERTED, ALL, then the following FHIR restful service have to be invoked:

- HTTP Method: GET
- Header Parameters:
  - Session token obtained by authentication method
  - Whatever parameter allowed from [FHIR API SPEC]
- FHIR Resource to be queried: Composition
- Search parameters:

Search Parameters			
Name	**Type**	**Mandatory**	**Value**
type	token	YES	“http://loinc.org|60591-5”
subject	reference	OPTIONAL	Optional parameter. If provided, the only allowed valued is the reference to the patient who

			performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
_include	string	YES	“*”

Examples:

1. GET <base url>/Composition?type=”http://loinc.org|60591-5”&_include=*
2. GET <base url>/Composition?subject=<REF TO AUTH USER>&type=”http://loinc.org|60591-5”&_include=*

If healthRecordTypes contains the value PATIENT_SUMMARY and ResponseFormat is one of UNSTRUCTURED, ALL, then the following FHIR restful service has to be invoked:

- HTTP Method: GET
- Header Parameters:
  - Session token obtained by authentication method
  - Whatever parameter allowed from [FHIR API SPEC]
- FHIR Resource to be queried: DocumentReference
- Search parameters:

Search Parameters			
Name	**Type**	**Mandatory**	**Value**
subject	reference	YES	Optional parameter. If provided, the only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
type	token	YES	“http://loinc.org|60591-5”
date	date	OPTIONAL	Optional parameter that has to be used if the value of input parameter fromDate is not null. In this case it must be used chained to the "ge" comparative operator in order to retrieve all MedicationRequest instances that have been authored starting from fromDate.

Examples:

1. GET <base url>/DocumentReference?type=”http://loinc.org|60591-5”
2. GET <base url>/DocumentReference?subject=<REF TO AUTH USER>&type=”http://loinc.org|60591-5”

Otherwise, if ResponseFormat passed by client is ALL, then both queries described above MUST be executed. In this case, the search is spread over both FHIR resource types DocumentReference (unstructured) and Composition (structured).

If healthRecordTypes contains the value PRESCRIPTION, and ResponseFormat is one of STRUCTURED_CONVERTED, STRUCTURED_UNCONVERTED, ALL, then the corresponding FHIR restful service to be invoked is the following one:

- HTTP Method: GET
- Header Parameters:
    - Session token obtained by authentication method
    - Whatever parameter allowed from [FHIR API SPEC]
- FHIR Resource to be queried: MedicationRequest
- Allowed search parameters:

Search Parameters			
Name	**Type**	**Mandatory**	**Value**
patient	reference	OPTIONAL	Optional parameter. If provided, the only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
authoredon	date	OPTIONAL	Optional parameter that has to be used if the value of input parameter fromDate is not null. In this case it must be used chained to the "ge" comparative operator in order to retrieve all MedicationRequest instances that have been authored starting from fromDate.

Examples:

1. GET <base url>/MedicationRequest?patient=<REF TO AUTH USER>
2. GET <base url>/MedicationRequest?patient=<REF TO AUTH USER>&authoredon=ge2019-03-14
3. GET <base url>/MedicationRequest?authoredon=ge2019-03-14

If healthRecordTypes contains the value PRESCRIPTION, and ResponseFormat is one of UNSTRUCTURED, ALL, then the corresponding FHIR restful service to be invoked is the following one:

- HTTP Method: GET
- Header Parameters:
    - Session token obtained by authentication method
    - Whatever parameter allowed from [FHIR API SPEC]
- FHIR Resource to be queried: DocumentReference
- Allowed search parameters:

Search Parameters			
Name	**Type**	**Mandatory**	**Value**
subject	reference	YES	Optional parameter. If provided, the only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
type	token	YES	"http://loinc.org|57828-6"
date	date	OPTIONAL	Optional parameter that has to be used if the value of input parameter fromDate is not null. In this case it must be used chained to the "ge" comparative operator in order to retrieve all MedicationRequest instances that have been authored starting from fromDate.

Examples:

1. GET <base url>/DocumentReference?patient=<REF TO AUTH USER>&type="http://loinc.org|57828-6"
2. GET <base url>/DocumentReference?date=ge2019-03-14&type="http://loinc.org|57828-6"

If ResponseFormat passed by client is ALL, then both queries described above MUST be executed. In this case, the search is spread over both FHIR resource types DocumentReference (unstructured) and MedicationRequest (structured).

Otherwise, if healthRecordTypes contains the value DIAGNOSTIC_REPORT, ResponseFormat is one of STRUCTURED_CONVERTED, STRUCTURED_UNCONVERTED, ALL, then the corresponding FHIR restful service to be invoked is the following one:

- HTTP Method: GET
- Header Parameters:
  - Session token obtained by authentication method
  - Whatever parameter allowed from [FHIR API SPEC]
- FHIR Resource to be queried: DiagnosticReport
- Allowed search parameters:

Search Parameters			
Name	**Type**	**Mandatory**	**Value**
subject	reference	OPTIONAL	Optional parameter. If provided, the only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly

			assumed to be the patient who performed the authentication.
date	date	OPTIONAL	Optional parameter that has to be used if the value of input parameter fromDate is not null. In this case it must be used chained to the "ge" comparative operator in order to retrieve all DiagnosticReport instances that have been created starting from fromDate.

Examples:

1. GET <base url>/DiagnosticReport?subject=<REF TO AUTH USER>
2. GET <base url>/DiagnosticReport?subject=<REF TO AUTH USER>&date=ge2012-01-01
3. GET <base url>/DiagnosticReport?date=ge2012-01-01

If healthRecordTypes contains the value DIAGNOSTIC_REPORT, ResponseFormat is one of UNSTRUCTURED, ALL, then the corresponding FHIR restful service to be invoked is the following one:

- HTTP Method: GET
- Header Parameters:
  - Session token obtained by authentication method
  - Whatever parameter allowed from [FHIR API SPEC]
- FHIR Resource to be queried: DocumentReference
- Allowed search parameters:

Search Parameters			
Name	**Type**	**Mandatory**	**Value**
subject	reference	OPTIONAL	Optional parameter. If provided, the only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
type	token	YES	“http://loinc.org|11502-2”
date	date	OPTIONAL	Optional parameter that has to be used if the value of input parameter fromDate is not null. In this case it must be used chained to the "ge" comparative operator in order to retrieve all DiagnosticReport instances that have been created starting from fromDate.

Examples:

1. GET <base url>/DocumentReference?subject=<REF TO AUTH USER>&type="http://loinc.org|11502-2"
2. GET <base url>/DocumentReference?type="http://loinc.org|11502-2"&date=ge2012-01-01

If ResponseFormat passed by client is ALL, then both queries described above MUST be executed. In this case, the search is spread over both FHIR resource types DocumentReference (unstructured) and DiagnosticReport (structured).

Finally, for what concerns the requests of health data of type DISCHARGE_REPORT, it will be specified in the next version of the current deliverable, after release of deliverable [D2.7].

Operation getLastRecord(healthRecordType : HealthRecordType, sessionId : String, responseFormat : ResponseFormat) : HealthRecord
This method returns the most recent version of a specific kind of health data of a citizen. The kind of health data requested is specified by the client with the argument healthRecordType.

If healthRecordTypes contains the value PATIENT_SUMMARY and ResponseFormat is one of STRUCTURED_CONVERTED, STRUCTURED_UNCONVERTED, ALL, then the corresponding FHIR restful service to be invoked is the following one:

- HTTP Method: GET
- Header Parameters:
  - Session token obtained by authentication method
  - Whatever parameter allowed from [FHIR API SPEC]
- FHIR Resource to be queried: Composition
- Allowed search parameters:

Search Parameters			
Name	**Type**	**Mandatory**	**Value**
Type	token	YES	"http://loinc.org|60591-5"
subject	reference	OPTIONAL	Optional parameter. If provided, the only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
_include	string	YES	"*"
_sort		YES	-date
_count		YES	1

Examples:

1. GET <base url>/Composition?type="http://loinc.org|60591-5"_include=*&_sort=-date&count=1

If healthRecordTypes contains the value PATIENT_SUMMARY and ResponseFormat is one of UNSTRUCTURED, ALL, then the corresponding FHIR restful service to be invoked is the following one:

- HTTP Method: GET
- Header Parameters:
  - Session token obtained by authentication method
  - Whatever parameter allowed from [FHIR API SPEC]
- FHIR Resource to be queried: DocumentReference
- Search parameters:

Search Parameters			
Name	**Type**	**Mandatory**	**Value**
subject	reference	YES	Optional parameter. If provided, the only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
type	token	YES	"http://loinc.org|60591-5"
date	date	OPTIONAL	Optional parameter that has to be used if the value of input parameter fromDate is not null. In this case it must be used chained to the "ge" comparative operator in order to retrieve all MedicationRequest instances that have been authored starting from fromDate.
_sort		YES	-date
_count		YES	1

Examples:

1. GET <base url>/DocumentReference?type="http://loinc.org|60591-5"&_sort=-date&count=1

If ResponseFormat passed by client is ALL, then both queries described above MUST be executed. In this case, the search is spread over both FHIR resource types DocumentReference (unstructured) and Composition (structured).

Otherwise, if healthRecordTypes contains the value PRESCRIPTION and ResponseFormat is one of STRUCTURED_CONVERTED, STRUCTURED_UNCONVERTED, ALL, then the corresponding FHIR restful service to be invoked is the following one:

- HTTP Method: GET

- Header Parameters:
  - Session token obtained by authentication method
  - Whatever parameter allowed from [FHIR API SPEC]
- FHIR Resource to be queried: MedicationRequest
- Allowed search parameters:

Search Parameters			
Name	**Type**	**Mandatory**	**Value**
patient	reference	OPTIONAL	Optional parameter. If provided, the only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
_sort		YES	-authoredon
_count		YES	1

Examples:

1. GET <base url>/MedicationRequest?patient=<REF TO AUTH USER>&_sort=-authoredon&_count=1
2. GET <base url>/MedicationRequest?&_sort=-authoredon&_count=1

If healthRecordTypes contains the value PRESCRIPTION, and ResponseFormat is one of UNSTRUCTURED, ALL, then the corresponding FHIR restful service to be invoked is the following one:

- HTTP Method: GET
- Header Parameters:
  - Session token obtained by authentication method
  - Whatever parameter allowed from [FHIR API SPEC]
- FHIR Resource to be queried: DocumentReference
- Allowed search parameters:

Search Parameters			
Name	**Type**	**Mandatory**	**Value**
subject	reference	YES	Optional parameter. If provided, the only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
type	token	YES	“http://loinc.org|57828-6”

date	date	OPTIONAL	Optional parameter that has to be used if the value of input parameter fromDate is not null. In this case it must be used chained to the "ge" comparative operator in order to retrieve all MedicationRequest instances that have been authored starting from fromDate.
_sort		YES	-date
_count		YES	1

Examples:

1. GET <base url>/DocumentReference?type=”http://loinc.org|57828-6”&_sort=-date&_count=1
2. GET <base url>/DocumentReference?patient=<REF TO AUTH USER>&type=”http://loinc.org|57828-6”&_sort=-date&_count=1

If ResponseFormat passed by client is ALL, then both queries described above MUST be executed. In this case, the search is spread over both FHIR resource types DocumentReference (unstructured) and MedicationRequest (structured).

Otherwise, if healthRecordTypes contains the value DIAGNOSTIC_REPORT and ResponseFormat is one of STRUCTURED_CONVERTED, STRUCTURED_UNCONVERTED, ALL, then the corresponding FHIR restful service to be invoked is the following one:

- HTTP Method: GET
- Header Parameters:
  - Session token obtained by authentication method
  - Whatever parameter allowed from [FHIR API SPEC]
- FHIR Resource to be queried: DiagnosticReport
- Allowed search parameters:

Search Parameters			
Name	**Type**	**Mandatory**	**Value**
subject	reference	OPTIONAL	Optional parameter. If provided, the only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
_sort		YES	-date
_count		YES	1

Examples:

1. GET <base url>/DiagnosticReport?&_sort=-date&_count=1
2. GET <base url>/DiagnosticReport?subject=<REF TO AUTH USER>&_sort=-date&_count=1

If healthRecordTypes contains the value DIAGNOSTIC_REPORT, ResponseFormat is one of UNSTRUCTURED, ALL, then the corresponding FHIR restful service to be invoked is the following one:

- HTTP Method: GET
- Header Parameters:
    - Session token obtained by authentication method
    - Whatever parameter allowed from [FHIR API SPEC]
- FHIR Resource to be queried: DocumentReference
- Allowed search parameters:

Search Parameters			
Name	**Type**	**Mandatory**	**Value**
subject	reference	OPTIONAL	Optional parameter. If provided, the only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
type	token	YES	“http://loinc.org|11502-2”
date	date	OPTIONAL	Optional parameter that has to be used if the value of input parameter fromDate is not null. In this case it must be used chained to the "ge" comparative operator in order to retrieve all DiagnosticReport instances that have been created starting from fromDate.
_sort		YES	-date
_count		YES	1

Examples:

1. GET <base url>/DocumentReference?subject=<REF TO AUTH USER>&type=”http://loinc.org|11502-2”&date=ge2012-01-01&_sort=-date&_count=1
2. GET <base url>/DocumentReference?type=”http://loinc.org|11502-2”&date=ge2012-01-01&_sort=-date&_count=1

If ResponseFormat passed by client is ALL, then both queries described above MUST be executed. In this case, the search is spread over both FHIR resource types DocumentReference (unstructured) and DiagnosticReport (structured).

Finally, for what concerns the requests of health data of type DISCHARGE_REPORT, it will be specified in the next version of the current deliverable, after release of deliverable [D2.7].

Operation getRecord(recordId : String, sessionId : String) : HealthRecord
The realization of this method in FHIR, relies on the direct access to a specific resource instance, by using an absolute URI retrieved by fetching another instance of health data (belonging to the authenticated citizen). In R2D protocol, ids of resources can be discovered only by browsing an instance of health data returned by search methods.
Direct access to health data by id is authorized only if the referenced data belongs to the authenticated user.

Operation nextPage(bundle : Bundle, sessionId : String) : Bundle
The realization of this method in FHIR, relies on the built-in pagination functionalities defined directly by FHIR specifications of Bundles resource. A Bundle contains the attribute named next contained in the main attribute (of type array) called link, the next attribute contains the entire URI of the service to be invoked in order to retrieve the next page of data.

3.3.5. R2D over extended eHDSI

This section describes a partial realization of the R2D can be based on the eHDSI API specifications [NCPeH ARCH SPECS]. Operations of conceptual R2D (previously described in this deliverable) are mapped to specific eHDSI SOAP invocations. In particular for each conceptual operation will be specified:

- the eHDSI interface(s) involved;
- the methods of the interfaces to be invoked;
- The arguments provided to methods and, if needed, the constraints defined over arguments values.

Current eHDSI protocol states [EHDSI SYS SPECS] that in order to invoke business methods for retrieving health data of a citizen, a preliminary phase named citizen identification must be executed by an HCP authenticated to the National eHealth system. While in this document it is described an Extended eHDSI where in addition to the current defined protocol, a citizen is also allowed to authenticate to its country NCP in order to request his health data avoiding the preliminary citizen identification phase made by an HCP.

Furthermore, current eHDSI technical specifications do not cover the whole kind of health data defined in [EUCBEHR REC] by the EU Commission, they cover only the request of the Patient Summary and of the ePrescription / eDispensation. For this reason, some of the operations defined by the R2D cannot be translated (for the moment) into eHDSI service invocations.

Operation getAllRecords(fromDate : Date, sessionId : String, responseFormat : ResponseFormat) : Bundle
Realization of method getAllRecords() follow the same rules of method getRecords() when passed all values of the enumeration HealthRecordType. For this reason both methods are described in the following paragraph.

Operation getRecords(healthRecordTypes : HealthRecordType[], fromDate : Date, sessionId : String, responseFormat : ResponseFormat) : Bundle

The method getRecords() allows a client to retrieve different kinds of health data related to the same patient. The client specifies what are the kind of health data to be retrieved by using the input parameter named healthRecordTypes : HealthRecordType[] (every value passed as input in the array of HealthRecordType corresponds to retrieve a specific type of FHIR Resource). For each value passed in the input array of HealthRecordType a specific eHDSI service must be executed, and the overall result of this method is the logical union of all results obtained by the individual query performed. Due to the current eHDSI limitations this method can be translated only for values: PATIENT_SUMMARY and PRESCRIPTION.

If healthRecordTypes contains the value PATIENT_SUMMARY, then the corresponding eHDSI service to be invoked is the following one:

- eHDSI Service Interface: PatientService
- Service Interface method: list()
- Header Parameters:
    - Session token obtained by authentication method
    - epSOS HCP Identity Assertion
- Operation arguments: an instance of ListPatientRequest containing:

Search Parameters		
Name	**Mandatory**	**Value**
Patient Identifier	YES	The only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
epSOS CDA template qualifier	OPTIONAL	Code defining the output format of requested health data. It may be an epSOS CDA common template or document coded accorded to the source of health data. The value of this argument is related to the value of the argument responseFormat and must be mapped as defined in the following list: ● STRUCTURED_CONVERTED: “urn:epSOS:ps:ps:2010” ● STRUCTURED_UNCONVERTED: “urn:epSOS:ps:ps:2010” ● UNSTRUCTURED: “urn:ihe:iti:xds-sd:pdf:2008” ● ALL: null

If healthRecordTypes contains the value PRESCRIPTION, then the corresponding eHDSI service to be invoked is the following one:

- eHDSI Service Interface: OrderService
- Service Interface method: list()
- Header Parameters:
    - Session token obtained by authentication method
    - epSOS HCP Identity Assertion
- Operation arguments: an instance of ListOrderRequest containing:

Search Parameters		
Name	**Mandatory**	**Value**
Patient Identifier	YES	The only allowed valued is the reference to the patient who performed authentication. If not provided, it IS implicitly assumed to be the patient who performed the authentication.
epSOS CDA template qualifier	OPTIONAL	Code defining the output format of requested health data. It may be an epSOS CDA common template or document coded accorded to the source of health data. The value of this argument is related to the value of the argument responseFormat and must be mapped as defined in the following list: ● STRUCTURED_CONVERTED: “urn:epSOS:ps:ps:2010” ● STRUCTURED_UNCONVERTED: “urn:epSOS:ps:ps:2010” ● UNSTRUCTURED: “urn:ihe:iti:xds-sd:pdf:2008” ● ALL: null

Operation getLastRecord(healthRecordType : HealthRecordType, sessionId : String, responseFormat : ResponseFormat) : HealthRecord
Realization of this method follow the same rules of previous method, with the addition of a final sorting to reorder the set of data returned from the NCP from the most recent to the oldest.

If healthRecordTypes contains the value PATIENT_SUMMARY, then the corresponding set of returned data has to be sorted in descending way using the release date of the Patient Summary as sort criteria.

If healthRecordTypes contains the value PRESCRIPTION, then the corresponding set of returned data has to be sorted in descending way using the release date of the Prescription as sort criteria.

Operation getRecord(recordId : String, sessionId : String, responseFormat : ResponseFormat) : HealthRecord
With the current version of eHDSI there is no explicit method to retrieve referenced health data. It is assumed that every set of health data returned from a NCP contains all referenced health data in the returned bundle.

Operation nextPage(bundle : Bundle, sessionId : String) : Bundle
With the current version of eHDSI there is no explicit reference to pagination of results, so in this version of the protocol data cannot be segmented into pages, they’ll be retrieved always in a single page. This method will always return an empty Bundle.

4. CONCLUSIONS AND NEXT STEPS

The objective of this report was to deliver the initial version of the design and the specification of both the D2D and the R2D protocols. With the implementation of D2D it will become feasible to achieve the exchange of healthcare related data between a healthcare practitioner and a mobile application of a citizen, without the usage of internet connection, whereas with the implementation of R2D it will become feasible to exchange healthcare related data between any EHR or S-EHR Cloud and a mobile application of a citizen, with the usage of internet. In the same notion as the other reports of the InteropEHRate project, this document presents a first draft of the intended content of the two libraries and their further functionality purposes. However, it should be mentioned that other two updated versions of this report are planned to be released. The one is planned to be released on March 2020, whilst the final one is planned to be released on March 2021, both of them including the relevant updates, of both the D2D and the R2D protocols, that will have taken place until then. In more detail, regarding the specification of the D2D protocol it should be mentioned that in the 1st version of the D2D protocol specification [D4.1], the specification was performed based on the Android Bluetooth API since the D2D protocol is currently in a very immature phase, and we would like to perform additional research on the different platforms for identifying the Bluetooth profile that can satisfy all the requirements as specified in D2.1 [D2.1]. In the next version of the D2D protocol specification, the D2D protocol specification will be independent from any API in order for any developer that would like to use this protocol to be able to use the listed operations or develop her own operations, in order to implement the corresponding data exchange between a S-EHR and an HCP application.

REFERENCES

- **[D2.1]** InteropEHRate consortium. *D2.1 : User Requirements for cross-border HR integration - V1*. InteropEHRate project, 2019. www.interopehrate.eu/resources
- **[D2.4]** InteropEHRate consortium. *D2.4 : InteropEHRate Architecture - V1*. InteropEHRate project, 2019. www.interopehrate.eu/resources
- **[D2.7]** InteropEHRate consortium. *D2.7 : FHIR profile for EHR interoperability - V1*. InteropEHRate project, Due on September 2019. www.interopehrate.eu/resources
- **[D3.1]** InteropEHRate consortium. D3.1 : *Specification of S-EHR mobile privacy and security conformance levels - V1*. InteropEHRate project, Due on March 2020. www.interopehrate.eu/resources
- **[D3.3]** InteropEHRate consortium. D3.3 : *Specification of remote and D2D IDM mechanisms for HRs Interoperability - V1*. InteropEHRate project, 2019. www.interopehrate.eu/resources
- **[D4.8]** InteropEHRate consortium. *D4.8 : Specification of protocol and APIs for research health data sharing - V1*, Due on March 2020. www.interopehrate.eu/resources
- **[ANT+]** ANT+, Website: https://www.thisisant.com/consumer/ant-101/what-is-ant
- **[BLUETOOTH]** Bluetooth Core Specification, Website: https://www.bluetooth.com/specifications/bluetooth-core-specification/
- **[ENOCEAN]** EnOcean Self-powered IoT, Website: https://www.enocean.com/en/
- **[NFC]** NFC forum, Website: https://nfc-forum.org/
- **[RFID]** How RFID Works, Website: https://electronics.howstuffworks.com/gadgets/high-tech-gadgets/rfid.htm
- **[ZIGBEE]** Zigbee alliance, Website: https://www.zigbee.org/
- **[WIFIDIRECT]** Wi-Fi Direct, Website: https://www.wi-fi.org/discover-wi-fi/wi-fi-direct
- **[ZWAVE]** Z-wave, Website: https://www.z-wave.com/
- **[SDDB]** Li, Sing, and Jonathan Knudsen. *Beginning J2ME: from novice to professional*. Apress, 2006.
- **[BCS]** Bluetooth Core Specification, https://www.bluetooth.com/specifications/bluetooth-core-specification/
- **[BCSv4.0]** Bluetooth Core Specification version 4.0, https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=229737
- **[SPP]** Serial Port Profile, https://learn.sparkfun.com/tutorials/bluetooth-basics/bluetooth-profiles
- **[RFCOMM]** RFCOMM protocol, https://www.amd.e-technik.uni-rostock.de/ma/gol/lectures/wirlec/bluetooth_info/rfcomm.html
- **[BASEBAND]** Bluetooth Baseband, https://www.amd.e-technik.uni-rostock.de/ma/gol/lectures/wirlec/bluetooth_info/baseband.html
- **[LMP]** Link Manager Protocol, https://www.amd.e-technik.uni-rostock.de/ma/gol/lectures/wirlec/bluetooth_info/lmp.html
- **[L2CAP]** Logical Link Control and Adaptation Protocol, https://www.amd.e-technik.uni-rostock.de/ma/gol/lectures/wirlec/bluetooth_info/l2cap.html
- **[OSI]** The protocol stack, http://www.informit.com/articles/article.aspx?p=27591&seqNum=5
- **[GSM]** GSM ts07.10, https://www.gefos-leica.cz/data/.../877301_leica_ts03_07_10_eql_v1-0-0_en.pdf
- **[SDP]** Service Discovery Protocol Layer, https://www.amd.e-technik.uni-rostock.de/ma/gol/lectures/wirlec/bluetooth_info/sdp.html

- **[ANDROIDBT]** Android Bluetooth Overview, https://developer.android.com/guide/topics/connectivity/bluetooth
- **[EUCBEHR REC]** COMMISSION RECOMMENDATION of 6.2.2019 on a European Electronic Health Record exchange format.
- **[EUCBEHR ANNEX]** ANNEX to the Commission Recommendation of 6.2.2019 on a European Electronic Health Record exchange format.
- **[EHDSI SYS SPECS]** EHDSI System Architecture Specification v2.1.0, 01 June 2017.
- **[NCPeH ARCH SPECS]** NCPeH System Architecture Specification v2.1.0, 01 June 2017.
- **[FHIR API SPEC]** FHIR RESTful API R4 (version 4.0.0), Website: https://www.hl7.org/fhir/http.html
- **[OPENEHR]** https://www.openehr.org/

ANNEX

Users' questionnaire

Communication Requirements

No.	Requirement	Answer to Requirement
CR1	What is the maximum delay in communication that is acceptable to you?	A few seconds
CR2	Would you prefer the data to be exchanged in a distance of a few centimeters between the 2 devices, or in a distance of no more than 10 meters?	There is not any preference

Table 3 - Communication Requirements

Data Requirements

No.	Requirement	Answer to Requirement
DR1	Do you foresee to exchange only textual data? Will you also exchange images and videos?	Both, textual data and images
DR2	For how long would you like the Physician to keep the accessed data after the Citizen leaves?	It depends on the situation
DR3	Would you like the Citizen to update her EHR with the newly derived data (i.e. the data that will emerge from her visit to the Physician)?	Yes

Table 4 - Data Requirements

Security Requirements

No.	Requirement	Answer to Requirement
SR1	Should the data exchange part be reliable (e.g. requirement of confirmation messages)?	Yes
SR2	How important is the authentication of the parties?	Very important

SR3	Is the transferred data confidential?	Yes

Table 4 - Security Requirements

User Interaction Requirements

No.	Requirement	Answer to Requirement
UI1	How many types of parties will be involved? Only Physicians and Citizens?	HCPs and citizens
UI2	Who will initiate the process? The Physician or the Citizen?	The Citizens
UI3	How would the Physician explain to the Citizen the data she wants to access? How would the Citizen understand the Physician's demands?	By providing an initial list of attribute names

Table 5 - User Interaction Requirements