D4.5

Design of libraries for remote and D2D HR exchange - V2

ABSTRACT

This deliverable describes the second version of the design of the libraries offered by the InteropEHRate Framework as a reference implementation of the device-to-device (D2D) and the remote-to-device (R2D) health record exchange protocols. A detailed description is being provided for these libraries, including their interactions with the involved applications, in terms of how the libraries can be used by any S-EHR app and HCP App, regarding the specified D2D and R2D protocols. The current deliverable is intended for developers and manufacturers that are interested in designing and building either mobile or web applications that aim at exploiting and reusing either the D2D or the R2D or both of these two functionalities offered by the InteropEHRate framework, in the context of their applications.

Delivery Date	January 14th, 2021
Work Package	WP4
Task	T4.2
Dissemination Level	Public
Type of Deliverable	Report
Lead partner	UPRC

InteropEHRate project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826106.

This document has been produced in the context of the InteropEHRate Project which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826106. All information provided in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose.

CONTRIBUTORS

	Name	Partner
Contributors	Thanos Kiourtis, Aikaterini Aggeli, Argyro Mavrogiorgou	UPRC
Contributors	Francesco Torelli, Alessio Graziani	ENG
Reviewers	Chrysostomos Symvoulidis	BYTE
Reviewers	Nicu Jalba	SIVECO

LOG TABLE

Version	Date	Change	Author	Partner
0.1	2020-07-23	Provided Initial ToC	Thanos Kiourtis, Argyro Mavrogiorgou	UPRC
0.2	2020-09-02	Updated sections related with D2D - Provided new ToC	Thanos Kiourtis, Argyro Mavrogiorgou	UPRC
0.3	2020-09-28	Updated sections related with R2D	Alessio Graziani	ENG
0.4	2020-10-14	Updated abstract, introduction and conclusions section	Thanos Kiourtis, Aikaterini Aggeli	UPRC
0.5	2020-11-06	Alignment of D2D libraries' operations with current implementation	Thanos Kiourtis, Argyro Mavrogiorgou	UPRC
0.6	2020-11-30	Proof-reading of document, updates on Section 1.4	Thanos Kiourtis	UPRC
0.7	2020-12-21	Sent for internal review	Thanos Kiourtis	UPRC
0.8	2021-01-08	Addressed comments from the internal reviewers	Thanos Kiourtis	UPRC
0.9	2021-01-11	Quality check	Argyro Mavrogiorgou	UPRC
Vfinal	2021-01-14	Final check and submission	Francesco Torelli Laura Pucci	ENG

ACRONYMS

Acronym	Term and definition
D2D	Device-to-Device
R2D	Remote-to-Device
EHR	Electronic Health Record
S-EHR	Smart Electronic Health Record
MD2D	Mobile Device-to-Device
TD2D	Terminal Device-to-Device
MD2DI	Mobile Device-to-Device Interface
TD2DI	Terminal Device-to-Device Interface
M-D2D-E	Mobile Device-to-Device Exchange
M-R2D-E	Mobile Remote-to-Device Exchange
M-R2D-SM	Mobile Remote-to-Device Security
T-D2D-E	Terminal Device-to-Device Exchange
HCO	Healthcare Organization
CEF	Connecting Europe Facility
eHDSI	eHealth Digital Service Infrastructure
NCP	National Contact Point
API	Application Programming Interface
FHIR	Fast Healthcare Interoperability Resources
UML	Unified Modeling Language
SQL	Structured Query Language
R2DI	Remote-to-Device Interface
HCP	Healthcare Practitioner

TABLE OF CONTENT

LIST OF FIGURES

1. INTRODUCTION

1.1. Scope of the document

The main goal of this document is to deliver the second version of the design of the libraries offered by the InteropEHRate Framework as a reference implementation of the device-to-device (D2D) and the remote-to-device (R2D) health record exchange protocols. The current document outlines for both protocols (i.e. D2D and R2D will be realized as libraries enabling their exploitation in various implementation contexts for health data exchange) the interfaces of their libraries describing the Java methods offered by the libraries to the HCP app and to the S-EHR app, which allows to send and receive the messages of the protocols.

In greater detail, this deliverable describes two libraries for the D2D protocol and one library for the R2D protocol. Regarding the D2D protocol, the first library is a Java-based component that can be nested in any Android application (Android Version 4.3 or higher). It offers a set of Java operations for establishing a D2D connection and allowing a mobile app of a Citizen to exchange his/her personal health records using the D2D protocol. The second library is a Java-based component that can be embedded in any Java application (Web or Desktop applications). It offers a set of Java operations enabling the application (used by a HCP) to send and receive the data of a citizen near him/her. Regarding the R2D protocol, the main objective of the library is to simplify the usage of R2D protocol to mobile application developers in order to foster the adoption and the propagation of R2D. It offers a set of Java operations for allowing a mobile application of a Citizen to receive his/her personal health records using the R2D protocol, whereas the mobile application developers will use R2D without the need to know all the technical details of the underlying R2D concrete protocols (FHIR or eHDSI) and technologies.

All the libraries ensure the corresponding required security levels by exploiting the components of the Security Library. The design of the security library is provided in deliverable D3.10 - Design of libraries for HR security and privacy services - V2 [D3.10]. Hence, the purpose of this document is to firstly describe the name and the usage of the libraries from the side of the involved applications, including the interactions with their offered and required Java interfaces.

1.2. Intended audience

The current document is mainly intended for developers, and manufacturers that are interested in designing and building either S-EHR applications or HCP applications who desire to exploit and reuse either the D2D or the R2D or both these two functionalities offered by 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 adopted by other systems and applications. Apart from that, the document is intended to researchers as well, as they may be interested in understanding the way that those 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 design of the libraries implementing the D2D protocol, in the form of external operations and the way that they are invoked.
- Section 3 describes the remote health data exchange protocol (R2D), describing in deep detail the purpose of the existence of the protocol, whereas stating the design of the libraries implementing the R2D protocol, in the form of both internal and external operations and the way that they are invoked.
- Section 4 describes the data model that is currently being used for both the D2D and the R2D protocols.
- Section 5 outlines the conclusions of the current document, including future developments and updates of the two libraries.

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

With regards to the previous version of the deliverable (D4.4 Specification of remote and D2D protocol and APIs for HR exchange V1 **[D4.4]**]), the following changes have been performed for each separate section of the current document.

Section 1 - Introduction:

- Scope of the document: a minor text update has been provided regarding the overall purpose and usage of the D2D and R2D libraries that implement the functionality of the protocols.

Section 2 - Design of the D2D Libraries:

- Section S-EHR side D2D Library
    - Updated public interfaces figure including the new version of the interfaces and the offered operations
    - Updated the operations' arguments of the D2DConnection interface
    - Updated the operations' arguments of the D2DConnectionListeners interface
    - Added the getVitalSigns operation to the D2DHRExchange interface
    - Deleted the onConsentRequestReceived operation from the D2DHRExchangeListeners interface
    - Added the onPrescriptionReceived operation to the D2DHRExchangeListeners interface
- Section HCP side D2D Library
    - Updated public interfaces figure including the new version of the interfaces and the offered operations
    - Updated the operations' arguments of the D2DConnection interface
    - Deleted the getConsent operation from the D2DHRExchange interface
    - Added the getPrescription and the getLaboratoryResults operation to the D2DHRExchange interface
    - Deleted the onConsentResponseReceived operation from the D2DHRExchangeListeners interface

- Added the onNoConformantPatientSummaryReceived operation to the D2DHRExchangeListeners interface
- Added new example of the T-D2D-E usage, including a sequence UML diagram showcasing the overall compliance checking functionality

Section 3 - Design of the R2D Library for mobile:

- Section R2D External view
  - Changed high level Component Diagram for removal of component MR2DFacade;
- Section R2D Internal view
  - Changed low level Component Diagram: removed component MR2DFacade, added component ProgressiveExecutor.
  - Changed class diagram: remove class MR2DFacade, added ProgressiveExecutor, changed interface of MR2DFactory, changed interface of R2DServerRegistry.
  - Changed sequence diagram showing instantiation of MR2D.
  - Changed sequence diagram showing execution of method getAllRecords() over FHIR: removed MR2Facade, added creation of stack of DAOs.
  - Changed sequence diagram showing execution of method getLastRecord() over FHIR: removed MR2Facade.
  - Changed sequence diagram showing execution of method getAllRecords() over EHDSI: removed MR2Facade, added creation of stack of DAOs.
  - Changed sequence diagram showing execution of method getLastRecord() over EHDSI: removed MR2Facade.
  - Added a new class diagram showing the DAOs hierarchy. The LaboratoryResultDAO and the MedicalImageDAO have been added for the second release of the design.

Section 5 - Conclusion and Next Steps:

- Minor updates have been provided based on the current conclusions and our future goals for the implementation of the D2D and R2D libraries, explaining the reasons why an overall libraries' re-design is mandatory in the next and final version of this deliverable **[D4.6]**, so as for the D2D and R2D libraries to have similar APIs, and be more easily implementable.

2. DESIGN OF THE D2D LIBRARIES

The D2D protocol defines the set of operations and the exchanged messages 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. More details about the D2D protocol specification can be found in D4.2 - Specification of remote and D2D protocol and APIs for HR exchange V2 [D4.2]. This section of the document (Section 2), provides the design of the libraries including a short description of D2D Libraries that can be used by any S-EHR app and HCP App, describing the name and the usage of the D2D Library from the side of the S-EHR app, and the name and the usage of the D2D Library from the side of the HCP app. Moreover, the description of the Public Java Components contained in each library is defined, including a description of the offered and required interfaces of those components. In addition, the description of the interactions of the components of the libraries takes place, whereas the dependencies from third party libraries are also depicted. To this end, the operations of the libraries interfaces are also described through providing their internal interactions.

2.1. D2D Libraries

The D2D protocol will define the operations represented by the interfaces that will be offered by the mobile application and the healthcare's organization application communicating with the S-EHR app and the HCP app accordingly. These interfaces will be included in both applications, 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 ones involved in the overall interaction, for exchanging the consent of accessing each one's personal data, the healthcare related data, and the evaluation data. 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 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. 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 1 displays the overall connection between a citizen and an HCP, including the aforementioned interfaces, as well as the involved applications.

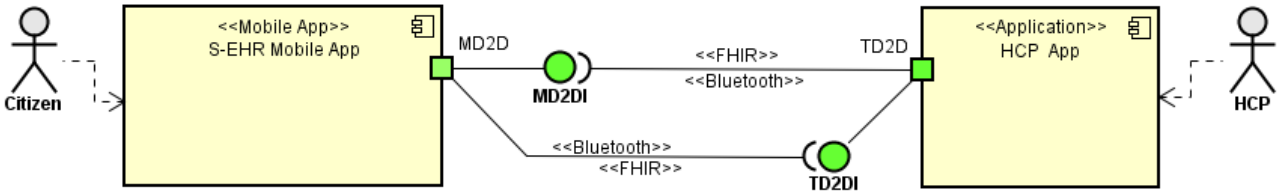

Figure 1 - D2D protocol interfaces

These interfaces will be included into the libraries which are especially designed for serving the purposes of the D2D protocol from the side of the S-EHR app and the HCP app.

2.1.1. S-EHR-side D2D library

Regarding the S-EHR-side D2D library (i.e. M-D2D-E), this will contain the total of the operations that will be needed from the side of the S-EHR application developer to interact with the library and finally with the

HCP application. This library will contain different operations that will have to be invoked in a specific sequence for implementing the purposes of the D2D protocol, regarding the S-EHR app. As described previously, this library is a Java-based component that can be nested in any Android application (version 8 or higher). It offers a set of Java operations for establishing a D2D connection and allowing a mobile app of a Citizen to exchange her personal health records using the D2D protocol.

2.1.1.1. *Components*

The M-D2D-E library incorporates a set of components (Figure 2) offering different functionalities and capabilities to the developer. These components can be offered publicly (i.e. Public components), including two major component categories: (a) the Mobile D2D Security Management that includes all the operations and functionalities related to security operations, and (b) the Mobile D2D HR Exchange that includes all the operations and functionalities related to communication operations. The second component category (i.e. Mobile D2D HR Exchange) includes two additional component categories, namely Bluetooth Connection and Data Exchange components, which are related to the functionalities for performing the connection through the Bluetooth short-range wireless communication technology and the overall data exchange actions accordingly.

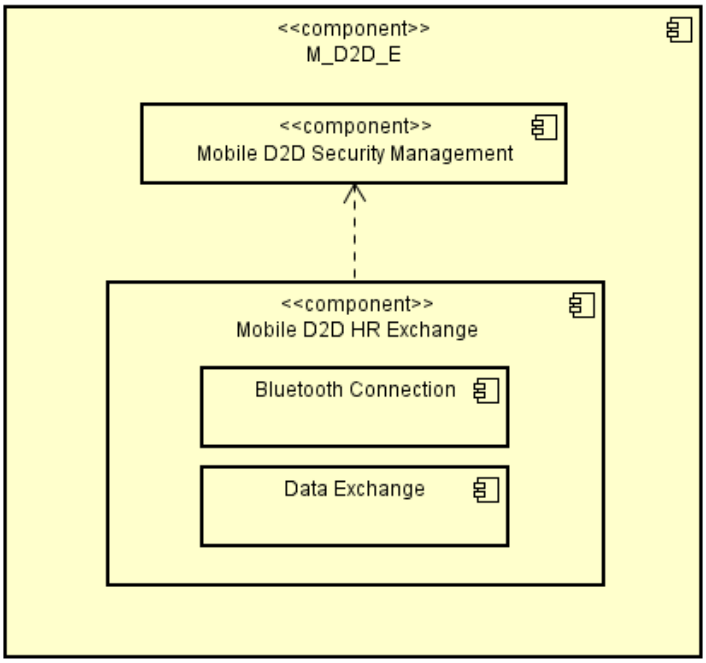

Figure 2 - M-D2D-E Public Java Components

2.1.1.2. *Public Interfaces*

The components defined in Section 2.1.1.1 are offering specific interfaces, categorized into two main categories, the offered and the required ones. The offered interfaces contain the operations which are offered by the M-D2D-E library and can be used by the developer without the need of any implementation from the developer's side, whereas the required interfaces contain operations whose implementation is not offered, and the developer has to implement specific callback operations that the specific interface will invoke. These interfaces are depicted in Figure 3 and explained below.

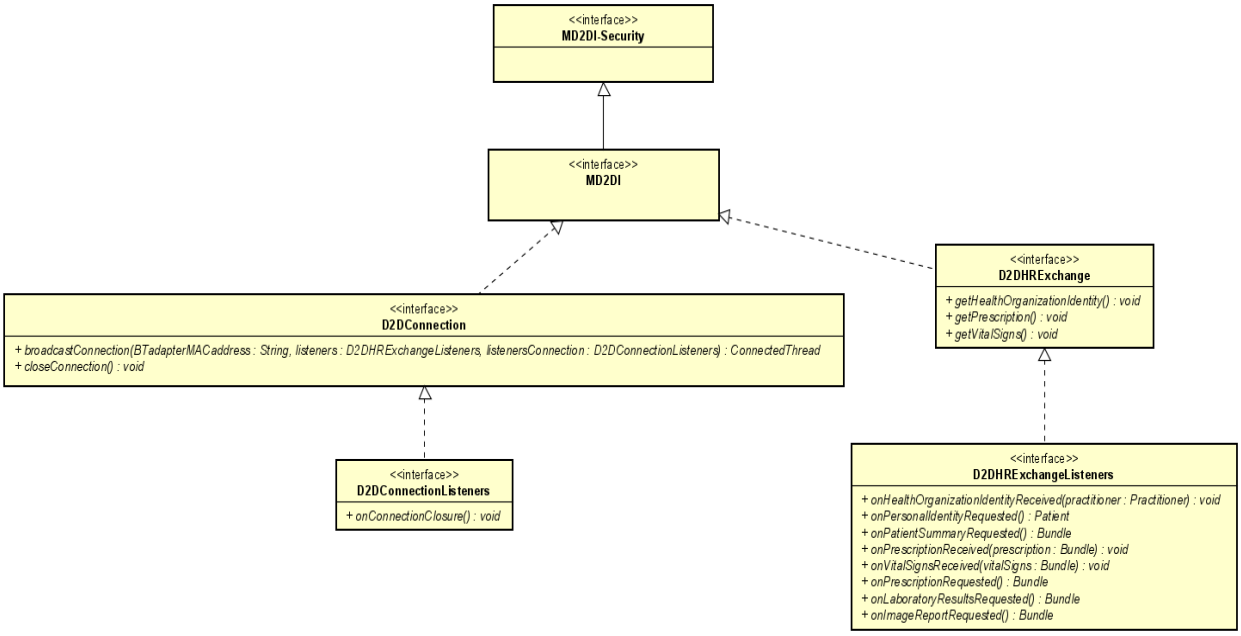

Figure 3 - M-D2D-E Public Java Components Interfaces

MD2DI-Security

MD2DI-Security is the name of the interface that is offered by the Mobile D2D Security Management component, containing the operations related to the S-EHR app and its interactions with the Security Library. More details regarding this interface as well as the provided operations can be found in D3.10 - Design of libraries for HR security and privacy services - V2 **[D3.10]**. It should be noted that this interface is offered to the developer without the need of implementing the included operations (i.e. Offered interface).

MD2DI

MD2DI is the name of the interface that is offered by the Mobile D2D HR Exchange component, containing the operations for letting the S-EHR app interact with the M_D2D_E library and finally perform communication actions with the HCP app, by invoking these operations. MD2DI is a nested interface containing additional interfaces for facilitating this communication process. These interfaces will be the: (a) D2DConnection, (b) D2DConnectionListeners, (c) D2DHRExchange and (d) D2DHRExchangeListeners. It should be noted that the D2DConnection and the D2DHRExchange interfaces are offered to the developer without the need of implementing the included operations, whereas for the D2DConnectionListeners and the D2DHRExchangeListeners it is required from the developer to implement specific callback operations that will be invoked by the aforementioned interfaces.

D2DConnection

The D2DConnection interface contains all the operations that have to be invoked for performing the bluetooth connection, regarding the side of the M-D2D-E library.

Operation broadcastConnection

Name	broadcastConnection
Description	This operation is invoked by the S-EHR app for connecting with the HCP who advertises a specified Bluetooth connection.
Arguments	• BTadapterMACaddress: a string that contains the MAC address of the Bluetooth adapter that is used from the side of the HCP. • listeners: an object containing all the implemented methods that need to be invoked from the D2D library in order for the exchange of data to be successful. • listenersConnection: an object containing the implemented method that needs to be invoked from the D2D library in order for the Bluetooth connection closure to be successful.
Return Value	This operation will return the new thread that was opened for listening for incoming messages.
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above.

Operation closeConnection

Name	closeConnection
Description	This operation is invoked by the S-EHR app for closing the bluetooth connection between the two devices.
Arguments	-
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.

Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above.

D2DConnectionListeners

The D2DConnectionListeners interface contains all the operations that have to be invoked for listening to the actions related to the bluetooth connection closure, regarding the side of the M-D2D-E library.

Operation onConnectionClosure

Name	onConnectionClosure
Description	This operation is invoked by the D2D library to notify the S-EHR app when there is a connection closure.
Arguments	-
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above. • The session is still valid.

D2DHRExchange

The D2DHRExchange interface contains all the operations that have to be invoked for performing the data exchange processes, regarding the side of the M-D2D-E library.

Operation getHealthOrganizationIdentity

Name	getHealthOrganizationIdentity
Description	This operation is invoked by the S-EHR app for requesting the Healthcare organization identity data from the HCP as provided from the HCP app.
Arguments	-

Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation getPrescription

Name	getPrescription
Description	This operation is invoked by the S-EHR app for requesting the prescription data from the HCP as provided from the HCP app.
Arguments	-
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation getVitalSigns

Name	getVitalSigns
Description	This operation is invoked by the S-EHR app for requesting the vital signs data from the HCP as provided from the HCP app.
Arguments	-
Return Value	-

Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above. • The session is still valid.

D2DHRExchangeListeners

The D2DHRExchangeListeners interface contains all the operations that have to be invoked for listening to the actions related to the exchange of specific types of data, regarding the side of the M-D2D-E library.

Operation onHealthOrganizationIdentityReceived

Name	onHealthOrganizationIdentityReceived
Description	This operation is invoked by the D2D library for informing the S-EHR app that the Healthcare Organization personal identity has been received from the side of the HCP app.
Arguments	• practitioner: the HCP's demographic data, with a reference to the data related to the HCP's organization in the form of a FHIR object.
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onPersonalIdentityRequested

Name	onPersonalIdentityRequested
Description	This operation is invoked by the D2D library for requesting the patient's personal identity (i.e. demographic data) from the S-EHR app.
Arguments	-

Return Value	This operation will return the Personal Identity (i.e demographic data) from the patient in the form of a FHIR object containing specific details that identify the citizen.
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onPatientSummaryRequested

Name	onPatientSummaryRequested
Description	This operation is invoked by the D2D library for requesting the Patient Summary of the citizen from the S-EHR app.
Arguments	-
Return Value	This operation will return the Patient Summary of the citizen in the form of a FHIR object (Bundle) containing specific details about the patient's medical history.
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onPrescriptionReceived

Name	onPrescriptionReceived
Description	This operation is invoked by the D2D library for informing the S-EHR app that the Healthcare Organization prescriptions have been received from the side of the HCP app.
Arguments	• prescription: the HCP's prescription data in a form of Bundle (i.e. FHIR Resource Bundle).
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onVitalSignsReceived

Name	onVitalSignsReceived
Description	This operation is invoked by the D2D library for informing the S-EHR app that the Healthcare Organization vital signs have been received from the side of the HCP app.
Arguments	• vitalSigns: the HCP's vital signs in a form of Bundle (i.e. FHIR Resource Bundle).
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onPrescriptionRequested

Name	onPrescriptionRequested
Description	This operation is invoked by the D2D library for requesting the prescriptions of the citizen from the S-EHR app.
Arguments	-
Return Value	This operation will return the prescriptions of the citizen in the form of a FHIR object (Bundle) containing specific details about the patient's medication requests.
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onLaboratoryResultsRequested

Name	onLaboratoryResultsRequested
Description	This operation is invoked by the D2D library for requesting the laboratory results of the citizen from the S-EHR app.
Arguments	-
Return Value	This operation will return the laboratory results of the citizen in the form of a FHIR object (Bundle) containing different details about the patient's medical tests.
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onImageReportRequested

Name	onImageReportRequested
Description	This operation is invoked by the D2D library for requesting an image report of the citizen from the S-EHR app.
Arguments	-
Return Value	This operation will return the image report of the citizen in the form of a FHIR object (Bundle) containing specific details about the report and a medical image converted into bytes.
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The smartphone is enabled with Bluetooth v4.0 and above. • The session is still valid.

2.1.1.3. *Example of usage of M-D2D-E*

The following sequence diagram (Figure 4) shows the fundamental steps executed by the S-EHR app in order to retrieve the Health Organization Identity from the HCP app, using the operations getHealthOrganizationIdentity(). The first part of the sequence diagram shows the creation of the listeners for being notified of the requested process, while the second part shows the sequence of invocations of operations described in the previous sections. However, it should be noted that some of these operations are described in the upcoming section (Section 2.1.2) but are described in this example, since both libraries need to communicate with each other for demonstrating an end-to-end example. It is important to state that the following sequence does not show the real complexity and the complete interactions between components, because its main objective is to focus on interfaces, methods and data used by the S-EHR app.

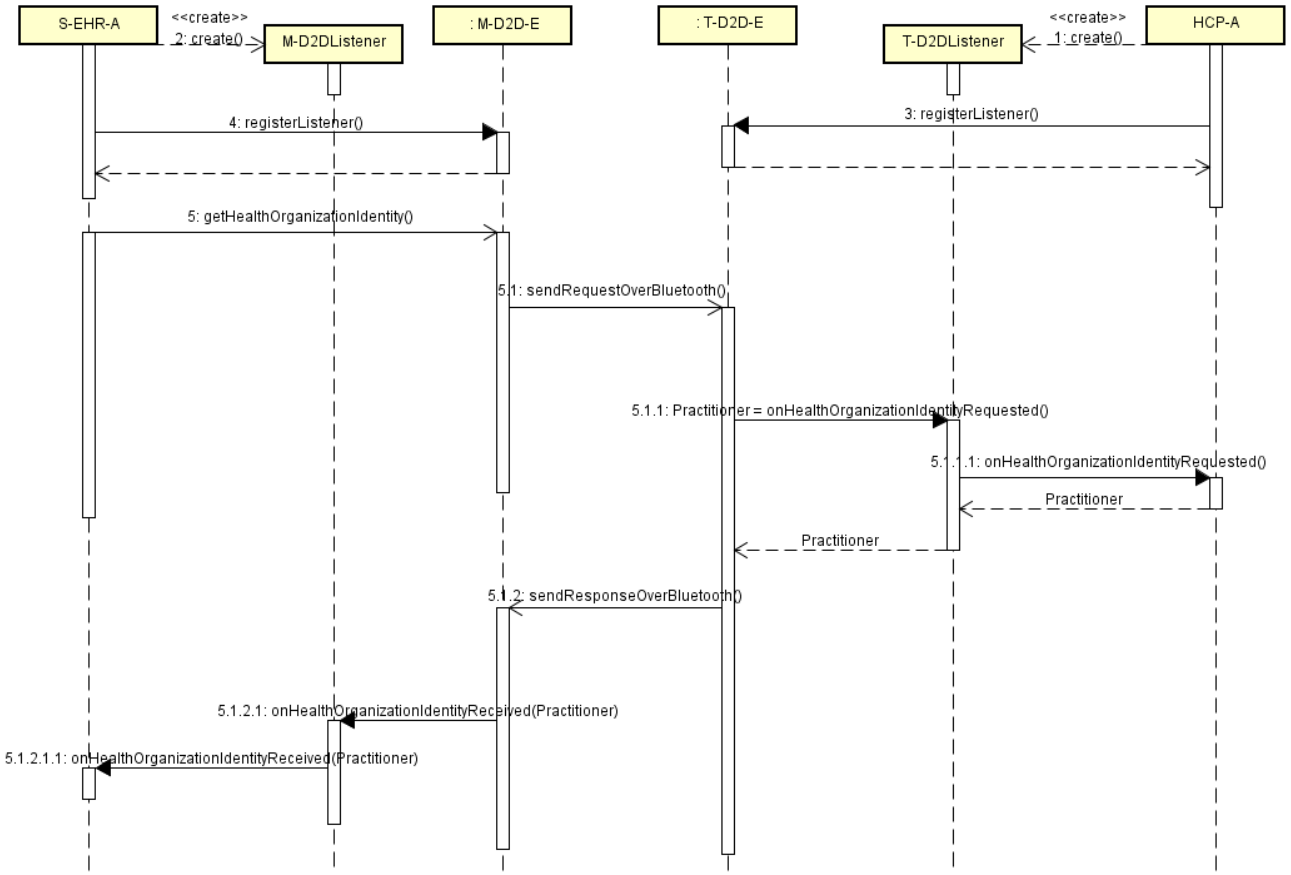

Figure 4 - Example of retrieving the Health Organization Identity

Step 1: Creation of the T-D2DListener for creating asynchronous callbacks from the side of the HCP app (HCP-A), in order to implement the code to run when an event occurs.

Step 2: Creation of the M-D2DListener for creating asynchronous callbacks from the side of the S-EHR app (S-EHR-A), in order to implement the code to run when an event occurs.

Step 3: The S-EHR app registers the listener to track it and pass on the events to it.

Step 4: The HCP app registers the listener to track it and pass on the events to it.

Step 5: The S-EHR app is invoking the getHealthOrganizationIdentity() operation for retrieving the Health Organization identity from the side of the HCP app. This request is sent through the Bluetooth communication (Step 5.1) where the T-D2DListener, as soon as it listens to this request (Step 5.1.1), through the onHealthOrganizationIdentityRequested() operation, it receives the response to this request by providing a Practitioner Object (Step 5.1.1.1). This Object is transferred through the Bluetooth communication (Step 5.1.2), where the M-D2DListener, as soon as it listens to the response to the request (Step 5.1.2.1), through the onHealthOrganizationIdentityReceived(Practitioner) operations, it provides the transferred object back to the S-EHR app (Step 5.1.2.1.1).

2.1.1.4. *Third Party Libraries*

The M-D2D-E library is currently dependent on two external libraries (Figure 5). These libraries are: (a) the HAPI FHIR Library, and (b) the Android Bluetooth API.

HAPI FHIR Library

The HAPI FHIR Library v4.1.0 **[HAPI]** is being used to define model classes for the resource type and datatype defined by the FHIR specification, based on the current data model that is described in D2.8 - FHIR Profile for EHR Interoperability **[D2.8]**. In the case of the M-D2D-E library, the HAPI FHIR Library is being used for transferring FHIR Resources in the form of FHIR objects (e.g. Patient Resource, Practitioner Resource) through the operations offered by the D2DHRExchangeListeners interface. Currently, the HAPI FHIR Library is used as a Gradle dependency in the Gradle file of the M-D2D-E library.

Android Bluetooth API

The Android Bluetooth API **[ANDROID BLUETOOTH]** is being used to support the Bluetooth network stack, which allows a device to wirelessly exchange data with other Bluetooth devices. In the case of the M-D2D-E library, the Android Bluetooth API is being used for Bluetooth connection and the wireless exchange of data through the operations offered by the D2DConnection and the D2DConnectionListeners interfaces. Currently, the Android Bluetooth API is being provided as a Maven dependency in the pom.xml file of the project of the M-D2D-E library.

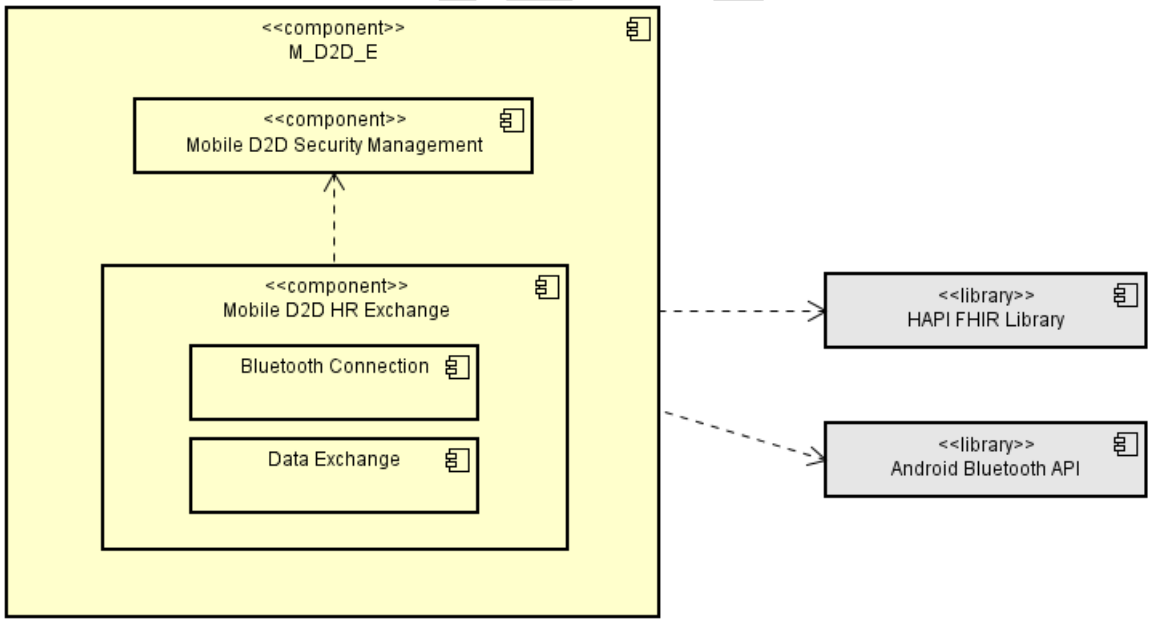

Figure 5 - M-D2D-E Third Party Libraries

2.1.2. HCP-side D2D library

Regarding the HCP-side D2D library (T-D2D-E), this will contain the total of the operations that will be needed from the side of the HCP application developer to interact with the library and finally with the S-EHR application. This library will contain different operations that will have to be invoked in a specific sequence for implementing the purposes of the D2D protocol, regarding the HCP app. As described above, the second library is a Java based component that can be embedded in any Java based application. It offers a set of operations for establishing a D2D connection and enabling the application used by a HCP to send and receive data of a Citizen near her.

2.1.2.1. *Components*

The T-D2D-E library contains a set of components (Figure 6) offering different functionalities and capabilities to the developer. These components can be offered publicly (i.e. Public components), including two major component categories: (a) the Terminal D2D Security Management that includes all the operations and functionalities related to security operations, and (b) the Terminal D2D HR Exchange that includes all the operations and functionalities related to communication operations. The second component category (i.e. Terminal D2D HR Exchange) includes two additional component categories, namely Bluetooth Connection and Data Exchange components, which are related to the functionalities for performing the connection through the bluetooth short-range wireless communication technology and the overall data exchange actions accordingly.

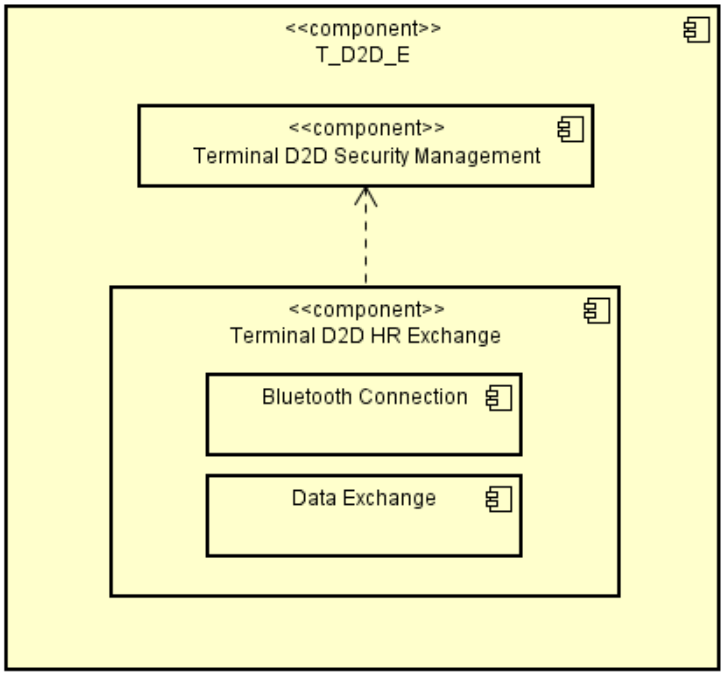

Figure 6 - T-D2D-E Public Java Components

2.1.2.2. *Public Interfaces*

The components defined in Section 2.1.2.1 are offering specific interfaces, categorized into two main categories, the offered and the required interfaces. The offered interfaces contain the operations which are offered by the T-D2D-E library and can be used by the developer without the need of any implementation from the developer's side, whereas the required interfaces contain operations whose implementation is

not offered, and the developer has to implement specific callback operations that the specific interface will invoke. These interfaces are depicted in Figure 7 and explained below.

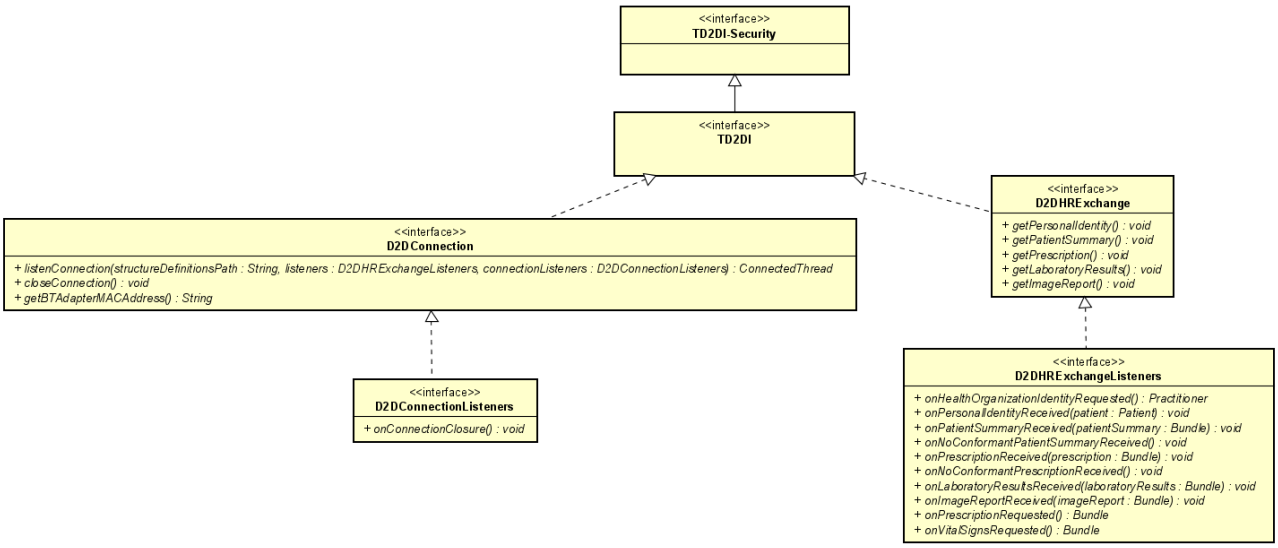

Figure 7 - T-D2D-E Public Java Components Interfaces

TD2DI-Security

TD2DI-Security is the name of the interface that is offered by the Terminal D2D HR Exchange component, containing the operations related to the HCP app and its interactions with the Security Library. More details regarding this interface as well as the provided operations can be found in **[D3.10]**. It should be noted that this interface is offered to the developer without the need of implementing the included operations (i.e. Offered interface).

TD2DI

TD2DI is the name of the interface that is offered by the Terminal D2D HR Exchange component, containing the operations for letting the HCP app interact with the T-D2D-E library and finally perform communication actions with the S-EHR app, by invoking these operations. TD2DI is a nested interface containing additional interfaces for facilitating this communication process. These interfaces will be the: (a) D2DConnection, (b) D2DConnectionListeners, (c) D2DHRExchange and (d) D2DHRExchangeListeners. It should be noted that the D2DConnection and the D2DHRExchange interfaces are offered to the developer without the need of implementing the included operations, whereas for the D2DConnectionListeners and the D2DHRExchangeListeners it is required from the developer to implement specific callback operations that will be invoked by the aforementioned interfaces.

D2DConnection

The D2DConnection interface contains all the operations that have to be invoked for performing the bluetooth connection, regarding the side of the T-D2D-E library.

Operation listenConnection

Name	listenConnection
Description	This operation is invoked by the HCP app for creating a server socket waiting for a device to connect and start a new thread when one is found.
Arguments	• listeners: an object containing all the implemented methods that need to be invoked from the D2D library in order for the exchange of data to be successful. • listenersConnection: an object containing the implemented method that needs to be invoked from the D2D library in order for the bluetooth connection closure to be successful. • structureDefinitionsPath: a string that contains the path of the folder that contains the InteropEHRate profiles.
Return Value	This operation will return the new thread that was opened for listening for incoming messages.
Exceptions	• Security exceptions related to Bluetooth connection. • Network exceptions related to Bluetooth connection failure.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above.

Operation closeConnection

Name	closeConnection
Description	This operation is invoked by the HCP app for closing the bluetooth connection between the two devices.
Arguments	-
Return Value	-
Exceptions	• Security exceptions related to Bluetooth connection. • Network exceptions related to Bluetooth connection failure.

Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above.

Operation getBTAdapterMACAddress

Name	getBTAdapterMACAddress
Description	This operation is invoked by the HCP app to get the current device's Bluetooth Adapter MAC Address.
Arguments	-
Return Value	This operation returns this device's Bluetooth MAC Address in the requested format.
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above.

D2DConnectionListeners

The D2DConnectionListeners interface contains all the operations that have to be invoked for listening to the actions related to the Bluetooth connection closure, regarding the side of the T-D2D-E library.

Operation onConnectionClosure

Name	onConnectionClosure
Description	This operation is invoked by the D2D library to notify the HCP app when there is a connection closure.
Arguments	-
Return Value	-

Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

D2DHRExchange

The D2DHRExchange interface contains all the operations that have to be invoked for performing the data exchange processes, regarding the side of the T-D2D-E library.

Operation getPersonalIdentity

Name	getPersonalIdentity
Description	This operation is invoked by the HCP App for requesting the Personal Identity data from the citizen.
Arguments	-
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP App is enabled with Bluetooth V4.0 and above. • The session is still valid.

Operation getPatientSummary

Name	getPatientSummary
Description	This operation is invoked by the HCP App for requesting the patient summary data from the citizen.
Arguments	-
Return Value	-

Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation getPrescription

Name	getPrescription
Description	This operation is invoked by the HCP App for requesting the prescription data from the citizen.
Arguments	-
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation getLaboratoryResults

Name	getLaboratoryResults
Description	This operation is invoked by the HCP App for requesting the laboratory result's data from the citizen.
Arguments	-
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.

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

Operation getImageReport

Name	getImageReport
Description	This operation is invoked by the HCP App for requesting the image report data from the citizen.
Arguments	-
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

D2DHRExchangeListeners

The D2DHRExchangeListeners interface contains all the operations that have to be invoked for listening to the actions related to the exchange of specific types of data, regarding the side of the T-D2D-E library.

Operation onHealthOrganizationIdentityRequested

Name	onHealthOrganizationIdentityRequested
Description	This operation is invoked by the D2D library for getting the Healthcare Organization personal identity from the HCP app.
Arguments	-
Return Value	This operation will return the Healthcare Organization identity in the form of a FHIR object containing specific details that identify the Organization.

Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onPersonalIdentityReceived

Name	onPersonalIdentityReceived
Description	This operation is invoked by the D2D library for informing the HCP app that the Personal identity data (i.e. demographic data) of the citizen has been received from the side of the S-EHR app.
Arguments	• patient: a patient's demographic data in the form of a FHIR object.
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onPatientSummaryReceived

Name	onPatientSummaryReceived
Description	This operation is invoked by the D2D library for informing the HCP app that the Patient Summary of the citizen has been received from the side of the S-EHR app.
Arguments	• patientSummary: a patient's summary in a form of Bundle (i.e. FHIR Resource Bundle)
Return Value	-

Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onNoConformantPatientSummaryReceived

Name	onNoConformantPatientSummaryReceived
Description	This operation is invoked by the D2D library for informing the HCP app that the Patient Summary of the citizen that was received from the side of the S-EHR app is not compliant with the InteropEHRate profiles.
Arguments	-
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onPrescriptionReceived

Name	onPrescriptionReceived
Description	This operation is invoked by the D2D library for informing the HCP app that the prescription data of the citizen has been received from the side of the S-EHR app.
Arguments	• prescription: a patient's prescription data in a form of Bundle (i.e. FHIR Resource Bundle)
Return Value	-

Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onNoConformantPrescriptionReceived

Name	onNoConformantPrescriptionReceived
Description	This operation is invoked by the D2D library for informing the HCP app that the prescription data of the citizen that was received from the side of the S-EHR app is not compliant with the InteropEHRate profiles.
Arguments	-
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onLaboratoryResultsReceived

Name	onLaboratoryResultsReceived
Description	This operation is invoked by the D2D library for informing the HCP app that the laboratory results of the citizen has been received from the side of the S-EHR app.
Arguments	• laboratoryResults: a patient's laboratory results in a form of Bundle (i.e. FHIR Resource Bundle)
Return Value	-

Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onImageReportReceived

Name	onImageReportReceived
Description	This operation is invoked by the D2D library for informing the HCP app that the image report data of the citizen has been received from the side of the S-EHR app.
Arguments	• imageReport: a patient's image report in a form of Bundle (i.e. FHIR Resource Bundle)
Return Value	-
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onPrescriptionRequested

Name	onPrescriptionRequested
Description	This operation is invoked by the D2D library for getting the prescription data from the HCP app.
Arguments	-
Return Value	This operation will return the prescription data in the form of a Bundle (i.e. FHIR Resource Bundle) containing medication requests.

Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

Operation onVitalSignsRequested

Name	onVitalSignsRequested
Description	This operation is invoked by the D2D library for getting the vital signs data from the HCP app.
Arguments	-
Return Value	This operation will return the vital signs in the form of a Bundle (i.e. FHIR Resource Bundle).
Exceptions	• Security exceptions related to Bluetooth state. • Network exceptions related to Bluetooth state.
Preconditions	• The HCP app is enabled with Bluetooth v4.0 and above. • The session is still valid.

2.1.2.3. *Example of usage of T-D2D-E*

The following sequence diagram (Figure 8) shows the fundamental steps executed by the HCP app in order to retrieve the Patient Summary from the S-EHR app, using the operations getPatientSummary(). The first part of the sequence diagram shows the creation of the listeners for being notified of the requested process, while the second part shows the sequence of invocations of operations described in the previous sections. It is important to state that the following sequence does not show the real complexity and the complete interactions between components, because its main objective is to focus on interfaces, methods and data used by the HCP app.

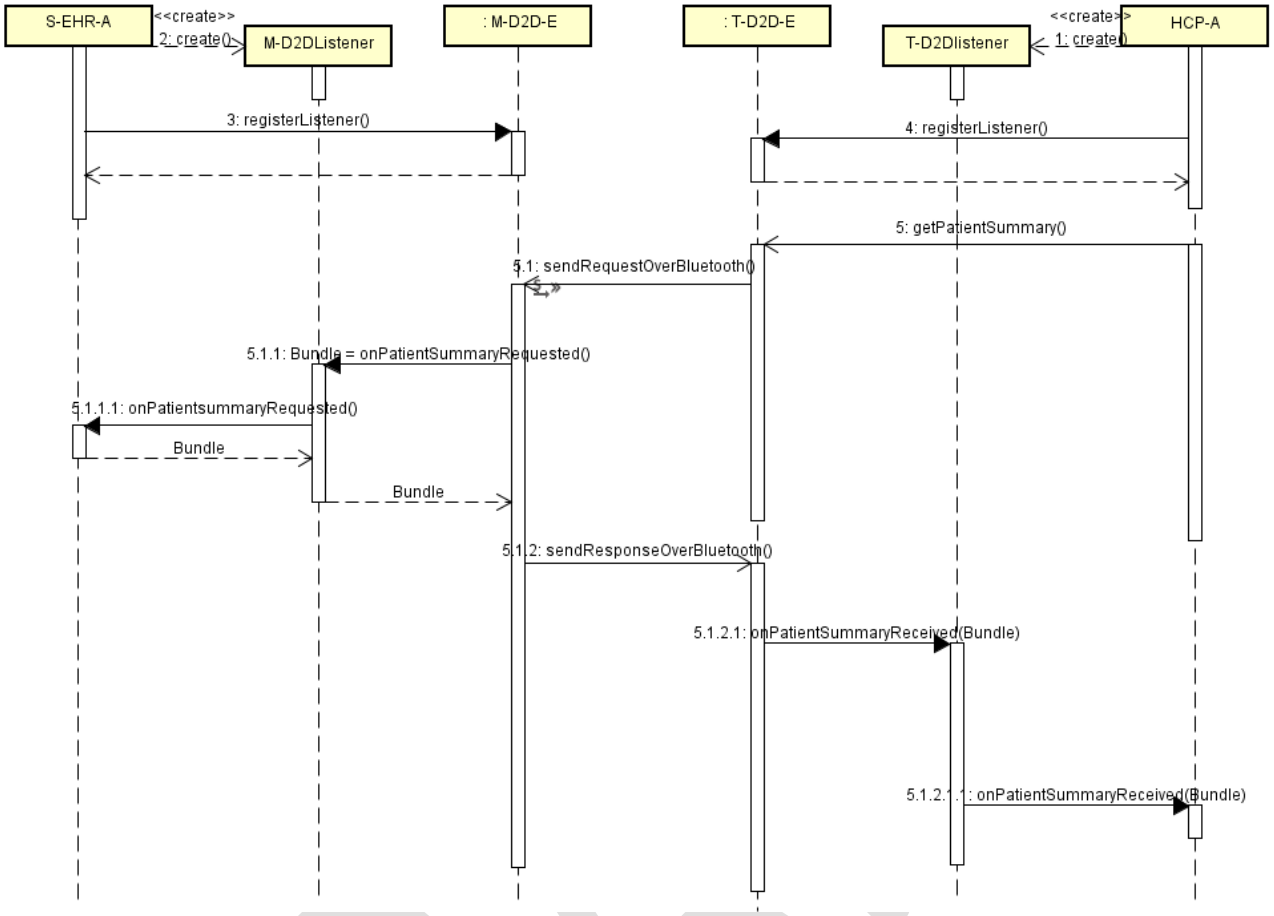

Figure 8 - Example of retrieving the Patient Summary

Step 1: Creation of the T-D2DListener for creating asynchronous callbacks from the side of the HCP app (HCP-A), in order to implement the code to run when an event occurs.

Step 2: Creation of the M-D2DListener for creating asynchronous callbacks from the side of the S-EHR app (S-EHR-A), in order to implement the code to run when an event occurs.

Step 3: The S-EHR app registers the listener to track it and pass on the events to it.

Step 4: The HCP app registers the listener to track it and pass on the events to it.

Step 5: The HCP app is invoking the getPatientsummary() operation for retrieving the Patient Summary from the side of the S-EHR app (Step 5). This request is sent through the Bluetooth communication (Step 5.1) where the M-D2DListener, as soon as it listens to this request, through the onPatientSummaryRequested() operation (Step 5.1.1), it receives the response to this request by providing a Bundle (Step 5.1.1.1). This Bundle is transferred through the Bluetooth communication (Step 5.1.2), where the T-D2DListener, as soon as it listens to the response to the request (Step 5.1.2.1), through the onPatientSummaryReceived(Bundle) operation, it provides the transferred object back to the HCP app (Step 5.1.2.1.1).

The following sequence diagram (Figure 9) shows the same example that was explained in detail before but with the extension of checking the compliance of the data set received. In more detail, on step 5 instead of calling onPatientSummaryReceived (Bundle) automatically, a compliance check to the InteropEHRate profiles happens. In this way the HCP will receive only data sets that are conformant to the profiles offered by the interopEHRate project. The D2D library will validate the resources coming on the HCP automatically and if they are not conformant the Bluetooth connection will close, and a message will pop up on the screen of the HCP App (onNoConformantPatientSummaryReceived()).

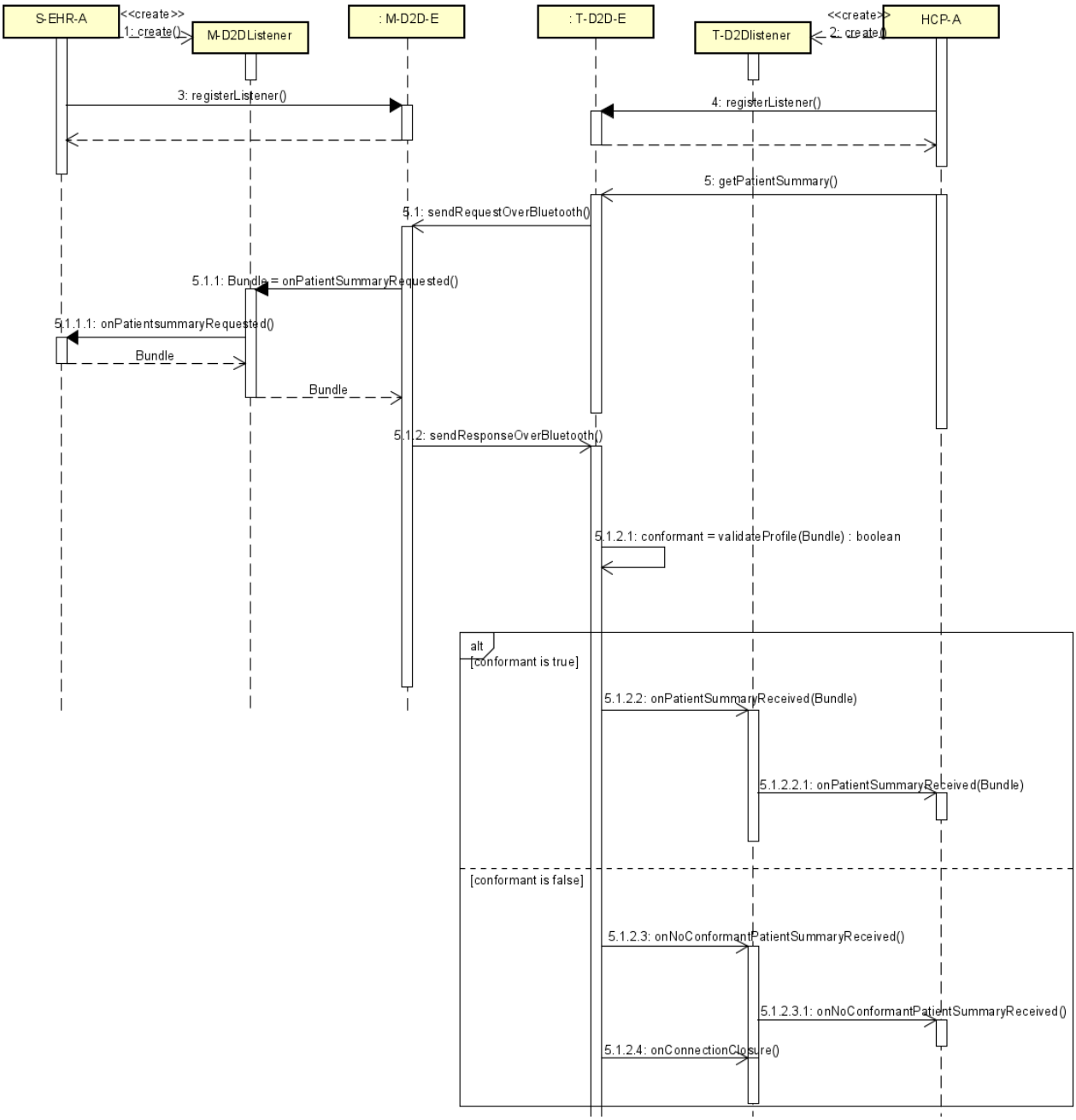

Figure 9 - Example of retrieving the Patient Summary with Compliance Checking

2.1.2.4. *Third Party Libraries*

The T-D2D-E library is currently dependent on two external libraries (Figure 9) that contain the interfaces and the offered operations. These libraries are: (a) the HAPI FHIR Library, and (b) the Bluecove Library.

HAPI FHIR Library

The HAPI FHIR Library v4.1.0 **[HAPI]** is being used to define model classes for the resource type and datatype defined by the FHIR specification, based on the current data model that is described in D2.8 - FHIR Profile for EHR Interoperability **[D2.8]**. In the case of the T-D2D-E library, the HAPI FHIR Library is being used for transferring FHIR Resources in the form of FHIR objects (e.g. Patient Resource, Practitioner Resource) through the operations offered by the D2DHRExchangeListeners interface. Currently, the HAPI FHIR Library is being used as a Gradle dependency in the Gradle file of the T-D2D-E library.

Bluecove Library

The Bluecove Library v2.1.0 **[BLUECOVE]** is being used to support Mac OS X, WIDCOMM, BlueSoleil and Microsoft Bluetooth stack found in Windows XP SP2 or Windows Vista and WIDCOMM and Microsoft Bluetooth stack on Windows Mobile, allowing a device to wirelessly exchange data with other Bluetooth devices. In the case of the T-D2D-E library, the Bluecove Library is being used for Bluetooth connection and the wireless exchange of data through the operations offered by the D2DConnection and the D2DConnectionListeners interfaces. Currently, the Bluecove Library is being provided as a Maven dependency in the pom.xml file of the project of the T-D2D-E library.

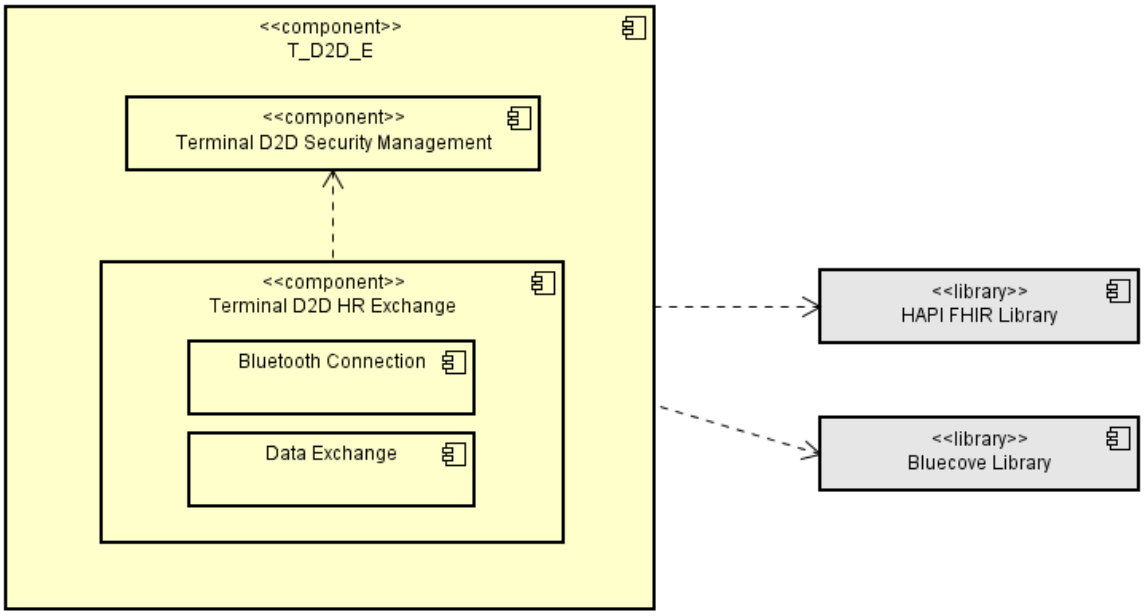

Figure 10 - T-D2D-E Third Party Libraries

3. DESIGN OF THE R2D LIBRARY FOR MOBILE

This section of the document (Section 3), provides the design of the M-R2D-E library, describing the usage of the R2D Access Library from the side of the S-EHR app. Moreover, the description of the Public Java Components contained in each library is defined, including a description of the OFFERED and REQUIRED interfaces of those components. In addition, the description of the interactions of the components of the libraries takes place, whereas the dependencies from third party libraries are also depicted. To this end, the private components of the libraries are described, in combination with their internal interactions.

3.1. R2D Access Libraries

As defined in D4.2 - Specification of remote and D2D protocol and APIs for HR exchange V2 **[D4.2]** the *R2D Access* protocol defines the set of operations used for enabling the exchange of health data between a local or National EHR and the S-EHR App with the usage of the internet. In order to simplify the adoption of R2D by developers of apps, the InteropEHRate project develops a library for mobile named M-R2D-E. The objective of this library is to allow the usage of R2D without the need for developers to know all the technical details of the underlying R2D concrete protocols and technologies. The M-R2D-E library acts as a proxy for an health care system compliant to R2D Access specifications.

3.1.1. R2D Access Library - External view

This section provides a description of the external view of M-R2D-E library, it describes the interfaces of classes and the structure of data directly used by S-EHR App or by any generic app using M-R2D-E. The external view of the M-R2D-E is composed by the following UML diagrams:

- a component diagram showing: i) the interfaces used by the S-EHR app to interact with the main components of the library, ii) the dependencies with other libraries of the InteropEHRate project, or third party libraries;
- a class diagram defining the interface of the components used by the S-EHR app;
- a class diagram showing the data model used by the S-EHR app;
- a sequence diagram showing the interactions between S-EHR and M-R2D-E components in order to perform a basic transaction based on R2D protocol.

This section is not intended to provide a full detailed design of the internal structure of M-R2D-E library, but only to describe the basic usage of the library focusing on the following fundamental actions:

- how to instantiate the library;
- how to invoke methods of R2D;
- how to browse results.

3.1.1.1. *Components*

The following component diagram provides an overall view of the M-R2D-E structure, it shows the external interfaces provided to clients, the internal components implementing the interfaces and their main dependencies from other internal components or external libraries.

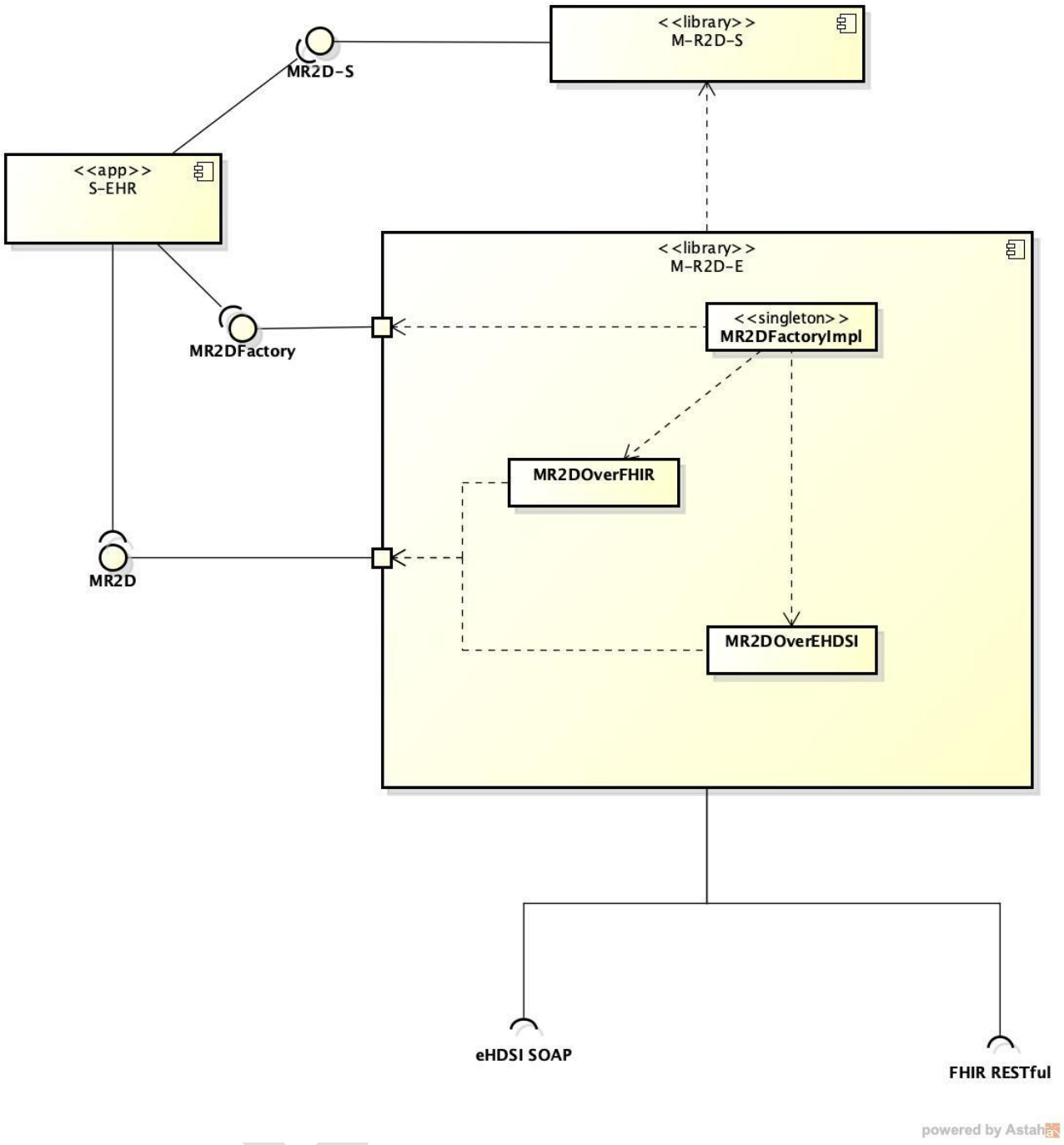

Figure 11 - M-R2D-E component diagram

As shown in this diagram, M-R2D-E library is an extension of M-R2D-S library, this means that M-R2D-E inherits methods from M-R2D-S, in particular it inherits methods regarding security and citizen authentication. A full description of the design of M-R2D-S library is out of the scope of this deliverable, while it is part of the deliverable **[D3.10]**, although the sequence diagrams shown in the next sections of this deliverable, will make conceptual references to authentication methods provided by M-R2D-S library (adding these methods in the diagrams is only for the purpose of description and not for technical specifications).

The diagram shows that users of the M-R2D-E library must interact with the following two interfaces: MR2DFactory and MR2D (including the methods inherited from M-R2D-S). The MR2DFactory interface defines methods for creating instances of MR2D (instances of components implementing the MR2D interface), while the MR2D interface defines methods for exchanging health data with an R2D server.

Internally, the MR2D interface is implemented by two components: MR2DOverFHIR and MR2DOverEHDSI, both classes are created by invoking methods of the singleton component MR2DFactoryImpl. Use of MR2D implies a first step of instantiation, delegated to the factory class, and a second step with the direct use of the business methods provided by the MR2D instance.

3.1.1.2. *Public Interfaces*

The following diagram shows the definition of the interfaces used to interact with an R2D Access compliant health care system: MR2D and MR2DFactory.

These interfaces define methods for invoking the operations of the R2D Access protocol that are described in deliverable **[D4.2]** .
MR2DFactory is the interface of a factory class used to create instances of MR2D, while MR2D is the interface that defines the business operations to concretely invoke R2D Access operations.

MR2D interface has been designed without having in mind the concrete (underlying) protocol used to implement R2D Access, but it has been designed taking into account only the requirements of a typical health app that needs to periodically import health data from a remote repository invoking as few operations as possible, managing large amount of data in an asynchronous way and without having to deal with the complexity of the underlying protocol.

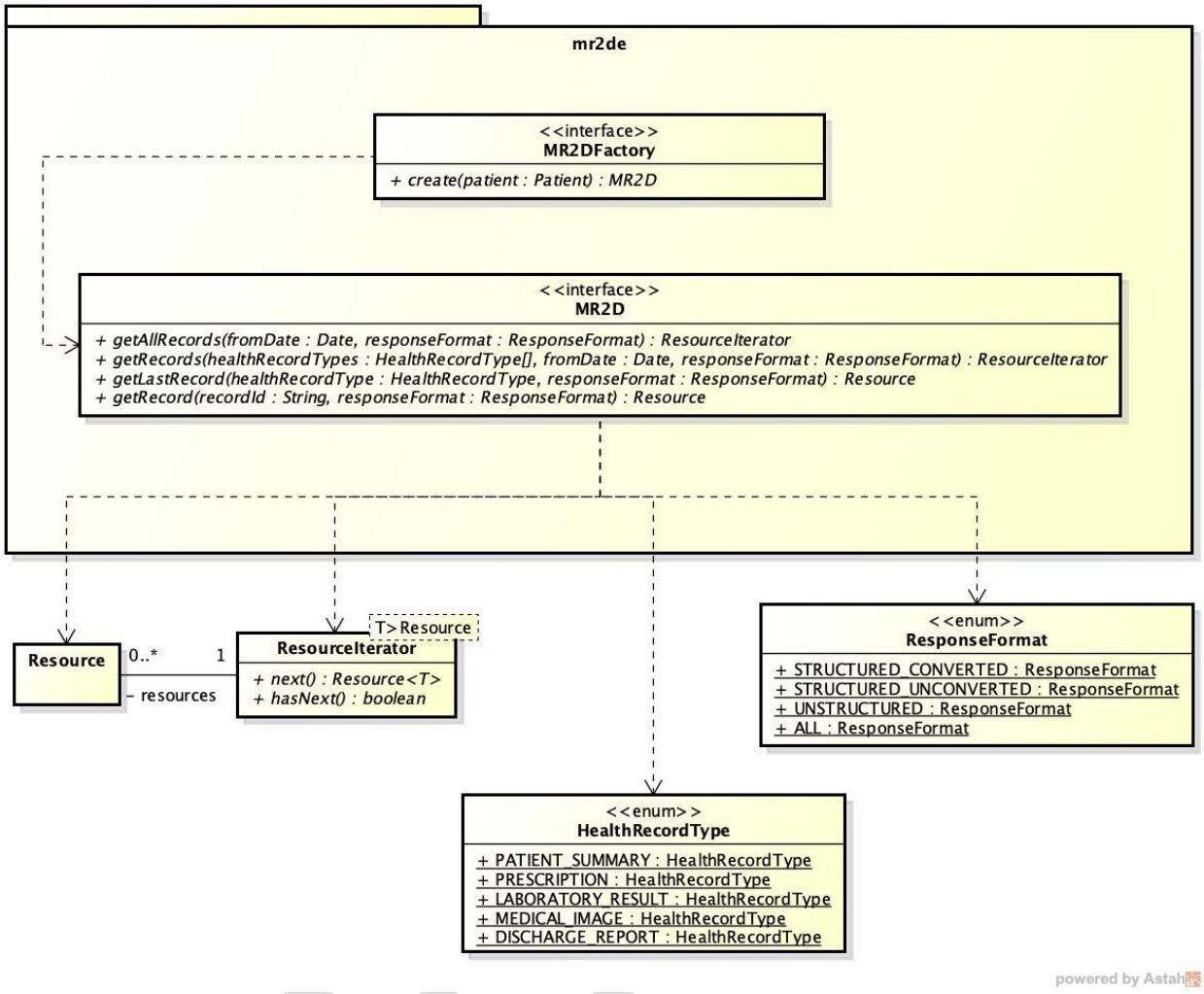

Figure 12 - M-R2D-E main interfaces

The following paragraphs provide detailed descriptions of the two interfaces defined above.

Interface MR2D

As already said before, this interface defines the methods for invoking R2D Access retrieval operations. Retrieval of health data with MR2D is type based, the defined methods are designed to retrieve all health data (belonging to the authenticated citizen) of a certain type, except the method getRecord(), all the other methods need to know what is the type of health data that must be retrieved. The types defined in M-R2D-E are represented by the values of the HealthRecordType enumeration: PATIENT_SUMMARY, PRESCRIPTION, LABORATORY_RESULT, MEDICAL_IMAGE and DISCHARGE_REPORT.

The methods of the interface have been designed to retrieve health data from the NCP in different ways, leaving to the client the freedom to retrieve a lot of data in a single invocation or retrieving smaller chunks of data with several invocations.

Operation getAllRecords

Name	getAllRecords
Description	This method returns all the health data stored in the responding R2D server of the authenticated citizen, it allows the client to retrieve all the citizen data in a single operation. Optionally, the client may specify a date indicating the date after which health data must have been produced.
Arguments	• Date fromDate: a date indicating the day after which the requested health data must have been produced. • ResponseFormat responseFormat: one of the predefined ResponseFormat enumeration values (STRUCTURED_CONVERTED, STRUCTURED_UNCONVERTED, UNSTRUCTURED, ALL) identifying the output format of the requested health data.
Return Value	An instance of ResourceIterator for iterating over health records returned from the execution of the underlying requests
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 using methods provided from the M-R2D-SM library. • The session is still valid.

Operation getRecords

Name	getRecords
Description	This method returns all the health data stored in the responding R2D server of the authenticated citizen, belonging to the specified types. Differently from the getAllRecords() method, the getRecords() method allows the client to retrieve one or more specific types of health data in a single request. If used with all the values of the HealthRecordType enumeration this method is conceptually equivalent to the getAllRecords() method. The client invoking this method uses the HealthRecordType[] parameter to define what kind of health data he is interested in. Optionally, the client may specify a date indicating the date in which health data must have been

	produced.
Arguments	• HealthRecordType[] healtRecordTypes: an array of the predefined HealthRecordType enumeration (PATIENT_SUMMARY, PRESCRIPTION, LABORATORY_RESULT, MEDICAL_IMAGE, DISCHARGE_REPORT) 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. • ResponseFormat responseFormat: one of the predefined ResponseFormat enumeration values (STRUCTURED_CONVERTED, STRUCTURED_UNCONVERTED, UNSTRUCTURED, ALL) identifying the output format of the requested health data.
Return Value	An instance of ResourceIterator for iterating over health records returned from the execution of the underlying requests
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 using methods provided from the M-R2D-SM library. • The session is still valid.

Operation getLastRecord

Name	getLastRecord
Description	This method allows a client to request the most recent version of one specific type of health data belonging to a citizen.
Arguments	• HealthRecordType healthRecordType: this argument specifies the type of health data requested by the client. • ResponseFormat responseFormat: one of the predefined ResponseFormat enumeration values (STRUCTURED_CONVERTED, STRUCTURED_UNCONVERTED, UNSTRUCTURED, ALL) identifying the output format of the requested health data.
Return Value	An instance of a subclass of org.hl7.fhir.model.Resource

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 using methods provided from the M-R2D-SM library. • 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: the ID of the health record to be retrieved id of a health data. • ResponseFormat responseFormat: one of the predefined ResponseFormat enumeration values (STRUCTURED_CONVERTED, STRUCTURED_UNCONVERTED, UNSTRUCTURED, ALL) identifying the output format of the requested health data.
Return Value	An instance of org.hl7.fhir.model.Resource
Exceptions	• Security exceptions related to the validation of the session. • Security exceptions related to the ownership of data (only health data of the authenticated citizen can be accessed). • Network exceptions related to failure during remote communication.
Preconditions	• The citizen has successfully executed the authentication using methods provided from the M-R2D-SM library. • The session is still valid.

Interface MR2DFactory

MR2DFactory is the interface for creating instances of MR2D. In the M-R2D-E library, a client is not allowed to directly create instances of concrete subclasses of MR2D, the library does not contain any public subclass

of MR2D. The only way to obtain an instance of MR2D is by using the methods provided by MR2DFactory interface and providing to it the needed parameters.

Operation create

Name	create
Description	Creates an instance of a concrete implementation of MR2D.
Arguments	• patient: instance of a org.hl7.fhir.model.Patient containing all data of the logged Citizen. The attribute address is MANDATORY.
Return Value	An instance of MR2D.
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 using methods provided from the M-R2D-SM library. • The session is still valid.

Interface ResourceIterator

This interface defines the methods of an Iterator for instances of class org.hl7.fhir.model.Resource.

Operation hasNext

Name	hasNext
Description	Returns a boolean indicating if there is still another item to iterate
Arguments	void
Return Value	boolean
Exceptions	N/A

Preconditions	N/A

Operation next

Name	next
Description	Returns the next element of an iterator and moves the iterator index on the next item (if there's one)
Arguments	void
Return Value	An instance of a subclass of org.hl7.fhir.model.Resource
Exceptions	N/A
Preconditions	N/A

3.1.1.3. *Example of usage of M-R2D-E*

In this section there is a practical example (represented by an UML sequence diagram) of usage of M-R2D-E, the example shows how to create an instance of MR2D, how to invoke one of the retrieval methods and how to iterate the results. In particular it shows the usage of the methods getAllRecords(). The first part of the sequence diagram shows the authentication phase executed using M-R2D-SM methods (the diagram shows a conceptual version of the M-R2D-SM interface), while the second shows the sequence of invocations of classes M-R2D-E classes described in the previous sections.

It is important to state that the following sequence does not show the real complexity and the complete interactions between components (especially during the creation of the instance of R2D), because its main objective is to focus on interfaces, methods and data directly used by the S-EHR app and not to internal complexity that is the subject of the second section of this chapter.

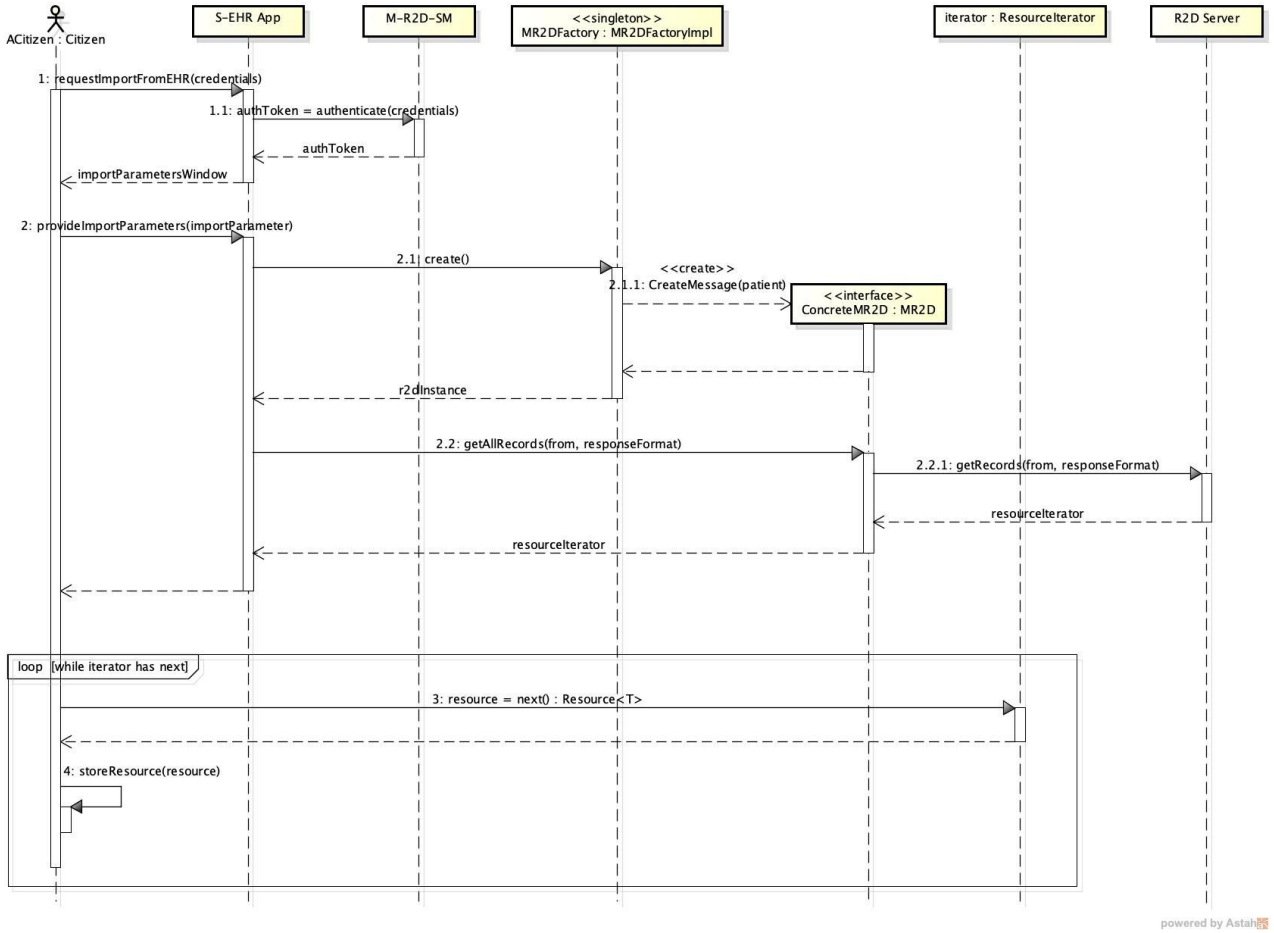

Figure 13 - Sequence diagram for getAllRecords()

- **Step 1**: a citizen requests the S-EHR app to import his health data from a health care system compliant with R2D specifications.
    - **Step 1.1**: The S-EHR app starts the user authentication to the health care system using credentials provided by the citizen. If the authentication succeeds the S-EHR app obtains the authentication token, and shows to the citizen the window for requesting the import options (what type of health data and from which date the import should start).
- **Step 2**: the citizen chooses the needed import options and then starts the process of importing his health data in the S-EHR app.
    - **Step 2.1**: in order to start the import, the S-EHR invokes the method MR2DFactoyImpl.create() to obtain an instance of MR2D.
    - **Step 2.2**: the S-EHR app uses the MR2D to invoke the getAllRecords() method providing the requested import parameters. This method returns an instance of ResourceIterator.
        - **Step 2.2.1**: the MR2D submits the request to the R2D Server and collect the returned results.
- **Step 3**: the S-EHR app uses the iterator returned by the GetAllRecords() method, to iterates over the results retrieved (within this loop the S-EHR app invokes the method next() of the Iterator to gain access to the current item in the Iterator).
- **Step 4**: S-EHR app stores the last item read from the iterator to its internal database and continues to iterate over results until the iterator has been completely fetched.

Interacting with R2D protocol, using M-R2D-E library implies the following mandatory points:

- authenticating user to its National EHR using API of M-R2D-SM library;
- creating instances of MR2D using methods of MR2DFactory class;
- using the MR2D instance obtained by R2Factory for submitting query to the R2D server;
- using ResourceIterator to iterate over results returned by R2D server (health data are retrieved as FHIR resource).

3.1.1.4. *Third Party Libraries*

HAPI FHIR Library

As for the T_D2D_E library, the M-R2D-E uses the HAPI FHIR Library v4.1.0 **[HAPI]** to define model classes for the resource type and datatype defined by the FHIR specification, based on the current data model that is described in D2.8 - FHIR Profile for EHR Interoperability **[D2.8]**. In the case of the M-R2D-E library, the HAPI FHIR Library is being used for transferring FHIR Resources in the form of FHIR objects (e.g. Patient Resource, Practitioner Resource) through the operations offered by the MR2D interface.

eHDSI SDK Library

The eHDSI SDK is being used for interacting with an eHDSI compliant NCP. eHDSI SDK defines data and proxy classes to submit requests to an eHDSI compliant NCP.

3.1.2. R2D Access Library - Internal view

This chapter provides architectural details about the internal structure of the software realizing the library. These internal details do not affect the external usage of the library as it has been described in previous sections, but describe the solutions adopted by the library to face the challenges imposed by the context.

In order to completely understand the rationale behind some design decisions, it is better to recall the role of the library: the M-R2D-E library acts as an intermediate layer between an app and a health care system providing an R2D interface, not only hiding technical details of the two underlying protocols, but also providing some operations not offered natively by the protocols . Furthermore, the M-R2D-E library has been conceived to foster the adoption of R2D from app developers, providing a standard and unified interface to access health care systems of different hospitals.

R2D Access protocol is specified and implemented with FHIR (reference implementation), but it may also be implemented using languages and protocols different from FHIR and HTTP, for instance the eHDSI protocol has many similarities with R2D Access and may be considered as a good candidate to become an alternative implementation of R2D Access. The design of the M-R2D-E library is strongly influenced by this possibility, and many technical decisions (concerning software architecture of the library) have been taken to realize a mobile library able to interact with R2D Access protocols implemented in more than language. The architectural solution described in this section is based upon several internal adapters ad hoc developed for each underlying protocol implementing the MR2D interface. These adapters are called:

- MR2DOverFHIR: this is the MR2D implementation based on FHIR, it uses third party libraries provided by the HAPI FHIR open source project.
- MR2DOverEHDSI: this is the MR2D implementation based on eHDSI, it uses third party libraries containing proxy clients for eHDSI / SOAP protocol.

The following component diagram is an evolution of the one shown before, it contains several additional components used internally by MR2DOverFHIR and MR2DOverEHDSI. These components are: ProgressiveExecutor, EHDSIDAO and FHIRDAO, their role is to support MR2DOverFHIR and MR2DOverEHDSI to execute complex operations with a specific protocol.

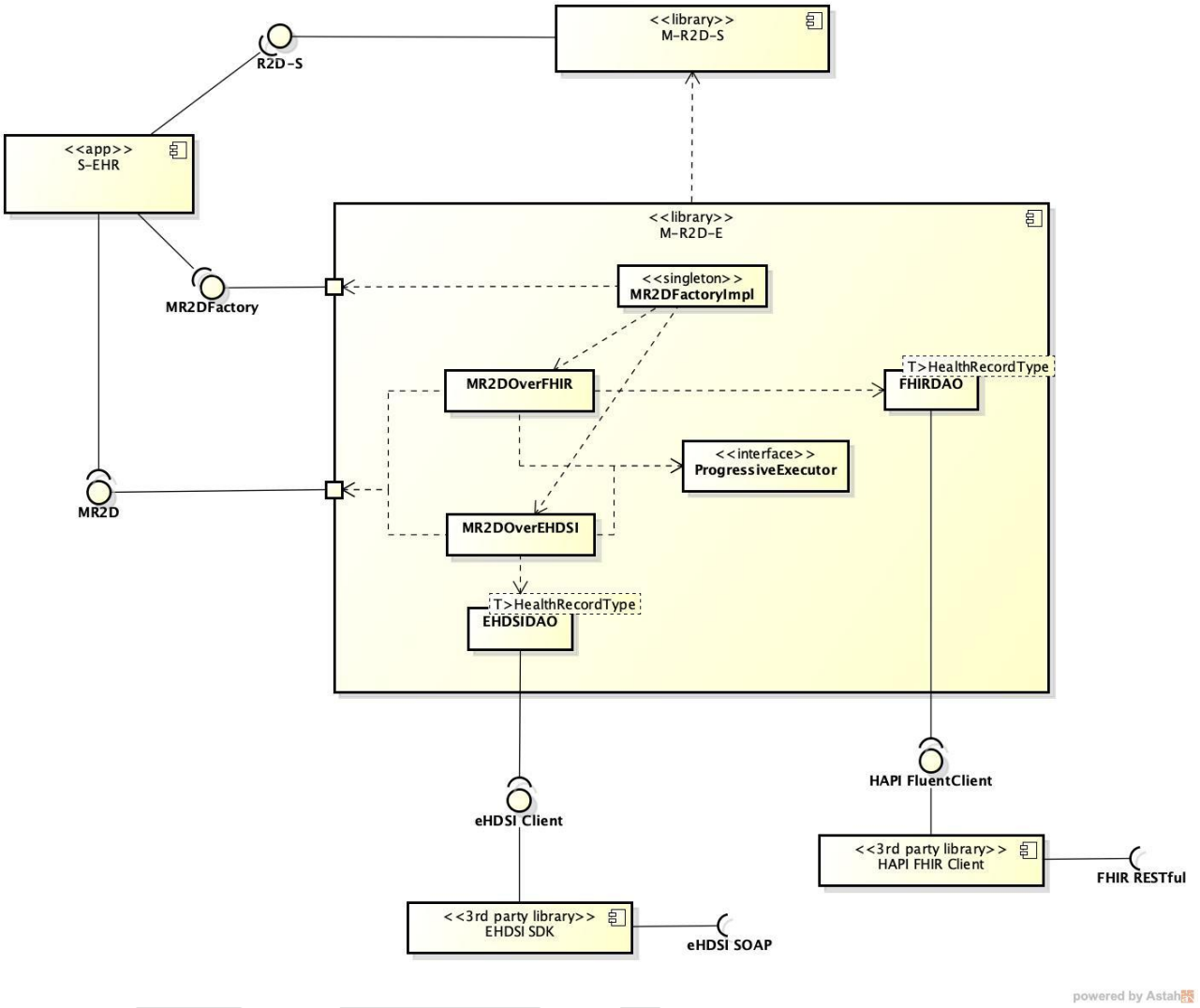

Figure 14 - Component diagram of M-R2D-E

The components shown in the previous diagram are concretely realized by classes, the internal structure of these classes, their interface and the relationships between them are described in the following class diagram.

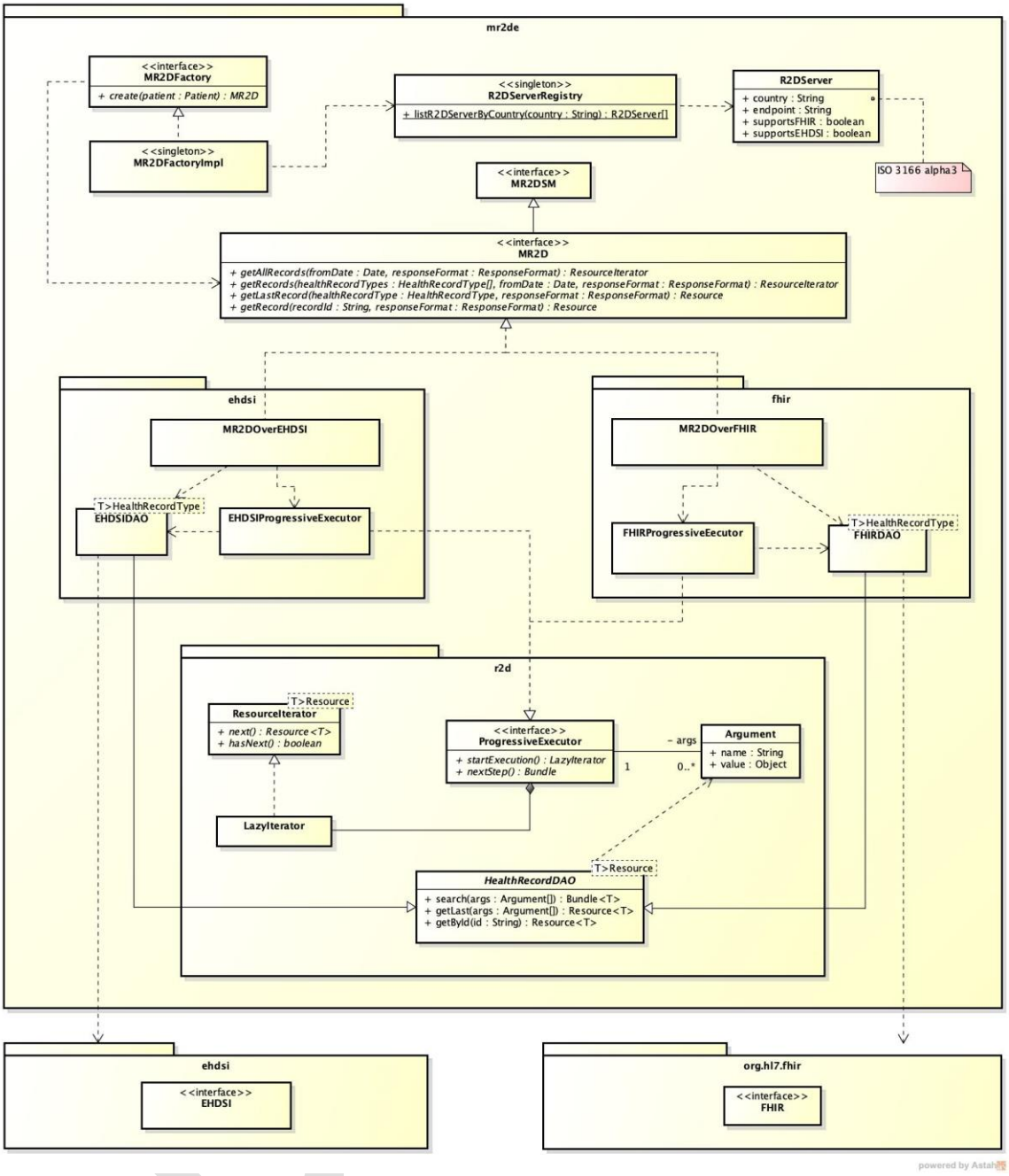

Figure 15 - Class diagram of M-R2D-E library

This diagram is an evolution of all the diagrams shown until now, it shows details about classes used by MR2DOverFHIR and MR2DOverEHDSI to execute complex operations with their protocols.

The main class used by both is the ProgressiveExecutor (package r2d) whose role is to coordinate the execution of operations that requires multiple interactions with the server to retrieve all requested data (for instance when a server provides paged results or when different types of Resource need to be queried). The direct interaction between the executor and the remote server is a responsibility delegated to the HealthRecordDAO class and its descendants (FHIRDAO and EHDSIDAO). As shown in the diagram, DAO classes are Generic classes, every instance is able to handle one specific type of HealthRecordType using one specific protocol, providing concrete methods for: i) searching a set of HealthRecordType, ii)

retrieving the most recent HealthRecordType, iii) getting one specific instance of HealthRecordType by its unique ID.

Complex operations are performed by M-R2D-E with lazy techniques, lazy capabilities are provided by the class LazyIterator and are triggered directly by the S-EHR while iterating over the Iterator. Every time that the Iterator has reached the end of its buffer it asks for the next bunch of data, asking them to the ProgressiveExecutor until the overall operation has been performed and all data have been retrieved. This behaviour and all other complex behaviours of M-R2D-E are described in detail in the next sections (using UML sequence diagrams), accompanied by full description of the interfaces of the used classes. The first sequence diagram shown refers to the internal detail regarding the business logic of instantiation of MR2D:

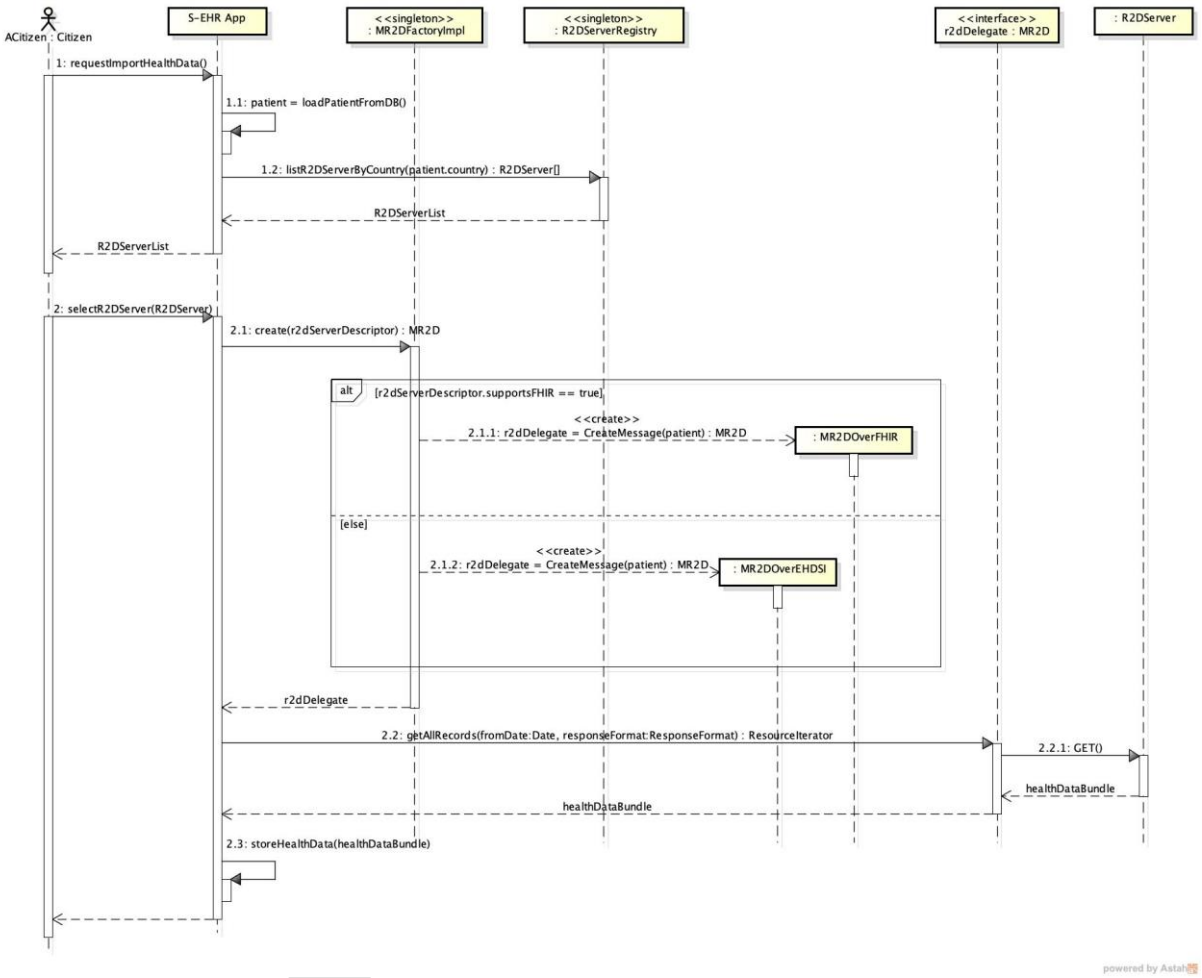

Figure 16 - Sequence diagram of the instantiation of MR2D

- **Step 0** (not shown in the sequence diagram): the citizen has performed the authentication.
- **Step 1**: the Citizen requests the S-EHR App to import his / her health data.
    - **Step 1.1**: the s-EHR app creates an instance of Patient and fills it with patient's data loaded from its internal database.
    - **Step 1.2**: the S-EHR invokes method R2DServerRegistry.listR2DServerByCountry(), providing as input parameter the value of the attribute patient.country. This method returns a list with the registered R2D Servers of the provided country, this list is provided to the Graphical Interface to show the list to the citizen.

- **Step 2**: the citizen selects the R2D Server to import data from and starts the import.
    - **Step 2.1**: the S-EHR app invokes the static method create() of class MR2DFactoryImpl, providing as arguments, the instance of Patient created at step 1.
        - **Step 2.1.1**: optional step executed if the R2D Server supports FHIR API (FHIR is the main choice). The MR2DFactoryImpl creates an instance of class MR2DOverFHIR, providing to the constructor the instance of the patient and returns it to the caller.
        - **Step 2.1.2**: optional step executed if the R2D Server supports EHDIS API (and does not support FHIR API). The MR2DFactoryImpl creates an instance of class MR2DOverEHDSI, providing to the constructor the instance of the patient and returns it to the caller.
    - **Step 2.2**: the S-EHR App invokes method getAllRecords using the instance of MR2D obtained from step 2.1.1.
        - **Step 2.2.1**: the MR2D translates the request into the corresponding operations of the protocol and the submit the request to the R2D Server. The obtained bundle is then returned to the caller.
    - **Step 2.3**: health data are stored locally in the S-EHR App.

So, the MR2D instantiation process, takes into account the capabilities of the referenced R2D Server, if the server supports FHIR, this protocol will be the preferred one, even if the server would implement more than one protocol. At the end of the instantiation process, the S-EHR app receives the instance of a business class implementing the MR2D interface (without knowing what concrete class has been instantiated), and will use it to retrieve data from the R2D Server.

This first sequence diagram has been shown here because it is common for both MR2DOverFHIR and MR2DOverEHDSI, while the next sections will show separate sequence diagrams showing how the same MR2D method is realized with FHIR and with EHDSI. Before going into this description, it is better to have a complete knowledge of the interfaces of ProgressiveExecutir and HealthRecordDAO.

Interface ProgressiveExecutor

ProgressiveExecutor is the interface of a class that is able to manage the execution of complex operations that requires more than one interaction with the R2D Server to be completely executed.

The methods provided by MR2D must be translated by M-R2D-E library in concrete operations provided by the underlying protocols implementing R2D specifications. Covering the gap between how a method is exposed to the S-EHR and how it is realized by the protocol is the core responsibility of the M-R2D-E library. For instance, when the S-EHR App invokes the method:

```
MR2D.getRecords([LABORATORY_RESULT, MEDICAL_IMAGES], 01/01/2020);
```

It is requesting all health data of type Laboratory Result and Medical Images produced after the 1st January 2020. Whether this method can be executed or not with a single request to the R2DServer depends only on the used implementation of the R2D protocol. Using the FHIR implementation of R2D this operation cannot be executed with a single request and is translated in this way:

1. query LABORATORY_RESULT produced after 01/01/2020 of the authenticated citizen;

2. retrieve next page of query started at point 1 until LABORATORY_RESULT have been completely fetched;
3. query MEDICAL_IMAGES produced after 01/01/2020 of the authenticated citizen, otherwise ;
4. retrieve the next page of the query started at point 3, until MEDICAL_IMAGES have been completely fetched, then the overall operation is terminated.

Coordinating a complex operation is the main responsibility of the ProgressiveExecutor, to accomplish its tasks the ProgressiveExecutor collaborates with the LazyIterator, the link between an instance of ProgressiveExecutor and an instance of LazyIterator is created by the ProgressiveExecutor itself and cannot be altered.

This behaviour is described in details in the sequence diagrams contained in the following sections.

Operation ProgressiveExecutor.start()

Name	start
Description	USed to start execution of a complex operation
Arguments	void
Return Value	An instance of LazyIterator
Exceptions	• Security exceptions related to the validation of the session. • Security exceptions related to the ownership of data (only health data of the authenticated citizen can be accessed). • Network exceptions related to failure during remote communication.
Preconditions	• The citizen has successfully executed the authentication using methods provided from the M-R2D-SM library. • The session is still valid.

Operation ProgressiveExecutor.next()

Name	next

Description	used to execute the next part of a complex operation
Arguments	void
Return Value	org.hl7.fhir.model.Bundle
Exceptions	• Security exceptions related to the validation of the session. • Security exceptions related to the ownership of data (only health data of the authenticated citizen can be accessed). • Network exceptions related to failure during remote communication.
Preconditions	• The citizen has successfully executed the authentication using methods provided from the M-R2D-SM library. • The session is still valid.

Class HealthRecordDAO <HealthRecordType>

This abstract class defines the characteristics (interface) of a class whose purpose is to translate and submit requests to an R2D Server using a specific protocol. The set of requests that this class must be able to execute is defined by the methods of its interface: search(), getLast() and getById().

M-R2D-E library needs several concrete implementations of this class in order be able to request all defined requests to an NCP, using FHIR or eHDSI, and for all kind of health data managed (PATIENT_SUMMARY, LABORATORY_RESULT, MEDICAL_IMAGES, PRESCRIPTION, DISCHARGE_REPORT). Searching for Prescriptions in FHIR is quite different from searching Prescriptions on eHDSI, hiding this difference to the ProgressiveExecutor is the purpose of this class.

Independently from the parameters explicitly provided by the citizen, the search scope of every HealthRecordDAO is defined by two implicit and immutable factors: the first corresponds to the Template parameter provided during instantiation (defines the type of health data to be searched), the second corresponds to the authenticated citizen. This means that an implementation of *HealthRecordDAO <PRESCRIPTION>* is only able to search for Prescriptions belonging to the authenticated Citizen even if no parameters have been provided to the method.

Internal Operation search()

Name	search

Description	Submit a search request to the R2D Server using the provided arguments.
Arguments	A set of class Argument corresponding to the parameters provided by the invoking client
Return Value	an instance of org.hl7.fhir.model.Bundle <T>
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 using methods provided from the M-R2D-SM library. • The session is still valid.

Internal Operation getLast()

Name	getLast
Description	Submit a search to the R2D Server, retrieving the most recent instance of a specific kind of health data belonging to the Citizen.
Arguments	A set of class Argument
Return Value	an instance of org.hl7.fhir.model.Resource <T>
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 using methods provided from the M-R2D-SM library. • The session is still valid.

Internal Operation getById()

Name	getById

Description	Submit a search to the R2D Server, retrieving a specific instance of health data belonging to the Citizen and identified by its ID.
Arguments	A set of class Argument
Return Value	an instance of org.hl7.fhir.model.Resource <T>
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 using methods provided from the M-R2D-SM library. • The session is still valid.

The following class diagram shows the concrete DAO hierarchy of the MR2DE library:

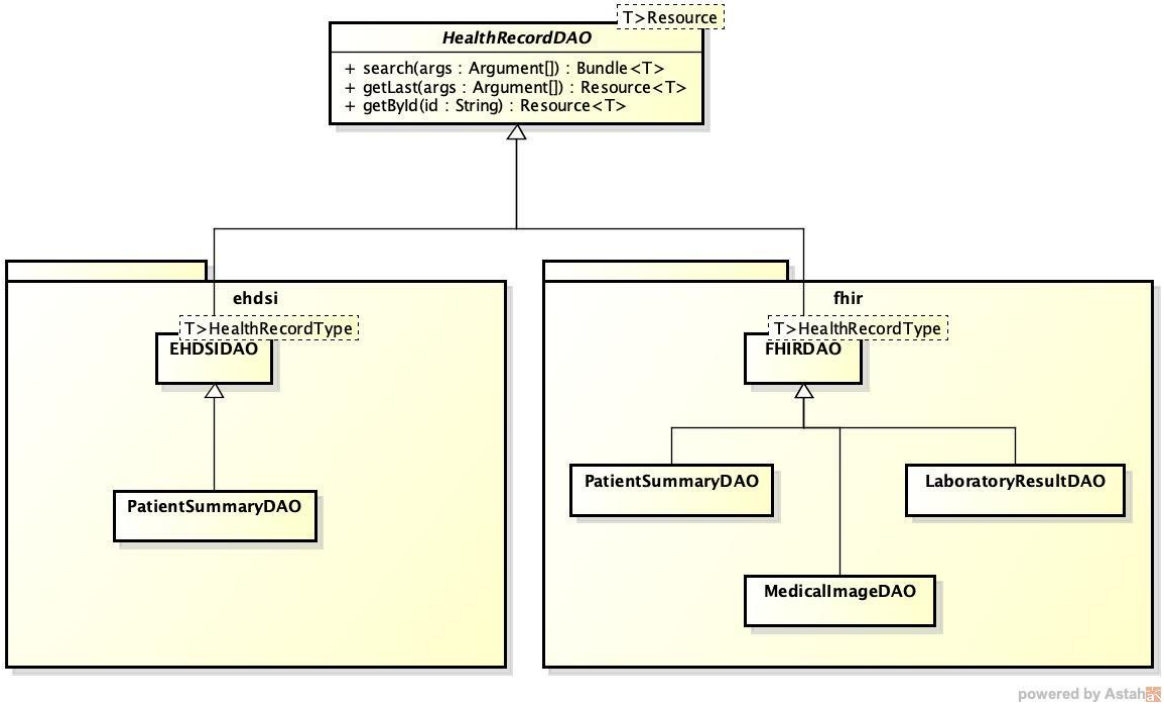

Figure 17 - Concrete DAO hierarchy of the MR2DE library

Class LazyIterator

This class is an implementation of the interface ResourceIterator (described in the previous sections) that loads its data in a lazy way collaborating with LazyQueryExecutor.

3.1.2.1. *MR2DOverFHIR*

This section provides sequence diagrams showing how methods of MR2D are concretely realized by the class MR2DOverFHIR.

Sequence Diagram of method GetAllRecords

This sequence diagram shows the collaborations between classes to execute the method GetAllRecords().

The execution of this method is requested by the S-EHR app to download all types of health data of the authenticated citizen (PATIENT_SUMMARY, LABORATORY_RESULT, MEDICAL_IMAGES, PRESCRIPTION, DISCHARGE_REPORT) in a single request. FHIR specifications does not allow to execute the logical union of several queries, so this union is performed locally (as described before) by M-R2D-E, executing the different queries (that compose the overall result) with lazy techniques, while S-EHR app is iterating over the initial results.

For purposes of simplicity, the following sequence diagram does not show the initial steps (shown before) performed to execute: i) authentication and ii) instantiation of R2D; these steps, even if not shown, must be considered as executed by the Citizen.

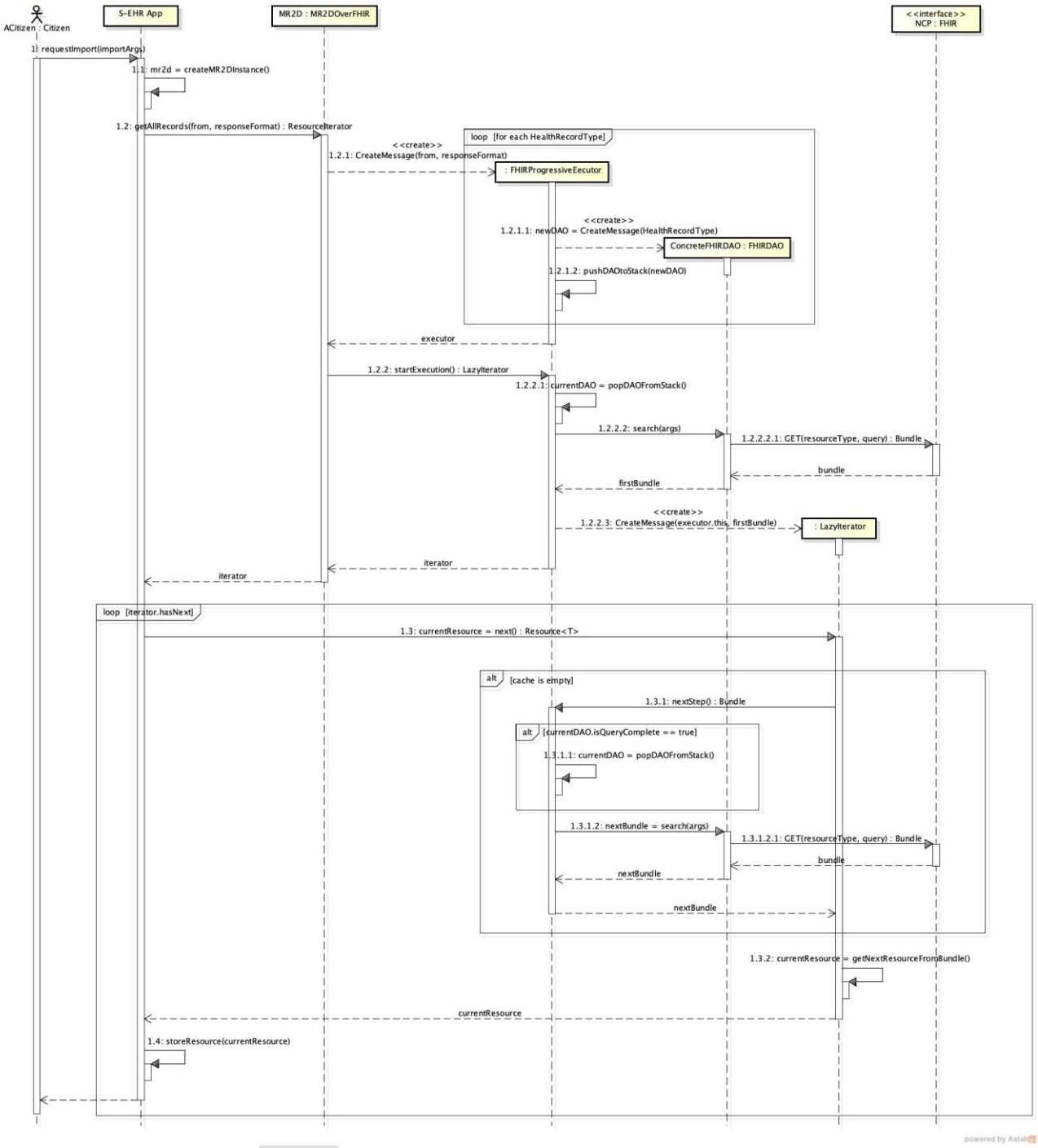

Figure 18 - Sequence diagram of GetAllRecords method executed with FHIR protocol

- **Step 0** (not shown in the sequence diagram): the citizen has performed the authentication.
- **Step 1**: the citizen has requested the S-EHR App the import of his / her health data providing the import options needed by the app.
    - **Step 1.1**: the S-EHR App creates an instance of MR2D as shown in the sequence diagram regarding the instantiation of MR2D. In this case, the library will create an instance of MR2DOverFHIR.
    - **Step 1.2**: the S-EHR app invokes method getAllRecords() on the instance of MR2D obtained at Step 1.1.

- **Step 1.2.1**: the S-EHR App creates an instance of ProgressiveExecutor providing the arguments needed to the constructor.
    - **Step 1.2.1.1**: the ProgressiveExecutor creates an instance of FHIRDAO for each type of health data of Enumeration HealthRecordType.
    - **Step 1.2.1.2**: the created instance of DAO is pushed in a stack (will be used later).
- **Step 1.2.2**: MR2DOverFHIR invokes the method startExecution() on the instance of ProgressiveExecutor created at the step before. The execution of the complex operation has started.
    - **Step 1.2.2.1**: the ProgressiveExecutor retrieves the first available FHIRDAO from the stack and makes it the current DAO.
    - **Step 1.2.2.2**: the ProgressiveExecutor invokes the method search() on the instance of FHIRDAO (retrieved in the step before) providing the necessary arguments.
        - **Step 1.2.2.2.1**: the FHIRDAO creates the corresponding FHIR request and submits it to the R2D Server. The obtained Bundle is returned to the caller;
    - **Step 1.2.2.3**: the ProgressiveExecutor creates an instance of LazyIterator providing the retrieved bundle as input parameter, then the iterator is returned to the initial caller (S-EHR). The first part of the getAllRecords method has been executed, the S-EHR app can now iterate over the results.

- **Step 1.3**: the S-EHR App starts iterating over query results, invoking the method next() of LazyIterator. This step is executed until the LazyIterator has no more records to get (lazyIterator.hasNext() == false).
    - **Step 1.3.1**: this step is executed optionally only if the lazy iterator cache is empty. Every time that the S-EHR App asks for the next item of the iterator, the LazyIterator must check if the item can be retrieved directly from the cache or if the cache needs to be refilled with a new bunch of data retrieved from R2DServer. If the cache is empty, the LazyIterator invokes the method next() of the ProgressiveExecutor to inform it to retrieve the next bundle of health data.
        - **Step 1.3.1.1**: the ProgressiveExecutor asks the currentDAO if the query is complete, if it is completed then the ProgressiveExecutor gets the next DAO from the stack and makes it the current DAO.
        - **Step 1.3.1.2**: the ProgressiveExecutor invokes method search() of the current DAO, requesting the next bundle of data.
            - **Step 1.3.1.2.1**: the FHIRDAO creates the corresponding FHIR request and submits it to the R2D Server. The obtained Bundle is returned to the caller (LazyIterator);
    - **Step 1.3.2**: the LazyIterator extracts the next health record from the bundle stored in the cache and returns it to the S-EHR App.
- **Step 1.4**: the S-EHR app stores the current health record in its database.

Sequence Diagram of method GetLastRecord

This sequence diagram shows the collaborations between components to execute the method GetLastRecord(). The main difference between this method and the GetAllRecord() methods described before, is that GetLastRecord is not executed with a multi steps query, because it is satisfied with just one single remote request to the NCP.

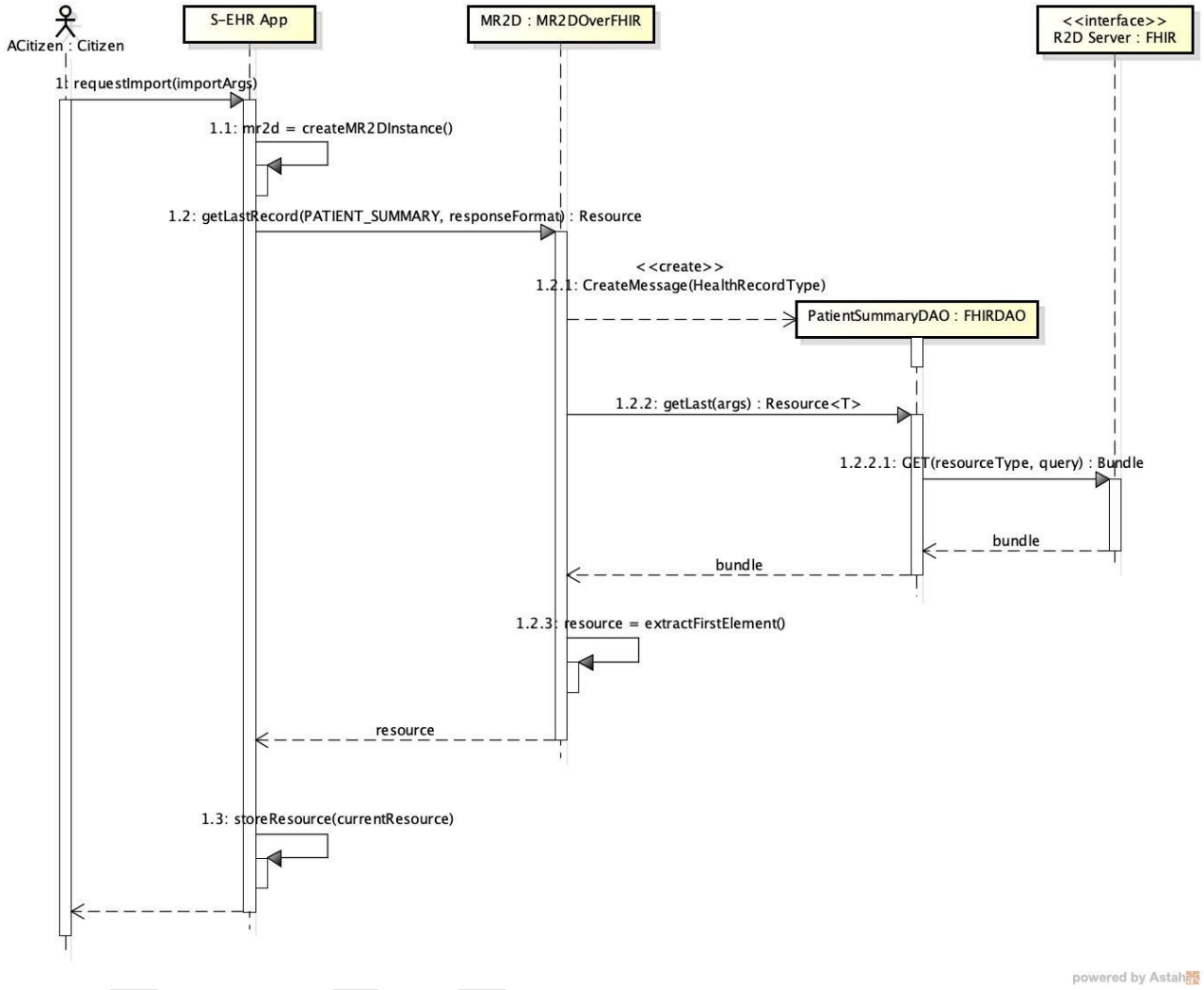

Figure 19 - Sequence diagram of GetLastRecord method executed with FHIR protocol

- **Step 0** (not shown in the sequence diagram): the citizen has performed the authentication.
- **Step 1**: the citizen has requested the S-EHR App the import of his / her health data providing the import options needed by the app.
    - **Step 1.1**: the S-EHR App creates an instance of MR2D as shown in the sequence diagram regarding the instantiation of MR2D. In this case, the library will create an instance of MR2DOverFHIR.

    - **Step 1.2**: the S-EHR app invokes method getLastRecord() on the instance of MR2D previously created, asking for the most recent version of the Patient Summary of the citizen.
        - **Step 1.2.1**: MR2DOverFHIR creates the instance of HealthRecordDAO able to handle the health data of type PATIENT_SUMMARY.

- **Step 1.2.2**: MR2DOverFHIR invokes method getLast() of the class HealthRecordDAO, providing the needed input parameters as an array of Argument.
  - **Step 1.2.2.1**: the HealthRecordDAO creates the corresponding FHIR request and submits it to the R2D server. The obtained Bundle is returned to the caller;
- **Step 1.2.3**: MR2DOverFHIR extracts the first item from the bundle and returns it to the caller.

- **Step 1.3**: the S-EHR app stores the Patient Summary in its database

3.1.2.2. *MR2DOverEHDSI*

This section provides sequence diagrams showing how methods of MR2D are concretely realized by the class MR2DOverEHDSI.

Sequence Diagram of method GetAllRecords

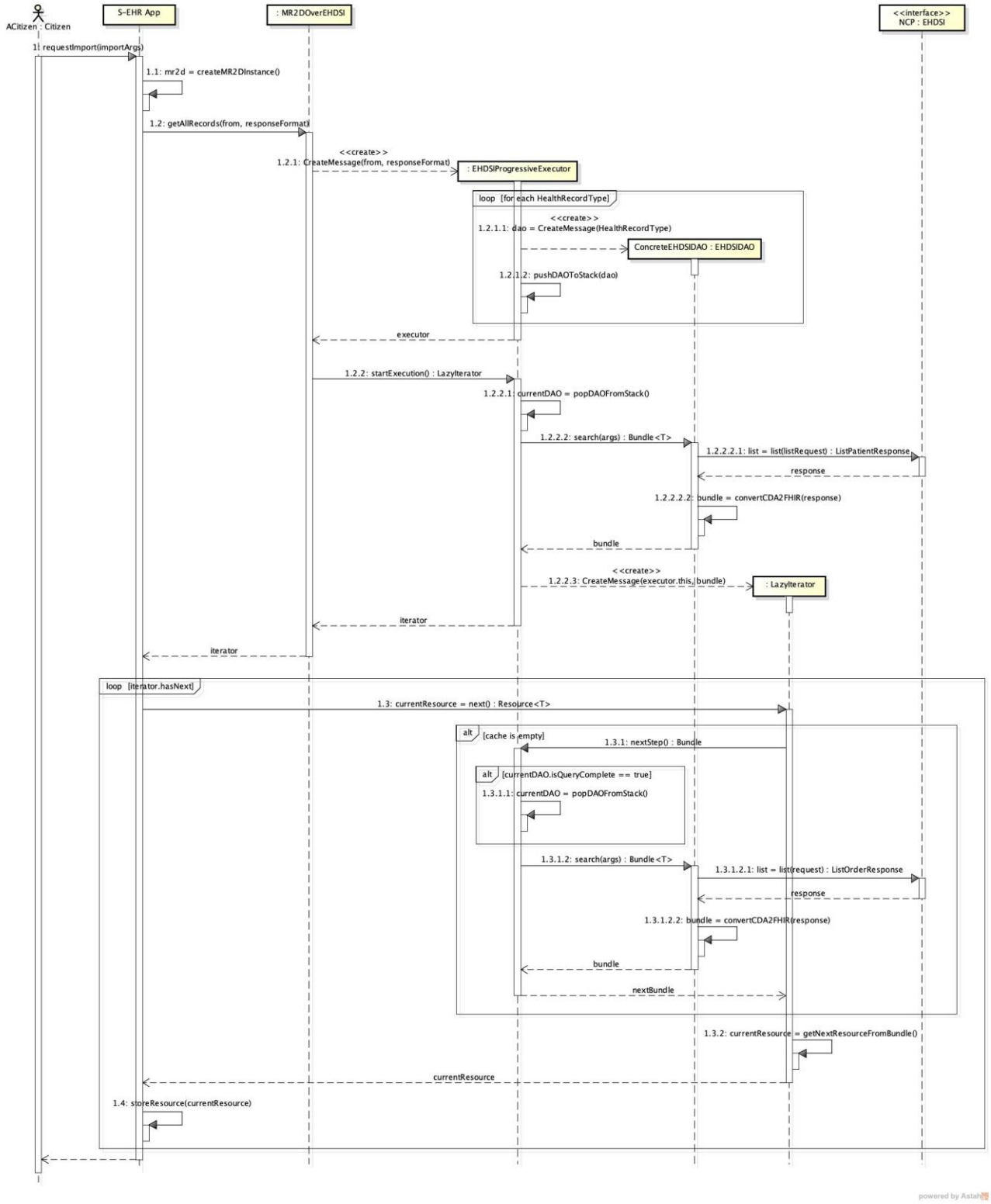

Figure 20 - Sequence diagram of GetAllRecords method executed with EHDSI protocol

- **Step 0** (not shown in the sequence diagram): the citizen has performed the authentication.

- **Step 1**: the citizen has requested the S-EHR App the import of his / her health data providing the import options needed by the app.
    - **Step 1.1**: the S-EHR App creates an instance of MR2D as shown in the sequence diagram regarding the instantiation of MR2D. In this case, the library will create an instance of MR2DOverEHDSI.
    - **Step 1.2**: the S-EHR app invokes method getAllRecords() on the instance of MR2D obtained at Step 1.1.
        - **Step 1.2.1**: the S-EHR App creates an instance of ProgressiveExecutor providing the arguments needed to the constructor.
            - **Step 1.2.1.1**: the ProgressiveExecutor creates an instance of EHDSIDAO for each type of health data of Enumeration HealthRecordType.
            - **Step 1.2.1.2**: the created instance of DAO is pushed in a stack (will be used later).
        - **Step 1.2.2**: MR2DOverEHDSI invokes the method startExecution() on the instance of ProgressiveExecutor created at the step before. The execution of the complex operation has started.
            - **Step 1.2.2.1**: the ProgressiveExecutor retrieves the first available EHDSIDAO from the stack and makes it the current DAO.
            - **Step 1.2.2.2**: the ProgressiveExecutor invokes the method search() on the instance of EHDSIDAO (retrieved in the step before) providing the necessary arguments.
                - **Step 1.2.2.2.1**: the EHDSIDAO creates the corresponding EHDSI request and submits it to the R2D Server.
                - **Step 1.2.2.2.2**: The list of health data obtained from the step before is converted from CDA to FHIR and then returned to the caller;
            - **Step 1.2.2.3**: the ProgressiveExecutor creates an instance of LazyIterator providing the retrieved bundle as input parameter, then the iterator is returned to the initial caller (S-EHR). The first part of the getAllRecords method has been executed, the S-EHR app can now iterate over the results.
- **Step 1.3**: the S-EHR App starts iterating over query results, invoking the method next() of LazyIterator. This step is executed until the LazyIterator has no more records to get (lazyIterator.hasNext() == false).
    - **Step 1.3.1**: this step is executed optionally only if the lazy iterator cache is empty. Every time that the S-EHR App asks for the next item of the iterator, the LazyIterator must check if the item can be retrieved directly from the cache or if the cache needs to be refilled with a new bunch of data retrieved from R2DServer. If the cache is empty, the LazyIterator invokes the method next() of the ProgressiveExecutor to inform it to retrieve the next bundle of health data.
        - **Step 1.3.1.1**: the ProgressiveExecutor asks the currentDAO if the query is complete, if it is completed then the ProgressiveExecutor gets the next DAO from the stack and makes it the current DAO.

        - **Step 1.3.1.2**: the ProgressiveExecutor invokes method search() of the current DAO, requesting the next bundle of data.
            - **Step 1.3.1.2.1**: the EHDSIDAO creates the corresponding EHDSI request and submits it to the R2D Server.
            - **Step 1.3.1.2.2**: The list of health data obtained from the step before is converted from CDA to FHIR and then returned to the caller;
    - **Step 1.3.2**: the LazyIterator extracts the next health record from the bundle stored in the cache and returns it to the S-EHR App.
- **Step 1.4**: the S-EHR app stores the current health record in its database.

Sequence Diagram of method GetLastRecord

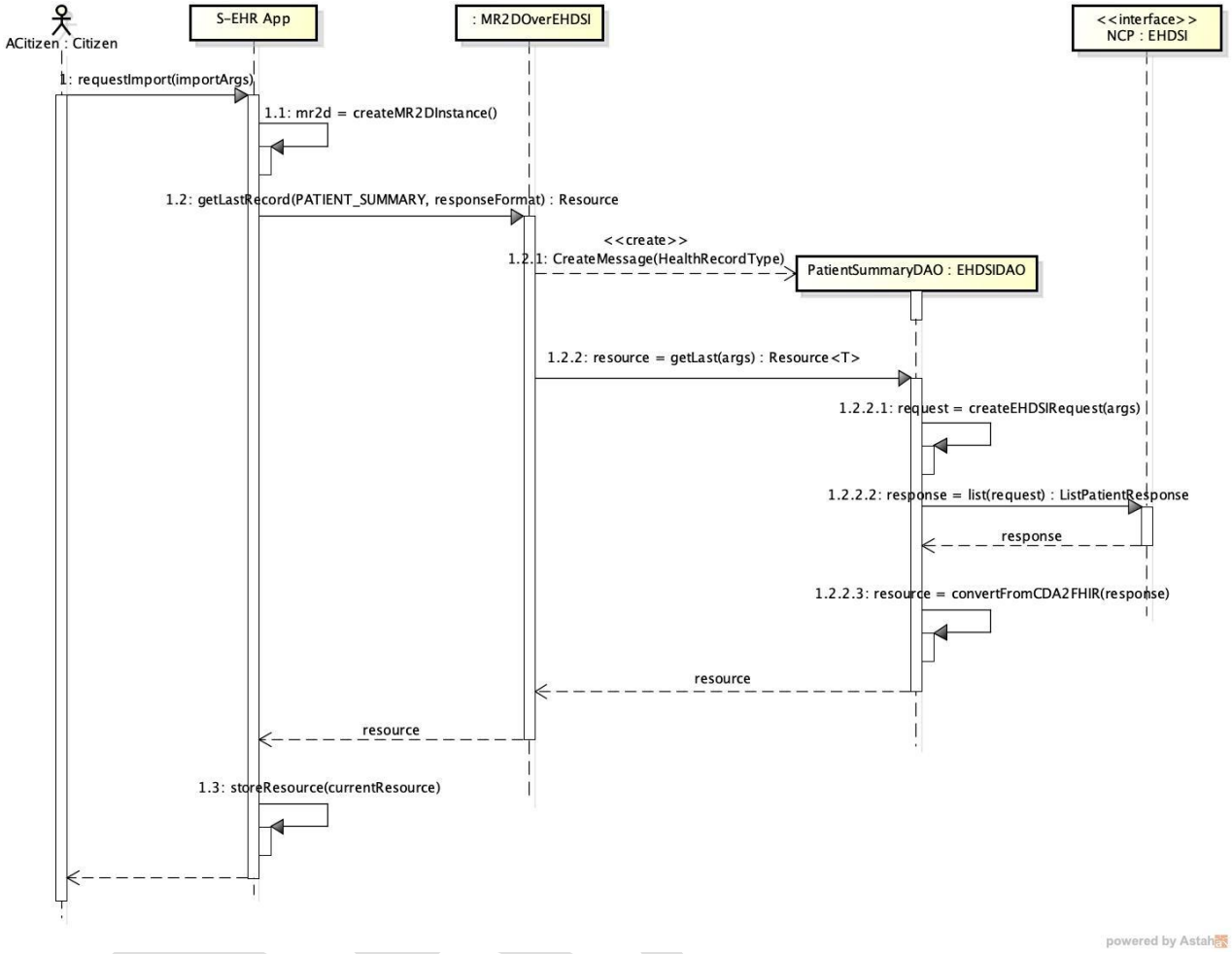

Figure 21 - Sequence diagram of GetLastRecord method executed with EHDSI protocol

- **Step 0** (not shown in the sequence diagram): the citizen has performed the authentication.
- **Step 1**: the citizen has requested the S-EHR App the import of his / her health data providing the import options needed by the app.
    - **Step 1.1**: the S-EHR App creates an instance of MR2D as shown in the sequence diagram regarding the instantiation of MR2D. In this case, the library will create an instance of MR2DOverEHDSI.
    - **Step 1.2**: the S-EHR app invokes the method getLastRecord() on the instance of MR2D previously created.
        - **Step 1.2.1**: MR2DOverEHDSI creates and configures an instance of PatientSummaryDAO to execute the specified request.
        - **Step 1.2.2**: MR2DOverEHDSI invokes method getLast() of the class HealthRecordDAO, providing the needed input parameters as an array of Argument.
            - **Step 1.2.2.1:** the PatientSummaryDAO creates the corresponding EHDSI request.

- **Step 1.2.2.2**: the PatientSummaryDAO submits the EHDSI request to the R2D Server.
- **Step 1.2.2.3**: the PatientSummaryDAO converts the CDA results in FHIR then the obtained Bundle is returned to the caller.

- **Step 1.3**: the S-EHR app stores the resource in its database

4. D2D AND R2D DATA MODEL

The data model used by the D2D and R2D libraries is mostly composed bydata compliant to FHIR specifications. Aside from some primitive types, operations of D2D and R2D mainly manage data that belong to subclasses (FHIR data model supports hierarchy) of *org.hl7.fhir.model.Resource*, that is the base class of every data handled by the FHIR API. The FHIR data model represents the common layer between D2D and R2D protocols, where they both allow exchange of health data represented as FHIR resources. The following class diagram shows in a simplified schema the FHIR resources used in the exchange of health data executed within the D2D and R2D protocols. More details regarding an initial version of the Data Model and the Interoperability Profile that is used in the InteropEHRate project can be found in D2.8 - FHIR profile for EHR interoperability - V2 [D2.8].

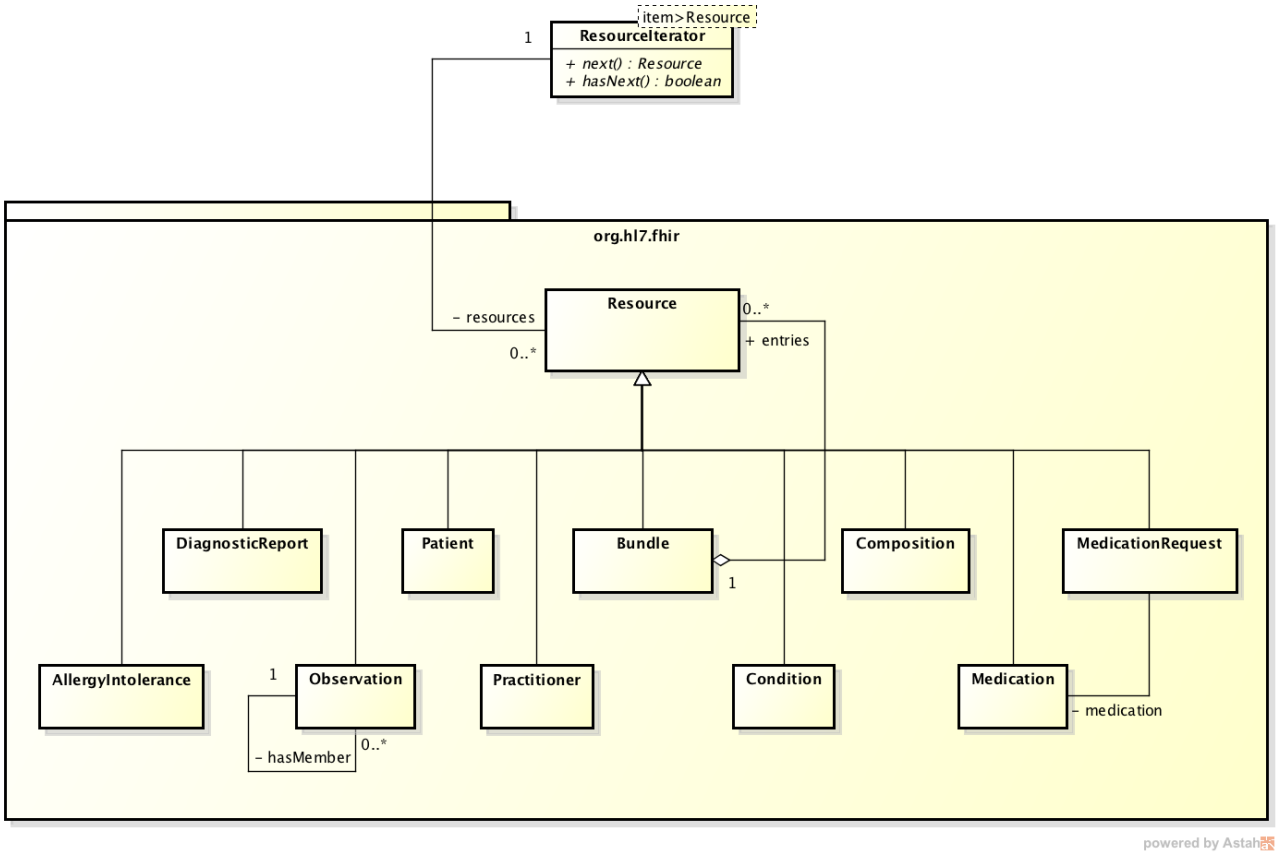

Figure 22 - D2D and R2D Data Model

Although the complete FHIR specifications can be found in **[FHIR]**, the following table contains a short description (extracted from FHIR specifications) of each class represented in the previous class diagram.

Resource	A resource is an entity that: • represents a medical or administrative data; • has a known identity (a URL) by which it can be addressed; • identifies itself as one of the types of resources defined in this specification; • contains a set of structured data items as described by the definition of the resource type; • has an identified version that changes if the contents of the resource change; It is the base class of all kinds of data that can be exchanged using FHIR.
Practitioner	FHIR resource that represents a person who is directly or indirectly involved in the provisioning of healthcare.
Patient	FHIR resource that contains demographics and other administrative information about an individual or animal receiving care or other health-related services.
DiagnosticReport	FHIR resource that contains findings and interpretation of diagnostic tests performed on patients or groups of patients. The report includes clinical context such as requesting and provider information, and some mix of atomic results, images, textual and coded interpretations, and formatted representation of diagnostic reports.
Observation	FHIR resource that represents a measurement or simple assertions made about a patient, device or other subject.
AllergyIntolerance	FHIR resource that represents a risk of harmful or undesirable, physiological response which is unique to an individual and associated with exposure to a substance.
Medication	FHIR resource primarily used for the identification and definition of a medication for the purposes of prescribing, dispensing, and administering a medication as well as for making statements about medication use.
MedicationRequest	FHIR resource that represents an order or request for both supply of the medication and the instructions for administration of the medication to a patient.
Condition	FHIR resource that represents a clinical condition, problem, diagnosis, or other event, situation, issue, or clinical concept that has risen to a level of concern.

Composition	FHIR resource that represents a set of healthcare-related information that is assembled together into a single logical package that provides a single coherent statement of meaning, establishes its own context and that has clinical attestation with regard to who is making the statement.
Bundle	A container for a collection of resources.

5. CONCLUSIONS AND NEXT STEPS

The objective of this report is to deliver the second version of the design of the libraries offered by the InteropEHRate Framework as a reference implementation of the device-to-device (D2D) and the remote-to-device (R2D) health record exchange protocols. To this end, this document presents a second version of the intended content of the libraries and their further functionality purposes. Following this version, one last update of this report is planned to be released.

With regards to the D2D libraries, the next planned version will contain a major update in the design of these libraries, since it has been identified that the APIs of the D2D and the R2D Access libraries should be more aligned, and have more similarities among them. These similarities will refer mainly to the operations and the listeners that will be offered from the side of the D2D libraries, for exchanging the requested data between the S-EHR app and the HCP app. In that way, the two applications will follow a client-server architecture, where the HCP app will have the role of the client that will send requests to the S-EHR app that will have the role of the server. This change will give to the HCP App the possibility to download from the S-EHR app only specific health data selected by the HCP, to limit the amount of exchange data, and will give the possibility to send new health data to the S-EHR App also without downloading any health data. The increased similarity among the APIs will make it easier for developers to implement additional functionalities or even reuse the D2D and R2D libraries, in a similar way, reducing the complexity of understanding the differences and the similarities in the offered APIs.

With regards to the R2D libraries, the next steps include minor updates regarding the R2D Access protocol, and more specifically the adding of classes for handling health data which is of type Prescriptions and Discharge Report. Furthermore, the support for java.nio.Stream will be added in order to provide a smarter way to browse health data retrieved with queries. The main novelty about the next version of R2D libraries will be the support for the other two remote protocols that are currently under development: R2D Backup and R2D Emergency, allowing respectively a S-EHR App and a HCP App to interact with a S-EHR Cloud.

The final update is planned to be released in December 2021 ([D4.6]), including the aforementioned major updates, of both the libraries that implement the operations for the communication between the involved applications, either for the purposes of the D2D or the R2D protocols.

REFERENCES

- **[D2.8]** InteropEHRate Consortium, *D2.8 - FHIR profile for EHR interoperability - V2*, 2020. www.interopehrate.eu/resources
- **[D3.10]** InteropEHRate Consortium, *D3.10 - Design of libraries for HR security and privacy services - V2*, 2020. www.interopehrate.eu/resources
- **[D4.2]** InteropEHRate Consortium, *D4.2 - Specification of remote and D2D protocol and APIs for HR exchange - V2*, 2020. www.interopehrate.eu/resources
- **[D4.4]** InteropEHRate Consortium, *D4.4 - Design of libraries for remote and D2D HR Exchange - V2*, 2020. www.interopehrate.eu/resources
- **[D4.6]** InteropEHRate Consortium, *D4.6 - Design of libraries for remote and D2D HR Exchange - V3*, 2021. www.interopehrate.eu/resources
- **[HAPI]** HAPI FHIR Library, Website: https://hapifhir.io/
- **[FHIR]** HL7 FHIR specifications, Website: http://hl7.org/fhir/
- **[ANDROID BLUETOOTH]** Android Bluetooth API, Website: https://developer.android.com/guide/topics/connectivity/bluetooth
- **[BLUECOVE]** Bluecove Library, Website: http://www.bluecove.org/