D4.4

Design of libraries for remote and D2D HR exchange - V1

ABSTRACT

This deliverable describes the initial 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 two libraries, including their interactions with the involved applications, in terms of how the libraries can be used by any S-EHR app, HCP App, as well as by any NCP 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	October 31st 2019
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, Argyro Mavrogiorgou	UPRC
Contributors	Francesco Torelli, Alessio Graziani	ENG
Reviewers	Chrysostomos Symvoulidis	BYTE
Reviewers	Nicu Jalba	SIVECO

LOGTABLE

Version	Date	Change	Author	Partner
0.1	2019-08-23	Provided Initial ToC	Thanos Kiourtis, Argyro Mavrogiorgou	UPRC
0.2	2019-09-13	Updated ToC and text, based on internal review.	Thanos Kiourtis, Argyro Mavrogiorgou, Francesco Torelli, Alessio Graziani	UPRC, ENG
0.3	2019-09-23	Provided input regarding the D2D libraries for both sides	Thanos Kiourtis, Argyro Mavrogiorgou, Dimitris Laliotis, Konstantinos Vidakis	UPRC
0.4	2019-09-30	Updated Abstract Section, Conclusions Section, References and Proofread Section 1 and Section 2.	Thanos Kiourtis, Argyro Mavrogiorgou, Konstantinos Vidakis	UPRC
0.5	2019-11-10	Provided input regarding the R2D libraries and the D2D and R2D Data Model.	Thanos Kiourtis, Argyro Mavrogiorgou, Konstantinos Vidakis, Alessio Graziani	UPRC, ENG
0.6	2019-11-20	Finalized input, proof-read and sent for internal review	Thanos Kiourtis, Argyro Mavrogiorgou,	UPRC, ENG

			Konstantinos Vidakis, Alessio Graziani	
0.7	2019-10-22	Internal review	Chrysostomos Symvoulidis	BYTE
0.8	2019-10-23	Internal review	Nicu Jalba	SIVECO
0.9	2019-10-24	Updates based on internal review	Thanos Kiourtis, Argyro Mavrogiorgou, Konstantinos Vidakis, Alessio Graziani	UPRC, ENG
1.0	2019-10-29	Quality Review	Thanos Kiourtis, Argyro Mavrogiorgou, Konstantinos Vidakis, Alessio Graziani	UPRC, ENG
VFinal	2019-10-31	Final review and version for submission	Laura Pucci	ENG

ACRONYMS

Acronym	Term and definition
D2D	Device-to-Device
R2D	Remote-to-Device
HER	Electronic Health Record
S-HER	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

TABLE OF CONTENT

LIST OF FIGURES

1. INTRODUCTION

1.1. Scope of the document

The main goal of this document is to deliver the initial 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 apps 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 D3.9 - Design of libraries for HR security and privacy services - V1 [**D3.9**]. 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. Updated with respect to previous version (if any)

Since this document is the initial version of the design of the libraries for D2D and R2D health record exchange, there does not exist any update with regards to previous related documents.

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 D4.1 - Specification of remote and D2D protocol and APIs for HR exchange V1 **[D4.1]**. 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 bluetooth 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 bluetooth operations and services by the S-EHR app for interconnecting, exporting messages and receiving requests from the HCP app, while the second interface (TD2DI) will be responsible for offering the bluetooth operations and services by the HCP app for similar types of tasks from the S-EHR app. 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.

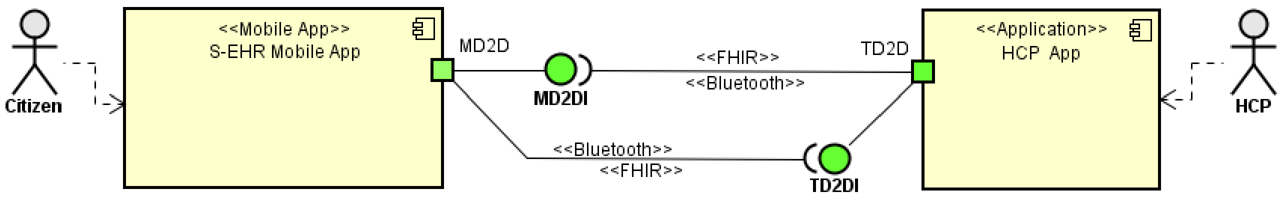


Figure 1 - D2D protocol interfaces

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.

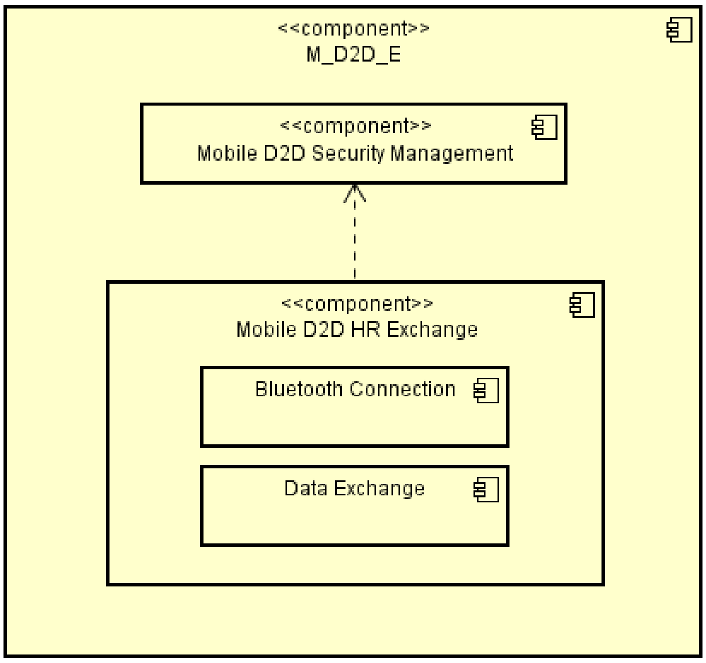


Figure 2 - M-D2D-E Public Java Components

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.

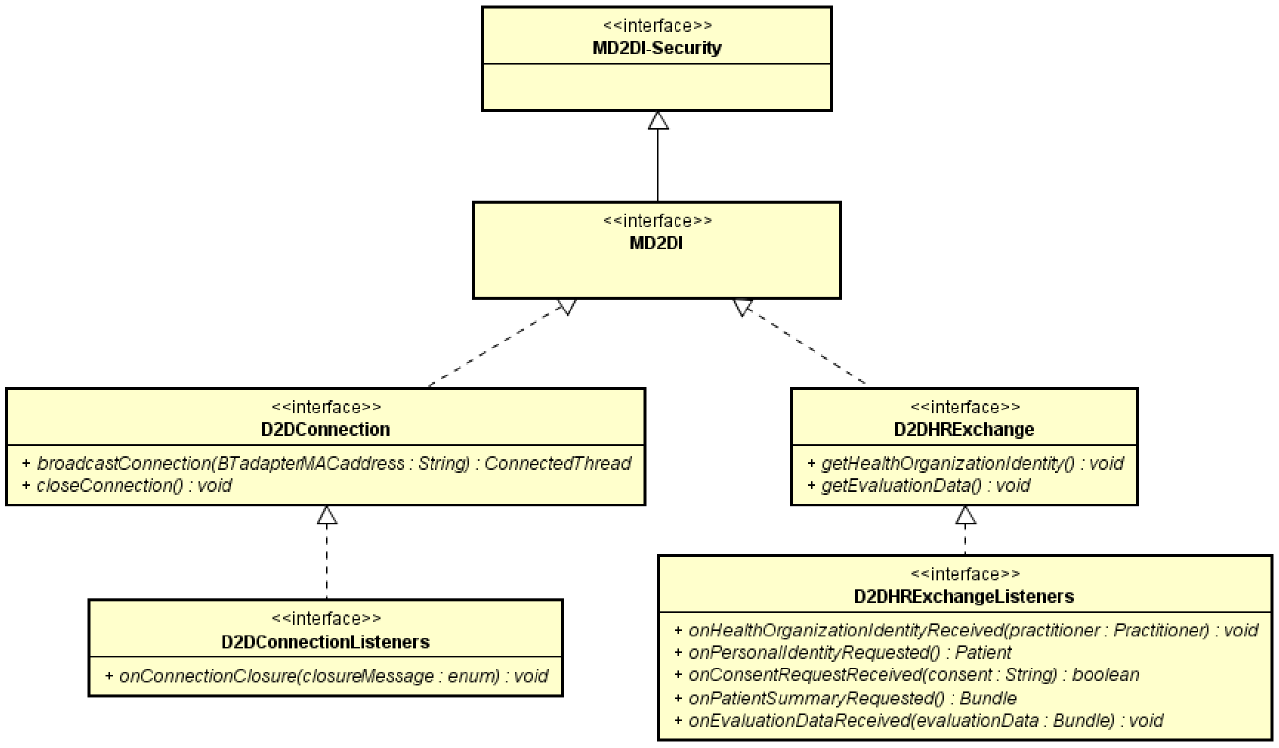

<<interface>>
MD2DI-Security
<<interface>>
MD2DI
<<interface>>
D2DConnection
+ broadcastConnection(BTadapterMACaddress : String) : ConnectedThread
+ closeConnection() : void
<<interface>>
D2DHRExchange
+ getHealthOrganizationIdentity() : void
+ getEvaluationData() : void
<<interface>>
D2DConnectionListeners
+ onConnectionClosure(closureMessage : enum) : void
<<interface>>
D2DHRExchangeListeners
+ onHealthOrganizationIdentityReceived(practitioner : Practitioner) : void
+ onPersonalIdentityRequested() : Patient
+ onConsentRequestReceived(consent : String) : boolean
+ onPatientSummaryRequested() : Bundle
+ onEvaluationDataReceived(evaluationData : Bundle) : void

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 D3.9 - Design of libraries for HR security and privacy services - V1 **[D3.9]**. 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.

(a) 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.
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.

(b) 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	● closureMessage: a coded message that represents details about the reason the connection closure happened.
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.

(c) 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 getEvaluationData

Name	GetEvaluationData
Description	This operation is invoked by the S-EHR app for requesting the evaluation data of the patient summary from the HCP.
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.

(d) 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 onConsentRequestReceived

Name	onConsentRequestReceived
Description	This operation is invoked by the D2D library for getting the consent details from the HCP App and requesting the consent response from the S-EHR app.
Arguments	● consent: a string that represents all the information a citizen needs to give consent for his data
Return Value	This operation will return a boolean that represents the answer regarding the case that the citizen has approved the consent to provide her data or not.

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 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 onEvaluationDataReceived

Name	onEvaluationDataReceived
Description	This operation is invoked by the D2D library for informing the S-EHR app that the evaluation data of the patient summary has been received from the side of the HCP app.
Arguments	● evaluationData: evaluation 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.

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.

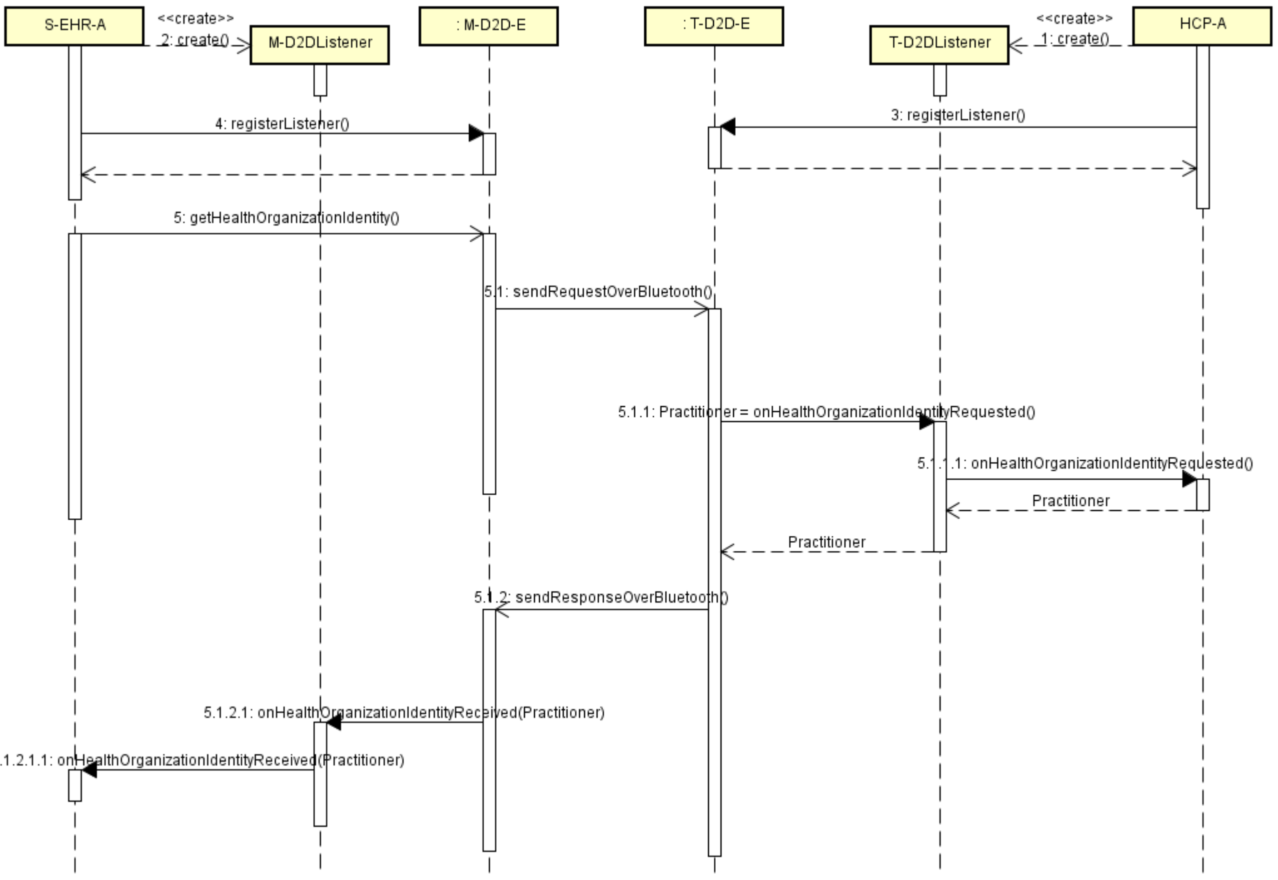

S-EHR-A
<<create>>
2: create()
M-D2DListener
: M-D2D-E
: T-D2D-E
T-D2DListener
<<create>>
1: create()
HCP-A
4: registerListener()
3: registerListener()
5: getHealthOrganizationIdentity()
5.1: sendRequestOverBluetooth()
5.1.1: Practitioner = onHealthOrganizationIdentityRequested()
5.1.1.1: onHealthOrganizationIdentityRequested()
Practitioner
Practitioner
5.1.2: sendResponseOverBluetooth()
5.1.2.1: onHealthOrganizationIdentityReceived(Practitioner)
5.1.2.1.1: onHealthOrganizationIdentityReceived(Practitioner)

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 D2.7 - Interoperability Profile Implementable Level Specification **[D2.7]**. 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.

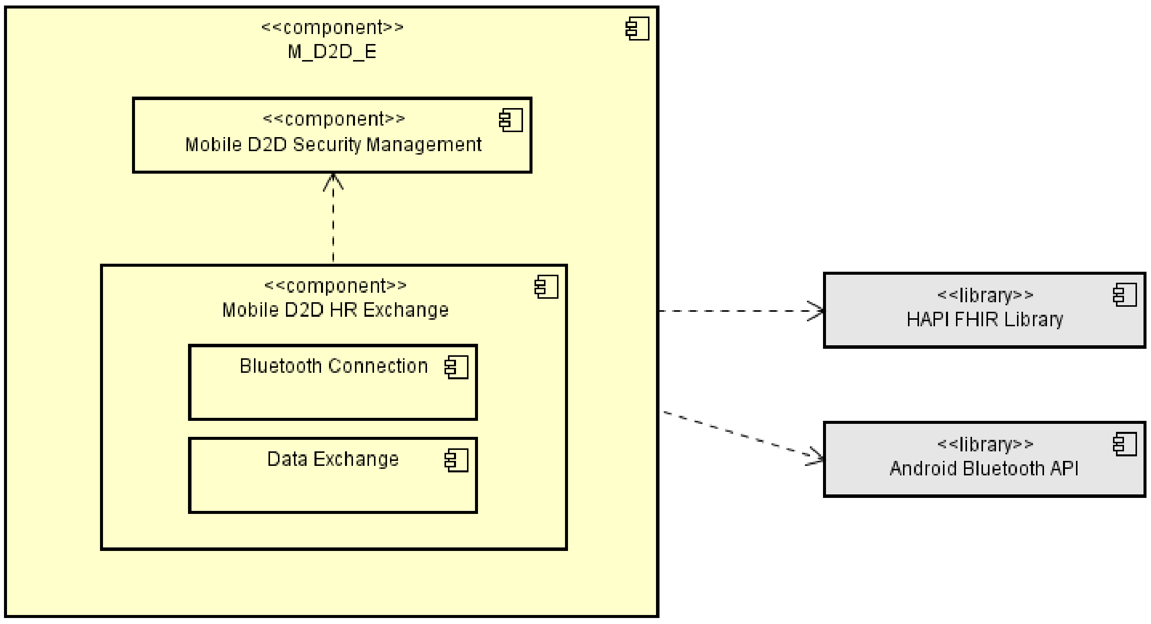


Figure 5 - M-D2D-E Third Party Libraries

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.

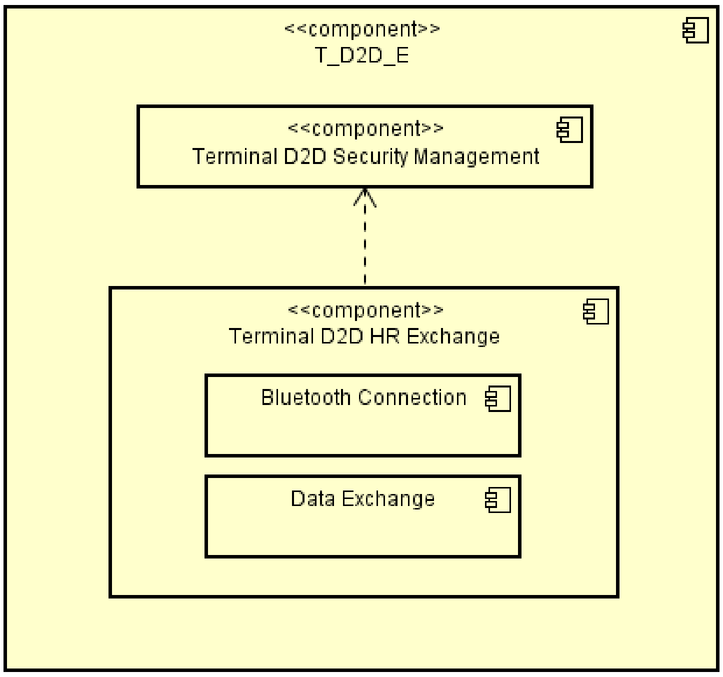


Figure 6 - T-D2D-E Public Java Components

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.

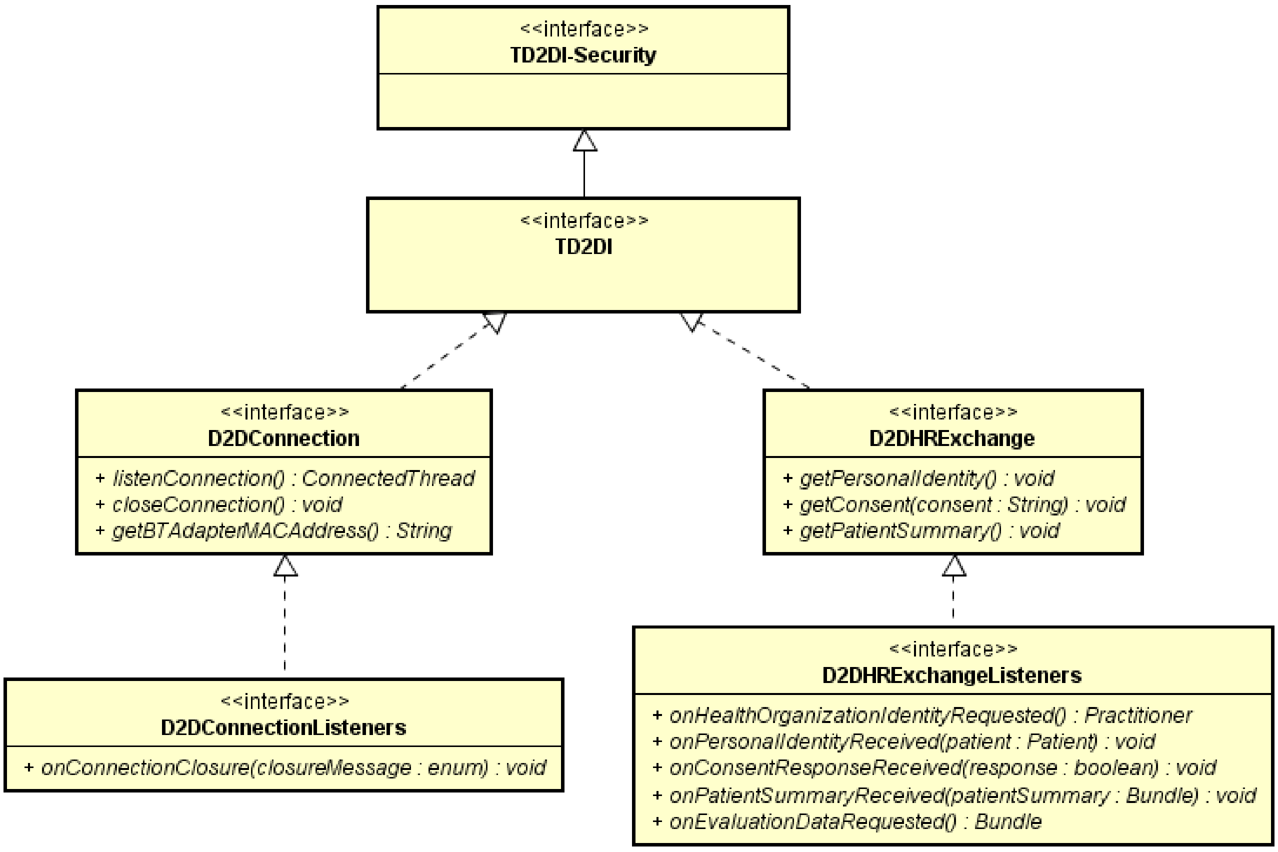

<<interface>>
TD2DI-Security
<<interface>>
TD2DI
<<interface>>
D2DConnection
+ listenConnection() : ConnectedThread
+ closeConnection() : void
+ getBTAdapterMACAddress() : String
<<interface>>
D2DHRExchange
+ getPersonalIdentity() : void
+ getConsent(consent : String) : void
+ getPatientSummary() : void
<<interface>>
D2DConnectionListeners
+ onConnectionClosure(closureMessage : enum) : void
<<interface>>
D2DHRExchangeListeners
+ onHealthOrganizationIdentityRequested() : Practitioner
+ onPersonalIdentityReceived(patient : Patient) : void
+ onConsentResponseReceived(response : boolean) : void
+ onPatientSummaryReceived(patientSummary : Bundle) : void
+ onEvaluationDataRequested() : Bundle

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 **[D3.9]**. 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.

(a) 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	-
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.

(b) 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	● closureMessage: a coded message that represents details about the reason the connection closure happened.
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.

(c) 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 getConsent

Name	getConsent
Description	This operation is invoked by the HCP App for requesting the consent from the citizen.
Arguments	● consent: a string that represents all the information a citizen needs to give consent for his data.
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.

(d) 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 onConsentResponseReceived

Name	onConsentResponseReceived
Description	This operation is invoked by the D2D library for informing the HCP app that the response on consent of the citizen has been received from the side of the S-EHR app.
Arguments	● response: a boolean that represents if the patient gives consent to send his data or not.
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 onEvaluationDataRequested

Name	onEvaluationDataRequested
Description	This operation is invoked by the D2D library for getting the evaluation data from the HCP app.
Arguments	-
Return Value	This operation will return the evaluation data in the form of a Bundle (i.e. FHIR

	Resource Bundle) containing observations of the Patient Summary.
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.

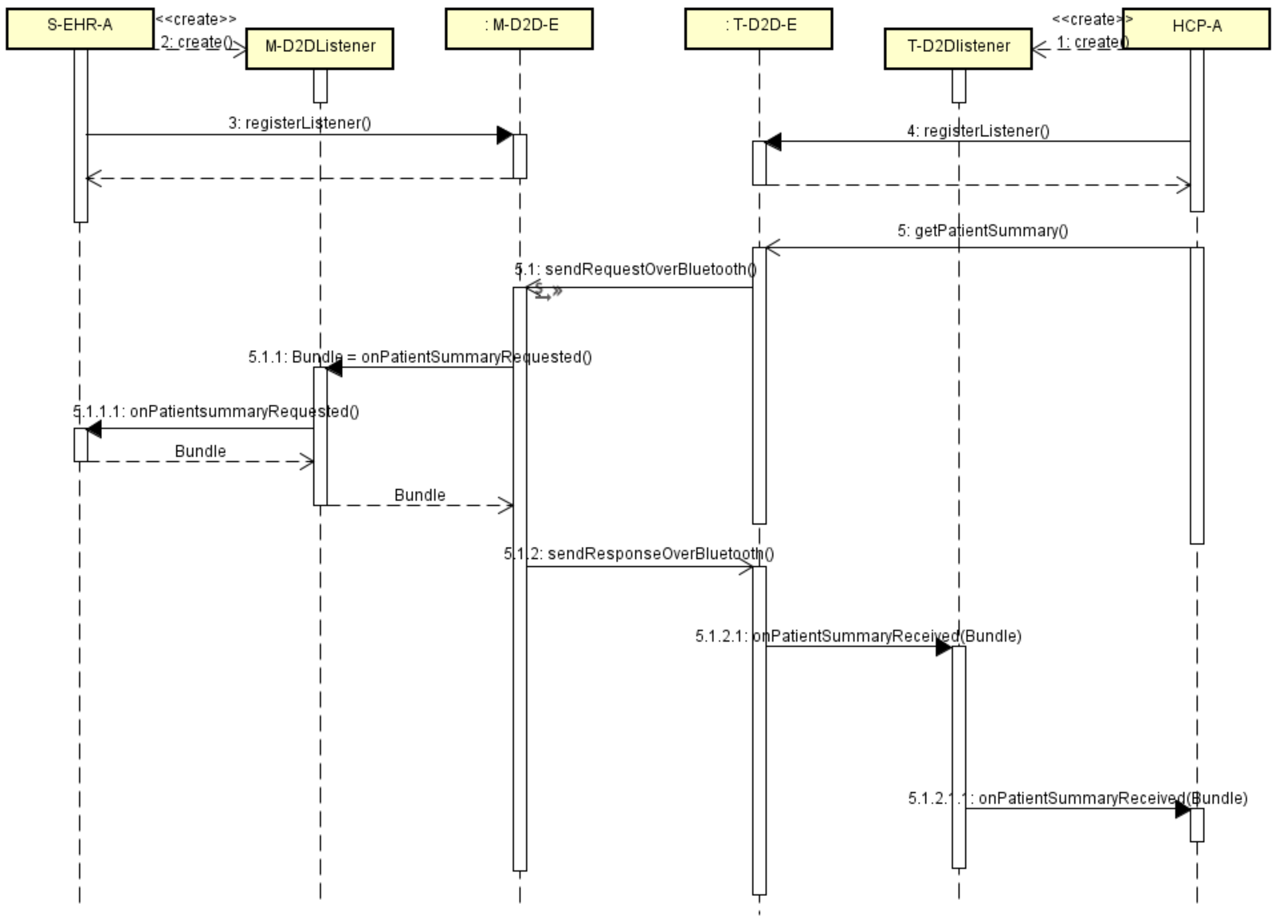

S-EHR-A
<<create>>
2: create()
M-D2DListener
: M-D2D-E
: T-D2D-E
T-D2Dlistener
<<create>>
1: create()
HCP-A
3: registerListener()
4: registerListener()
5: getPatientSummary()
5.1: sendRequestOverBluetooth()
5.1.1: Bundle = onPatientSummaryRequested()
5.1.1.1: onPatientsummaryRequested()
Bundle
Bundle
5.1.2: sendResponseOverBluetooth()
5.1.2.1: onPatientSummaryReceived(Bundle)
5.1.2.1.1: onPatientSummaryReceived(Bundle)

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).

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 D2.7 - Interoperability Profile Implementable Level Specification **[D2.7]**. 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.

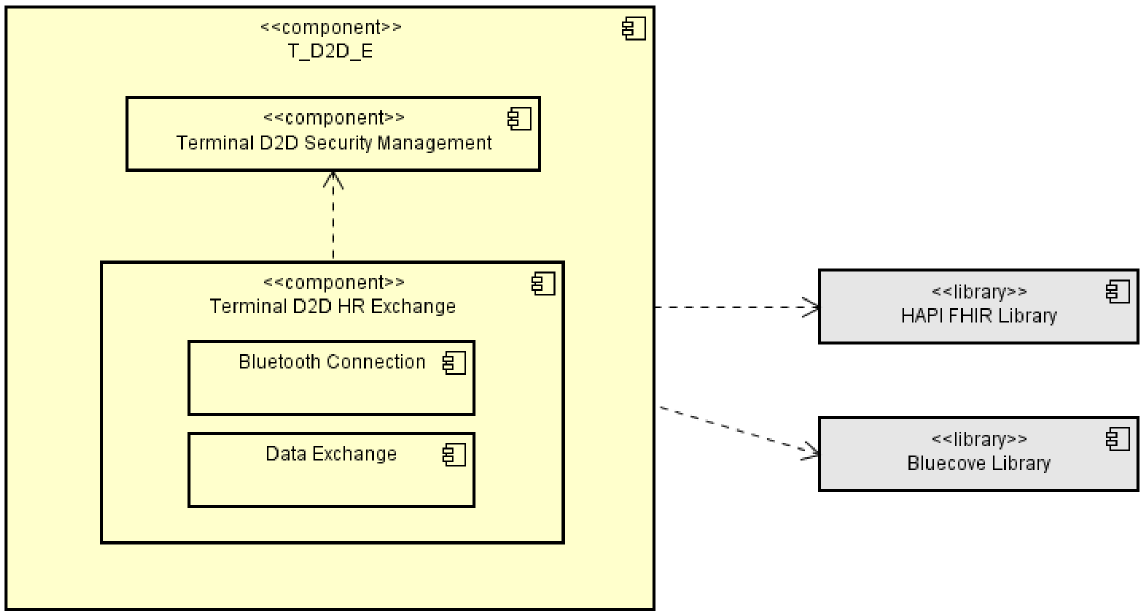


Figure 9 - T-D2D-E Third Party Libraries

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 name and the usage of the R2D 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 Library

As defined in D4.1 - Specification of remote and D2D protocol and APIs for HR exchange V1 **[D4.1]**, the R2D protocol defines the set of operations used for enabling the exchange of health data between a local or National EHR or a S-EHR Cloud and the S-EHR App with the usage of the internet. In 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 knowing all the technical details of the underlying R2D concrete protocols (FHIR or eHDSI) and technologies. The M-R2D-E library acts as a proxy for a NCP compliant to R2D specifications.

3.1.1. R2D 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 steps:

- 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.

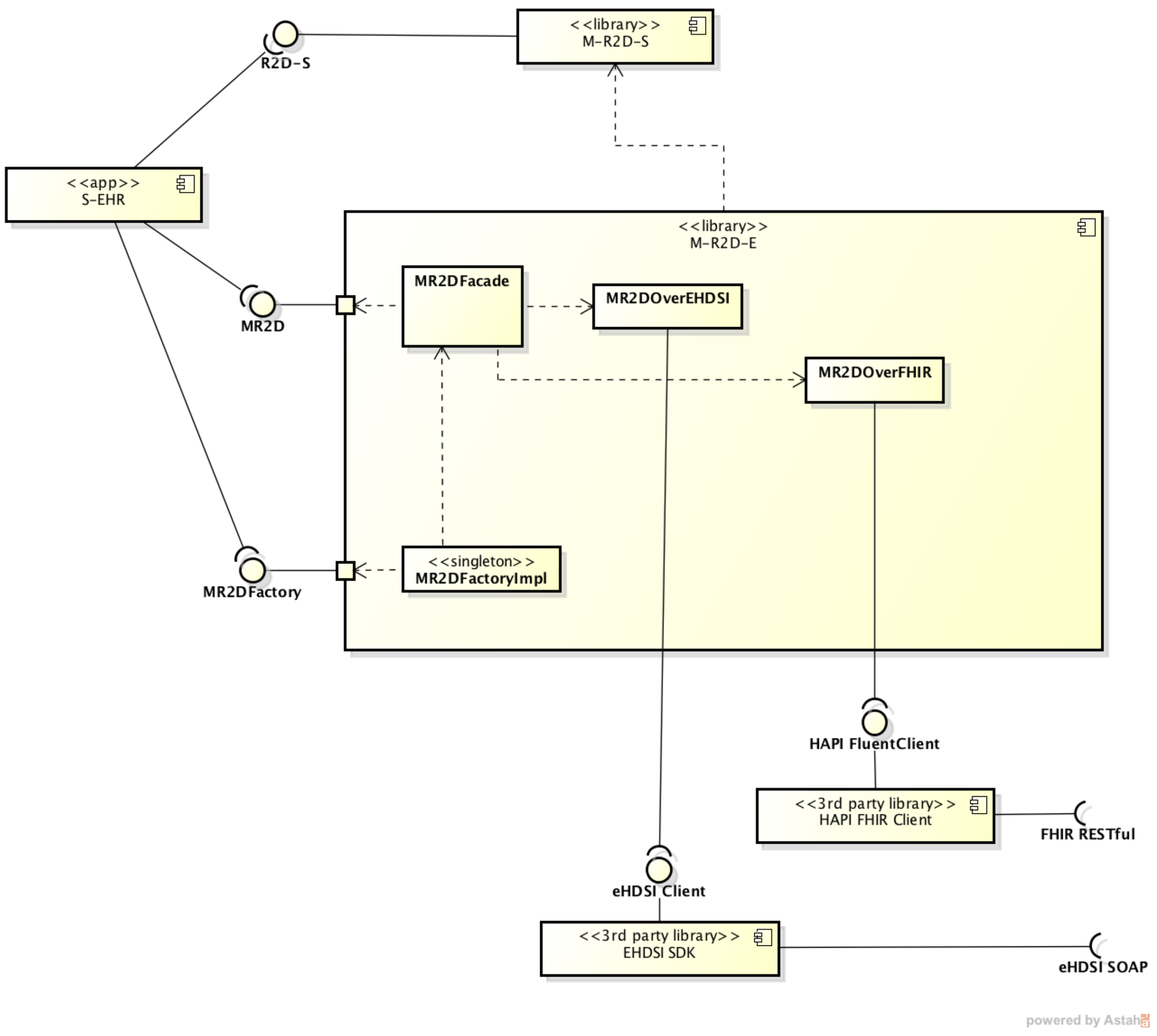


Figure 10 - M-R2D-E component diagram

In order to use the M-R2D-E library, the S-EHR app must interact mainly with two components, MR2DFactoryImpl and MR2DFacade through the interface they provide: the MR2DFactory and the MR2D. The MR2DFactory defines methods for creating instances of classes implementing the MR2D interface, while the MR2D interface defines methods for exchanging data with a NCP.
As the R2D protocol is based over two concrete protocols (FHIR and eHDSI), the main responsibility of the M-R2D-E is to convert generic requests made by the client, into concrete requests submitted to a specific NCP and previously converted in accordance to the protocol adopted by the NCP. This responsibility is delegated, by the MR2DFacade, to two specific components:

- MR2DOverFHIR: this component is able to interact with a NCP using FHIR. In order to do this, the MR2DOverFHIR component uses third party libraries provided by the HAPI FHIR open source project.
- MR2DOverEHDSI: this component is able to interact with a using eHDSI. In order to do this, the MR2DOverEHDSI component uses third party libraries containing proxy client for eHDSI / SOAP protocol.

3.1.1.2. Public Interfaces

The following diagram shows the complete definition of the interfaces MR2DFactory and R2D. The Java interface (MR2D) offered by the R2D library allows to invoke operations of R2D interface offered by the NCP node described in deliverable D4.1 (Specification of remote and D2D protocol and APIs for HR exchange). For this reason the interface offered by the library mirrors the one offered by the NCP node. The main differences between the two interfaces is the absence, on the MR2D interface, of the operation called nextBundle whose purpose was to navigate the pages that make up the results of a query. This responsibility has been moved to the class named ResourceIterator, that allows fetching the results of a query in a more natural way just by using the classic method next of an Iterator. This specific argument will be described later, in the Internal View section of this chapter.

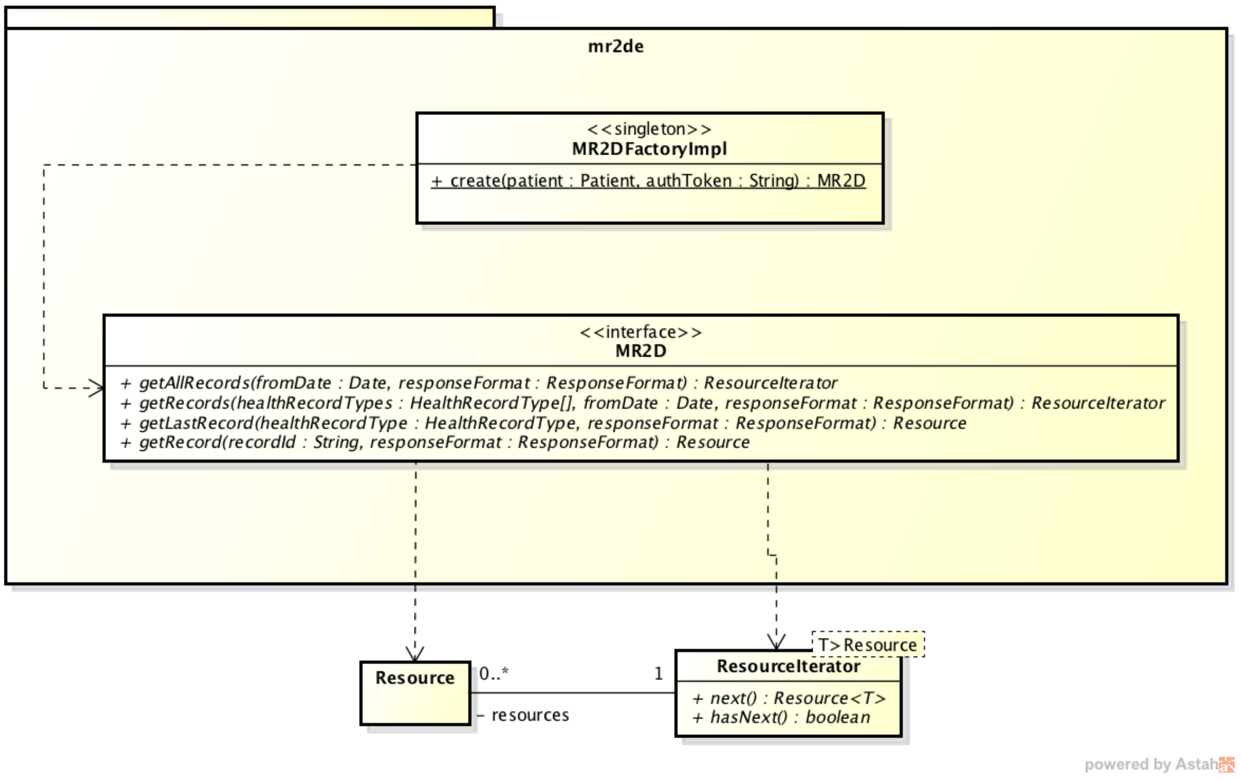

mr2de
<<singleton>>
MR2DFactoryImpl
+ create(patient : Patient, authToken : String) : MR2D
<<interface>>
MR2D
+ getAllRecords(fromDate : Date, responseFormat : ResponseFormat) : ResourceIterator
+ getRecords(healthRecordTypes : HealthRecordType[], fromDate : Date, responseFormat : ResponseFormat) : ResourceIterator
+ getLastRecord(healthRecordType : HealthRecordType, responseFormat : ResponseFormat) : Resource
+ getRecord(recordId : String, responseFormat : ResponseFormat) : Resource
Resource
0..*
1
- resources
T>Resource
ResourceIterator
+ next() : Resource<T>
+ hasNext() : boolean
powered by Astah

Figure 11 - M-R2D-E main interfaces

Interface MR2D

Defines all the methods for extracting data of a single patient from an NCP. 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 choose how to design the import functionality: retrieving 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 the logical union of all kinds of health data belonging to a citizen and stored in an NCP, allowing the client to retrieve these data in a single operation. Optionally, the client may specify a date indicating the date in 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 the logical union of some specific kinds of health data belonging to a citizen and stored in an NCP, allowing the client to retrieve the requested health data in a single operation. 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, DIAGNOSTIC_REPORT, 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 only one specific kind of health data belonging to a citizen.
Arguments	● HealthRecordType healtRecordType: an array of the predefined HealthRecordType enumeration (PATIENT_SUMMARY, PRESCRIPTION, DIAGNOSTIC_REPORT, DISCHARGE_REPORT) containing the types of health data requested by the client. ● String sessionId: a valid session token, representing the user who successfully executed the login. This sessionId is obtained directly from the platform after successful execution of login method. ● ResponseFormat responseFormat: one of the predefined ResponseFormat enumeration values (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: a valid id of a health data. ● String sessionId: a valid session token, representing the user who successfully executed the login. This sessionId is obtained directly from the platform after successful execution of login method. ● ResponseFormat responseFormat: one of the predefined ResponseFormat enumeration values (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. ● 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 MR2DFactoryImpl (Interface MR2DFactory)

MR2DFactoryImpl is a concrete class, providing static methods for creating instances of R2D. 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 MR2DFactory methods 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. ● authToken: the authorization token obtained by invoking the authentication method of M-R2D-SM library.
Return Value	An instance of a class implementing the MR2D interface.
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

The following sequence diagram shows the fundamental steps executed by S-EHR app in order to retrieve data from an NCP using the method 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 used by the S-EHR app. The internal complexity of M-R2D-E library is the focus of the second section of this chapter.

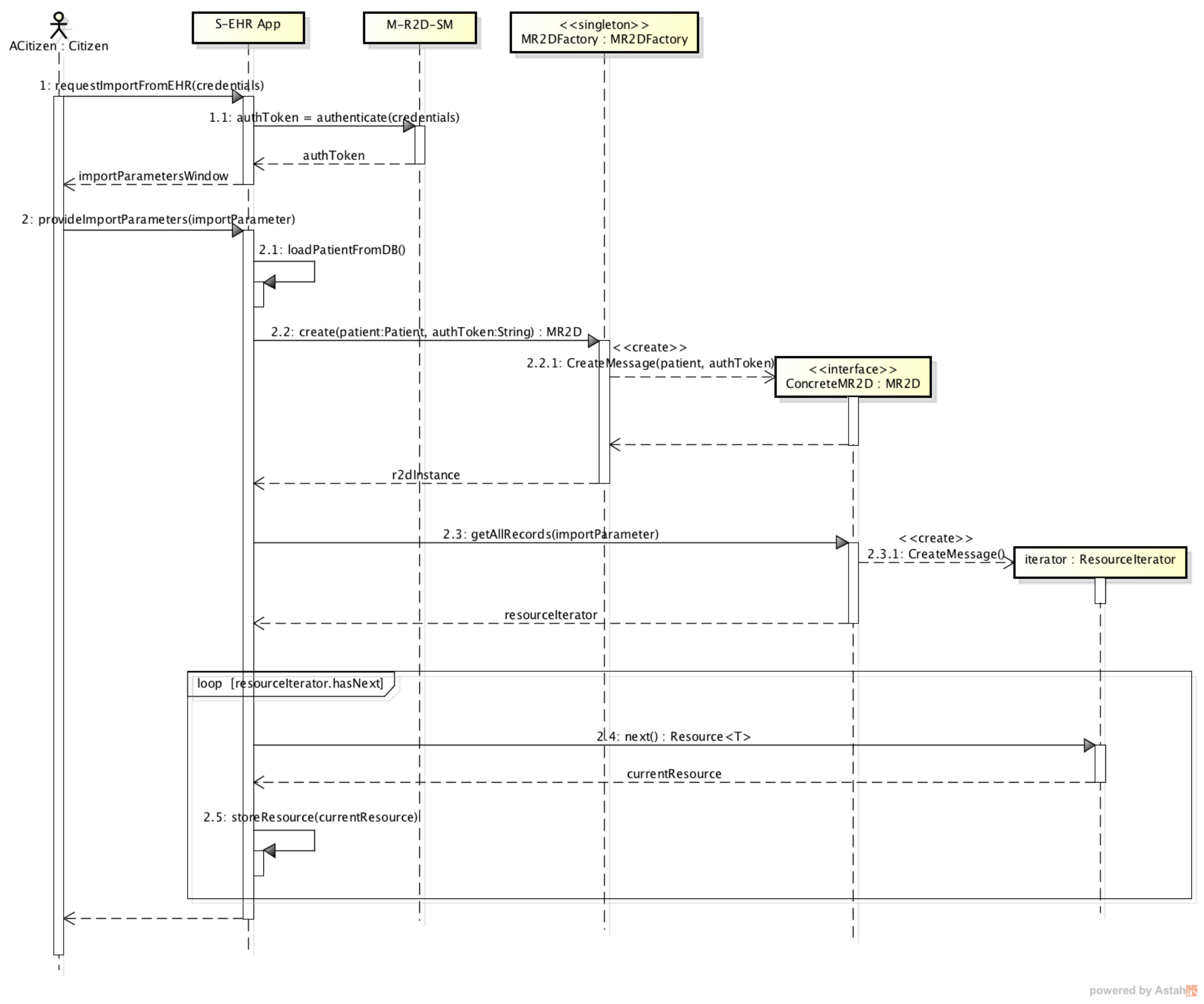

ACitizen : Citizen
S-EHR App
M-R2D-SM
<<singleton>> MR2DFactory : MR2DFactory
1: requestImportFromEHR(credentials)
1.1: authToken = authenticate(credentials)
authToken
importParametersWindow
2: provideImportParameters(importParameter)
2.1: loadPatientFromDB()
2.2: create(patient:Patient, authToken:String) : MR2D
<<create>>
2.2.1: CreateMessage(patient, authToken)
<<interface>> ConcreteMR2D : MR2D
r2dInstance
2.3: getAllRecords(importParameter)
<<create>>
2.3.1: CreateMessage()
iterator : ResourceIterator
resourceIterator
loop [resourceIterator.hasNext]
2.4: next() : Resource<T>
currentResource
2.5: storeResource(currentResource)
powered by Astah

Figure 12 - Sequence diagram for getAllRecords()

- **Step 1**: a citizen requests to S-EHR app to import his health data from the national EHR.
- **Step 1.1**: The S-EHR app starts the user authentication to the national EHR, 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 kind of health data and from which date the import should start).
- **Step 2**: the citizen chooses the right import options and then starts the importing of his health data in the S-EHR app.
- **Step 2.1**: in order to invoke the method for creating an instance of MR2D, the S-EHR app loads from the DB the instance of the Patient containing data of the authenticated citizen.
- **Step 2.2**: S-EHR app invokes the static method create() of class MR2DFactoryImpl.
- **Step 2.2.1**: the MR2DFactoryImpl uses provided parameters to detect what is the nationality of the citizen in order to connect to the right NCP using the proper connector (the connector that adopts the real protocol supported by the NCP). Before ending, this method return to S-EHR app the instance of a concrete (and private) class implementing MR2D interface.
- **Step 2.3**: S-EHR app uses the MR2D instance returned from the R2DFactoryImpl, to invoke the getAllRecords() method and providing the import parameters defined by the citizen. This method

returns an instance of ResourceIterator used by S-EHR app to iterate through the set of health data returned from the method.

- **Step 2.4**: the S-EHR app is iterating over results produced by method previously invoked. 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 2.5**: S-EHR app store the last item read from iterator to its internal database.

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

1. authenticating user to its National EHR using API of M-R2D-SM library;
2. creating instances of MR2D using API of MR2DFactory class;
3. using the MR2D instance obtained by R2Factory for submitting query to National EHR;
4. using ResourceIterator to iterate over results produced by MR2D method invocation.
5. managing health data retrieved as FHIR resource (instances of concrete classes of package org.hl7.fhir.model);

3.1.1.4. *Third Party Libraries*

(a) 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 D2.7 - Interoperability Profile Implementable Level Specification **[D2.7]**. 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.

(b) eHDSI SDK Library

The eHDSI SDK is being used to 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 Library Internal view

The internal view of M-R2D-E provides architectural details about the internal structure of the library. These details does not affect the external usage of the library described in previous section.

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 has been conceived to foster the adoption of R2D from app developers, simplifying the import health data from a national infrastructure.
In the next few years, EU Member States are expected to provide NCPs offering FHIR or eHDSI interfaces, it will be possible to download health data in a standard way from all EU Member States. M-R2D-E acts as an intermediate layer between the app and the NCP, not only hiding technical details of the two underlying protocols, but also providing some virtual operations not offered natively by the protocols. These two factors have been the most influencing factors in M-R2D-E internal design.

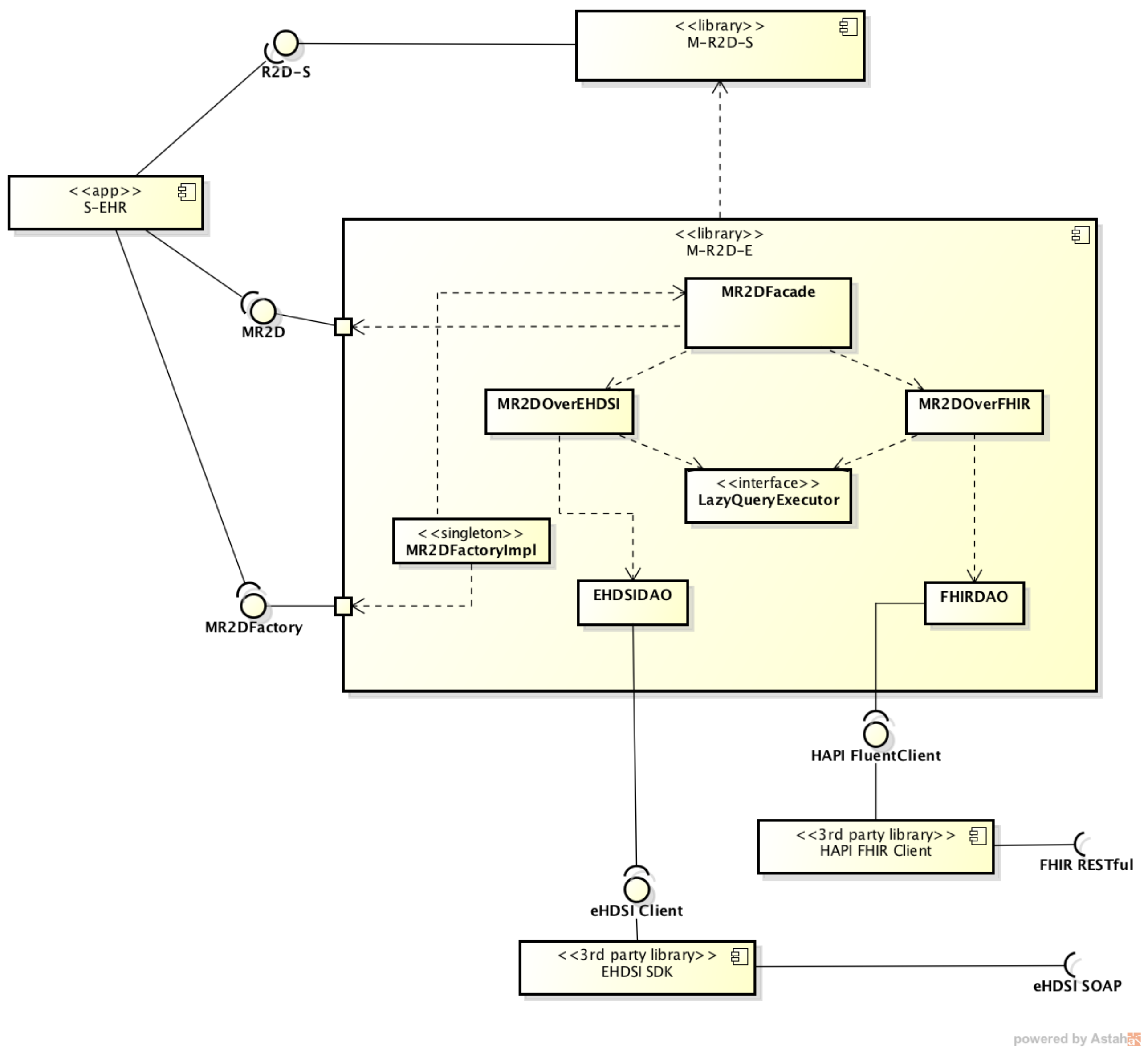


Figure 13 - Component diagram of M-R2D-E

The above diagram shows some additional components that were not shown in the previous component diagram, these components are named: LazyQueryExecutor, EHDSIDAO and FHIRDAO. Their role is to support MR2DOverFHIR and MR2DOverEHDSI to execute specific tasks over a specific protocol.

The relationships between these classes and the role that each one plays in the overall architecture is explained in the next sections of this chapter.

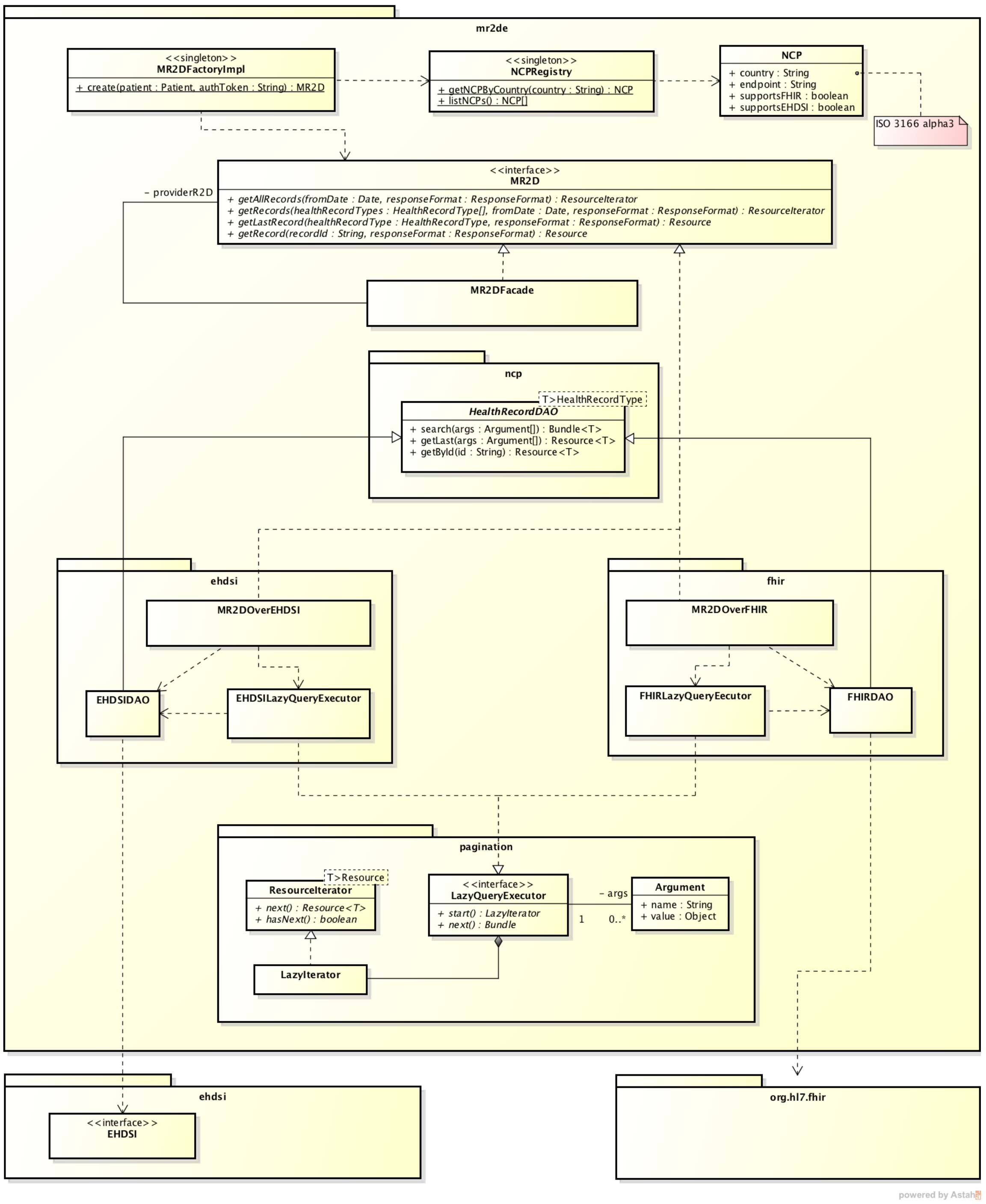

mr2de
<<singleton>>
MR2DFactoryImpl
+ create(patient : Patient, authToken : String) : MR2D
<<singleton>>
NCPRegistry
+ getNCPByCountry(country : String) : NCP
+ listNCPs() : NCP[]
NCP
+ country : String
+ endpoint : String
+ supportsFHIR : boolean
+ supportsEHDSI : boolean
ISO 3166 alpha3
- providerR2D
<<interface>>
MR2D
+ getAllRecords(fromDate : Date, responseFormat : ResponseFormat) : ResourceIterator
+ getRecords(healthRecordTypes : HealthRecordType[], fromDate : Date, responseFormat : ResponseFormat) : ResourceIterator
+ getLastRecord(healthRecordType : HealthRecordType, responseFormat : ResponseFormat) : Resource
+ getRecord(recordId : String, responseFormat : ResponseFormat) : Resource
MR2DFacade
ncp
T>HealthRecordType
HealthRecordDAO
+ search(args : Argument[]) : Bundle<T>
+ getLast(args : Argument[]) : Resource<T>
+ getById(id : String) : Resource<T>
ehdsi
MR2DOverEHDSI
EHDSIDAO
EHDSILazyQueryExecutor
fhir
MR2DOverFHIR
FHIRLazyQueryEecutor
FHIRDAO
pagination
T>Resource
ResourceIterator
+ next() : Resource<T>
+ hasNext() : boolean
<<interface>>
LazyQueryExecutor
+ start() : LazyIterator
+ next() : Bundle
- args
1
0..*
Argument
+ name : String
+ value : Object
LazyIterator
ehdsi
<<interface>>
EHDSI
org.hl7.fhir
powered by Astah

Figure 14 - Class diagram of M-R2D-E library

The MR2DFacade class acts has a dispatcher of requests to another class (one of MR2DOverFHIR and MR2DOverEHDSI) that implements the same MR2D interface and that acts as delegated class (the relationship between an instance of MR2DFacade and its delegated MR2D is stored in the attribute *providerR2D* of class MR2DFacade). The main capability of the delegated class is the ability to translate all operations provided by the MR2D interface into specific transactions executed using the rules of one of the supported protocols (the name of the classes are self-explanatory). MR2DOverFHIR and MR2DOverEHDSI are two specialized implementations of MR2D, the first is able to interact to an NCP using FHIR language, while the second is able to interact using eHDSI protocol. MR2DFacade is a concrete implementation of MR2D able to *speak* both languages.

The relationship between an instance of MR2DFacade and its delegated class is established during the execution of the create() method of the MR2DFactoryImpl class, creating this connection is the main responsibility of the MR2DFactoryImpl class. There is no method (public or private) in the current API of M-R2D-E able to break this link.

The following sequence diagram shows the creation process:

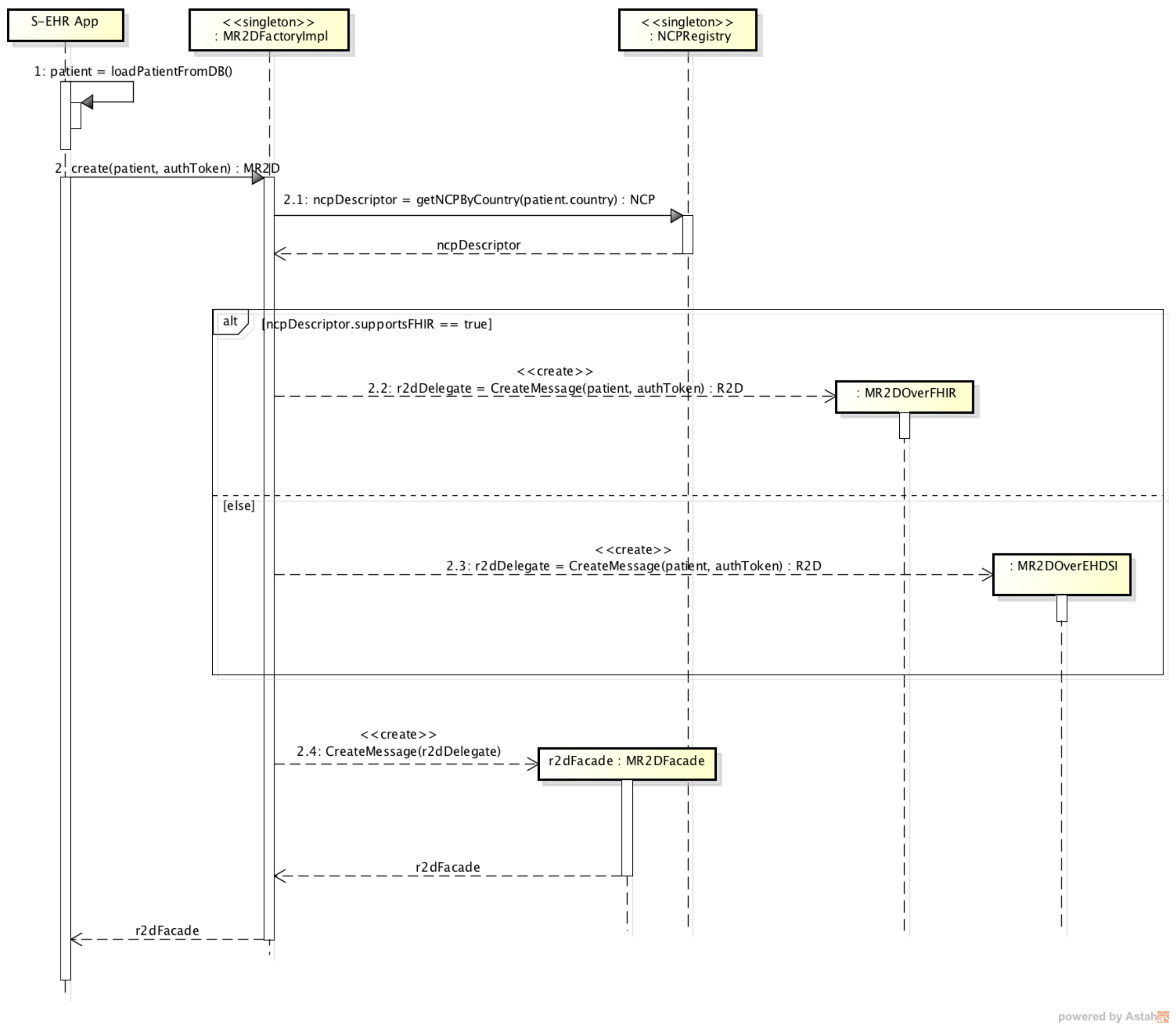

S-EHR App
<<singleton>> : MR2DFactoryImpl
<<singleton>> : NCPRegistry
1: patient = loadPatientFromDB()
2: create(patient, authToken) : MR2D
2.1: ncpDescriptor = getNCPByCountry(patient.country) : NCP
ncpDescriptor
alt
[ncpDescriptor.supportsFHIR == true]
<<create>>
2.2: r2dDelegate = CreateMessage(patient, authToken) : R2D
: MR2DOverFHIR
[else]
<<create>>
2.3: r2dDelegate = CreateMessage(patient, authToken) : R2D
: MR2DOverEHDSI
<<create>>
2.4: CreateMessage(r2dDelegate)
r2dFacade : MR2DFacade
r2dFacade
r2dFacade
powered by Astah

Figure 15 - Sequence diagram of the factory method create()

- **Step 1**: the s-EHR app creates an instance of Patient and fills it with patient's data loaded from its internal database.
- **Step 2**: the S-EHR app invokes the static method create() of class MR2DFactoryImpl, providing as arguments, the instance of Patient created at step 1, and the authentication token obtained by the authentication method executed using M-R2D-SM.
    - **Step 2.1**: MR2DFactoryImpl invokes the static method getNCPByCountry() of class NCPRegistry, in order to retrieve information about the NCP of the county of the patient.
    - **Step 2.2**: optional step executed only if the NCP supports FHIR API. The MR2DFactoryImpl creates an instance of class R2OverFHIR, providing as argument to the constructor, the patient and the authentication token.
    - **Step 2.3**: optional step executed only if the NCP does not supports FHIR API. The MR2DFactoryImpl creates an instance of class R2OverEHDSI, providing as argument to the constructor, the patient and the authentication token.
    - **Step 2.4**: The MR2DFactoryImpl creates an instance of MR2DFacade class providing as input parameter the instance of MR2D subclass created at step 2.2 / 2.3.

As shown in previous class diagram, both classes MR2DOverFHIR and MR2DOverEHDSI use specialized classes, but also some common class, in particular these two classes: LazyQueryExecutor and HealthRecordDAO. The first is used to manage the execution of complex queries that require more than one interaction with the NCP, while the second is used to translate requests into the language of the supported protocols.

The following sections provide definitions of the interfaces of LazyQueryExecutor and HealthRecordDAO, and a concrete example of their usage by means of specific sequence diagrams.

Interface LazyQueryExecutor

LazyQueryExecutor defines the interface of a class that is able to manage the execution of a complex query. In R2D a complex query is a query made up by different parts, that requires several interactions with the NCP, but provided as a single operation to S-EHR app (the complexity of the underlying set of queries is hidden to the client).
The execution of all the parts that make up a complex query is started invoking the start() method. This method returns an instance of class LazyR2DIterator (an implementation of the interface ResourceIterator) linked with the LazyQueryExecutor that creates it. When the cache of data handled by the LazyR2DIterator is close to being empty (depending on the lazy policy implemented), the LazyR2DIterator must invoke the method next() of its LazyQueryExecutor in order to asks for the next bunch of data. The LazyQueryExecutor will submit a request to the NCP asking for the next chunk of data, until all data have been fetched from the NCP.

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

Internal Operation start()

Name	start
Description	USed to start execution of a complex query (that requires more than one interaction with the NCP)
Arguments	void
Return Value	An instance of LazyR2DIterator
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.

Internal Operation next()

Name	next
Description	used to execute a part of a complex query
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 <HealtRecordType>
This abstract class defines the characteristics (interface) of a class whose purpose is to submit requests to an NCP using one 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 submit all defined requests to an NCP, using FHIR or eHDSI, and for all kind of health data managed (PATIENT_SUMMARY, PRESCRIPTION, DIAGNOSTIC_REPORT, DISCHARGE_REPORT). Searching for Prescriptions in FHIR is quite different from searching Prescriptions on eHDSI, hiding this is difference to a LazyQueryExecutor is the reason behind the conception 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 NCP 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 NCP, 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 NCP, 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.

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

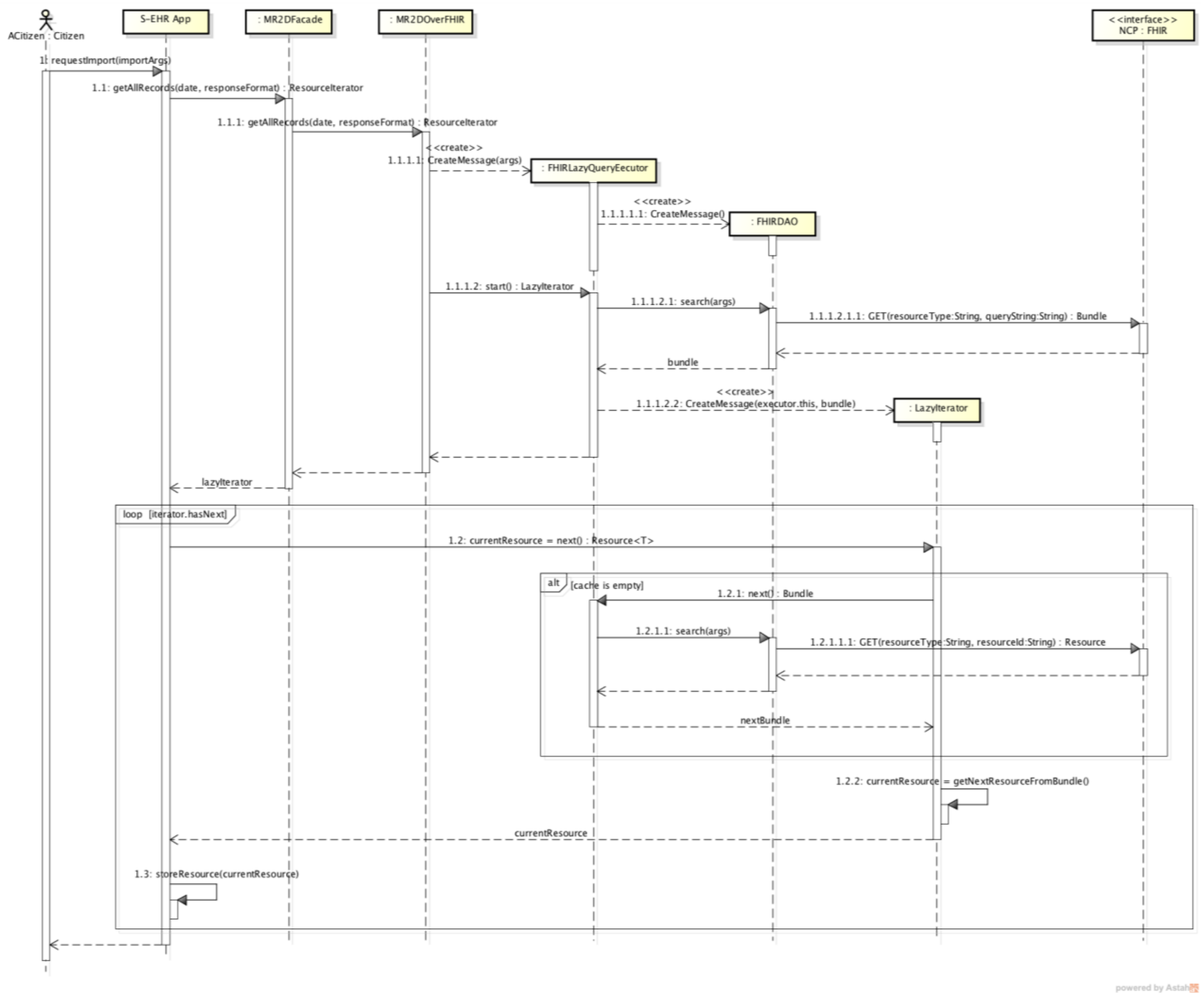
This section describes the design details of MR2DOverFHIR, that is the implementation of MR2D specialized in interacting with an NCP supporting FHIR protocol. This class in order to submit requests with FHIR protocol, collaborates with the following classes: FHIRLazyQueryExecutor and FHIRDAO, a concrete implementation of class HealthRecordDAO <HealtRecordType>.

GetAllRecords

This sequence diagram shows the collaborations between components to execute the method GetAllRecords(). The execution of this method is requested by S-EHR app in order to download all kinds of health data of the patient (patient summary, prescriptions, diagnostic reports, discharge reports) from the NCP, in a single request. FHIR specifications does not allow to execute the logical union of several queries, so this union is performed locally 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.

The GetAllRecord() method has been used as an example to show how the designed software is able to execute a multi steps query execution.

For purposes of simplicity, the following sequence diagram does not show the initial steps (showed 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.



ACitizen : Citizen
S-EHR App
: MR2DFacade
: MR2DOverFHIR
<<interface>> NCP : FHIR
1: requestImport(importArgs)
1.1: getAllRecords(date, responseFormat) : ResourceIterator
1.1.1: getAllRecords(date, responseFormat) : ResourceIterator
<<create>>
1.1.1.1: CreateMessage(args)
: FHIRLazyQueryEecutor
<<create>>
1.1.1.1.1: CreateMessage()
: FHIRDAO
1.1.1.2: start() : LazyIterator
1.1.1.2.1: search(args)
1.1.1.2.1.1: GET(resourceType:String, queryString:String) : Bundle
bundle
<<create>>
1.1.1.2.2: CreateMessage(executor.this, bundle)
: LazyIterator
lazyIterator
loop [iterator.hasNext]
1.2: currentResource = next() : Resource<T>
alt [cache is empty]
1.2.1: next() : Bundle
1.2.1.1: search(args)
1.2.1.1.1: GET(resourceType:String, resourceId:String) : Resource
nextBundle
1.2.2: currentResource = getNextResourceFromBundle()
currentResource
1.3: storeResource(currentResource)
powered by Astah

Figure 16 - Sequence diagram of GetAllRecords method executed with FHIR protocol

- **Step 0** (not shown in the sequence diagram): the citizen has performed the authentication and has created an instance of MR2D (MR2DOverFHIR) using the method R2FactoryImpl.create().
- **Step 1**: the citizen asks for the import of his health data from the national EHR, and provides import options needed by the app.
- **Step 1.1**: the S-EHR app invokes method getAllRecords() on the instance of MR2D previously created.
    - **Step 1.1.1**: the MR2DFacade dispatches the request to the specific MR2D provider, an instance of MR2DOverFHIR.
    - **Step 1.1.1.1**: MR2DOverFHIR creates an instance of FHIRLazyQueryExecutor providing as input parameter an array of Argument (derived by the import options chosen by the citizen.
    - **Step 1.1.1.1.1**: FHIRLazyQueryExecutor creates all the instances of FHIRDAO that it needs in order to perform the request.
    - **Step 1.1.1.2**: MR2DOverFHIR invokes method start() of FHIRLazyQueryExecutor, starting the execution of the complex query.
    - **Step 1.1.1.2.1**: FHIRLazyQueryExecutor invokes method search() of the class HealthRecordDAO, starting the search of the requested kind of health data of the citizen.

  - **Step 1.1.1.2.1.1**: the HealthRecordDAO creates the corresponding FHIR request and submit it to the NCP. The obtained Bundle is returned to the caller;
  - **Step 1.1.1.2.2**: creates an instance of LazyIterator providing the bundle as input parameter. The iterator is returned to the caller, the first part of the getAllRecords has been executed, the S-EHR app now can iterate over the results.
- **Step 1.2**: S-EHR stars iterating over query results, invoking the method next() of ResourceIterator.
  - **Step 1.2.1** (executed optionally only if the cache is empty): the LazyIterator invokes the method next() of its FHIRLazyQueryExecutor, asking for the next bunch of data.
  - **Step 1.2.1.1**: FHIRLazyQueryExecutor invokes method search() of the class HealthRecordDAO, requesting the next page of data.
  - **Step 1.2.1.1.1**: the HealthRecordDAO creates the corresponding FHIR request and submit it to the NCP. The obtained Bundle is returned to the caller (LazyIterator);
  - **Step 1.2.2**: the LazyIterator returns the current instance of Resource from the Bundle
- **Step 1.3**: the S-EHR app stores the resource in its database.

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.

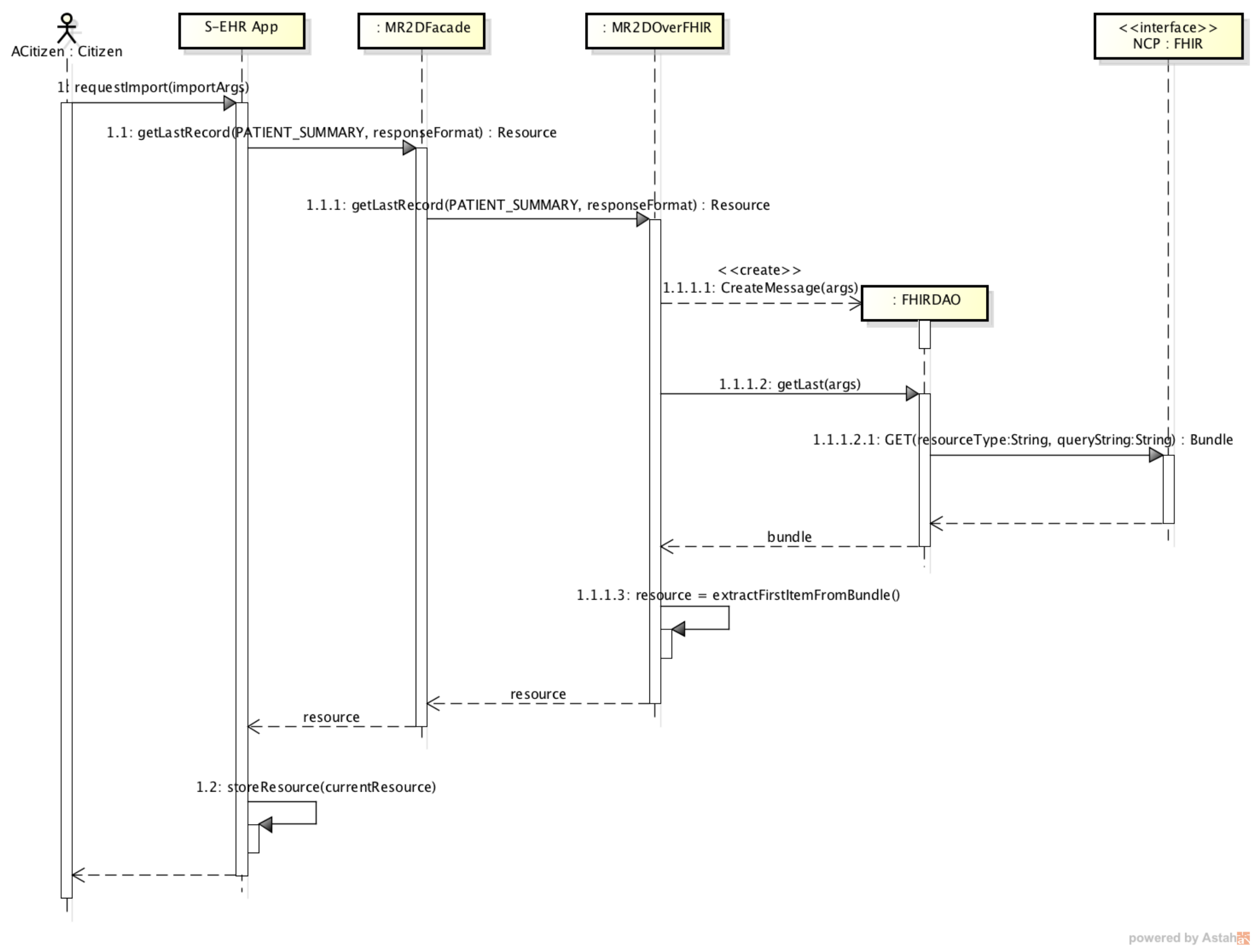

ACitizen : Citizen
S-EHR App
: MR2DFacade
: MR2DOverFHIR
<<interface>> NCP : FHIR
1: requestImport(importArgs)
1.1: getLastRecord(PATIENT_SUMMARY, responseFormat) : Resource
1.1.1: getLastRecord(PATIENT_SUMMARY, responseFormat) : Resource
<<create>> 1.1.1.1: CreateMessage(args)
: FHIRDAO
1.1.1.2: getLast(args)
1.1.1.2.1: GET(resourceType:String, queryString:String) : Bundle
bundle
1.1.1.3: resource = extractFirstItemFromBundle()
resource
resource
1.2: storeResource(currentResource)
powered by Astah

Figure 17 - Sequence diagram of GetLastRecord method executed with FHIR protocol

- **Step 0** (not shown in the sequence diagram): the citizen has performed the authentication and has created an instance of MR2D (MR2DOverFHIR) using the method R2FactoryImpl.create().
- **Step 1**: the citizen asks for the import of his patient summary from the national EHR, and provides import options needed by the app.
- **Step 1.1**: the S-EHR app invokes method getLastRecord() on the instance of MR2D previously created.
    - **Step 1.1.1**: the MR2DFacade dispatches the request to the specific MR2D provider, an instance of MR2DOverFHIR.
    - **Step 1.1.1.1**: MR2DOverFHIR creates an instance of HealthRecordDAO to perform the specific request.
    - **Step 1.1.1.2**: MR2DOverFHIR invokes method getLast() of the class HealthRecordDAO, providing the needed input parameters as an array of Argument.
    - **Step 1.1.1.2.1**: the HealthRecordDAO creates the corresponding FHIR request and submit it to the NCP. The obtained Bundle is returned to the caller;
    - **Step 1.1.1.3**: MR2DOverFHIR extracts the first item from the bundle and returns it to the caller.
- **Step 1.2**: the S-EHR app stores the resource in its database

3.1.2.2. MR2DOverEHDSI

This section describes the design details of class MR2DOverEHDSI, that is the implementation of MR2D specialized in interacting with an NCP supporting the eHDSI protocol. This class, in order to submit requests with eHDSI protocol, collaborates with the following classes: EHDSILazyQueryExecutor and EHDSIDAO a concrete implementation of class HealthRecordDAO <HealtRecordType>.

This section shows the sequence diagrams of the same two methods showed in the previous section(getAllRecords() and getLastRecord()) but implemented for eHDSI.

GetAllRecords

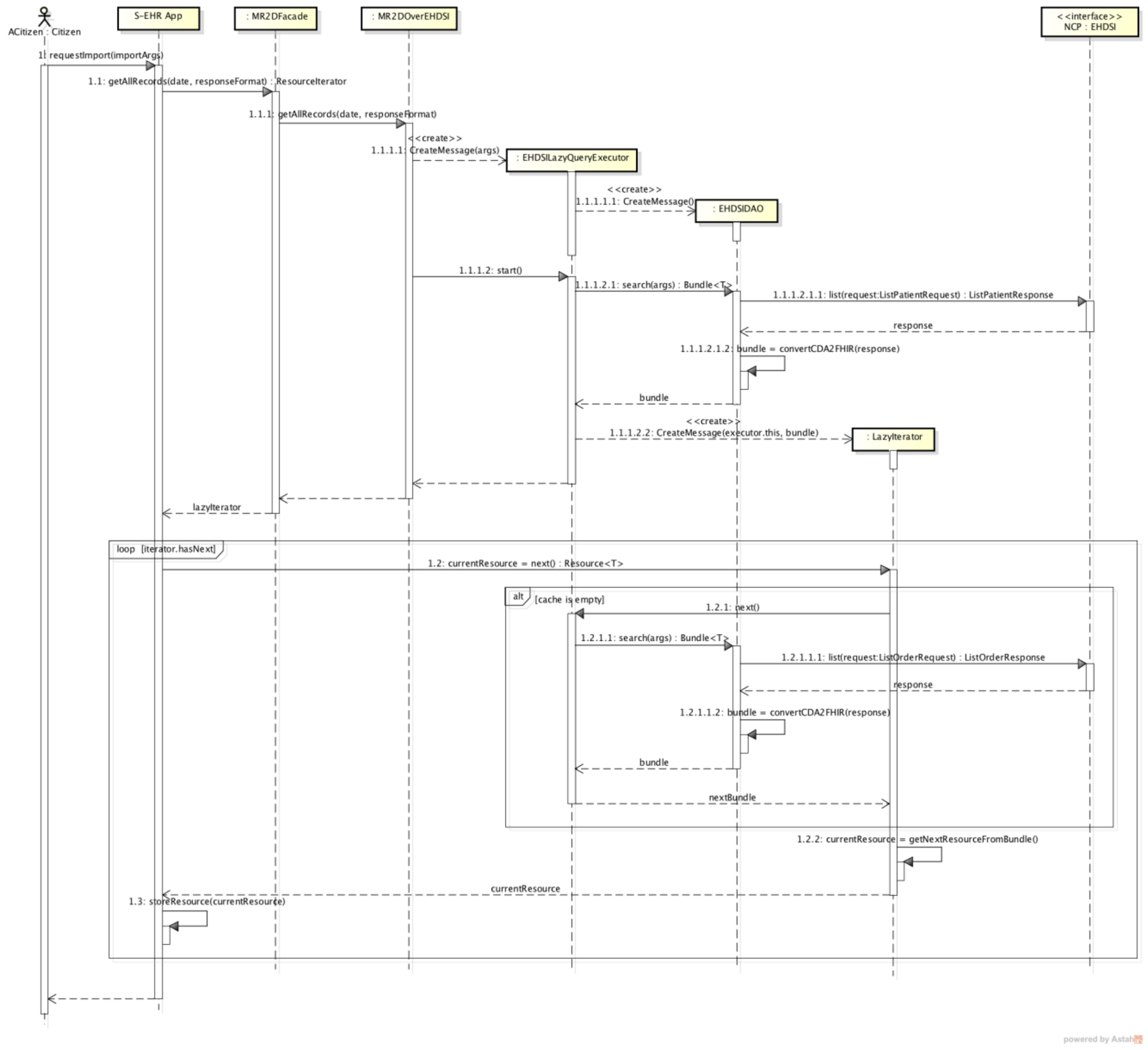

ACitizen : Citizen
S-EHR App
: MR2DFacade
: MR2DOverEHDSI
<<interface>> NCP : EHDSI
1: requestImport(importArgs)
1.1: getAllRecords(date, responseFormat) : ResourceIterator
1.1.1: getAllRecords(date, responseFormat)
<<create>> 1.1.1.1: CreateMessage(args)
: EHDSILazyQueryExecutor
<<create>> 1.1.1.1.1: CreateMessage()
: EHDSIDAO
1.1.1.2: start()
1.1.1.2.1: search(args) : Bundle<T>
1.1.1.2.1.1: list(request:ListPatientRequest) : ListPatientResponse
response
1.1.1.2.1.2: bundle = convertCDA2FHIR(response)
bundle
<<create>> 1.1.1.2.2: CreateMessage(executor.this, bundle)
: LazyIterator
lazyIterator
loop [iterator.hasNext]
1.2: currentResource = next() : Resource<T>
alt [cache is empty]
1.2.1: next()
1.2.1.1: search(args) : Bundle<T>
1.2.1.1.1: list(request:ListOrderRequest) : ListOrderResponse
response
1.2.1.1.2: bundle = convertCDA2FHIR(response)
bundle
nextBundle
1.2.2: currentResource = getNextResourceFromBundle()
currentResource
1.3: storeResource(currentResource)
powered by Astah

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

- **Step 0** (not shown in the sequence diagram): the citizen has performed the authentication and has created an instance of MR2D (MR2DOverEHDSI) using the method R2FactoryImpl.create().

- **Step 1**: the citizen asks for the import of his health data from the national EHR, and provides import options needed by the app.
- **Step 1.1**: the S-EHR app invokes method getAllRecords() on the instance of MR2D previously created.
    - **Step 1.1.1**: the MR2DFacade dispatches the request to the specific MR2D provider, an instance of MR2DOverEHDSI.
    - **Step 1.1.1.1**: MR2DOverEHDSI creates an instance of EHDSILazyQueryExecutor providing as input parameter an array of Argument (derived by the import options chosen by the citizen.
    - **Step 1.1.1.1.1**: EHDSILazyQueryExecutor creates all the instances of HealthRecordDAO that it needs in order to perform the request.
    - **Step 1.1.1.2**: MR2DOverEHDSI invokes method start() of EHDSILazyQueryExecutor, starting the execution of the complex query.
    - **Step 1.1.1.2.1**: EHDSILazyQueryExecutor invokes method search() of the class HealthRecordDAO, starting the search of the requested kind of health data of the citizen.
        - **Step 1.1.1.2.1.1**: the HealthRecordDAO creates the corresponding EHDSI request and submit it to the NCP.
        - **Step 1.1.1.2.1.1**: the results in CDA format, are converted to FHIR format, the obtained Bundle is returned to the caller.
    - **Step 1.1.1.2.2**: creates an instance of LazyIterator providing the bundle as input parameter. The iterator is returned to the caller, the first part of the getAllRecords has been executed, the S-EHR app now can iterate over the results.
- **Step 1.2**: S-EHR stars iterating over query results, invoking the method next() of ResourceIterator.
    - **Step 1.2.1** (executed optionally only if the cache is empty): the LazyIterator invokes the method next() of its FHIRLazyQueryExecutor, asking for the next bunch of data.
    - **Step 1.2.1.1**: FHIRLazyQueryExecutor invokes method search() of the class HealthRecordDAO, requesting the next page of data.
    - **Step 1.2.1.1.1**: the HealthRecordDAO creates the corresponding EHDSI request and submit it to the NCP. The obtained results are converted to CDA and then returned to the caller (LazyIterator);
    - **Step 1.2.2**: the LazyIterator returns the current instance of Resource from the Bundle
- **Step 1.3**: the S-EHR app stores the resource in its database.

GetLastRecord

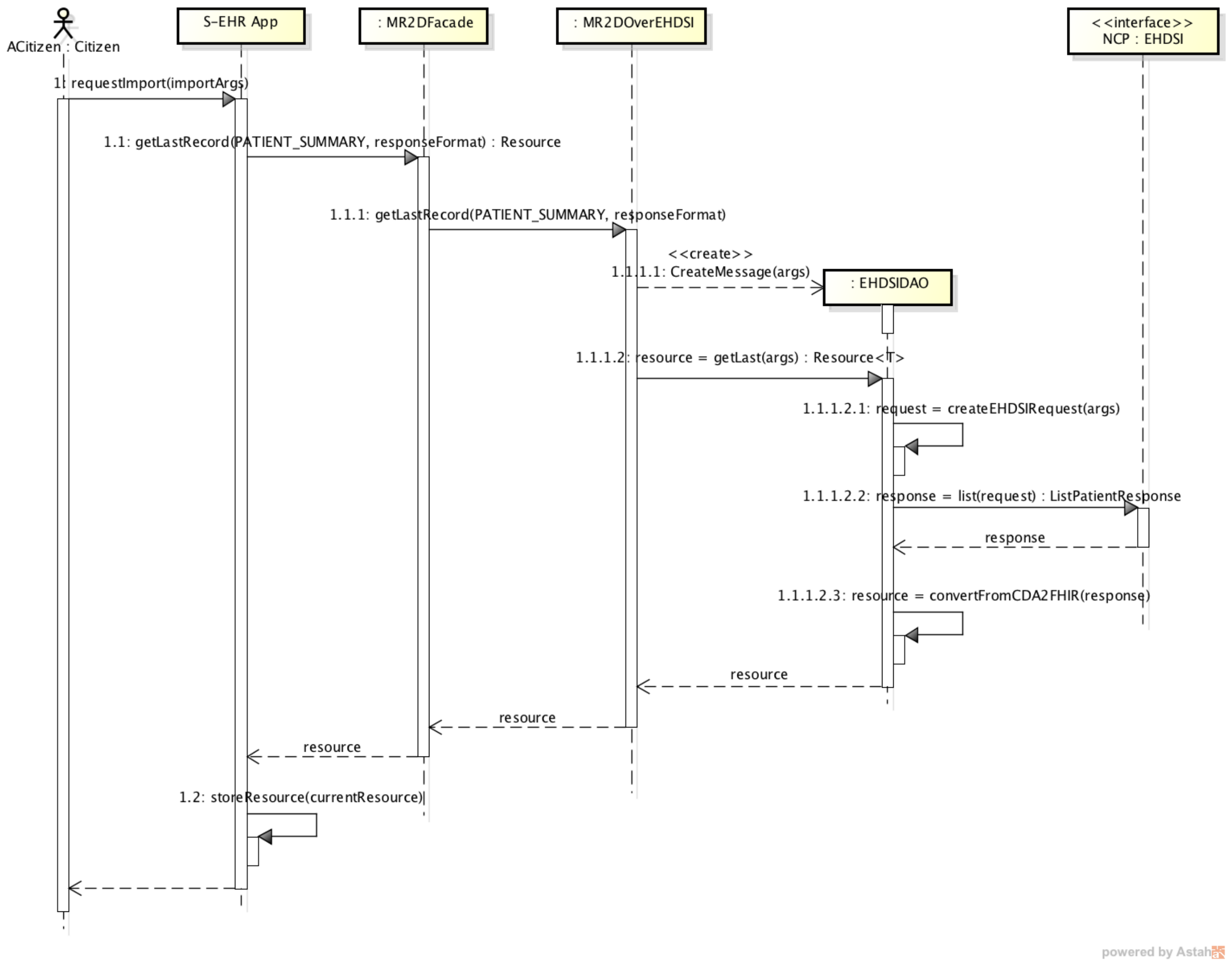

ACitizen : Citizen
S-EHR App
: MR2DFacade
: MR2DOverEHDSI
<<interface>> NCP : EHDSI
1: requestImport(importArgs)
1.1: getLastRecord(PATIENT_SUMMARY, responseFormat) : Resource
1.1.1: getLastRecord(PATIENT_SUMMARY, responseFormat)
<<create>> 1.1.1.1: CreateMessage(args)
: EHDSIDAO
1.1.1.2: resource = getLast(args) : Resource<T>
1.1.1.2.1: request = createEHDSIRequest(args)
1.1.1.2.2: response = list(request) : ListPatientResponse
response
1.1.1.2.3: resource = convertFromCDA2FHIR(response)
resource
resource
resource
1.2: storeResource(currentResource)
powered by Astah

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

- **Step 0** (not shown in the sequence diagram): the citizen has performed the authentication and has created an instance of MR2D (MR2DOverEHDSI) using the method R2FactoryImpl.create().
- **Step 1**: the citizen asks for the import of his patient summary from the national EHR, and provides import options needed by the app.
- **Step 1.1**: the S-EHR app invokes method getLastRecord() on the instance of MR2D previously created.
    - **Step 1.1.1**: the MR2DFacade dispatches the request to the specific MR2D provider, an instance of MR2DOverEHDSI.
    - **Step 1.1.1.1**: MR2DOverEHDSI creates and configure an instance of HealthRecordDAO to perform the specific request.
    - **Step 1.1.1.2**: MR2DOverEHDSI invokes method getLast() of the class HealthRecordDAO, providing the needed input parameters as an array of Argument.
        - **Step 1.1.1.2.1** and **Step 1.1.1.2.2**: the HealthRecordDAO creates the corresponding EHDSI request and submit it to the NCP.
        - **Step 1.1.1.2.3**: the HealthRecordDAO converts the CDA results in FHIR, the obtained Bundle is returned to the caller.

- ○ **Step 1.1.1.3**: MR2DOverFHIR extracts the first item from the bundle and returns it to the caller.
- **Step 1.2**: 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 by data 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 D2.7 - FHIR profile for EHR interoperability - V1 **[D2.7]**.

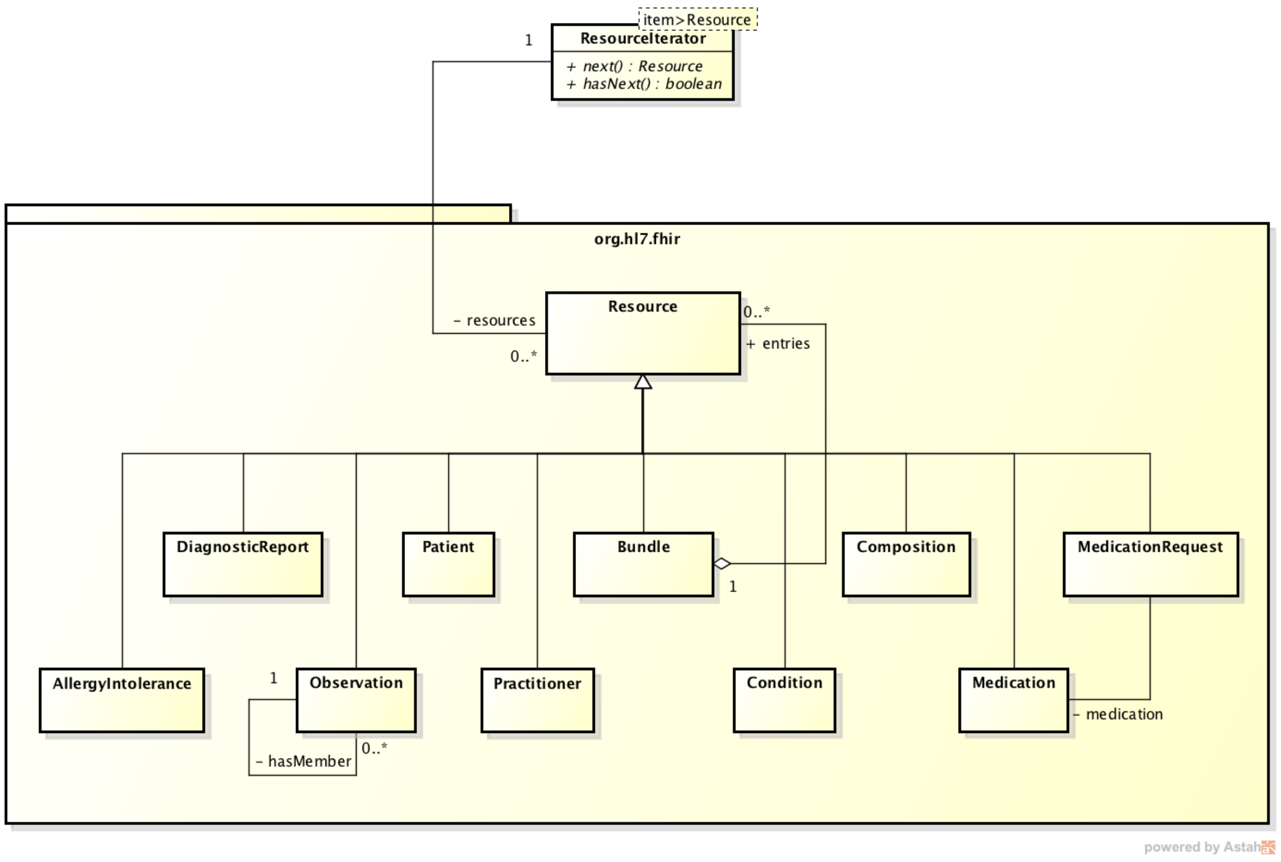


Figure 20 - D2D and R2D Data Model

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 initial 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 first draft of the intended content of the libraries and their further functionality purposes. Following this draft, two updates of this report are planned to be released. The first update is planned to be released on December 2020, whilst the second update is planned to be released on December 2021, both of them including the relevant 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. In the next version of the design of libraries for remote and D2D HR exchange, based on the current implementation, the needs as well as the additional functionalities that will be required, a new version of the libraries' design will be released for the intended audience.

REFERENCES

- **[D2.7]** InteropEHRate Consortium, *D2.7 - FHIR profile for EHR interoperability - V1*,2019. www.interopehrate.eu/resources
- **[D3.9]** InteropEHRate Consortium, *D3.9 - Design of libraries for HR security and privacy services - V1*,2019. www.interopehrate.eu/resources
- **[D4.1]** InteropEHRate Consortium, *D4.1 - Specification of remote and D2D protocol and APIs for HR exchange - V1*,2019. 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/**